# Private, Distributed, and Searchable Content Providers

by

Binh Vo

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

COLUMBIA UNIVERSITY

June 2015

**Private, Distributed, and Searchable Content Providers**

by

Binh Vo

Submitted to the Department of Computer Science
on Jun 19, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

## Abstract

In this thesis I will show that, by leveraging efficient data structures and algorithms for indexing and secure computation, we can create practical systems for anonymous and private search, communication, and content distribution. I will improve and extend existing work in private search, which only addresses the problem where a client stores his own data encrypted on a server and wishes to be able to search his records remotely without revealing the their content. I do so by addressing a broader scenario, in which one or more servers store their own data, and a number of users wish to be able to issue queries across these records, without the server learning about the types of queries users are running, and without users learning anything about the remote databases besides the results of their searches. I also improve upon the field of anonymous communication systems, where prior systems focused on addressed communication in a unicast setting. I will discuss how we can create anonymous communication systems that work on a publish-subscribe basis, allowing communication to reach many people while solving the issue of how to establish communication without prior relationships.

Next, I will discuss anonymous credential systems, and how to make them feasible for real-world scenarios. These systems can be useful for anonymously enforcing policies and managing privileges on a per-user basis. Our final challenge is to provide a scalable anonymous communication system that can deliver our queries while maintaining our privacy requirements. I will do this using a publish-subscribe architecture.

I will show how all of these advancements can be accomlished by leveraging Bloom Filters, Onion Routing, Re-routable Encryption, and Yao Garbled Circuits to create anonymity preserving systems that operate in real time.

# Contents

# Chapter 1

# Introduction

In this thesis, we focus generally on the problem of private content sharing systems. Specifically, we will examine systems for anonymous and/or private search, communication, and authentication. We will extend the envelope of what can be accomplished functionally by these systems while still protecting user identities, and in doing so create the tools we need to create a new system that enable private, distributed, and searchable content providers. These systems allow users to make data publicly available, while protecting personal information. This can cover a range of possible things: the identity of the providers, the identities of recipients receiving the content, the nature of the content being requested or that of the remaining available content. We aim to develop a private content sharing systems that also support complex anonymous queries broadcast to the network so that users can find content that they are interested in. To accomplish this, we design various new systems for private search, anonymous communication, and anonymous credentials.

Chapter 2 will formally lay out the problem we are aiming to solve and concretely define our goals and requirements.

Chapter 3 will discuss related works.

Chapter 4 will discuss existing systems and algorithms that we will use as building blocks to build our own solutions.

Chapter 5 describes approaches for anonymous database search. These cover scenarios wherein a single server has data it wishes to make available for search to

users, while preventing them from learning information other than the results of their searches, and also preventing the server from learning the content of their queries. Such systems form the basis for anonymously locating and retrieving data of interest.

Chapter 6 describes systems for anonymous publish-subscribe. These allow multicast routing of messages based on topic rather than individual addressing. They work on a push basis and prevent users from determining the publishers of messages they receive, or the subscribers who receive the messages they publish. Such systems provide new and more flexible means for anonymously communicating with unknown recipients, and can be used to route messages for many different kinds of applications

Chapter 7 describes a system for providing anonymous and revocable credentials. While many such systems existed prior, we will discuss how to achieve properties that are necessary to make them practical for real world applications.

Finally, chapter 8 describes how we will use the previously described systems to create search systems that support multi-user and multi-content-provider scenarios.

## 1.1 Motivations for private content sharing systems

With increasing amounts of censorship and privacy concerns on the internet there is a growing need for identity anonymization and protected communication. This applies to general communication, content sharing, identity management, and search of private data. Anonymous content sharing systems can be used to replace newsgroups or article databases for sensitive topics such as medical issues or political sensitive topics like Falun Gong published materials which are often censored by their government.

For example, users with sensitive illnesses such as the HIV positive may wish to reach out to others, or talk about their experiences, without revealing their status to others in their personal life or on a public forum. Similarly, those reading their content may also wish to protect their identities and hide their interest in these subjects.

There are existing systems for anonymous publishing. One well known system for widescale document distribution is the Free Haven project [35], a peer-to-peer file-sharing system. In this system, users can publish documents, making them freely

available without revealing their identities. It is based on a community of servers that distribute storage of split shares of published documents. Recipients broadcast requests throughout the storage space, and those with pieces of interest return them encrypted. The documents are associated with private key encryption pairs to maintain ownership through updates and deletions. This system only protects the publisher, and not those searching or receiving the published material.

Another similar system is FreeNet [28], also a peer-to-peer file-sharing system, but one with routed document requests rather than universally broadcast ones. Documents are associated with hashes of descriptive keyword strings, and migrated over time so that similar documents tend to migrate to geographically close servers on the network topology. Queries are then sent on a hill-climbing search over these lexical hashes. It is more scalable, and has more flexible document retrieval (keyword search rather than simple unique-name lookup) than FreeHaven, however it does not protect recipient identity, only that of the document owners.

We will aim to provide a system with more comprehensive privacy guarantees, protecting the identities of both publisher and recipient, and also the nature of content being searched for.

## 1.2 Improvements of existing systems and new building blocks

Private search systems enable a user to search a database that resides remotely without revealing the content of his query to the remote server. We will further aim to protect the identity of the querier from the server. While many existing scenarios assume the querier is searching his own data that he has encrypted and stored remotely, we will address the scenario where the data is owned by the remote entity, who wishes to make it available for public search. This can be considered the atomic part of our system which we will then try to scale to large numbers of queriers issuing searches to large numbers of servers. We will improve these with anonymous creden-

tial systems that allow users to obtain credentials that they can later anonymously prove ownership of to enforce per-user policies.

An anonymous publish-subscribe communication network allows users to anonymously route messages, while addressing them by topic rather than identity. In other words, instead of senders choosing the recipients of their transmissions, recipients can instead subscribe to topics of interest that senders publish messages to. Because of this paradigm, it is not necessary for publishers and subscribers to have any prior relationship, or indeed to even know of each other's existence. This is a natural fit for anonymous communications.

## 1.3 Design of private content sharing systems

By combining these building blocks, we will construct a distributed, anonymous content sharing network. This network will allow users to publish content, make it searchable so that consumers can find content of interest to them, and control the incoming searches, all while protecting the identities of both parties. We will aim to provide both query functionality and content retrieval in such a way that both the identities of the consumers and providers are protected from each other and from other parties. We will also protect the nature of the queries, the nature of remaining content unrelated to the queries, and the specifics of the resulting documents themselves. We will also aim to enable providers to enforce policies on the types of queries they will allow in order to protect their content in the face of an anonymous querier who could otherwise potentially be asking for any or all of his data. Although there are numerous anonymous publishing systems, there are none that provide all of the protections we will aim to, as well as supporting anything but the most simple of queries.

Applying this idea to our earlier example of discussion groups for sensitive medical conditions, we can imagine what kinds of issues we would need to address and what kinds of attackers we would need to defend against. On the one hand, we have legitimate users, those who are trying to either share articles of interest, or those

receiving them or searching for them. On the other we have those who are trying to interfere with this operation; denial of service would be one obvious example of this, but a less obvious one may be simply antagonizing a community with false or disruptive articles, which may prompt us to want some form of accountability or blacklistability. We also have those who wish to de-anonymize, to either identify queriers or data owners, or to examine some of the queries or content they should not be privy to.

# Chapter 2

# Problem Statement

The hypothesis of this research is as follows: a system can be created that enables users to share content in a distributed and anonymous fashion, with certification authority-regulated search capability. This will be made possible by creating and combining new systems for secure database search, anonymous credentials, and anonymous publish-subscribe routing.

We will discuss the following types of private content sharing systems:

- A private search system that allows a server to host an index that queriers can issue queries to remotely. The queriers identities should not be revealed to the server in doing so, and the content of the queries should be protected from the server. The system should allow the server to enforce policies, which themselves are not revealed to the querier.

- An anonymous publish-subscribe network. Users should be able to subscribe to topics of interest, thus receiving messages which are published to matching topics. Both senders and receivers should remain anonymous through these interactions.

- A system that allows users to anonymously obtain and demonstrate ownership of credentials. The system should support multiple organizations that can assign credentials to individuals they trust, while protecting the anonymity of those owners from others.

- A distributed private search system that supports multiple hosts in a network. Users will be able to issue queries anonymously into the system and fetch results from matching hosts with neither party learning the identity of the other.

## 2.1 Private Search requirements

The general scenario that we consider includes multiple parties who possess private sensitive data, which they are willing to share under certain very specific circumstances. In particular, they will permit secure, anonymous search between two parties which we call the querier (or user) and the server. Specifically, it must be possible for a party to submit keywords and find out which documents in the database of another party contain these keywords. The security and privacy guarantees of the functionality should be that the server does not learn the query nor does the user learn anything more about the database than the relevant search results. In addition, only authorized users are allowed to search a particular server's database; at the same time the server should not find out the identity of the querying party. Both new authorizations and revocations should be possible in the system.

In order to handle the conflicting needs of user anonymity and authorization, we add a third party who is trusted to serve as an intermediary in the communication path between the two parties. The trusted party is to be trusted to know and protect the identities of the participants and enforce correct authorization before allowing queries to reach the server. However, we still aim to protect the content of the queries and results from the trusted party, which does not need this information to carry out its responsibilities.

As a searchable database system we thus need to support the following functions:

- `Create(R, d)`*:* Database $d$ is instantiated with a list of records $R$.
- `Insert(r, d)`*:* Record $r$ is added to database $d$.
- `Delete(i, d)`*:* The record at index $i$ is removed from database $d$.
- `Update(i, r, d)`*:* The record at index $i$ in database $d$ is replaced with $r$.

- `Query(u, q, d)`*:* User $u$ issues query $q$ against $d$

And as a private search system, we must guarantee certain properties:

**Definition 1.** Querier anonymity: *Let $d$ be the owner of a database, and $c$ be a client issuing a query $q$. Let $H$ be the set of $h$ honest clients in the system (including $c$). Let $A$ be any collaboration of entities not in $H$, including other clients, $d$, entities related to the operation of the system, and outside observers. These entities may issue any number and type of queries, and observe the results. $A$ cannot then identify $p$ given $m$ with probability greater than $\frac{1}{h}$.*

This captures the idea that agents outside of a querier against the database should be able to gain an advantage in identifying them beyond what they would be able to do via random guessing.

**Definition 2.** Querier privacy: *Let $d$ be the owner of a database, and $c$ be a client issuing a query $q$. Let $H$ be the set of $h$ honest clients in the system (including $c$). Let $A$ be any collaboration of entities not in $H$, including other clients, $d$, entities related to the operation of the system, and outside observers. These entities may issue any number and type of queries, and observe the results. $A$ cannot then determine the contents of $q$.*

This captures the idea that a query's contents should not be revealed to anyone in the system beyond the querier themselves.

**Definition 3.** Database privacy: *Let $d$ be the owner of a database, and $c$ be a client issuing a set of queries $Q$. Let $A$ be any collaboration of entities including all clients of the system, entities related to the operation of the system, and outside observers. These entities observe the results of $Q$. $A$ then learns exactly the content of the database that answers $Q$ and nothing else.*

This captures the idea that queries do not expose more of the content of a database than is appropriate.

Our system will improve over existing systems by accomplishing this within sub-linear search times.

## 2.2 Anonymous Publish-Subscribe requirements

The anonymous publish-subscribe building block of our system will serve to route queries from interested users to content providers. To do this, it will aim to provide publish-subscribe functionality while protecting sender and receiver identities. This means that in terms of functionality, it will allow users to subscribe and unsubscribe to topics, and publish to topics, ensuring that published messages on a topic are delivered to all of its subscribers. More concretely, the system provides the following functions to its users:

- `Subscribe(u, t)`*:* User $u$ specifies interest in topic $t$. The system maintains an internal, protected subscription of the tuple $(u, t)$. $u$ listens for messages sent with the topic $t$.
- `Unsubscribe(u, t)`*:* The system removes any subscription of $(u, t)$ if present, and $u$ ceases to listen for relevant messages.
- `Publish(m, t)`*:* A message $m$ is sent into the system under topic $t$. For every subscription tuple $(u_i, t_i)$ s.t. $t$ matches $t_i$, $m$ will be sent to $u_i$.

In the above functions, the nature of a topic and what constitutes a match between publication and subscription topics are left undefined. A variety of different matching types can be supported by a publish-subscribe system depending on what it is trying to accomplish. The most basic of these is exact string matching; in other words users subscribe specifically to a unique topic, and receive messages that are published exactly to that topic string. This is useful for establishing communication between defined clusters of users, such as newsgroups.

We will also deal with less concrete groupings, and allow users to instead define communication on one or more dimensions of ranges so that we can define geometric shapes of users. In such a case, a topic would consist of one or more labels, each being associated with an integer value within some pre-defined and limited range. A subscription would then be a list of label-range pairs indicating what range of values to accept along each axis. This can be useful for applications such as geographically

based communication, or alert systems that are notified of values in certain ranges generated from physical sensor networks. Thus we have matching types on strings, integer ranges, and ranges across multiple dimensions:

- *Labels:* Each topic is a human-readable string. A publication and subscription are deemed to match if their topics are identical.
- *Ranges:* A publication topic is a numerical value $v$ (either integer or float). A subscription consists of a tuple $(l, h)$ s.t. $l \leq h$. A publication and subscription are deemed to match if $l \leq v \leq h$.
- *Multi-attribute ranges:* A publication topic is a list of tuples $(t, v)$. A subscription topic is a list of tuples $(t, l, h)$. A publication and subscription are deemed to match if for every tuple in the subscription $(t_s, l_s, v_s)$, there exists at least one tuple in the publication $(t_p, v_p)$ s.t. $t_s = t_p$ and $l_s \leq v_p \leq h_s$.

Our system will be able to make guarantees that messages will be successfully delivered. It should also make guarantees in regards to the amount of excess delivery that occurs. Delivery of messages that were not subscribed to is acceptable to an extent, since the receiver can simply ignore them himself. By default, these systems are not designed to prevent users from subscribing to any topic of their choosing, so it is not considered a leakage for them to receive extra messages. If it *were* desirable to prevent such leakages, that could be achieved independently using encryption systems for each topic. Hence, for the underlying message delivery system, excessive message receipt is an efficiency issue, not a security one. To be called anonymous, the system should ensure that using the basic publish and subscribe functionality does not compromise one's identity. In other words, we aim to prevent interactors and third parties from identifying two parties: the publishers and the subscribers. The specifics of this protection, which parties are prevented from identifying the participants and under what circumstances, are dependent on the implementing system.

We begin with correctness definitions:

- *Completeness:* For every publication $(m_p, t_p)$ and subscription $(u_s, t_s)$, if $t_p$ and $t_s$ match, then $m_p$ will be delivered to $u_s$.
- *Non-excessiveness:* For every publication $(m_p, t_p)$ and subscription $(u_s, t_s)$, if $t_p$ and $t_s$ do not match, then $m_p$ will be delivered to $u_s$ with probability $Pr \leq \epsilon$.

These capture the requirement that messages be delivered to those who are subscribed to them, and that the system not produce undue load by delivering them to a large amount of uninterested parties. More complicated are the security definitions. First is publisher anonymity: we will guarantee that no adversary can learn the identity of the publisher of any message.

**Definition 4.** Publisher anonymity: *Let $p$ be the publisher of message $m$, and let $H$ be the set of $h$ honest users in the system. Let $A$ be any collaboration of entities not in $H$, including subscribers, entities related to the operation of the system, outside observers, and other publishers. These entities may enter any number and type of published messages as publishers,and observe the outputs as subscribers. $A$ cannot then identify $p$ given $m$ with probability greater than $\frac{1}{h}$.*

We allow an adversary to collude with or compromise any number of other users (both publishers and subscribers) in the system. They may then for any length of time take any of the actions those entities might take: publishing messages, subscribing to topics, and observing the messages received as a result of those subscriptions or through the normal routing of other messages in the system. They may do so in an adaptive fashion, choosing what types of publications or subscriptions to issue based on observations from previous messages, including the one they are attempting to de-anonymize. They may also attempt to subvert the system by refusing to forward messages the protocol would otherwise require them to, and observe the results of such actions in terms of additional traffic sent. We claim that our system will prevent such an adversary from identifying the publisher of any given message with probability any better than random guessing from amongst the pool of non-compromised users.

Next is subscriber anonymity, which encapsulates the protection of the identities of users who are subscribed to a topic. There are two types of anonymity we wish to

protect:

**Definition 5.** Topic subscriber anonymity: *Let $t$ be a topic for which there are $s$ subscribers out of a group of $S$ total participants in the system. Let $A$ be any collaboration of entities having $A_s$ subscribers, and possibly including entities related to the operation of the system, outside observers, and publishers. These entities may enter any number and type of published messages as publishers,and observe the outputs as subscribers. $A$ cannot identify determine if user $u$ is subscribed to $t$ with probability greater than $\frac{s-A_s}{S-A_s}$.*

This captures subscriber anonymity in the first direction, an adversary should not be able to identify the subscribers of a given topic with probability greater than random guessing. Again, we assume an adversary may compromise any number of users in the system, and learn whatever information it can by taking all actions normally available to those compromised users. It may again also learn adaptively, using observations from previous messages to form new publications and subscriptions to enter into the system. It may do so indefinitely over the lifetime of a subscription. We claim our system will prevent such an adversary from identifying any subscriber of a given topic with probability better than random guessing from amongst the pool of non-compromised users.

**Definition 6.** Subscription anonymity: *Let $t$ be a topic and $s$ be a user subscribed to $t$. Let $A$ be any collaboration of entities including those related to the operation of the system, outside observers, other subscribers, and publishers. These entities may enter any number and type of published messages as publishers,and observe the outputs as subscribers. Given $s$, $A$ cannot identify $t$ with probability greater than $\frac{1}{T}$ where $T$ is the total number of possible topics.*

This captures the opposite direction: an adversary should not be able to, given a user, determine what topics he is subscribed to. This will assume the same types of powers for the adversary as with subscriber anonymity, and the adversary will attempt to defeat our system by guessing from amongst the pool of possible subscriptions.

These are a completely new type of anonymous communication system, and extend the capability of existing anonymous communication systems by allowing publish-subscribe addressing in a push architecture instead of specific destination addressing.

## 2.3 Anonymous Credential System Requirements

The credential systems we build support interactions between the following entities:

- *Users:* A user $\mathsf{U}$ will obtain and demonstrate credentials.
- *Organizations:* Organizations will grant credentials to *Users*.
- *Registration Authority:* A centrally trusted authority who will manage membership of *Users* and *Organizations* in the system.

Practical implementations may also have additional entities, subdividing *Organizations* into *Banks* and *Employers* that use credentials to manage finances or employment details. They may also support entities such as *Tax Authorities* who are responsible for aggregating income for *Users*.

Between these identities, we need to support the following operations:

- $\mathsf{Generate\ Master\ Identity}$, where $\mathsf{U}$generates his single master secret identity, i.e., $\mathsf{Ms}_{\mathcal{U}}$.
- $\mathsf{Register}$, where $\mathsf{U}$ registers to $\mathsf{RA}$ and validates $\mathsf{Ms}_{\mathcal{U}}$.
- $\mathsf{ShowMS}$, where $\mathsf{U}$ demonstrates knowledge of his $\mathsf{RA}$-validated $\mathsf{Ms}_{\mathcal{U}}$; in our system, this procedure is extended to include $\mathsf{ShowMSAttribute}$ operations, where $\mathsf{U}$ proves that his $\mathsf{Ms}_{\mathcal{U}}$ has a particular attribute.
- $\mathsf{ObtainMembership}$, where $\mathsf{U}$ and Organization $\mathsf{org}$ collaborate for $\mathsf{U}$ to become a member of $\mathsf{org}$. Depending on the type of $\mathsf{org}$, $\mathsf{ObtainMembership}$ prerequisites may vary.

- UpdateMS, where U and RA collaborate for U to obtain a new $\mathsf{Ms}_{\mathcal{U}}$; all past U's registrations through his ex-$\mathsf{Ms}_{\mathcal{U}}$ are also be updated.

- RecoverMS, where the U-authorized people collaborate to recover $\mathsf{Ms}_{\mathcal{U}}$; this procedure only takes place in case of emergency.

Our system, as mentioned before, is privacy preserving. In the context of an identity management system, privacy can be interpreted as the combination of *unlinkability* and *anonymity*, namely no one should be able to link a particular system activity to a different one as having originated by the same individual (*unlinkability*) or to a particular identity (*anonymity*). It is critical that privacy provisions are conditional: a misbehaving party will have both his identity and entire activity revealed.

Another fundamental requirement of our system is *deployability*. *Deployability* requires that we take in consideration the current settings of various organizations realized in our protocols: (a) their functionality and how "misbehavior" is defined in them, i.e., under what conditions should their members be considered as malicious, (b) their participation, since we have to optimize our protocols' efficiency accordingly.

Master secret credential *unforgeability* and *forward secrecy* are two more of our requirements. In particular, we require that no credential can be created without the participation of the authorized authorities, such that the MSShow operation accepts(*unforgeability*). *Forward Secrecy* requires that no past activities of a particular individual are traced through the UpdateMS or RecoverMS procedures.

## 2.4 Private, Distributed, and Searchable Content Provider Requirements

In total, we aim to support the following functionality:

- `Register(u)`: A new user $u$ enters the system.

- `Create − Owner(o)`: A new organization $o$ enters the system.

- `Provide(u, m, d)`*:* User $u$ makes content $m$ available under description $d$. The system maintains an internal, protected entry of the tuple $(u, m, d)$. $u$ listens for queries sent that could match the description $d$.

- `Query(u, q)`*:* A user $u$ sends a query $q$ into the system. For every entry in the system $(u_i, m_i, d_i)$ s.t. $d_i$ matches $q$, and there exists some tuple $(u_p, c_p, p_p)$ and some tuple $(u_q, c_q)$ such that $c_p = c_q$, $u_p = u_i$ and $q$ is accepted by policy $p_p$, and there exists a tuple $(u_p, c_q, n)$ s.t. $n > 0$, $u$ receives a unique handle $h_i$ that can be used to fetch $m_i$ and $(u_p, c_q, n)$ is replaced by $(u_p, c_q, n-1)$

- `Retrieve(u, h)` A user $u$ who has received a handle $h$ from a `Query` submits $h$ to the system. He is sent the corresponding content $m$.

In providing this functionality, we aim to protect the participants in various ways.

- *Provider anonymity:* For a provided tuple $(u, c, d)$, it should be impossible to identify $u$, either from the existence of the tuple in the system, ownership of a handle $h$ for that tuple's content, or from queries issued or content retrieved from a handle resulting from a query that matched $d$. In other words, so long as there are $h$ honest users in the system, an adversary consistent of any number of other users, content providers, and other entities involved in the operation of the system, and permitted to issue any number of queries and view the results or monitor the communications that pass through them through the proper execution of the system (i.e. honest but curious) should not be able to guess $u$ with probability greater than $\frac{1}{h}$.

- *Querier anonymity:* For a query $(u, q)$ issued to the system, it should be impossible for any other entities to identify $u$ from the passage of the query through the system. So long as there are $h$ honest users in the system, an adversary consistent of any number of other users, content providers, and other entities involved in the operation of the system, and permitted to issue any number of queries and view the results or monitor the communications that pass through

them through the proper execution of the system (i.e. honest but curious) should not be able to guess $u$ with probability greater than $\frac{1}{h}$.

- *Query privacy:* For a query $(u, q)$ issued into the system, it should be impossible for any other entities to learn anything about $q$. A collusion of entities involving all other queriers and data owners should not be able to determine $q$ from handling its execution.

- *Content privacy:* For a provided tuple $(u, c, d)$, it should be impossible for anyone to learn anything about $c$, except by successfully obtaining a handle to it through `Query` and fetching it with `Retrieve`. In other words, a collusion of entities including all clients and other data owners, issuing any number of queries unrelated to $c$ into the system, and observing the results, should not be able to learn any of the content of $c$.

# Chapter 3

# Related Work

## 3.1 Private Search

The problem of secure search can be considered in general as a problem of secure multiparty computation ([44, 75, 77]) and be solved with general techniques such as [44]. Although this approach provides strong privacy guarantees for the participating parties, limiting the amount of leaked information to what is inherently leaked by the result of the computation, it has one significant drawback: the complexity of generic secure multiparty computation schemes is very inefficient and unusable for practical purposes.

Most of the papers that present encrypted search techniques that allow keyword search address a different type of search scenario from private search, namely database outsourcing [5,9–11,22,30,69,72,73]. 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 maintains 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.

The idea behind most encrypted search papers is to provide capability to the querying party to decrypt cipher texts that encrypt the same word as the their query

[9–11,69]. However, this approach necessitates computational complexity linear in the number of searchable token, which introduces too high cost for practical applications. In fact Bellare et al. [6] 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.

The problem of private database management can be solved by general purpose secure computation schemes [45, 56, 76, 78]. These solutions, are generally at least linear time in database size, and thus not useful for practical purposes for very large databases. Oblivious RAM (ORAM) [46] can be used to completely hide the client's query pattern, and can also be used as a tool to achieve sublinear amortized time for secure computation if we allow to leak the program running time[47,57]. Nonetheless, computational costs are still prohibitively high, rendering these solutions impractical for very large scale databases.

Private Information Retrieval protocols (PIR) [25] consider a scenario where the client wishes to retrieve the $i$th record of the server's data, keeping the server oblivious of the index $i$. Symmetric PIR protocols [43] additionally require that client should not learn anything more than the requested record. While most PIR and SPIR protocols support record retrieval by index selection, Chor et al. [24] considered PIR by keyword. Although these protocols have sublinear communication complexity, their computation is polynomial in the number of records, and inefficient for practical uses.

Another approach would be to use fully homomorphic encryption. In 2009, Gentry [39] showed that is at least theoretically possible. Despite this breakthrough and many follow up works, current constructions are too slow for practical use. For example, it is possible to homomorphically compute 720 AES blocks in two and a half days [40].

Little work has appeared on practical, private search on a large data. In order to achieve efficiency, weaker security (some small amount leakage) has been considered. The work of [62, 65] supports single keyword search and conjunctions. However, the solution does not scale well to databases with a large number of records (say

millions); its running time is linear in the number of DB records. A more efficient solution towards this end was proposed in [34]. However, they only considered single keyword search.

Searchable encryption has also been studied in the public key setting [2,6,9,12,68]. Here, many users can use the server public key to encrypt their own data and send it to the server.

The closest solution to the type of large scale efficient practical private search we are interested in is a very recent extension [51] of the SSE solution [20], which additionally (to the SSE requirements) addresses data privacy against the client.[51] support the same class of functions as [20] (discussed above). In the worst case, such as when the client has little *a priori* information about the DB and chooses a sub-optimal term to appear first in the query term, the complexity of the [51] solution can be linear in the DB size.

## 3.2 Anonymous Publish-Subscribe

Non-anonymous publish subscribe systems were developed a great deal by TIBCO, who developed the Rendezvous system [70], which introduced wildcard topic matching, used a de-centralized architecture which supported topic priority in routing. The most currently used publish-subscribe system is PubSubHubbub [36]. PubSubHubbub is an extension of the RSS web feed protocol, but improves upon it by implementing the delivery of messages using a push mechanism. In other words, feed updates are pushed from the sender immediately to the receivers rather than waiting for them to poll the feed, making it a publish-subscribe protocol. Similarly, there are cloud-based content distribution mechanisms designed to ensure secure, but not identity-hidden, delivery of messages [15, 32, 74]. None of these systems, however, provide any anonymity protection. They are intended to be used between openly known clients.

There are numerous anonymous data distribution systems besides those that work on a publish-subscribe basis. Tor provides a simple anonymous routing network that

relays messages through a number of nodes with layered encryption, such that unless an attacker can either compromise all nodes on the path or monitor both the beginning and end, the sender cannot be identified [67]. Since messages are addressed, this is not a publish-subscribe system of delivery. Also, the recipient's identity is not protected. One thing a publish-subscribe system handles well that an addressed system does not is broadcast delivery of messages to a wide audience. There are anonymous systems that do this, but not using a publish-subscribe model.

One well known system for widescale document distribution is the Free Haven project [35], a peer-to-peer file-sharing system. In this system, users can publish documents, making them freely available without revealing their identities. It is based on a community of servers that distribute storage of split shares of published documents. Recipients broadcast requests throughout the storage space, and those with pieces of interest return them encrypted. The documents are associated with private key encryption pairs to maintain ownership through updates and deletions.

Another similar system is FreeNet [28], also a peer-to-peer file-sharing system, but one with routed document requests rather than universally broadcast ones. Documents are associated with hashes of descriptive keyword strings, and migrated over time so that similar documents tend to migrate to geographically close servers on the network topology. Queries are then sent on a hill-climbing search over these lexical hashes. It is more scalable, and has more flexible document retrieval (keyword search rather than simple unique-name lookup) than FreeHaven, however it does not protect recipient identity, only that of the document owners.

Finally, another approach to anonymous distribution is TOR hidden services [67]. These allow a user to create a pseudonymous address through which they may be reached anonymously through the TOR network. Other users can then initiate connections through this address without revealing their own identities or knowing who they are contacting behind the pseudonymous address. These are not inherently multi-cast systems; each recipient must establish an individual connection, which creates additional load on the server when many clients are involved.

In all of these anonymous distribution systems, since users must expressly request

messages, they are not push systems and furthermore do not allow for continuous messages based upon a topic. They also are not generally intended for low-latency delivery of messages; a distributor stores a message to the network whereupon they will be fetched by an interested party at a later time. They are thus not publish-subscribe systems. Document publishing systems such as these are suited to applications where a sender wishes to send a limited amount of data in a short time to be made available over a much longer time period. Publish-subscribe systems, however, are better suited to applications where there will be an ongoing stream of data relating to a specific subject, and where messages have a shorter lifespan.

The only existing publish-subscribe system we are aware of that aims to provide anonymity is by Datta et al. [4, 33]. They propose a routing system based on maintaining multiple layers of weakly connected directed acyclic graphs. In this system, one or more sink nodes, which may change over time, become dissemination points receiving all publications and forwarding them to subscribers. However, anonymity is provided only by stating that the node a receiver gets a message from may not be the original publisher. However, an adversary would still know that that node could possibly be an original publisher. Without probabilistic analysis of this possibility, it is difficult to say how well protected the publishers actually are. Also, no mention is made as to how difficult it is to identify subscribers in the system. Further, the system is neither analyzed for efficiency and scalability, nor implemented, so it is unclear at what cost this protection comes. There is no guarantee that the shape of the directed graphs that forms over very large networks scales in an efficient manner.

## 3.3 Identity Management

There has been some work addressing the problem of online privacy. Brands [14] and Camenisch and Lysyanskaya [17] were the first to provide a general review of privacy issues caused by the extended online use of PKI and provided a series of constructions of privacy preserving credentials, tickets and certificates based on blind signatures and zero knowledge proofs. There have also been several constructions to

provide blacklistable anonymous credentials [18,59,63]. Although the aforementioned systems are able to individually provide strong and well-defined privacy guarantees, they do not refer to systems with multiple operations each requiring a different privacy level.

Centralized identity management systems applying the primitives of [14,17] have been suggested in the past. In Idemix [16] , Camenisch and Herreweghen developed additional functionality for service providers and credential issuers to configure and enforce resource access control and credential issuing decisions. Higgins [38], OpenID [37] and the iCard [79] Foundation are examples of frameworks handling many identities of the same user across different websites.

The PRIME project [48] is a European initiative for privacy preserving identity management for online commercial interactions. Although the existing work in the field refers to multiple types of user-interactions, they do not provide accountability when the user misbehaves, or consider real world issues deriving from master identity compromise. For example, the complete recovery of a user's online subscriptions, automatic invalidation of the corresponding compromised credentials, and advanced user-authentication to manage these operations.

## 3.4 Private Content Providers

There are several existing works in the area of anonymous document distribution. The first of these is the Free Haven project [35], a peer-to-peer file sharing system. In this system, users can publish documents, making them freely available without revealing their identities. It is based on a community of servers that distribute storage of split shares of published documents. Recipients broadcast requests throughout the storage space, and those with pieces of interest return them encrypted. The documents are associated with private key encryption pairs to maintain ownership through updates and deletions. This system does not aim to protect the identities of the recipients, only the content providers. Furthermore, it does not contain any type of search functionality, users must already be aware of the documents they wish to receive.

The universal broadcast nature of the system also limits its scalability.

Another similar system is FreeNet [28], also a peer-to-peer file sharing system, but one with routed document requests rather than universally broadcast ones. Documents are associated with hashes of descriptive keyword strings, and migrated over time so that similar documents tend to migrate to geographically close servers on the network topology. Queries are then sent on a hill-climbing search over these lexical hashes. It is more scalable, and has more flexible document retrieval (keyword search rather than simple unique-name lookup) than FreeHaven, however it again does not protect recipient identity, only that of the document owners. Query flexibility is still limited to simple keyword search, there is no support for more complex queries such as boolean operations or range or negation queries.

FreeNet's search functionality also does not aim to hide the nature of the keywords being queried. While neither anonymous nor distributed, there are existing search systems that aim to protect query and result content. A common technique used in encrypted search schemes [9, 69] is to use trapdoors derived from queries with the ability to determine if a cipher text matches the trapdoor to provide search ability over cipher texts. 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 [9] and [69] 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 [9] (allowing anyone to add to the searchable data) or private key [69] (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

[22, 30]. 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 [30] and [22] use deterministic trapdoors and leak the search pattern to the search server. Their search complexities, though, differ and [30] achieves linearity in the size of the matching set of document while [22] is merely linear in the number of documents. Curtmola et al. [30] 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.

# Chapter 4

# Building Blocks

In order to build our system, we build on several existing techniques. We will outline and summarize them here before describing our novel contributions.

## 4.1 Bloom Filters

Bloom Filters are an efficient data structure for probabilistically testing set membership [8]. They have a boundable false positive rate, with no chance for false negatives. This false positive rate can be made arbitrarily small by increasing the size of the Bloom Filter, and keeping it linear with the number of elements intended to be stored. A Bloom Filter can store elements from a universe of infinite size, as long as every element can be hashed into it.

A Bloom Filter is represented as a vector of bits. An empty filter is set to all zeros. It is configured with a group of $k$ hash functions that map elements from the universe of elements into the range of indexes of the bit vector. To insert an element, you hash it using each of the $k$ hash functions, then set each of the corresponding bits to 1. Multiple elements can set the same indexes to 1.

To test for membership for an element, hash it using each of the functions, and check if the corresponding bit has been set. If the element has ever been added this is guaranteed to return true, guaranteeing a zero false negative rate. However, it is possible for the bits to have been set by another element that hashed the same, or

by multiple other elements that have individually set the corresponding bits, thus allowing for a false positive rate.

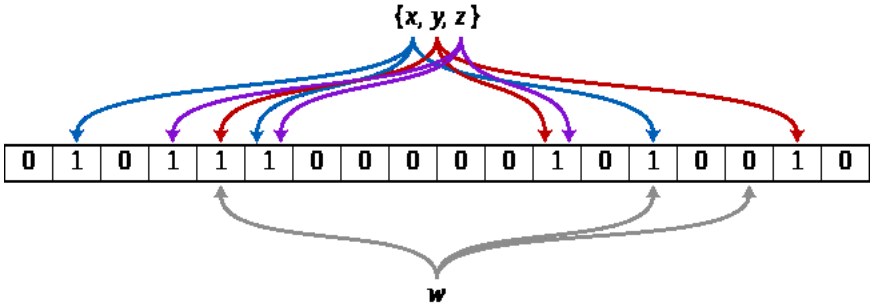



Figure 4-1: Bloom filters

The probability of having a false positive can be described by the following equation, where $k$ is the number of hash functions, $n$ is the number of elements inserted, and $m$ is the number of bits in the filter.

$$(1 - e^{-kn/m})^k$$

The optimal number of hash functions (that minimizes the required size of the bloom filter for a given number of elements and desired false positive rate), can be given by the following equation:

$$k = \frac{m}{n} \ln 2$$

Thus, to instantiate a bloom filter, one should determine the greatest number of elements that will be added to it, and decide what the maximum acceptable false positive rate is, then use the aforementioned equations to determine the number of distinct hash functions to use and the size of the bloom filter. It should be noted that for a given false positive rate, $m$ and $n$ scale linearly. The required size of a Bloom Filter is thus linear in the number of elements being stored (not the number of potential elements being drawn from).

### 4.1.1 Counting

In the standard Bloom Filter construction, there is no way to delete elements. This is because there is no simple way to determine which of the bits that have been set are unique to the element in question. Even if one were to use the hash functions to identify the bits associated with that element, it is possible those same bits were set by other elements that have been inserted as well.

The solution to this is to replace the bits with integers [13]. In a Counting Bloom Filter, we have a vector of small integer values instead of a vector of bits. To insert an element, compute all of its indices using the hash functions as before, and increment the corresponding integer. To test for membership, all one has to do now is compute the indices and check that all of the integers are greater than zero. Because they are stored as integers and not bits anymore, one can implement deletion by calculating the corresponding indices for an element to be deleted, and decrementing all of the appropriate integers. The size of the integers should be chosen to make overflow unlikely (a size of 3 or 4 bits is common). This thus makes the space requirements of a counting bloom filter larger by a factor of the number of bits used per integer.

### 4.1.2 Bit-slicing

In many applications of Bloom Filters, it is common to keep large arrays of filters, and to want to check a single element against all of them efficiently, to determine which, if any, contain matches. In the common storage format, this would require that we read the full size of all Bloom Filters for every query to check every index as specified by the Bloom Filter membership test protocol. However, it is possible to do much better than this.

To minimize the number of bits that need to be read to satisfy queries across a large number of Bloom filters, we can 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. 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 [80].

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.

This format would also allow us to handle boolean queries where we check for the existence of multiple elements combined with arbitrary boolean logic. 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

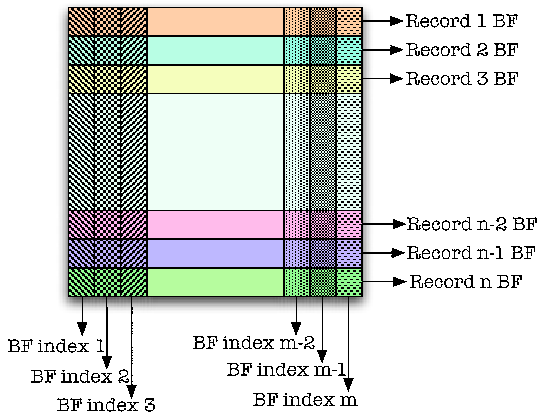



Figure 4-2: Multiple Bloom Filters Memory Storage

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.

### 4.1.3 Trees

While the slicing optimization can greatly reduce the amount of bits read per filter, it still scales linearly with the number of filters being read. Thus, with a low constant

overhead and a high asymptotic behavior, it is ideal for applications with limited numbers of filters. However for applications with massive numbers of filters, it is desirable to scale sub-linearly. For example, Bloom Filters could be used to index extremely large databases with billions of records or more.

One architecture designed to address this is tree-based Bloom Filter search indexes. In this format we choose a branching factor $b$, and create a tree of Bloom Filters. The leaves of the tree each hold one of the Bloom Filters. Then for each $b$ leaves, we create a parent node filter that contains every element present in any of its children. We repeat this going upwards until we have a single root filter that stores all elements present anywhere in the structure. Now to quickly scan the entire set of Bloom Filters and find all those that contain a particular element, we can simply traverse the tree. If a parent node returns true, we can proceed to its children, but if it does not then we know we can skip all of its children (since Bloom Filters have no false negatives). This can be massively more efficient for queries where only a small portion of the set contain matches, which is a common scenario for many types of database applications.

The maximum number of distinct elements that could be stored in a parent filter is $b$ times as many as each of its children. However, the number of parent filters is equal to $\frac{1}{b}$ the number of leaves. Since to maintain a consistent false positive rate, the size of the Bloom Filter scales linearly with the number of elements, we can infer that each level of the tree requires the same amount of space. Since the bottom level of the tree is equal to the space of storing all the filters originally, we know that the total space requirement is increased by a factor equal to the depth of the tree. The space cost can then be summarized as

$$\log_b(n)$$

This gives us a tradeoff: with a greater branching factor we have a smaller space cost but less benefit from tree search.

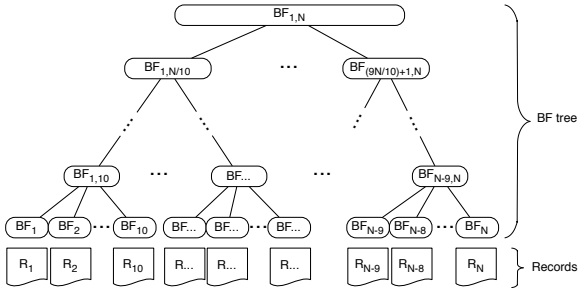



Figure 4-3: Tree-based Bloom Filter Storage

### 4.1.4 Ranged

Standard Bloom Filters support only set membership of specific elements. However, using this basic functionality, we can construct a system that allows ranged testing. That is, specific integer values on a range can be inserted, and tests can be done on subranges of that range. We now describe a system that enables multi-dimensional range queries using any underlying blackbox set membership system that can support boolean queries in conjunctive normal form. We will first describe our system as a general construction then discuss how some of the costs interact with the efficiency tradeoffs inherent in Bloom Filters, since we use Bloom Filters to match subjects in our central server system. This system can be used on top of our system to provide range-based topic matching.

#### General construction

Our general system introduces the following additional costs over its underlying search system:

- For each inserted point, we will need to insert into the underlying search system $d \lg r$ terms, where $d$ is the number of dimensions we are supporting and $r$ is the size of the range of values supported per dimension.
- For each query, we will need to issue a boolean query using up to $2d \lg(\frac{q}{2})$ query terms to the underlying search system, where $q$ is the size of the range being queried.

- The system presents repeatable unique identifiers for logarithmically cut sub-regions across all documents in a single query, and across multiple queries. If the underlying private search system does not guarantee full privacy of its queries, these can increase the information leakage over what would normally be incurred.

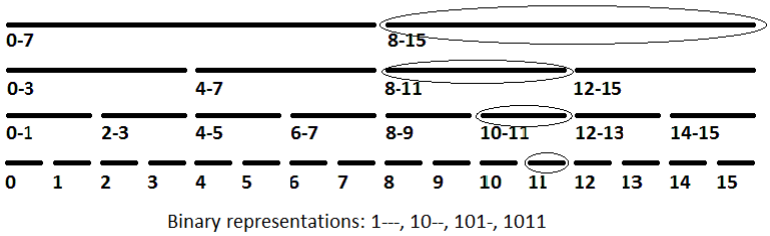



Figure 4-4: Terms used for inserting the value "11"

Our basic approach is to represent each ranged dimension as a binary value. Then, for each one, we create a strata for each digit of the value, and for each strata, divide the range into binary pieces according to the order of the digit, and assign each piece of each strata of each dimension a globally unique identifier. To insert a term to the search index, we insert the id of every piece that contains it (thus one term is inserted per dimension per strata, with a number of strata logarithmic in the size of the ranges). This is shown in Fig. 4-4.

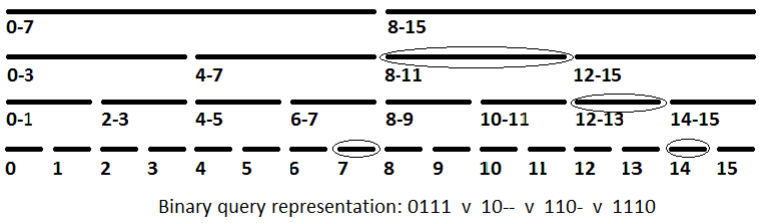



Figure 4-5: Boolean query for range "7-14"

To issue a query, now, we create a boolean OR query for each dimension. At each dimension, we start at the strata with the largest and least numerous pieces, and add

to the query the largest piece that fits entirely within the query range. We iterate to lower strata, adding pieces that fit entirely into the range without covering values that are already covered by existing pieces in the query, and continue, if necessary, to the lowest strata which contains every individual value in the full range. We then create an AND query across all of the dimensions, resulting in a query in conjunctive normal form. An example query on one dimension is shown in Fig. 4-5.

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

**Theorem 7.** *A contiguous range query on a single dimension cannot require more than* $2\lg(\frac{q}{2})$ *disjunctive terms.*

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

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

### Bloom filter construction

If the underlying system is based on Bloom filters, as with the SADS system, we can describe the tradeoffs listed in the general construction in terms of increased Bloom filter size according to the formulae associated with Bloom filter constructions.

The standard Bloom filter equation demands that in order to store $n$ items with a false positive rate of $p$, our filter needs to have size

$$m = \frac{-n \ln p}{(\ln 2)^2}$$

There are two factors that increase the necessary size of Bloom filters we must choose in order to maintain the same false positive rate. First, for every value inserted, there are now $d \lg r$ terms added, giving an increase in "effective" number of values for purposes of calculating proper sizes. Second, for every query, there are $2d \lg(\frac{q}{2})$ queries in CNF, any of which could be a false positive. If we assume in the worst case that a single false positive from a sub-query will cause a complete false positive, then we can give an upper bound on the multiplicative increase of false positive rate as $2d \lg(\frac{q}{2})$.

Thus, the total size of the bloom filter to ensure that the false positive rate does not exceed $p$ is

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

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

An issue of greater concern is the magnification of existing privacy concerns, especially if we are using a system like SADS, which does not guarantee full protection of the result patterns. Because our construction will query the same sub-regions across multiple records in a query, and across multiple queries, if the result privacy is not protected against the server, he may be able to learn about the values stored within

over time. For example, if a server sees that during a range query, two records had the same positive result for the same sub-region, it knows that they at the very least share a value in the same half-region (the largest possible sub-region). If over the course of multiple queries, it then sees those two documents match for a second sub-region, it knows that they at the very least share a value in the same quarter-region. Over time, and seeing a sufficiently varied number of queries, it may learn exactly which documents share specific values.

This is partially mitigated in the multi-dimensional case, since sub-regions of different dimensions cannot be differentiated, lending some additional obscurity. A user can further delay this learning curve by choosing maximum value ranges that are significantly larger than expected query ranges. However, this only slows the learning process, it does not prevent it. A more effective deterrence would require periodically refreshing the system by recreating new indexes.

## 4.2 Anonymous routing

Anonymous routing systems allow users to send addressed messages to single recipients, and protect their own identities in doing so. Obviously since the messages are addressed, the recipient's identity is not protected (except in the case of Hidden Services, which we will discuss later). There are two main ways to do this, via mixnets (which provide a stronger guarantee of anonymity, but at a cost in delivery latency) and onion routing (which provide a weaker guarantee of anonymity, but achieve delivery times that are more practical for real-time applications).

### 4.2.1 Mixnets

Mixnets are anonymous routing systems that protect sender anonymity by sending messages through several layers of a mixnet, a group of layered forwarding proxy servers [42,49]. A mixnet is arranged into a set of $n$ proxy servers. To send messages, each user then encrypts his message under a public key tied to the recipient, and then under a sequence of $n$ public encryption keys, one associated with each proxy server

of the mix. Tied to this message,and buried under all layers of encryption, will also be the address of the recipient and some randomness.

They then each send their message to the first node of the mixnet. Each proxy node will decrypt using its private key, and then they will re-permute all of the messages they have received and send them en masse to the next proxy server. This repeats through all layers until the last one, which upon decryption will see the final destinations of the messages and forward them accordingly. The recipients can then decrypt the messages using their private keys.

Since each layer of the mixnet applies its own permutation to the messages, and discards this information afterwards, then unless the nodes that a message traverses across all layers collude, no adversary can determine who originally sent a particular message. Since all messages traverse the same series of nodes, this system remains secure against traffic analysis and global passive adversaries. However, since message forwarding cannot begin until a full set of messages is ready to be permuted by the network, this can result in very high latency, and is unsuitable for applications which require realtime performance.

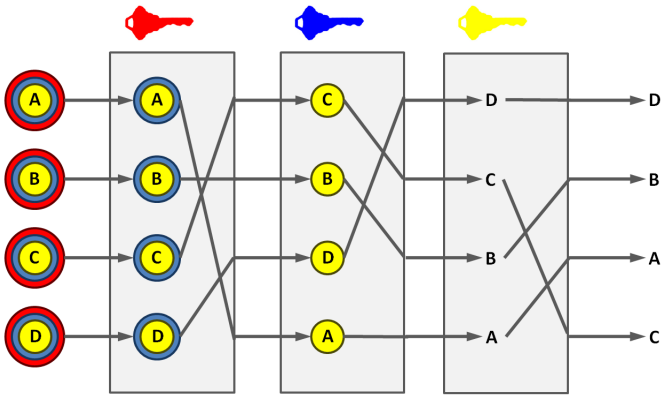



Figure 4-6: Mixnet architecture

### 4.2.2 Onion Routing

Onion-routing networks attempt to improve upon the efficiency of mixnets, but at the cost of not aiming to protect against global passive adversaries, and thus exposing

themselves to traffic analysis attacks [67]. The basic approach is the same, messages will be encrypted under several layers of encryption using the public keys of forwarding nodes. However, unlike with mixnets, messages will not be sent in batches, and the sequence of proxy nodes will be unique per message, chosen by the sender. Thus, the full method proceeds as follows:

1. The sender chooses a path of $m$ nodes, from a global directory of all participating nodes in the system. These nodes will serve as forwarding proxies for the message. The global directory will also list the public keys associated with each of these nodes.

2. The sender contacts the first node in the path, sending it a message encrypted under its public key, with a request to establish a connection, with a unique circuit ID, and the first part of a Diffie-Hellman handshake to establish a shared secret.

3. The first node will then respond and complete the Diffie-Hellman handshake.

4. These two nodes will now use their shared secret for all communications going forward.

5. The sender can now send relay messages through their encrypted connection. They will extend the communication to the next node in the chain by instructing the initial node to go through the same last three steps with the next node in the chain, establishing a shared secret between the two of them, and sending messages that only that node (and not the node being relayed through) can interpret by encrypting them using that node's public key.

6. This procedure can be extended through all of the chosen relay nodes, and only the originator will know the full sequence. Each node in the chain will only know the identities of the nodes before and after it.

7. After the chain has been extended to the final node, that node can be instructed to forward a message to the final destination through non-anonymous means

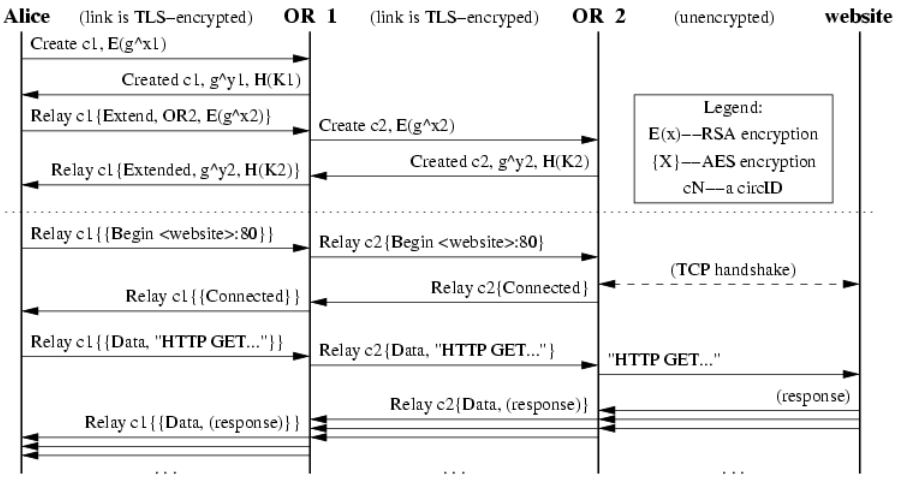



Figure 4-7: Onion routing circuit construction

Because in onion routing users establish a persistent connection through all of the nodes in their circuit, they can also provide Hidden Services, pseudonymous addresses whereby they can be contacted anonymously. To do so, a user established an onion circuit, associates an identifier with the endpoint and a public key for encryption, and stores them in a public directory. Other users can than contact that endpoint to initiate communication with the owner of the hidden service.

Because onion routing does not rely on batching messages, delivery latency is much improved over mixnets. However, since all communication circuits are unique, the system is vulnerable to global passive adversaries and traffic analysis. An adversary who can see all traffic in the network can easily deanonymize the sender of a message.

## 4.3 Yao garbled circuits

Yao's garbled circuits allow circuits to be evaluated obliviously by one party on hidden inputs provided by another party. Let $C$ be a Boolean circuit with $n$ input wires, $m$ gates, and one output wire; let $(1, \ldots, n)$ be the indices to the input wires and $q = n + m + 1$ be the index to the output wire. To generate a garbled circuit $\tilde{C}$, a pair of random keys $w_i^0, w_i^1$ are associated with each wire $i$ in the circuit; key $w_i^0$ corresponds to the value '0' on wire $i$, while $w_i^1$ corresponds to the value '1'. Then, for

each gate $g$ in the circuit, with its input wires $i, j$ and its output wire $k$, a garbled gate $\tilde{g}$ (consisting of four ciphertexts) is constructed so that it will enable one to recover $w_k^{g(b_i,b_j)}$ from $w_i^{b_i}$ and $w_j^{b_j}$ (refer to [23,54,58,66] for more detail.) The garbled circuit $\tilde{C}$ is simply the collection of all the garbled gates. By recursively evaluating the garbled gates, one can compute the garbled key $w_q^b$ given the keys $(w_1^{a_1}, \ldots, w_n^{a_n})$, where $b = C(a_1, \ldots, a_n)$. We will sometimes call wire keys corresponding to input/output *garbled input/output*, and denote them by $\tilde{a}$ and $\tilde{b}$, i.e., $\tilde{a} = (w_1^{a_1}, \ldots, w_n^{a_n}), \tilde{b} = w_q^b$. We will also use the notation of garbled evaluation $\tilde{b} = \tilde{C}(\tilde{a})$.

# Chapter 5

# Private Database Search

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 powerful tools. One such method 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.

This kind of tool thus 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, i.e., 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 attacks that may be detected when suspicious behavior is correlated across different domains. Automatic email filtering on encrypted email can be

viewed as the capability of the filtering point to search for keywords such as "urgent" on the data. 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 confidential.

We will discuss two systems for achieving this functionality, Secure Anonymous Database Search (SADS) and BlindSeer.

## 5.1 Secure Anonymous Database Search

Much of the existing work on encrypted search addresses scenarios for data storage outsourcing, where a party stores its data encrypted on an untrusted machine ([9, 10, 22, 31, 69]). The scenario that we consider differs from the above in that the querier and the owner of the data in our case are different parties. There is thus a different adversarial model: we must protect the database from the querier. Additionally, in our model we require that only authorized users are allowed to submit queries; furthermore, after authorization the users' anonymity is preserved with respect to the server. Schemes like PIR( [26]) and SPIR([43]) achieve such functionality but at a high computational cost that makes them unusable and another major goal for us is that the efficiency of the search algorithm should be usable for practical applications; as in [6], this means sub-linear search time in the size of the database. Protocols such as ([9, 22, 69, 71]) achieve linear search time; to improve this complexity, we may be willing to sacrifice strict definitions of privacy and security to a limited degree. Thus, our goal is to achieve well-defined and optimal security, privacy and anonymity guarantees, while still allowing efficient protocols. In order to accomplish this, we introduce the notion of reroutable encryption, and use it in conjunction with Bloom filter-based ([8]) search methods; these allow us to take advantage of known techniques for database signature file indexes.

We instantiate the reroutable encryption scheme into different ways to obtain an protocol for secure anonymous database search. As shown in [6], an efficiently-searchable encryption scheme has to be deterministic. Following their ideas, we in-

troduce a definition for secure private key deterministic encryption and construct a particular such encryption PH-DSAEP+, which also satisfies reencryption properties. We use this new encryption scheme for the instantiations of the reroutable encryption scheme used to submit queries and for the other instantiation used to return the search result we use standard semantically secure encryption.

The search itself is achieved efficiently by preprocessing the document set into a set of Bloom filters. Privacy-preserving search is known to require time linear in the total size of the document set. By using Bloom filters, we can mitigate this cost by paying it once during the pre-processing generation of the filters. From then on, queries against an individual document run in constant time. Furthermore, by properly organizing and querying these filters, we can take advantage of substantial existing work on bitsliced signatured files that is commonly used in database indexes ([80]). This organization allows us to further reduce the amount of data in the Bloom filters that must actually be read and processed by running queries across several Bloom filters in parallel.

### 5.1.1 Re-routable encryption

Reroutable encryption is a one-round, three-party computation protocol. One party, the sender, wishes to send a message to another party, the receiver. However, one or both of these parties may wish to have their identities protected from each other. This then requires a third party who is entrusted to choose communication pairs and protect this information. This third party is still not trusted with the content of the communication. Thus the sender must have a way to give the third party an ability to reveal information to various other parties without giving him that information directly. In addition, the third party may perform some secure computation on the data, in case the sender wishes to protect certain aspects of it from the receiver as well.

A useful primitive for our problem is an encryption scheme that allows a party to send an encrypted message to another party with neither being privy to the identity of the other. This can be done with a third party acting as an intermediary to choose

communication links. This party is the only one who knows the identities of both ends of the communication, but is not privy to the contents of the message. This is the underlying idea behind the functionalities of $F$ and $G$ introduced in Chapter 2. The problem is similar proxy encryption ([7]) and universal re-encryption for mixnets ([49]), but in our scenario the sender does not know the destination of its message and the public key encryption schemes used in these protocols are not appropriate for us. Additionally, our scheme allows the receiver to obtain partial information about the sent message; we also want to keep the key of the third party secret from the sender and the receiver. We now formally define the notion of *reroutable encryption*.

**Definition 8** (Reroutable Encryption)**.** *A reroutable encryption scheme consists of the tuple of algorithms* $(\mathsf{gen}, \mathsf{enc}, \text{ENC-TP}, \text{TRANS}, \text{DEC-R})$*:*

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

The idea behind the reroutable encryption is to allow the third party to control which are the allowed communication pairs and to transform the message that is being sent without learning anything about it. In the context of the search problem, we want the third party to ensure that only authorized users query a given server; it must also be able to forward answers from the server to the corresponding querier. Furthermore, all that the server may learn from the message is that which is necessary

for the search; this may not be the whole ciphertext. This information should be extractable from the ciphertext by the third party.

Apart from the search functionality, a reroutable encryption scheme will be applicable in cases when the third party is an intermediary in a protocol for secure multi-party computation. It is allowed to perform computation only on encrypted data; it must interact with the data owners to perform any computation that requires manipulation of the real data.

### Definition of Security for Reroutable Encryption

There are two main security requirements that the reroutable encryption scheme should guarantee. The first is the security of the sender's message with respect to all parties not including the sender and receiver. The second is the anonymity of the sender with respect to the receiver. The anonymity of the receiver with respect to the sender is trivial, given that the protocol does not involve communicating any information to the sender after he has encrypted the message. We summarize these two requirements is the following two definitions:

**Definition 9** (Message Security)**.** *Let $S$ be a security definition using an adversary $A$ and applicable to a general encryption scheme. Let $R = (\mathsf{gen}, \mathsf{enc}, \text{ENC-TP}, \text{TRANS}, \text{DEC-R})$ be a reroutable encryption scheme. We say that $R$ provides $S$-message security with respect to the third party if $(\mathsf{gen}, \mathsf{enc}, \text{DEC-R} \circ \text{ENC-TP})$ meets $S$ when $A$ is supplemented with $tpk_{\{S,R\}}$.*

This definition is intentionally non-specific. It thus applies to a general case of reroutable encryption schemes that might meet different definitions of security, depending on what underlying algorithms are used to implement them. We will instantiate this definition with a more specific one based on our instantiation of the reroutable encryption algorithm.

**Definition 10** (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 reroutable encryption scheme $(\mathsf{gen}, \mathsf{enc}, \text{ENC-TP}, \text{TRANS}, \text{DEC-R})$ 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-TP($\mathsf{enc}_{q_b}(m)$) *for* $b \leftarrow_R \{0,1\}$ *outputs a guess* $b'$*, the following holds*

$$|Pr[b = b'] - \frac{1}{2}| < negl(k).$$

A reroutable encryption scheme is secure if it meets both of the above definitions.

We will now show one method for constructing a reroutable encryption scheme from a given of type of encryption schemes that posses the following group property:

**Definition 11** (Encryption Group Property)**.** *Let* $\Pi = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ *be a private key encryption scheme. We say that* $\Pi$ *has a group property if the keys for encryption scheme form a group and for any message* $m$ *the following holds:*

$$\mathsf{enc}_{k1}(\mathsf{enc}_{k2}(m)) = \mathsf{enc}_{k1 \cdot k2}(m).$$

**Definition 12** (Simple Re-reroutable Encryption Construction)**.** *Let* $\Pi = (\mathsf{gen}', \mathsf{enc}', \mathsf{dec}')$ *be an encryption algorithm with the group property from Definition 11. We construct a reroutable encryption scheme* $(\mathsf{gen}, \mathsf{enc}, \text{ENC-TP}, \text{TRANS}, \text{DEC-R})$ *in the following way:*

- $\mathsf{gen}(1^k)$ - *Sender and Receiver independently run* $\mathsf{gen}'(1^k)$ *and to create* $sk$ *and* $rk$*, respectively. Sender, Receiver, and TP then run a secure multiparty computation with* $sk$ *as input from Sender,* $rk$ *as input from Receiver, and* $tpk = \frac{sk}{rk}$ *be an output for TP.*
- $\mathsf{enc}(sk, m) = \mathsf{enc}'(sk, m) = c$
- TRANS - *a function identifying Sender, Receiver pair*
- ENC-TP$(tpk, c) = \mathsf{enc}'(tpk, c) = \overline{c}$
- DEC-R$(rk, \overline{c}) = \mathsf{dec}'(rk, \overline{c}) = m$

We now show that the reroutable encryption scheme created with the above construction from is a secure reroutable encryption.

**Theorem 13.** *Let* $\Pi = (\mathsf{gen}', \mathsf{enc}', \mathsf{dec}')$ *be an encryption scheme with the group property from Definition 11 and a security definition* $S$*. The reroutable encryption* $(\mathsf{gen}, \mathsf{enc}, \text{ENC-TP}, \text{TRANS}, \text{DEC-R})$ *obtained from* $\Pi$ *using Construction 12 provides* $S$*-message security.*

*Proof.* Assume that $(\mathsf{gen}, \mathsf{enc}, \text{DEC-R}\circ\text{ENC-TP})$ does not meet the security definition $S$. Therefore there exists an adversary $\mathcal{A}$ that can obtain information $t$ about a message $m$ given $\mathsf{enc}(m)$. Since $\mathsf{enc}(m) = \mathsf{enc}'(m)$ 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 $\Pi$. □

**Theorem 14.** *Let* $\Pi = (\mathsf{gen}', \mathsf{enc}', \mathsf{dec}')$ *be an encryption scheme with the group property from Definition 11 and a security definition* $S$*. The reroutable encryption* $(\mathsf{gen}, \mathsf{enc}, \text{ENC-TP}, \text{TRANS}, \text{DEC-R})$ *obtained from* $\Pi$ *using Construction 12 ensures Sender anonymity w.r.t the receiver.*

*Proof.* Let $q_0$ and $q_1$ be the keys of two senders and $tp_0 = \frac{rk}{q_0}$ and $tp_1 = \frac{rk}{q_1}$ be the corresponding transformation keys at the third party. Let $m$ be any message. Now using the group property of $\Pi$ we have

$$\text{ENC-TP}(\mathsf{enc}_{q_0}(m)) = \mathsf{enc}_{tp_0}(\mathsf{enc}_{q_0}(m)) =$$
$$= \mathsf{enc}'_{tp_0}(\mathsf{enc}'_{q_0}(m)) = \mathsf{enc}'_{tp_0 \cdot q_0}(m) = \mathsf{enc}'_{rk}(m)$$

$$\text{ENC-TP}(\mathsf{enc}_{q_1}(m)) = \mathsf{enc}_{tp_1}(\mathsf{enc}_{q_1}(m)) =$$
$$= \mathsf{enc}'_{tp_1}(\mathsf{enc}'_{q_1}(m)) = \mathsf{enc}'_{tp_1 \cdot q_1}(m) = \mathsf{enc}'_{rk}(m)$$

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

### Secure Anonymous Database Search

The definition of reroutable encryption (Definition 8) captures all the properties that we required for the functionalities $F$ and $G$ that we defined presenting the setting for

secure anonymous search. However, since the set-ups for $F$ and $G$ are not completely symmetric, we have some differences in the particular instantiations of the six algorithms for the reroutable encryption. The most fundamental difference between them is the fact that we want to leak as little information as possible about the encrypted query sent from the user to the server while still sending enough for the proper search functionality. In the other direction, we want the user to receive the exact answer that was sent from the server as a query result. We capture these requirements in the following definition for secure anonymous database search.

**Definition 15** (Secure Anonymous Database Search (SADS))**.** *An anonymous secure search scheme consists of three algorithms* $(F, G, Search)$ *such that* $F$ *and* $G$ *are secure reroutable encryption schemes and the following hold:*

- *Given* $\overline{c} = \text{ENC-TP}_F(\mathsf{enc}_F(m))$ *the server can learn at most* $h(m)$ *information about* $m$*, where* $h$ *is a function chosen at the instantiation of the anonymous secure search protocol.*
- *The user learns the exact result sent from the server, i.e.,* $\text{DEC-R}_G(\text{ENC-TP}_G(\mathsf{enc}_G(m))) = m$.
- *Let* $S$ *be the exact search results for a query* $m$ *and* $S' = Search(\overline{c})$ *be the result set returned from the server who was given* $\overline{c}$*, produced by* ENC-TP *in the reroutable encryption scheme. Then*

$$\frac{|S| - |S'|}{|S|} \leq \epsilon,$$

*where* $\epsilon$ *is a small, fixed error rate.*

### 5.1.2 Private Key Deterministic Encryption

The properties that we required in Section 2 for the functionality $F$ were information hiding and a result suitable for searching. In [6] Bellare et al. suggest a deterministic and efficiently searchable encryption scheme. While this scheme provides optimal

security properties that allow efficient search on the ciphertexts, its public key orientation — where everybody can have access to the public key — is not appropriate for the our requirement for flexible authorization, especially because it does not allow easy revocation of users. We adapt the idea from [6] to a private key setting that we use to provide unique user keys that can be easily revoked; we also use a single server key for encryption of the database.

### Security Definition

In our notation, $x$ denotes a single message, $\mathbf{x}$ denotes a vector of messages with elements $\mathbf{x}[i]$ and $\mathbf{y} = \mathsf{enc}_{pk}(\mathbf{x})$ stands for $\mathbf{y}[i] = \mathsf{enc}_{pk}(\mathbf{x}[i])$ for all $i$.

**Definition 16** (Deterministic Private Key Encryption scheme)**.** *A private key encryption key scheme* $\Pi = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ *consists of three polynomial time algorithms:*

- $\mathsf{gen}(1^n) = s$ *generates a private key* $s$*;*
- $\mathsf{enc}_s(m) = c$ *encrypts a message* $m$ *with a key* $s$*;*
- $\mathsf{dec}_s(c) = m$ *is a deterministic algorithm that decrypts the plaintext* $x$ *from a ciphertext* $c$*.*

*A private key encryption scheme is deterministic if the encryption algorithm* $\mathsf{enc}$ *is deterministic.*

The security definition for public key deterministic schemes given in [6] captures the attack model relevant to deterministic schemes; accordingly, we use a similar definition of security for private key encryption schemes:

**Definition 17** (Private Key Encryption Privacy Adversary)**.** *A chosen ciphertext adversary* $\mathcal{A} = (A_1, A_2)$ *is a pair of polynomial time algorithms that share neither coins nor state and work in the following way:*

- $A_1$ *takes as input* $1^k$ *and return a vector* $\mathbf{x}$ *and some information* $t$*.*
- $A_2$ *takes as input* $1^k$ *and an encryption of* $\mathbf{x}$ *and tries to compute* $t$*.*

*Given an encryption scheme that uses random oracles $H_1, \ldots, H_n$, we say that $\mathcal{A}$ is a $(t, q_D, q_{H_1}, \ldots 1_{H_n})$ adversary if it runs in time $t$, issues $q_D$ decryption queries and $q_{H_i}$ queries to $H_i$ for $1 \leq i \leq n$.*

We use a characteristic of the adversary introduced in [6] that will be important in the following relaxed definition of security and the schemes that we will build. Intuitively it will measures the density of the plaintext domain that will be quantified with the probability of an adversary choosing a given plaintext.

**Definition 18** (Min-entropy)**.** *Let $\mathcal{A} = (A_1, A_2)$ be a private key encryption adversary. We say that $\mathcal{A}$ has min-entropy $\mu(n)$ if*

$$Pr[\boldsymbol{x}[i] = x : (\boldsymbol{x}, t) \leftarrow A_2(1^n)] \leq 2^{-\mu(n)}$$

*for all $1 \leq i \leq |\boldsymbol{x}|$ and all $x \in \{0,1\}^*$.*

We now state the relaxed definition for security of private key encryption schemes that we will use thoughout the paper.

**Definition 19** (DET-CCA)**.** *Let $\Pi^{det} = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ be a private key encryption scheme and $\mathcal{A}$ be an adversary against it. We conduct the following two experiments for:*

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

*We say that $\Pi^{det}$ is DET-CCA secure when the adversary advantage $\boldsymbol{Adv}^{DET-CCA}_{\pi,\mathcal{A}}$ against the encryption scheme is negligible where we define*

$$\boldsymbol{Adv}^{DET-CCA}_{\pi^{det},\mathcal{A}} =$$

$$= \quad Pr[\boldsymbol{DET\text{-}EXP}^{0}_{\Pi^{det},\mathcal{A}}(n) = 1] - Pr[\boldsymbol{DET\text{-}EXP}^{1}_{\Pi^{det},\mathcal{A}}(n) = 1].$$

**Note:** The relaxed definition of security stated above is achievable only when the adversary has high min-entropy $\mu(n) = \omega(\log(k))$, which is true for any adversary if the underlying plaintext domain is dense.

### 5.1.3 Encrypted Bloom Filter Search

We now show how to instantiate $F$, $G$ and *Search* to implement an SADS scheme according to the problem statement from Section 2.

#### Querier to Server Reroutable Encryption

In defining $F$, we have to achieve a balance between the privacy-preserving property that limits what the server will learn about the submitted query, while at the same time ensuring that it receives enough information to perform *efficient* and *accurate* searches. Deterministic encryption provides this type of balance. It guarantees DET-CCA security while allowing search schemes on the ciphertexts that run in sub-linear time, as do existing non-deterministic encryption schemes. We define $F$ to be a reroutable encryption scheme $(\mathsf{gen}, \mathsf{enc}, \text{TRANS}, \text{ENC-TP}, \text{DEC-R})$ such that:

- $\mathsf{gen}(1^k, Sender, TP, Receiver)$ produces three keys $(sk, tpk_{\{S,R\}}, rk)$ for the sender, the trusted party and the receiver in the scheme. Each party should get only its key and learn nothing about the keys of the other parties.

- $c = \mathsf{enc}(sk, m)$ =PH-DSAEP$+_{sk}(m)$.

- $\text{TRANS}(c, S, st_i) = (R, st_{i+1})$. The selection of the receiver is based on either an explicit user request or a matching rule in the third party that maps a querier to a server. Selection always includes a check of authorization; if such is missing,

$R = \perp$. The state $st_{i+1}$ will be obtained from $st_i$ by recording information which user has submitted with the query; this will be used later to route the response from the server correctly.

- ENC-TP$(c, tpk_{\{S,R\}}, st_i)$ =BF(PH-DSAEP$+_{tpk_{\{S,R\}}}(c)$) =BF$(\overline{c}) = \{b_1, \ldots, b_i\}$, where the $\{b_1, \ldots, b_i\}$ are the Bloom filter indexes for $\overline{c}$. We later show the algorithm for the BF function.

- DEC-R$(\{b_1, \ldots, b_i\}) = \{b_1, \ldots, b_i\}$, in this case the server does not need to perform any decryption.

In addition we apply the reroutable encryption $F$ on a hashed values of the user's query $H(m)$ where $H$ is some hash function; this provides further protection of the query against the server.

**Server to Querier Reroutable Encryption**

The primary functionality of $G$ is to protect against the third party learning information about the query results, rather than just redirecting them to the appropriate user. We want to prevent the third party from finding different queries that have the same result vector and thus be able to guess that the queries are semantically related. In this case, deterministic encryption will not suffice since the same result vectors will have the same encryption. However, we can now use non-deterministic semantically secure encryption since the user will decrypt the message at the end. Thus $G$ is defined as a reroutable encryption $(\mathsf{gen}, \mathsf{enc}, \text{TRANS}, \text{ENC-TP}, \text{DEC-R})$ where

- $\mathsf{gen}(1^k, Sender, TP, Receiver)$ produces three keys $(sk, tpk_{\{S,R\}}, rk)$ for the sender, the third party, and the receiver in the scheme. Each party should get only its key and learn nothing about the keys of the other parties.

- $c = \mathsf{enc}(sk, m) =$ PH-SAEP$+_{sk}(m)$.

- TRANS$(c, S, st_i) = (R, st_{i+1})$ where the third party has information in its state $st_i$ about the user who sent the query whose reply is returned.

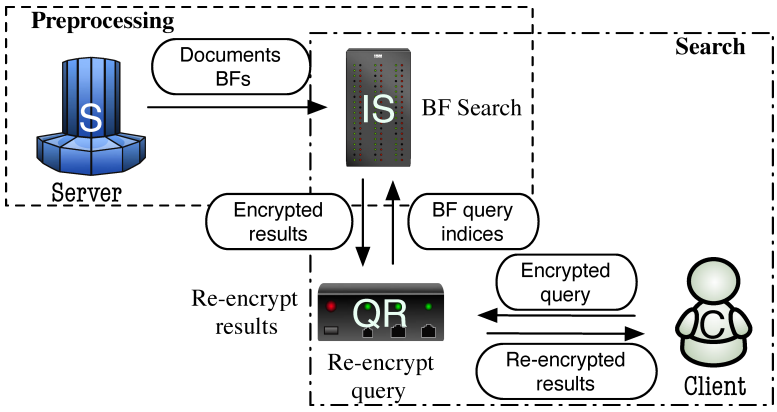



Figure 5-1: SADS overview.

- ENC-TP$(c, tpk_{\{S,R\}}, st_i)$ = PH-SAEP+$_{tpk_{\{S,R\}}}(c) = \overline{c}$.
- DEC-R$(\overline{c})$ = $\mathsf{dec}_{\text{PH-SAEP+}}(\overline{c})$.

### Bloom Filters Encrypted Search

We now consider the *Search* function and show how to optimize its computational complexity to a constant using Bloom filters. As we mentioned before, we will use Bloom filters to perform search on ciphertexts produced by PH-DSAEP+. We build Bloom filters from ciphertexts in the following way:

1. Given the expected number of entries in a Bloom filter and the false positive rate that we want to achieve, we calculate the optimal size of the Bloom filter and the number of hash functions that have to be used for each entry. We require that the size of the Bloom filter should be a power of 2 because of the specific hash function we are using. The size of the Bloom filters is a secondary concern to the efficiency and the false positive rate that the structures achieve.

2. Let us now define the function BF that we used in the ENC-TP algorithm of the reroutable encryption $F$. If we have a Bloom filter of size $2^n$ bits and we use $k$ hash functions per entry, we derive the Bloom filter indexes for an entry $m$ by taking the first $k$ blocks of length $n$ in the binary representation of $PH - DSAEP^{+}_{sk}(m)$ where $sk$ is a fixed key. The indexes derived in this way

are suitable for a Bloom filter since they have pseudorandomness property at least as good as a hash function, which is the property that determines the false positive rate of a Bloom filter.

A Bloom filter query has a constant search time and does not depend on the number of entries in the filter. At the same time, privacy-preserving search algorithm requires that each entry is touched. This is achieved at the time when the Bloom filter is built including information for each entry. In other words, the computation that is dependent on the number of entries to be searched is performed only once at the time of the filter creation.

**Key Generation**

The key generation algorithm is not our main focus since general multiparty computation techniques ([44, 75, 77]) 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 third party (TP) 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 $(\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ in which $pk_{TP}$ and $pk_S$ are public encryption keys for the third party and the sender.

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

At the end of the above message exchange the third party can compute:

$$k_{TP} = (k_R \cdot r_R) \cdot (k_S \cdot r_S)^{-1} \cdot r_S \cdot r_R^{-1} = (k_R) \cdot (k_S)^{-1}.$$

### Security of The Key Generation Protocol

Our adversarial model includes a single malicious party that does not follow the protocol. We do not protect against colluding parties; if any two parties collude, they can find the key of the third one. If the third party colludes with the sender or the receiver, they can compute from their keys the key of the the non-colluding party. Collusion between sender and receiver is not possible since these parties want to protect their data from each other. Accordingly, we show that if any of the parties is acting maliciously it cannot achieve anything more than producing an invalid third party key and it cannot learn any secret. Since the third party never sends a message in the protocol, it cannot affect in any way the information that it receives. Its only power is to compute its own key incorrectly but this way it cannot learn anything about the sender and receiver keys. The receiver also does not get any message during the execution of the protocol; thus it cannot learn anything about the sender's key. The receiver sends its random number to the sender; he can misbehave and send an incorrect value. Sending an incorrect value for $k_R \cdot r_R$ will have the same effect. However, since the randomness of the receiver is used together with the randomness from the sender, the receiver cannot influence the computation of the third party key in any way other than making it produce an invalid key about which he has no further information. The sender in the protocol gets just a random number from the receiver; this also does not reveal anything about the key of the receiver. He has to send $r_S \cdot r_R^{-1}$ to the third party and he can manipulate this value. However, the key computed for the third party also depends on the randomness of the receiver via $k_R \cdot r_R$; the only thing that the sender can achieve by sending an incorrect value is to produce an invalid third party key about which he has no further information. Sending an incorrect value for $k_S \cdot r_S$ will have the same effect.

### Server Preprocessing Step

The server has to generate Bloom filters for the documents to be used for the search of the encrypted query. Each Bloom filter will contain the unique stems of the words

in a document. To obtain the Bloom filter indexes for a stem $st$, the server computes PH-DSAEP+$_{sk}(H(st))$, where $sk$ is the server's key and $H$ is the hash function that the users will use when sending their queries. He then uses the first $k$ blocks of $n$ bits from the resulting digest as indexes for the Bloom filter, where $k$ is the number of hash functions used for the filter and $n$ is the size of the filter.

### Security Proof

**Theorem 20.** *The Encrypted Bloom Filter Search is an SADS scheme according to Definition 15.*

*Proof.* First we consider $F$. We will show that $F$ is a secure reroutable encryption where the receiver can find at most $H(m)$, where $H$ is the hash function that we applied to the query before sending it. The third party receives the encrypted message from the user and the encrypted result from the sender. Since obtaining any information about the result is at least as hard as obtaining information about the encrypted query because of the encryption schemes that are used, the power of the third party is equivalent to the power of that of an adversary against PH-DSAEP+ given $tpk_{\{S,R\}}$. Hence Definition 9 is satisfied. The user anonymity according to Definition 10 follows from the fact that the third party transforms all messages to encryptions with the server's key. Therefore $F$ is secure reroutable encryption. If the third party just reencrypted what was sent by the user to the key of the server without any further transformation, the server would be able to decrypt it and get $H(m)$. However, in practice the server is able to obtain even less information about the query, since he receives only the set of the Bloom filter indexes which arranged in the appropriate order given the first bits from the ciphertext.

We need to show that $G$ is a secure reroutable encryption scheme and that the user gets the exact message that was sent from the server. The view of the third party consists again of the encrypted message and the encrypted result. However, the third party has no information about the server's database; accordingly, learning information about the query is not useful for learning information about the result. Thus the power of the third party is equivalent to that of an adversary against PH-

| Corpus Size | 1K | 5K | 10K | 50K |
|---|---|---|---|---|
| Creation time | 10m7s | 44m51s | 1h20m20s | 6h39m23s |

Table 5.1: Creation times for indexes on various corpus sizes

SAEP+ given the transformation key; thus, Definition 9 is satisfied. Similarly to $F$, all results are converted to ciphertexts under the key of the recipient user, which suffices for Definition 10. It follows that $G$ is secure reroutable encryption. Clearly, the user gets the exact search result message, since the third party only converts the message to the user's key.

Finally, the accuracy of the search results — the third requirement for an SADS scheme — is guaranteed by the upper bound on the false positive rate obtained from the Bloom filter parameters.

□

### 5.1.4 Performance

Our system was implemented in C++. We ran experiments on a Ubuntu 8.04 Linux PC with a Pentium 4, 3.4 GHz cpu and 2GB of RAM. Four different corpus sizes (1K, 5K, 10K, 50K) were extracted from the Enron Email Dataset, available at `http://www.cs.cmu.edu/~enron/`. Each document was stemmed using the techniques provides by the Clair library [64], and the stems were inserted into the Bloom filter index in the same document order. 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 5.1 shows the time taken to create indexes for each of our 4 corpuses. As we can see 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 set of query terms, and grouped them by document frequency within the set. In each experiment, we ran a

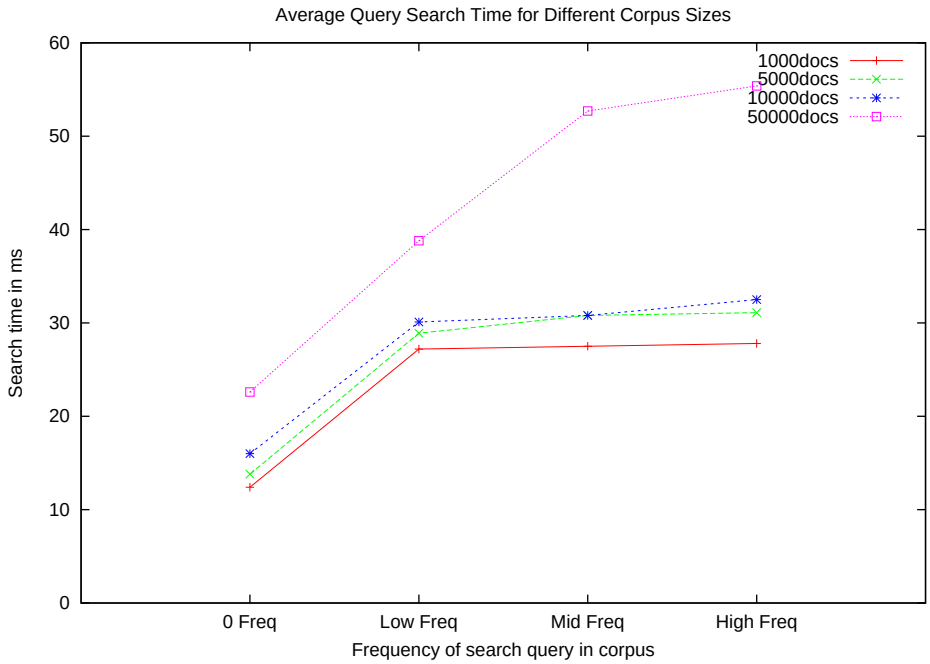



Figure 5-2: Time vs Term Frequency

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 5-2 shows the average time per query plotted against the frequency of the query terms being searched on for all four corpus sizes. 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 we can see, 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 bitslice 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

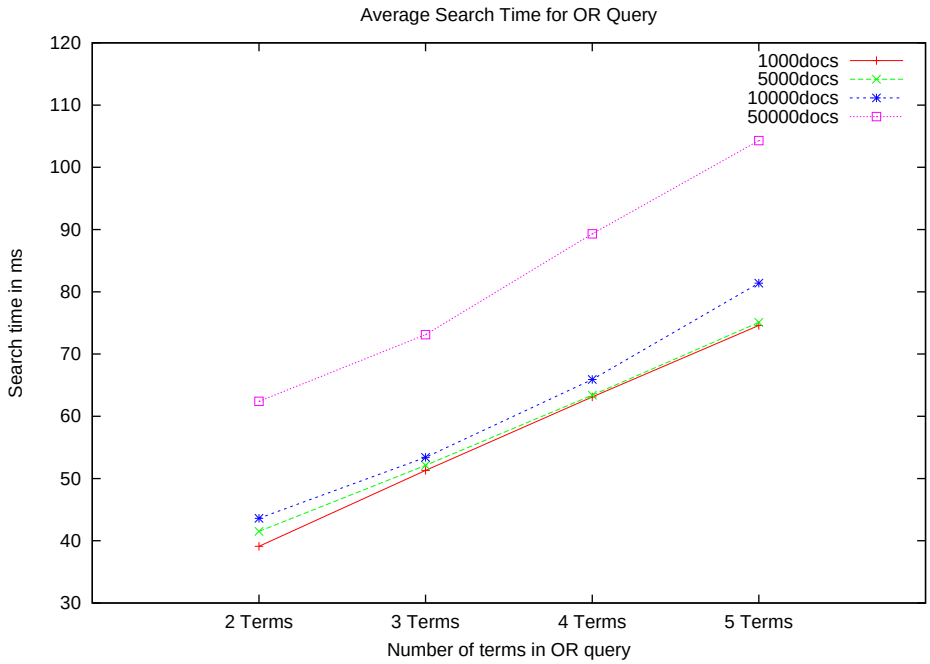



Figure 5-3: Time vs OR Query Terms

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 somewhat 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 5-3 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 5-4 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

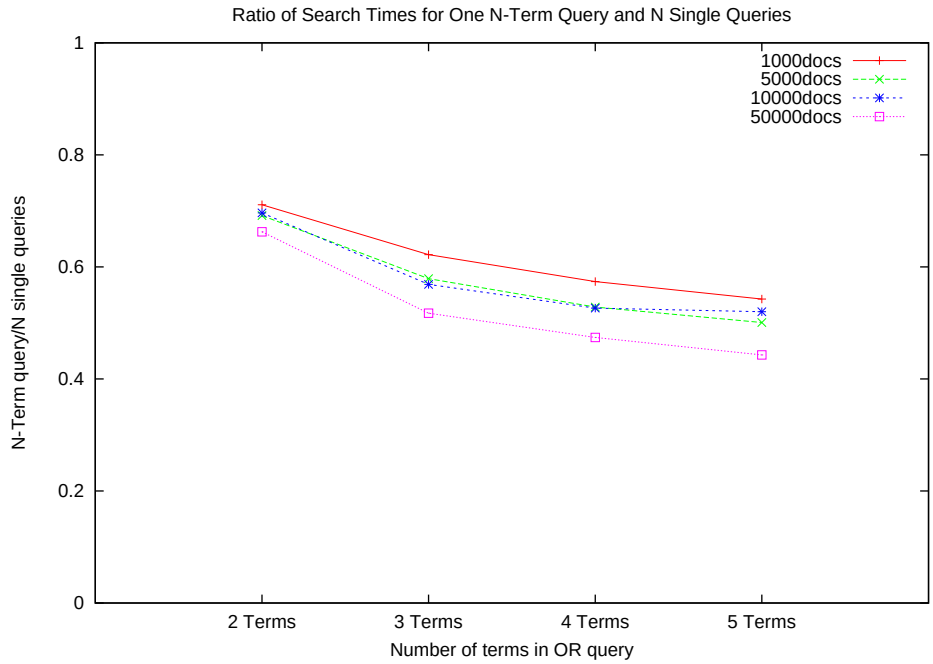
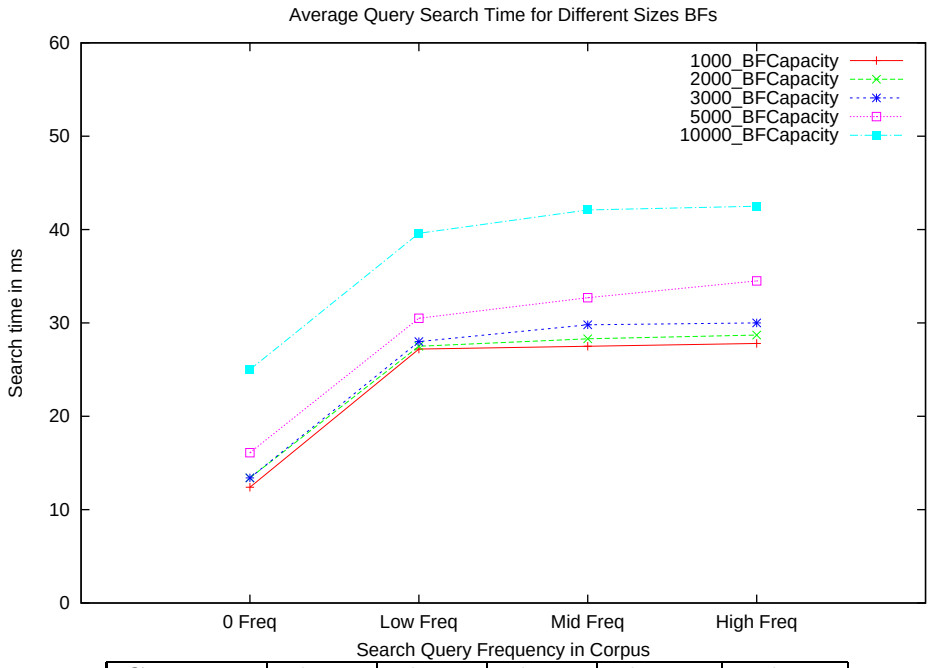



Figure 5-4: Ratio of OR Query time vs Individual Queries

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

Table 5.2: Ping Latency

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 5-5 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 because 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 5.2 shows network delays for different typical distances around the world as a point of comparison. Furthermore, on long running tests, analysis via gprof revealed




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

Figure 5-5: Time vs Term Frequency (varying BF size)

that 64.9% of time was spent running cryptographic operations. For queries, these are run by the client and trusted party, not the 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.

### 5.1.5 Document Retrieval

While the described system provides a means of issuing queries to a database and determining which records match, we still need a means of retrieving those records without revealing the results to the server. One way to do this is with private information retrieval techniques, however these are very expensive, and can be even more expensive when fetching large numbers of records, or records of individually great size. We present a system that is much more efficient, at the cost of requiring a trusted third party, and can be modularly implemented to extend any private search system that returns handles representing matches.

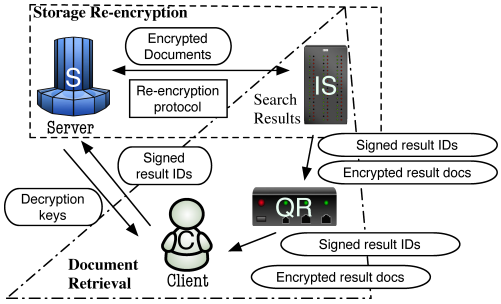



Figure 5-6: SADS with Document Retrieval.

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

Document retrieval poses its own challenge, especially when the data is not owned by the party retrieving it. In this scenario, returning additional data is a privacy leak for the data owner; at the same time, revealing the matching documents to the owner is a privacy leak for the retriever. Thus, the strongest security we would want to aim for would require us to touch the contents of the entire database [24]. This is a prohibitively expensive cost for applications that aim to work in "real time" over a large data set. One way to avoid this cost is to relax our security definition and allow leakage of the retrieval pattern (i.e. whether separate retrieval attempts touched the same documents). In the case of data outsourcing, this amount of privacy leakage easily suffices, since the untrusted server just searches for and returns the encrypted files that he stores to the owner who has the corresponding decryption keys [9, 22, 30]. This approach, however, is not applicable to the case of data sharing, where leaking the matching documents to the owner reveals more than the result pattern: he also knows the content of the documents, from which he can infer information about the

query.

This problem is similar to that addressed by private information retrieval protocols (PIR) [25, 43, 61], wherein a server holds a set of items from which a user wishes to retrieve one without revealing which item he is requesting. It differs slightly in that we wish to retrieve multiple items (corresponding to the search results). It also differs in that we require that the selected set be certified and that the user does not learn content of documents outside of it. There are PIR schemes that address this [43], but at additional cost. Thus, our problem could be addressed by simply running an appropriate PIR scheme once for each document result. However, PIR is already quite expensive for a single document, and running them multiply would only aggravate this.

We address this by constructing a document retrieval scheme that can be used on top of any other scheme that returns document IDs. Our scheme maintains efficiency by introducing an intermediary party who stores the encrypted files of the database and provides the matching ones to the querying party. This party is given limited trust to perform the search, but he should not be able to decrypt the stored files. In this case we need to provide the querier with the decryption keys for the result documents; these are known to the data owner, who must be able to provide the correct keys obliviously without learning the search results. In Figure 5-7 we present a protocol that realizes the document retrieval functionality between a data owner (S) and a client (C) with the help of an intermediary party (P). For the purposed of this protocol we assume that there is a search functionality $EncSearch$ that returns the IDs of the documents matching a query from the client. For a query $Q$ we denote $EncSearch(Q)$ the returned set of document IDs. The database of the server that is used for the protocol consists of documents $D_1, \ldots, D_n$. Our protocol also uses 1-out-of-n oblivious transfer (OT) functionality that allows two parties, one of which has input an array and the other has input an index in the array, to execute a protocol such that the latter party learns the array element at the position of his index and the former learns nothing. There are many existing instantiations of OT protocols, we use the protocol of [41], which allows best efficiency. The last tool for our constructions

is an encryption scheme with the following property (defined in more detail in [65], which also gives an instantiation for such a scheme):

**Definition 21** (Encryption Group Property)**.** *Let* $\Pi = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ *be a private key encryption scheme. We say that* $\Pi$ *has a group property if* $\mathsf{enc}_{k1}(\mathsf{enc}_{k2}(m)) = \mathsf{enc}_{k1 \cdot k2}(m)$ *holds for any keys* $k_1, k_2$ *and any message* $m$*.*

Intuitively, the security of this protocol is based on the secrecy of the permutation $\pi$, 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 2 of the re-encryption can be implemented using protocols for oblivious transfer [3, 29, 60].

## 5.2 Blind Seer

While SADS is an efficient and effective private search system for smaller sized databases, it still has a query time that scales linearly with the database size. This is insufficient for very large database sizes, which generally need to run in sub-linear time. It also provides no ability for the data owner to control what types of queries are being issued upon his data. Blind Seer is a database management system providing query functionality that scales sub-linearly with the number of records, making it applicable for potentially much larger datasets. It also supports private policy enforcement, allowing data owners to set rules on what kinds of queries can be run and enforce them without revealing the nature of the policies, or the queries being checked against. To do so, it makes use of Bloom filters, as with SADS, but uses Yao garbled circuit evaluation to check them remotely. The Bloom filters themselves are

**Storage Reencryption** (***preprocessing*** *phase*)

***Inputs:***
$S$ : $D_1, \ldots, D_n$, keys $k_1, \ldots, k_n$ and $k'_1, \ldots, k'_n$;
$P$ : permutation $\pi$ of length n ;
$S, P$ : $(\overline{GEN}, \overline{ENC}, \overline{DEC})$ satisfying Definition 21
***Outputs:***
$S : \perp$; $P$ : $\overline{ENC}_{k'_{\pi(i)}}(D_i)$ for $1 \leq i \leq n$
***Protocol:***

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

**Document Retrieval**

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

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

Figure 5-7: Protocol for Document Retrieval

stored in a tree structure, allowing a query to find matches without touching all of the records in the database.

### 5.2.1 Participants

Recall, our system consists of four participants: *server* S, *client* C, *index server* IS, and *query checker* QC. The server owns a database DB, and provides its encrypted searchable copy to IS, who obliviously services C's queries. QC, a logical player who can be co-located with and may often be an agent of S, privately enforces a policy over the query. This is needed to ensure control over hidden queries from C. Player interaction is depicted in Figure 5-8.

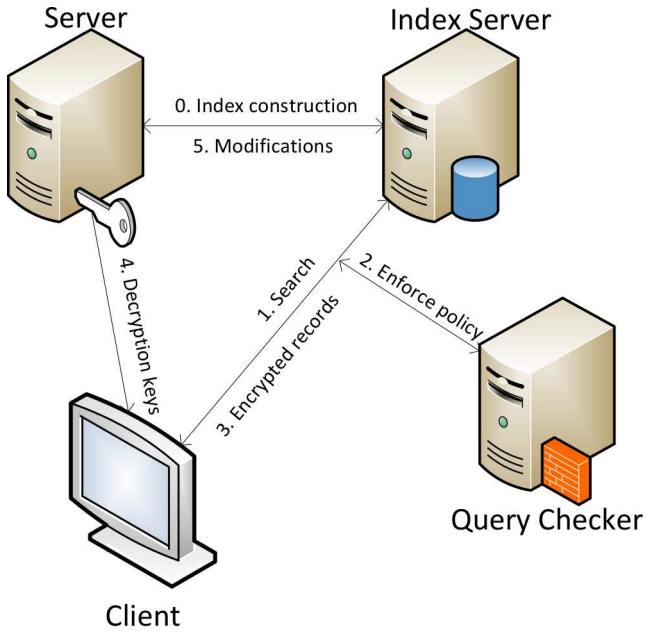



Figure 5-8: High-level overview of Blind Seer. There are three different operations depicted: preprocessing (step 0), database searching (step 1-4) and data modifications (step 5).

**Our approach.** We present a high-level overview of our approach and refer the reader to Section 5.2.2 for technical details. We adhere to the following general approach of building large secure systems, in which full security is prohibitively costly: in a large problem, we identify small privacy-critical subproblems, and solve those securely (their outputs must be of low privacy consequence, and are handled in plaintext). Then we use the outputs of the subtasks (often only a small portion of them will need to be evaluated) to complete the overall task efficiently.

We solve the large problem (encrypted search on large DB) by traversing an

encrypted search tree. This allows the subtasks of privately computing whether a tree node has a child matching the (arbitrarily complex) query to be designated as security-critical. Further, unlike the protected input and the internals of this subtask, its output, obtained in plaintext by $\mathsf{IS}$, reveals little private information, but is critical in pruning the search tree and achieving efficient sublinear (logarithmic for some queries) search complexity. Putting it together, our search is performed by traversing the search tree, where each node decision is made via very efficient secure function evaluation (SFE).

We use Bloom filters (BF) to store collections of keywords in each tree node. Bloom filters serve this role well because they support small storage, constant time access, and invariance of access patterns with respect to different queries and match outputs. For SFE, we use state-of-the-art Yao's garbled circuits.

Because of SFE's privacy guarantee in each tree node, the overall leakage (i.e. additional information learned by the players) essentially amounts to the traversal pattern in the encrypted search tree.

We discuss technical details of these and other aspects of the system, such as encrypted search tree construction, data representation, policy checking, etc., in Section 5.2.2. We stress that many of these details are technically involved.

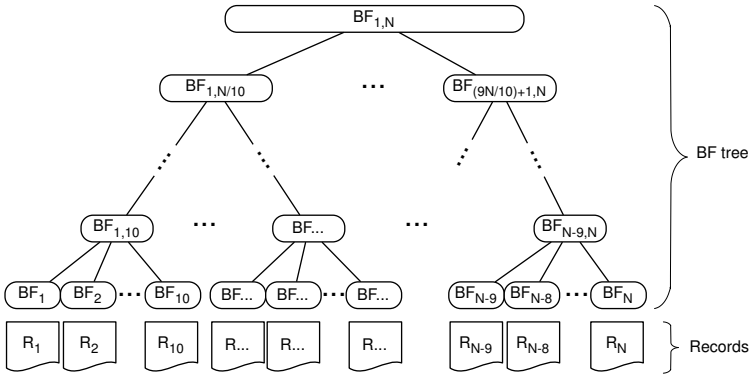
### 5.2.2 Basic System Design

In this section, we will begin by describing the basic system design supporting only simple private query. In the next section, we will augment this basic design with more features.

#### BF Search Tree

Our key data structure enabling sublinear search is a BF search tree for the database records. We stress that there is only one global search tree for the entire database. Let $n$ be the number of database records and $T$ be a balanced $b$-ary tree of height $\log_b n$ (we assume $n = b^z$ from some positive integer $z$ for simplicity). In our system, $b$ is set

to 10. In the search tree, each leaf is associated with each database record, and each node $v$ is associated with a Bloom filter $B_v$. The filter $B_v$ contains all the keywords from the (leaf) records that the node $v$ have (as itself or as its descendants). For example, if a node contains a record that has `Jeff` in the `fname` field, a keyword $\alpha =$ `'fname:Jeff'` is inserted to $B_v$. The length $\ell_v$ of $B_v$ is determined by the upper bound of the number of possible keywords, derived from DB schema, so that two nodes of the same level in the search tree have equal-length Bloom filters. The insertion of keywords is performed by shrinking the output of the hash functions $\zeta_{\ell_v}(\mathcal{H}(\alpha))$ to fit in the corresponding BF length $\ell_v$. Here, $\mathcal{H}$ is the set of hash functions associated with the root node BF. See Figure 5-9.




Let $(R_i, \ldots, R_n)$ be the overall database records. The Bloom filter $BF_{a,b}$ contains all the keywords of records $R_a, R_{a+1}, \ldots, R_b$.

Figure 5-9: Index structure: Bloom-filter based search tree.

**Search using a BF search tree.** Consider a simple recursive algorithm Search below. Let $\alpha$ and $\beta$ be keywords and $r$ the root of the search tree. Note that $\mathsf{Search}(\alpha \wedge \beta, r)$ will output all the leaves (i.e., record locations) containing both keywords $\alpha$ and $\beta$; any ancestor of a leaf has all the keywords that the leaf has, and therefore there should be a search path from the root to each leaf containing $\alpha$ and $\beta$. This algorithm can be easily extended to searching for any monotone Boolean formula of keywords.

$\mathsf{Search}(\alpha \wedge \beta, v)$:

If the BF $B_v$ contains $\alpha$ and $\beta$, then

If $v$ is a leaf, then output $\{v\}$.

Otherwise, output $\bigcup_{c:\text{ children of } v} \mathsf{Search}(\alpha \wedge \beta, c)$.

Otherwise, output $\emptyset$.

**Preprocessing**

Roughly speaking, in this phase, $\mathsf{S}$ gives an encrypted DB to $\mathsf{IS}$. To be more specific, by executing the following protocols, the two parties encrypt and permute the records, create a search tree for the permuted records, and prepare record decryption keys.

**Encrypting database index/records.** In this step, the server first permutes its DB to hide information of the order of records in the DB and then creates BF-search tree on this permuted DB; these DB and search tree are encrypted and sent to the index server.

1. (Shuffle and encrypt the records.) The server generates a key pair $(pk, sk)$ for a public-key semi-homomorphic (e.g., additively homomorphic) encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Given a database of $n$ records, the server $\mathsf{S}$ randomly shuffles the records. Let $(R_1, \ldots, R_n)$ be the shuffled records. $\mathsf{S}$ then chooses a random string $s_i$, and computes $\tilde{s}_i \leftarrow \mathsf{Enc}_{pk}(s_i)$ and $\tilde{R}_i = G(s_i) \oplus R_i$, where $G$ is a PRG.

2. (Encrypt the BF search tree.) $\mathsf{S}$ constructs a BF search tree $T$ for the permuted records $(R_1, \ldots, R_n)$. It then chooses a key $k$ at random for a PRF $F$. The Bloom filter $B_v$ in each node $v$ is encrypted as follows: $\tilde{B}_v = B_v \oplus F_k(v)$. (This encryption can be efficiently decrypted inside SFE evaluation by GC.)

3. (Share) Finally, the $\mathsf{S}$ sends the (permuted) encrypted records $(pk, (\tilde{s}_1, \tilde{R}_1), \ldots, (\tilde{s}_n, \tilde{R}_n))$ and the encrypted search tree $\{\tilde{B}_v : v \in T\}$ to the index server. The client will receive the PRF key $k$, and the hash functions $\mathcal{H} = \{h_i\}_{i=1}^{\eta}$ used in the Bloom filter generation.

**Preparing record decryption keys.** To save the decryption time in the on-line phase, the index server and the server precompute record decryption keys as follows:

(Blind the decryption keys) The index server $\mathsf{IS}$ chooses a random permutation $\psi : [n] \to [n]$. For each $i \in [n]$, it chooses $r_i$ randomly and computes $\tilde{s}'_{\psi(i)} \leftarrow \tilde{s}_i \cdot \mathsf{Enc}_{pk}(r_i)$. Then, it sends $(\tilde{s}'_1, \ldots, \tilde{s}'_n)$ to $\mathsf{S}$. Then, the server decrypts each $\tilde{s}'_i$ to obtain the blinded key $s'_i$. Note that it holds $s'_{\psi(i)} = s_i r_i$.

### Search

Our system supports any SQL query that can be represented as a monotone Boolean formula where each variable corresponds to one of the following search conditions: *keyword match*, *range*, and *negation*. So, without loss of generality, we support non-monotone formulas as well, modulo possible performance overhead (see how we support negations below). See Figure 5-10 as an example.

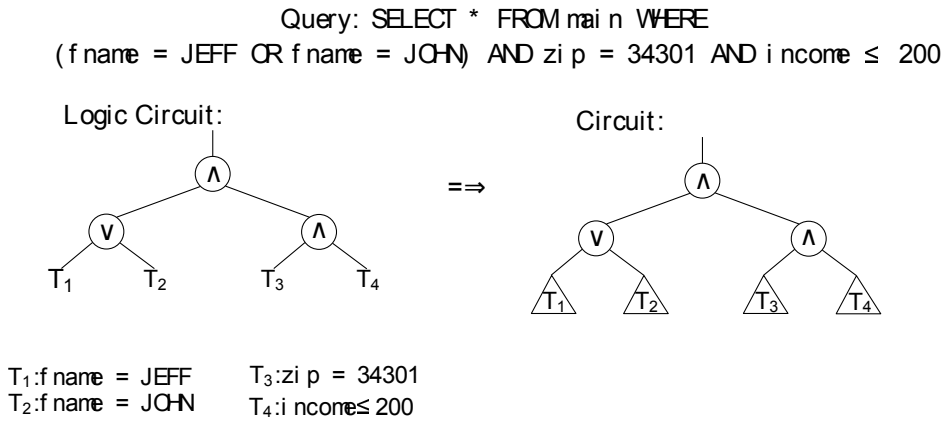



Figure 5-10: High level circuit representation of a query.

**Traversing the search tree privately.** The search procedure starts with the client transforming the query into the corresponding Boolean circuit. Then, starting from the root of the search tree, the client and the index server will compute this circuit $Q$ via secure computation. If the circuit $Q$ outputs true, the parties visit all the children of the node, and again evaluate this circuit $Q$ on those nodes recursively, until they reach leaf nodes; otherwise, the traversal at the node terminates. Note that evaluation of $Q$ outputs a single bit denoting the search result at that node. It is fully secure, and reveals no information about individual keywords.

In order to use secure computation, we need to specify the query circuit and the inputs of the two parties to it. However, since the main technicalities lie in constructing circuits for the variables corresponding to search conditions, we will describe how to construct those sub-circuits only; the circuit for the Boolean formula on top of the variables is constructed in a standard manner.

**Keyword match condition.**

We first consider a case where a variable corresponds to a keyword match condition. For example, in Figure 5-10 the variable $T_1$ indicates whether the Bloom filter $B_v$ in a given node $v$ contains the keyword $\alpha =$ `'fname:JEFF'`. Consider the Bloom filter hash values for the keyword $\alpha$, and let $Z$ denote the positions to be checked, i.e., $Z := \zeta_{\ell_v}(\mathcal{H}(\alpha))$. If the Bloom filter $B_v$ contains the keyword $\alpha$, the projected bits w.r.t. $Z$ should be all set, that is, we need to check

$$B_v{\downarrow_Z} \stackrel{?}{=} 1^\eta. \tag{5.1}$$

Recall that the index server has an encrypted Bloom filter $\tilde{B}_v = B_v \oplus F_k(v)$, and the client the PRF key $k$ and the hash functions $\mathcal{H} = \{h_i\}_{i=1}^{\eta}$. Therefore, the circuit to be computed should first decrypt and then check the equation (5.1). That is, the keyword match circuit looks as follows:

$$\mathsf{KM}((b_1, \dots b_\eta), (r_1, \dots, r_\eta)) = \bigwedge_{i=1}^{\eta} (b_i \oplus r_i).$$

Here, $(b_1, \dots, b_\eta)$ is from the encrypted BF and $(r_1, \dots, r_\eta)$ from the pseudorandom mask. That is, to this circuit $\mathsf{KM}$, the index server will feed $\tilde{B}_v{\downarrow_Z}$ as the first part $(b_1, \dots, b_\eta)$ of the input, and the client will feed $F_k(v){\downarrow_Z}$ as the second $(r_1, \dots, r_\eta)$. In order that the two parties may execute secure computation, it is necessary that the client compute $Z$ and send it (in plaintext) to the index server.

*Range/negation condition.* Consider the variable $T_4$ in Figure 5-10 for example. Using the technique from [65], we augment the BF to support inserting a number

$x \in \mathbb{Z}_{2^n}$, say with $n = 32$, and checking if the BF contains a number in a given range.

To insert an integer $a$ in a BF, *all* the canonical ranges *containing* $a$ are added in the filter. A canonical range with level $i$ is $[x2^i, (x+1)2^i)$ for some integer $x$, so for each level, there is only one canonical range containing the number $a$. In particular, for each $i \in \mathbb{Z}_n$, compute $x_i$ such that $a \in [x_i 2^i, (x_i+1)2^i)$ and insert '`r:income:`$i$`:`$x_i$'' to the Bloom filter.

Given a range query $[a, b)$, we check whether a canonical range *inside the given query* belongs to the BF. In particular, for each $i \in \mathbb{Z}_n$, find, if any, the minimum $y_i$ such that $[y_i 2^i, (y_i+1)2^i) \in [a, b)$ and the maximum $z_i$ such that $[z_i 2^i, (z_i+1)2^i) \in [a, b)$; then check if the BF contains a keyword '`r:income:`$i$`:`$y_i$'' or '`r:income:`$i$`:`$z_i$'. If any of the checks succeeds for some $i$, then output yes; otherwise output no. Therefore, a circuit for a range query is essentially ORs of keyword match circuits.

For example, consider a range query with $\mathbb{Z}_{2^4}$. When inserting a number 9, the following canonical ranges are inserted: $[9, 10)$, $[8, 10)$, $[8, 12)$, $[8, 16)$. Given a range query $[7, 11)$, the following canonical ranges are checked: $[7, 8)$, $[10, 11)$, $[8, 10)$. We have a match $[8, 10)$.

Negation conditions can be easily changed to range conditions; for example, a condition '`NOT work hrs = 40`' is equivalent to '`work hrs` $\leq$ `39 OR work hrs` $\geq$ `41`'.

*Overall procedure in a node.* In summary, we describe the protocol that the client and the index server execute in a node of the search tree.

1. The client constructs a query circuit corresponding to the given SQL query. Then, it garbles the circuit and sends the garbled circuit, Yao keys for its input, and the necessary BF indices.

2. The client and the index server execute OT so that IS obtains Yao keys for its input (i.e., encrypted BF). Then, the index server evaluates the garbled circuit and sends the resulting output Yao key to the client.

3. The client decides whether to proceed based on the result.

**Record Retrieval.** When the client and the index server reach some of the leaf

nodes in the tree, the client retrieves the associated records. In particular, if computing the query circuit on the $i$th leaf outputs success, the index server sends $(\psi(i), r_i, \tilde{R}_i)$ to the client. Then, the client sends $\psi(i)$ to $\mathsf{S}$, and gets back $s'_{\psi(i)}$. Note that it holds $s'_{\psi(i)} := s_i r_i$. The client $\mathsf{C}$ decrypts $\tilde{R}_i$ using $s_i$ and obtains the output record.

### 5.2.3 Advanced features

We now discuss advanced features such as query policies, and one-case indistinguishability. We also overview insert/delete/update operations from the server.

#### Policy Enforcement

The policy enforcement is performed through a three-party protocol among the query checker $\mathsf{QC}$ (holding the policy), the client $\mathsf{C}$ (holding the query), and the index server $\mathsf{IS}$. A policy is represented as a circuit that takes a query as input and outputs accept or reject. In our system, $\mathsf{QC}$ garbles this policy circuit, and $\mathsf{IS}$ evaluates the garbled policy circuit on the client's query. A key idea here is to have *the client and the query checker share the information of input/output wire key pairs in this garbled policy circuit;* then, the client can later construct a garbled query circuit (used in the search tree traversal) to be dependent on the output of the policy circuit. Assuming semi-honest security, this sharing of information can be easily achieved by having the client choose those key pairs (instead of $\mathsf{QC}$) and send them to $\mathsf{QC}$. The detailed procedure follows.

Before the tree search procedure described in the previous section begins, the client $\mathsf{C}$, the query checker $\mathsf{QC}$, and the index server $\mathsf{IS}$ execute the following protocol.

1. Let $\mathbf{q} = (q_1, \ldots, q_m) \in \{0,1\}^m$ be a string that encodes a query. The client generates Yao key pairs $\mathbb{W}_{\mathbf{q}} = ((w_1^0, w_1^1), \ldots, (w_m^0, w_m^1))$ for the input wires of the policy circuit, and a key pair $\mathbb{W}_x = (t^0, t^1)$ for the output wire. The client sends the key pairs $\mathbb{W}_{\mathbf{q}}$ and $\mathbb{W}_x$ to query checker $\mathsf{QC}$. It also sends the index server the garbled input $\tilde{\mathbf{q}} = (w_1^{q_1}, w_2^{q_2}, \ldots, w_m^{q_m})$.

2. Let $P$ be the policy circuit. $\mathsf{QC}$ generates a garbled circuit $\tilde{P}$ using $\mathbb{W}_{\mathbf{q}}$ as input key pairs, and $\mathbb{W}_x$ as the output key pair ($\mathsf{QC}$ chooses the other key pairs of $\tilde{P}$ at random). Then, $\mathsf{QC}$ sends $\tilde{P}$ to the index server.

3. The index server evaluates the circuit $\tilde{P}$ on $\tilde{\mathbf{q}}$ obtaining the output wire key $\tilde{x} = \tilde{P}(\tilde{\mathbf{q}})$. Note that $\tilde{x} \in \mathbb{W}_x$.

After the execution of this protocol, the original search tree procedure starts as described before. However, the procedure is slightly changed when evaluating a leaf node as follows:

1. Let $Q'(\mathbf{b}, \mathbf{r}, x) = Q(\mathbf{b}, \mathbf{r}) \wedge x$ be an augmented circuit, where $Q$ is the query circuit, $\mathbf{b}$ and $\mathbf{r}$ are the inputs from $\mathsf{IS}$ and $\mathsf{C}$ respectively, and $x$ is a bit representing the output from the policy circuit. The client $\mathsf{C}$ generates a garbled query circuit $\tilde{Q}'$ using wire key pair $\mathbb{W}_x$ for the bit $x$. Then, it sends $(\tilde{Q}', \tilde{\mathbf{r}})$ to the index server, where $\tilde{\mathbf{r}}$ is the garbled input of $\mathbf{r}$.

2. After obtaining the input keys $\tilde{\mathbf{b}}$ for $\mathbf{b}$ from OT with $\mathsf{C}$, the index server $\mathsf{IS}$ evaluates $\tilde{Q}'(\tilde{\mathbf{b}}, \tilde{\mathbf{r}}, \tilde{x})$ and sends the resulting output key to the client. Recall that it has already evaluated the garbled policy circuit $\tilde{P}(\tilde{\mathbf{q}})$ and obtained $\tilde{x}$.

3. The client checks the received key and decides to accept or reject.

Regarding privacy, the client learns nothing about the policy, since it never sees the garbled policy circuit. The index server obtains the topology of the policy circuit (from the garbled policy circuit).

Note that the garbled policy circuit is evaluated only once, before the search tree execution starts. Therefore, the policy checking mechanism introduces only a small overhead. It is also worth observing that, so far, we have not assumed any restriction on the policy to be evaluated. Since Yao-based computation can compute any function represented as a circuit, in principle, we could enforce any policy computable in a reasonable time (as long as it depends only on the query).

### One-case Indistinguishability

So far, in our system the index server learns how many records the client retrieved from the query. In many use cases, this leakage should be insignificant to the index server, in particular, when the number of returned results is expected to be, say, more than a hundred. However, there do exist some use cases in which this leakage is critical. For example, suppose that a government agent queries the passenger database of an airline company looking for persons of interest (POI). We assume that the probability that there is indeed a POI is small, and the airline or the index server discovering that a query resulted in a match may cause panic. Motivated from the above scenario, we consider a security notion which we call *one-case indistinguishability*.

**Motivation.** Consider a triple $(q, D_0, \mathbf{r})$ where $q$ is a query, and $D_0$ is a database with the query $q$ resulting in no record, but $\mathbf{r}$ satisfies $q$. Let $D_1$ be a database that is the same as $D_0$ except that a record is randomly chosen and replaced with $\mathbf{r}$. Let $\text{VIEW}_0$ (resp. $\text{VIEW}_1$) denote the view of $\mathsf{IS}$ when the client runs a query $q$ with the database $D_0$ (resp., $D_1$).

A natural start would be to require that for any such $(q, D_0, \mathbf{r})$, the difference between the two distributions $\text{VIEW}_0$ and $\text{VIEW}_1$ should be small $\epsilon$ (in the computational sense), which we call $\epsilon$ zero-one indistinguishability. However, it does not seem possible to achieve negligible difference $\epsilon$ without suffering significant performance degradation (in fact, our system satisfies this notion for a tunable small constant $\epsilon$). Unfortunately, this definition does not provide a good security guarantee when the difference $\epsilon$ is non-negligible, in particular, for the scenario of finding POIs. For example, let $\Pi$ be a database system with perfect privacy and $\Pi'$ be the same as $\Pi$ except that when it is 1-case (i.e., a query with one result record), the client sends the index server the message "the 1-case occurred" with non-negligible probability. It is easy to see that $\Pi'$ satisfies the definition with some non-negligible $\epsilon$, but it is clearly a bad and dangerous system.

**One-case indistinguishability.** Observe that in the use case of finding POIs, we don't particularly worry about "the 0-case", that is, it is acceptable if the airline

company sometimes knows that a query definitely resulted in no returned record. Motivated by this observation, this definition intuitively requires that if the a-priori probability of a 1-case is $\delta$, then a-posteriori probability of a 1-case is at most $(1+\epsilon)\delta$. For example, for $\epsilon = 1$, the probability could grow from $\delta$ to $2\delta$, but never more than that, no matter what random choices were made. Moreover, if the a-priori probability was tiny, the a-posteriori probability remains tiny even if unlucky random choices were made. In particular, consider $(q, D_0, \mathbf{r})$ and $D_1$ as before. Now consider a distribution $E$ that outputs $(b, v)$ where $b \in \{0, 1\}$ chosen with $\Pr[b = 1] = \delta$, and $v$ is the view of the index server when the query $q$ is run on $D_b$. The system satisfies $\epsilon$ one-case indistinguishability if for any $(q, D_0, \mathbf{r})$, $\delta$ and $v$, it holds

$$\Pr_E[b = 1|v] \leq (1 + \epsilon)\delta.$$

**Augmenting the design.** To achieve these indistinguishability notions, we change the design such that the client chooses a small random number of paths leading to randomly selected leaves. In particular, let $\mathcal{D}$ be the probability distribution on the number of random paths defined as follows:

$$\begin{aligned} \text{Distribution } \mathcal{D}: \quad & \text{For } 1 \leq x \leq \alpha - 1, \ \Pr_{\mathcal{D}}[x] = 1/\alpha. \\ & \text{For } x \geq \alpha, \ \Pr_{\mathcal{D}}[x] = (1/\alpha) \cdot 1/2^{x-\alpha+1}. \end{aligned}$$

Here, $\alpha$ is a tunable parameter. The client chooses $x \leftarrow \mathcal{D}$, and then it also chooses $x$ random indices $(j_1, \ldots, j_x) \leftarrow [n]^x$. When handling the query, the client superimposes the basic search procedure above with these random paths. Our system is $1/\alpha$ zero-one indistinguishable and $\epsilon$ one-case indistinguishable with $\epsilon = 1$. Intuitively, the leakage to the index server is the tree traversal pattern, and these additional random paths make the 0-case look like 1-case with a reasonably good probability.

If we slightly relax the definition and ignore views taking place with a tiny probability, say $2^{-20}$, we can even achieve both 1-case and 0-case indistinguishability at the same time; the probability of the number $x$ of fake paths is now $1/2^{|x-\alpha|+2}$ with

a parametrized center $\alpha$, say $\alpha = 20$ (except when $x = 0$, i.e., $\Pr[x = 0] = 1/2^{\alpha+1}$).

**Against the server.** One-case indistinguishability against the server is easily achieved by generating a sufficient number of dummy record decryption keys in the preprocessing phase; the index server will let the client know the (permuted) positions of the dummy keys. When zero records are returned from a query, the client asks for a dummy decryption key from the server. For brevity, we omit the details here, and exclude this feature in the security analysis.

#### Delete, Insert, and Update from the Server

Our system supports a basic form of dynamic deletion, insertion, and update of a record which is only available to the server. If it would like to delete a record $R_i$, then the server sends $i$ to the index server, which will mark the encrypted correspondent as deleted. For newly inserted (encrypted) records, the index server keeps a separate list for them with no permutation involved. In addition, it also keeps a temporary list of their Bloom filters. During search, the temporary list is also scanned linearly, after the tree. When the length of the temporary Bloom filter list reaches a certain threshold, all the current data is re-indexed and a new Bloom filter tree is constructed. The frequency of rebuilding the tree is of course related to the frequency of the modifications and also the threshold we choose for the temporary list's size. Our tree building takes one hour/100M records. Finally, update is simply handled by atomically issuing a delete and an insert command.

We note that updates is not our core contribution; we implement and report it here, but don't focus on its design and performance. A more scalable update system would use a BF tree rather than a list; its implementation is a simple modification to our system.

### 5.2.4 Security Analysis

We consider static security against a semi-honest adversary that controls at most one participant. We first describe an ideal functionality $\mathcal{F}_{db}$ parameterized with a

**Functionality $\mathcal{F}_{db}$**

**Parameter:** Leakage profile.

**Init:** Given input $(D, P)$ from $\mathsf{S}$, do the following:

1. Store the database records $D$ and the policy $P$. Let $n$ be the number records in $D$. Shuffle $D$ and let $(R_1, \ldots, R_n)$ be the shuffled records. Choose a random permutation $\pi : [n] \to [n]$. Construct a BF-search tree for $(R_1, \ldots, R_n)$ using the hash functions $\mathcal{H}$.
2. To handle the client's queries, it chooses hash functions $\mathcal{H} = \{h_i : \{0,1\}^* \to [\ell]\}_{i=1}^{\eta}$ for Bloom filters with parameters $(\eta, \ell)$ to maintain false positive rate of $10^{-6}$.
3. Finally, return a $\mathsf{DONE}_{\mathsf{init}}$ and the leakage to all parties.

**Query:** Given input $\mathbf{q}$ from $\mathsf{C}$, do the following:

1. Check if $\mathbf{q}$ is allowed by $P$. If the check fails, then disallow the query by setting $y = \emptyset$. Otherwise, for each $i \in [n]$, let $B_i \in \{0,1\}^{\ell'}$ be the Bloom filter associated with the $i$th leaf in the BF tree. For $i = 1, \ldots, n$, check if the query passes according to the filter $B_i$; if so, add $(i, R_i)$ to the result set $Y$.
2. Return $Y$ to $\mathsf{C}$ and return a $\mathsf{DONE}_{\mathsf{query}}$ message and leakage to all parties.

Figure 5-11: The Ideal Functionality $\mathcal{F}_{db}$

leakage profile in Figure 5-11, and then show that our system securely realizes the functionality where the leakage is essentially the search tree traversal pattern and the pattern of accessed BF indices.

For the sake of simplicity, we only consider security where there are no insert/delete/update operations,[1]and unify the server and the query checker into one entity. We also assume that all the records have the same length.

We use the DDH assumption (for ElGamal encryption and Naor-Pinkas OT), and our protocols are in the random oracle model (for Naor-Pinkas OT and OT extension). We also use PRGs and PRFs, and those primitives are implemented with AES.

[1]As access patterns are revealed, additional information for inserted/deleted/updated records is leaked. For example, $\mathsf{C}$ or $\mathsf{IS}$ may learn whether a returned record was recently inserted; they also may get advantage in estimating whether the query matched a recently deleted record. We stress that this additional leakage can be removed by re-running the setup of the search structure.

### 5.2.5 Security of Our System

With empty leakage profile, the ideal functionality $\mathcal{F}_{db}$ in Figure 5-11 captures the privacy requirement of a database management system in which each query is handled deterministically. The client obtains only the query results, but nothing more. The index server and the server learn nothing. Realizing such a functionality incurs a performance hit. Our system realizes the functionality $\mathcal{F}_{db}$ with the leakage profile described below. The security of our system can be proved from the security of the secure computation component, and is deferred to the full version.

**Leakage in Init.** Since the server has all the input, the leakage to $\mathsf{S}$ is none. The leakage to $\mathsf{C}$ is $n$, that is, the total number of records. The leakage to $\mathsf{IS}$ is $n$ and $|R_1|$.

**Leakage to $\mathsf{S}$ in each query.** We first consider the leakage to the server. The server is involved only when the record is retrieved. Let $((i_1, R_{i_1}), \ldots, (i_j, R_{i_j}))$ be the query results. Then, the leakage to the server is $(\pi(i_1), \pi(i_2), \ldots, \pi(i_j))$.

**Leakage to $\mathsf{C}$ in each query.** The leakage to the client is the BF-search tree traversal paths, that is, all the nodes $v$ in which the query passes according to the filter $B_v$.

**Leakage to $\mathsf{IS}$ in each query.** The leakage to the index server is a little more than that to the client. In particular, the nodes in the faked paths that the client generates due to one-case indistinguishability are added to the tree search pattern. Also, the topology of the query circuit and of the policy circuit is leaked to $\mathsf{IS}$ as well. Finally, the BF indices are also revealed to $\mathsf{IS}$ (although not the BF content), but assuming that the hash functions are random, those indices reveal little information about the query. However, based on this, after observing multiple queries, $\mathsf{IS}$ can infer some correlations a $\mathsf{C}$'s queries' keywords.

### 5.2.6 Discussion

**Leakage to the server.** We could wholly remove the leakage to the server by modifying the protocol as follows:

Remove the decryption key preparation (and blinded keys) in the preprocessing; instead, the client receives the secret key $sk$ from the server. The client (as the receiver) and the index server (as the sender) execute oblivious transfer at each leaf of the search tree. The choice bit of the client is whether the output of the query circuit is success. The two messages of the index server is the encrypted record and a string of zeros.

However, we believe that it is important for the server to be able to upper-bound the number of retrieved records. Without such control, misconfiguration on the query checker side may allow overly general queries to be executed, causing too many rows to be returned to the client; in contrast, in our approach, $\mathsf{S}$ releases record decryption keys at the end, and therefore it is easy to enforce the sanity check of the total number of returned records. Moreover, if $\mathsf{S}$ has a commercial DB, it may be convenient to implement payment mechanism in association with key release by $\mathsf{S}$.

**OR queries.** For OR queries passing the policy, our system leaks extremely small information. In particular, the leakage to the client is minimal, as the tree traversal pattern can be reconstructed from the returned records. As a consequence, if the client retrieves only document ids, the client learns nothing about the results for individual terms in his query. The leakage to the index server is similar. We believe that the topology of the SQL formula and the policy circuit reveals small information about the query and the policy. If desired, we can even hide those information using universal circuits [55] with a circuit size blow-up of a logarithmic multiplicative factor.

**AND queries.** For AND queries, the tree traversal pattern consists of two kinds of paths. The first are, of course, the paths reaching the leaves (query results). The second stop at some internal nodes due to our BF approach[2]; although the leakage from this pattern reveals more information about which node don't contain a given keyword, we still believe this leakage is acceptable in many use cases.

[2]For example, consider a query $q$ that looks for two keywords, say, $q = \alpha \wedge \beta$. Let $v$ be some node and $c_1, \ldots, c_b$ be the children of $v$ in the search tree. If $c_1$ contains only $\alpha$, and $c_2$ contains only $\beta$, then $v$ will contain both $\alpha$ and $\beta$, and so the node $v$ will pass the query; however, neither $c_1$ nor $c_2$ would.

We stress that the second leakage is related to the fact that a large linear running time seems to be *inherent* for some AND queries, irrespective of privacy, but depending only on the underlying database (see Section 5.2.11 for more detail). Therefore, if we aim at running most AND queries in sublinear time, the running time will inherently leak information on the underlying DB.

### 5.2.7 Implementation

We built a prototype of the proposed system to evaluate its practicality in terms of performance. The prototype was developed from scratch in C++ (a more than a year effort, almost two years including designing) and consists of about 10KLOC. In this section, we describe several interesting parts of the implementation that are mostly related to the scalability of the system.

**Crypto building blocks.** We developed custom implementations for all the cryptographic building blocks previously described. More specifically, we used the GNU Multiple Precision (GMP) library to implement oblivious transfers, garbled circuits and the semi-homomorphic key management protocol. The choice of GMP was mostly based on thread-safety. As for AES-based PRF, we used the OpenSSL implementation because it takes advantage of the AES-NI hardware instructions, thus delivering better performance.

**Parallelization.** The current implementation of Blind Seer supports parallel preprocessing and per-query threading when searching. For all the multi-threading features we used Intel's Threading Building Blocks (TBB) library. To enable multi-threaded execution of the preprocessing phase we created a 3-stage pipeline. The first stage is single-threaded and it is responsible for reading the input data. The second stage handles record preprocessing. This stage is executed in parallel by a pool of threads. Finally, the last stage is again single-threaded and is responsible for handling the encrypted records. Concurrently supporting multiple queries was straightforward as all the data structures are read-only. To avoid accessing the Bloom filter tree while it is being updated by a modification command, we added a global writer lock (which does

not block reads). Since we only currently support parallelization on a one-thread-per-query basis, it only benefits query throughput, not latency. However, long-running queries involve a large amount of interaction between querier and server that is independent and thus amenable to parallelization. The improvement we see in throughput is a good indicator for how much we could improve latency of slow queries by applying parallelization to these interactions.

**Bloom filter tree.** This is the main index structure of our system which grows by the number of records and the supported features (e.g., range). For this reason, the space efficiency of the Bloom filter tree is directly related to the scalability of the system. In the current version of our system we have implemented two space optimizations: one on the representation of the tree and another on the size of Bloom filter in each tree node.

Firstly, we avoided storing pointers for the tree representation, which would result in wasting almost 1G of memory for 100M records. This is achieved by using a flat array with fixed size allocations per record.

Secondly, we observed that naively calculating the number of items stored in the inner nodes by summing the items of their children is inefficient. For example, consider the case of storing the 'Sex' field in the database, which has only two possible values. Each Bloom filter in the bottom layer of the tree (leaves) will store either the value `sex:male` or `sex:female`. However, their parent nodes will keep space for 10 items, although the Sex field can have only two possible values. Thus, we estimate the number of items that need to be stored in a given level as the minimum between the cardinality of the field and the number of leaf-nodes of the current subtree. This optimization alone reduced the total space of the tree by more than 50% for the database we used in our evaluation.

**Keyword search and stemming.** Although we focus on supporting database search on structured data, our underlying system works with collections of keywords. Thus, it can trivially handle other forms of data, like keyword search over text documents, or even keyword search on text fields of a database. We actually do support the latter – in our system we provide this functionality using the special operator

`CONTAINED_IN(column, keyword)`. Also, we support stemming over keyword search by using the Porter stemming algorithm [1].

### 5.2.8 Evaluation

In this section, we evaluate our system. We first evaluate our system as a comparison with MySQL as a baseline, to establish what the performance cost of providing private search is. We then generalize the performance expectations of our system by performing a theoretical analysis based on the type of queries.

**Dataset.** The dataset we use in all of our tests for the first part of the evaluation is a generated dataset using learned probability distributions from the US census data and text excerpts from "The Call of the Wild", by Jack London. Each record in our generated database contains personal information generated with similar distributions to the census. It also contains a globally unique ID, four fields of random text excerpts ranging from $10 - 2000$ bytes from "The Call of the Wild", and a "fingerprint" payload of random data ranging from 50000 to 90000 bytes. The payload is neither searchable nor compressible, and is included to emulate reasonable data transfer costs for real-world database applications. The census data fields are used to enable various types of single-term queries such as term matching and range queries, and the text excerpts for keyword search queries.

**Testbed.** Our tests were run on a four-computer testbed that Lincoln Labs set up and programmed for the purpose of testing our system and comparing it to MySQL. Each server was configured with two Intel Xeon 2.66 Ghz X5650 processors, 96GB RAM (12x8 GB, 1066 MHz, Dual Ranked LV RDIMMs), and an embedded Broadcom 1GB Ethernet NICS with TOE. Two servers were equipped with a 50TB RAID5 array, and one with a 20TB array. These were used to run the owner and index server. MySQL was configured to build separate indices for each field. DB queries were not known in advance for MySQL or for our system.

### 5.2.9 Querying Performance

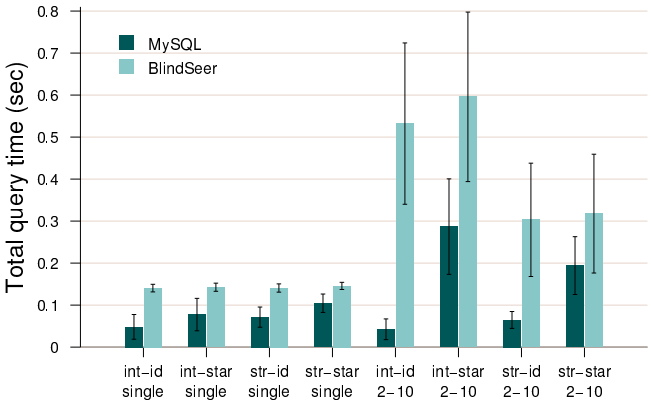



Figure 5-12: Comparison with MySQL for single-term queries that have a single result (first four bar groups) and 2 to 10 results (last four bar groups). The search terms are either strings (str) or integers (int) and the returned result is either the id or the whole record (star).

**Single term queries with a small result set.** Figure 5-12 shows a comparison of single term queries against MySQL. We expect the run time for both our system and MySQL to depend primarily on the number of results returned. The first four pairs show average and standard deviation for query time on queries with exactly one result in the entire database, and the latter four for queries with a few (2-10) results. Queries are further grouped into those which are run on integer fields (int) and string fields (str), and those which return only record ids (id) and those which return full record content (star). For each group, we executed 200 different queries to avoid caching effects in MySQL.

As we can see, for single result set queries, our system is very consistent. Unlike with MySQL, the type of query has no effect on performance, since all types are stored and queried the same way in the underlying Bloom filter representation. Also, the average time is dominated by the average number of results, which is slightly larger for integer terms. Unexpectedly, there is also no performance difference for returning record ids versus full records. This is likely because for a single record, the performance is dominated by other factors like circuit evaluation, tree traversal and key handling, rather than record transfer time. Overall, aside from some bad-case scenarios, we are generally less than 2× slower.

Variation in performance of our system is much larger when returning a few re-

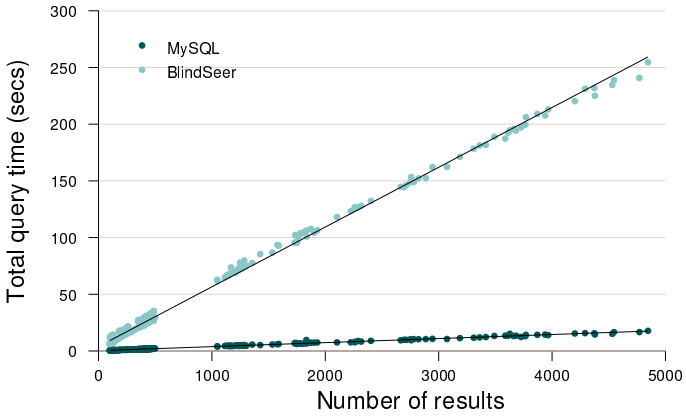



Figure 5-13: Comparison of the scaling factor with respect to the result set size, using single-term queries. Both MySQL and Blind Seer scale linearly, however, Blind Seer's constant factor is 15× worse (mostly due to increased network communication).

sults. This is because the amount of tree traversal that occurs depends on how much branching must occur. This differs from single result set queries, where each tree traversal is a single path. With the larger result sets, we can also begin to see increased query time for full records as opposed to record ids, although it remains a small portion of the overall run time.

**Scaling with result set size.** Figure 5-13 expands on both systems' performance scaling with the number of results returned. This experiment is also run with single term queries, but on a larger range of return result set sizes. As one would expect, the growth is fairly linear for both systems, although our constant factor is almost 15× worse. This indicates that for queries with a small result set, the run time is dominated by additive constant factors like connection setup for which we are not much slower than MySQL. However, the multiplicative constant factors involved in our interactive protocol are much larger, and grow to dominate run time for longer running queries. This overhead is mostly due to increased network communication because of the interactiveness of the search protocol. Although this is inherent, we believe that there is room for implementation optimizations that could lower this constant factor.

**Boolean queries.** Figure 5-14 shows our performance on various Boolean queries. The first three groups show average query time for 2-term AND queries. In each

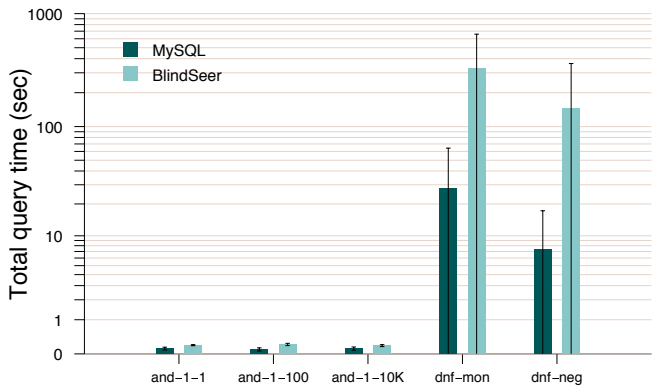



Figure 5-14: Boolean queries having a few results ($< 10$). The first three are two-term AND queries where one of the terms has a single result and the other varies from 1 to 10K results. The fourth group includes monotonic DNF queries with 4-9 terms, the last includes 5-term DNF queries with negations.

case, one term occurs only once in the database, resulting in the overall Boolean AND having only one match in the database. However, the second term increases up to 10000 results in the database. As we can see, our query performance does not suffer; as long as at least one term in a Boolean is infrequent we will perform well. The next two groups are more complex Boolean queries issued in disjunctive normal form, the latter including negations. The first one includes queries with 4-9 terms, and the second one, with 5 terms. These incur a larger cost, as the number of a results is larger and possibly a bigger part of the tree is explored. As we can see, MySQL incurs a proportionally similar cost.

We note that the relatively large variation shown in the graph is due to the different queries used in our test. Variation is much smaller when we run the same query multiple times.

**Parallelization.** We have implemented a basic form of parallelization in our system, which enables it to execute multiple queries concurrently. As there are no critical sections or concurrent modifications of shared data structures during querying, we saw the expected linear speedup when issuing many queries up to a point where the CPU might not be the bottleneck anymore. In our 16-core system, we achieved approximately factor 6x improvement due to this crude parallelization.

**Discussion.** We note several observations on our system, performance, bottlenecks,

etc.

Firstly, we note that our experiments are run on a fast local network. A natural question is how this would be translated into the higher-latency lower bandwidth setting. Firstly, there will be performance degradation proportional to bandwidth reduction, with the following exception. We could use the slightly more computationally-expensive, but much less communication intensive GESS protocol of [52] or its recent extension sliced-GESS [53], instead of Yao's GC. In reduced-bandwidth settings, where bandwidth is the bottleneck, sliced-GESS is about 3x more efficient than most efficient Yao's GC. Further, we can easily scale up parallelization factor to mitigate latency increases. Looking at this in a contrapositive manner, improving network bandwidth and latency would make CPU the bottleneck.

All search structures in our system are RAM-resident. Only the record payloads are stored on disk. Thus, disk should not be a bottleneck in natural scenarios.

### 5.2.10 Other Operations

Although querying is the main operation of our system, we also include some results of other operations. First, we start with the performance of the setup phase (preprocessing). Blind Seer took roughly two days to index and encrypt the 10TB data. As mentioned before, this phase is executed in parallel and is computationally efficient enough to be IO-bounded in our testbed. We note that the corresponding setup of MySQL took even longer.

Policy enforcement was another feature for which we wanted to measure overhead. However, in our current implementation, it cannot be disabled (instead, we use a dummy policy). We experimentally measured the overhead of enforcing the dummy policy versus more complex ones, but there was no noticeable difference. We plan to add the functionality to totally disable policy enforcement – because it is an optional feature – and measure its true performance overhead. Our expectation is that it will be minimal.

Finally, we performed several measurements for the supported modification commands: insert, update and delete. All of them execute in constant time in the order

of a few hundred microseconds. The more expensive part though is the periodic re-indexing of the data that merges the temporary Bloom filter list in the tree (see Section 5.2.3). In our current prototype, we estimated this procedure to take around 17 minutes, while avoiding re-reading the entire database. This can be achieved by letting the server store some intermediate indexing data during the initial setup and reusing it later when constructing the Bloom filter tree.

### 5.2.11 Theoretical Performance Analysis

In this section, we discuss the system performance for various queries by analyzing the number of visited nodes in the search tree. Let $\alpha_1, \ldots, \alpha_k$ be $k$ single term queries, and for each $i \in [k]$, let $r_i$ be the number of returned records for the query $\alpha_i$, and $n$ be the total number of records.

**OR queries.** Our system shows great performance with OR queries. In particular, consider a query $\alpha_1 \vee \cdots \vee \alpha_k$. The number of visited nodes in the search tree is at most $r \log_{10} n$, where $r = r_1 + \ldots + r_k$ is the number of returned records. Therefore, performance scales with the size of the result set, just like single term queries.

**AND queries.** The performance depends on the best constituent term. For the AND query $\alpha_1 \wedge \cdots \wedge \alpha_k$, the number of visited nodes in the search tree is at most $\min(r_1, \ldots, r_k) \cdot \log_{10} n$. Note that the actual number of returned records may be much smaller than $r_i$s. In the worst case, it may even be 0; consider a database where a half of the records contain $\alpha$ (but not $\beta$) and the other half $\beta$ (but not $\alpha$). The running time for the query $\alpha \wedge \beta$ in this case will probably be linear in $n$. However, we stress that this seems to be *inherent*, even without any security. Indeed, without setting up an index, every algorithm currently known runs in linear time to process this query.

This can be partially addressed by setting up an index, in our case by using a BF. For example, for AND queries on two columns, for each record with value `a` for column `A`, and value `b` for column `B`, the following keywords are added: `A:a, B:b, AB:a.b`. With this approach, the indexed AND queries become equivalent to single term queries. However, this cannot be fully generalized, as space grows exponentially

in the number of search columns.

**Complex queries.** The performance of CNF queries can be analyzed by viewing them as AND queries where each disjunct (i.e, OR query) is treated as a single term query. In general, any other complex Boolean query can be converted to CNF and then analyzed in a similar manner. In other words, performance scales with the number of results returned by the best disjunct when the query is represented in CNF. Note that we do *not* actually need to convert our queries to this form

(nor know anything about the data, in particular, which are high- or low-entropy terms) in order to achieve this performance (this aspect is even better than MySQL).

**Computation and Communication.** Both computational and communication resources required for our protocol are proportional to the query complexities described above.

**False Positives.** As our system is built on Bloom filters, false positives are possible. In our experiments, we set each BF false positive rate to $10^{-6}$. Assuming the worst-case scenario for us, where the DB is such that many of the search paths do reach and query the BFs at the leaves, this gives $10^{-6}$ false positive probability for each term of the query. Of course, the false positive is a tunable parameter of our system.

# Chapter 6

# Anonymous Publish-Subscribe

While prior to this work there have been many different kinds of anonymous communication systems, they have all worked on an address basis, and generally serve unicast applications. Users must choose a single recipient for their messages, and the system would relay those messages while protecting one or both identities. These include messaging systems and relays such as mixnets [21, 42] and onion routing networks [67] (discussed in Chapter 3), and publishing systems like FreeHaven [35] and FreeNet [28]. Anonymous relays allow users to send addressed messages while protecting their identities from the recipients and against all third parties. They may also allow users to create pseudonymous "addresses" that they can announce, whereby others may contact them without knowing who their true identities. Anonymous publishing systems allow users to store and advertise documents online that can then be freely accessed by the public without revealing the identity of the authors. They may also protect the identities of readers who are either accessing these documents or using search protocols that allow them to find documents of interest to them.

In the area of non-anonymous communication systems, one model for communication is publish-subscribe. In the publish-subscribe model, messages can be published to topics, rather than addressed to recipients. These are then multicast to the entire set of recipients that have previously subscribed to those topics. These topics can be anything from a set of specific match strings to ranged attributes on a multi-dimensional array. These types of messaging systems are typically implemented

using a third party who manages subscriptions and acts as a relay between publishers and subscribers, though distributed systems have been implemented which allow for greater scalability.

This paradigm adds a different kind of flexibility in that senders and recipients are decoupled and can operate without even knowing of each other's existence. This can be a more suitable mode of operation for many kinds of systems. For example, a chat or newsgroup application is more cleanly implemented where the speaker does not have to obtain and maintain an enumerated list of all people who are interested in what he has to say. In a normal addressing system, he would have to be aware of all the people he means to send messages to. In a publish-subscribe system, he only needs to publish his message on a topic, and all interested readers can subscribe to those topics without either party knowing of the other. Any system where addressing is preferably based upon the nature of the content rather than knowledge of the recipient can benefit from a publish-subscribe architecture.

In this chapter, we construct two systems to achieve anonymous publish-subscribe, meeting the requirements described in Chapter 2. They support a push publish-subscribe architecture: messages will be delivered to recipients without needing to be polled or requested on an individual basis. It also supports publication topics as string matches, integer ranges, and multi-attribute ranges mapping integer values to multiple labels.

Anonymous communication systems are of clear value for protecting sensitive data or interests. Although there are some systems that claim to provide anonymous publish/subscribe, neither the difficulty of identifying publishers nor the efficiency of the system is thoroughly analyzed. This is unfortunate, since the publish-subscribe paradigm is a natural match for anonymous communication. In many scenarios which require anonymous communication, there are two separate problems: how to establish relationships between sender and receiver when neither knows the other's identity, and then how to anonymously deliver their messages. Many anonymous communication systems do not address the first problem of how to establish anonymous relationships where meaningful communication should occur; this is left as outside the scope of the

system. But by its very nature, publish-subscribe aims to support communication based on content rather than by identity, and users need not concern themselves with the details of *finding* the entities they aim to communicate with anonymously. It thus naturally solves this issue.

Such a system could for example allow for newsgroups and real-time chat applications that discuss sensitive topics like medical conditions or radical political movements such as discussions between members of Falun Gong or Arab Spring. Since in a group discussion a user is already sending messages without requiring awareness of the recipients, it is a natural step to provide a guarantee that this identity remain anonymous.

These applications could not be efficiently met by existing anonymous communication systems, which do not support any form of multi-cast and work based on known-recipient addressing. Nor could they be met by anonymous publishing systems, which work on a pull-basis rather than a push-basis. This makes them unsuitable for real-time applications. Publish-subscribe systems naturally provide flexibility that is likely to be useful for any type of anonymous communication need, since they do not require assumptions about participant identity by other participants.

## 6.1 Central server routing

Our first solution will be based off of a central server that handles the logic of matching publications to subscriptions and routing. To guarantee anonymity from this server, and from other participants, both publishers and subscribers will connect to it through an obfuscating proxy, which is trusted not to collaborate with the server. There are thus four types of entities:

- *Server:* Stores subscriptions, matches publications, and routes messages. It should not be able to read message content, subject content, or identify senders or recipients.
- *Publisher:* Sends messages into the system.

- *Subscriber:* Sends subscriptions and receives matching messages from the system.

- *Proxy:* Entry point for communication between the server and publishers or subscribers. It is responsible for all contact with these entities, and for obscuring their identities from the server.

Trust is separated between the proxy and the server. The proxy will be able to see the identities of senders and recipients of messages, but will not be able to see the content of the messages being sent or the subjects they are being sent upon. The server will be able to see a deterministic encryption of this information, but will not know who is sending or receiving the messages. Although these deterministic encryptions can be matched to each other, since all origin points look identical to the server, he cannot link publishers. This separation of trust ensures to the user that no single entity can monitor his behavior.

To achieve this separation of information, we make use of a protocol called re-routable encryption [65]. This protocol allows for a sender and a receiver, each with unique symmetric encryption keys, and a third router entity. It provides a fast multi-party computation between the three parties, resulting in the router receiving a transformation key which then allows him to transform messages encrypted by the sender into messages encrypted by the receiver's key without being able to see or compute the cleartext on his own. This protocol allows us to efficiently realize the separation of trust between the server and proxy. These transformation keys will be generated between the server, proxy, and client (publisher or subscriber) once to introduce each participant to the system. Owning transformation keys allows the proxy to relay messages to and from the server without seeing their content or revealing the other communicating party without the expensive overhead of an obfuscating mixnet.

We also make use of Bloom filters [8] to manage and match large quantities of publications and subscriptions on the server end while obscuring topic content from the server. Bloom Filters allow matching against sets that can store any number

of elements with a boundable false positive rate that can be reduced by increasing Bloom Filter size relative to the number of terms stored.

The server will store an index of all subscriptions on a per-subscriber basis as a Bloom Filter index. Each subscriber is represented as one Bloom Filter storing all of his subscriptions. The exact nature of the subscriptions can be anything supported by Bloom Filters (exact topic keywords, ranges using our range-query protocol, multi-dimensional ranges, etc.) In our system, the subscribers will be pseudonymous from the server's point of view. If subscribers wish to prevent linkage between their subscriptions, they can do so by creating a pseudonym per subscription.

The proxies will use re-routable encryption to deliver messages from source to destination: deterministic for communication of subjects from publisher to server, and non-deterministic for communication of messages from publisher to server to subscriber. The proxy contains a transformation key from the server key to and from the key of each subscriber. This key must be computed between the user, proxy, and server once to join each new user into the system. The proxy maintains this mapping using the same pseudonyms used by the Bloom Filter index held on the message server. The server maintains its own encryption key $k_r$. To subscribe to a subject, a user generates his own key $k_u$, engages in secure multiparty computation with the proxy and server that results in the server learning transformation key $k_{\frac{r}{u}}$. He then deterministically encrypts his subject subscription under $k_u$ and sends it to the proxy, who transforms it to encryption under $k_r$ and forwards it to the server where it is stored, along with a pseudonym that the proxy would understand to correspond to the user. We are now ready for publishers to send messages to subscribers. The system and message path is thus laid out as in Fig. 6-1.

A publisher generates a message $m$, and encrypts it non-deterministically under $k_u$. He determines a subject $s$, and encrypts it deterministically under $k_u$. These are both transformed by the proxy to be encrypted under $k_r$ before being forwarded to the server. The server checks $s$ under deterministic encryption by $k_r$ against his BF index.

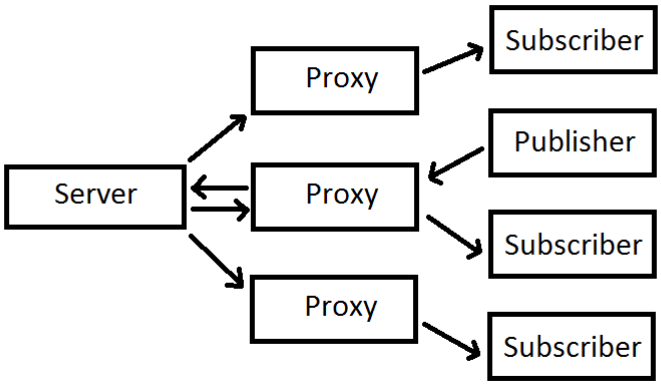



Figure 6-1: Centralized anonymous publish-subscribe system

He then re-randomizes the random component of $m$ encrypted by $k_r$ and sends it to the proxy, along with all the BF match identities. For each corresponding recipient $u'$, the proxy transforms $m$ to encryption under $k_{u'}$, and forwards the message to the corresponding users.

### 6.1.1 Security Analysis

That our system achieves *completeness* and *non-excessiveness* is easy to see: since Bloom Filters promise a zero false negative rate all messages will be properly routed to their appropriate destinations. By adjusting the Bloom Filter parameters, the amount of excess publications created by false positives can be made arbitrarily small.

We aim to protect the identity of the participants from the server, and the content of the messages and subjects from the proxy. This is under the assumption that the proxy and server do not collaborate with each other, and that the proxy does not collaborate with other publishers or subscribers. The server and proxy are trusted to be honest-but-curious; that is they will obey the protocols, but may attempt to learn more than they should from the results.

**Claim 22.** *Our system achieves trusted-party publisher anonymity.*

Let us assume that there were a full collaboration of all other meaningful entities

except for the proxy, who we will treat as a trusted party. The receiver sees only a delivered message which is entirely agnostic to its origin. Similarly, the server sees only the message and topic delivered by the proxy after transforming them to encryption under his own key. These would look identical regardless of which user originated the publication. Thus even if the adversary consisted of the server, the receiver, and any number of dishonest publishers, their combined view of any given message looks identical regardless of which honest user sent it. Therefore, they cannot gain an advantage in identifying the user.

**Claim 23.** *Our system achieves both trusted-party topic subscriber anonymity and trusted-party subscription anonymity.*

Again, we can assume collaboration between the server and any number of publishers publishing on a given topic. The subscription is delivered to the server by the proxy after transforming the encryption to his own key. This would look identical regardless of which user is subscribing to the topic. Thus for a given topic, he will see only a set of subscriptions which do not give any information that distinguishes between subscribers. In the reverse, given a subscriber, the subscriptions seen by the server do not look different whether or not he is the origin. Thus the server cannot gain any information that would help distinguish between subscribers given a topic, or identify topics given a subscriber.

The proxy is privy to both of these identities, and is thus treated as the trusted third party. However, he cannot see what content is being delivered or what topic it is being published to. This is given under the same assumptions as the underlying re-routable encryption scheme.

Owing to the use of Bloom Filters to match publications and subscriptions, there is an existing false positive rate. However, since the system is used for open subscription, this is a non-issue from a security standpoint. The recipient can simply ignore any messages he is not interested in. As we mentioned before, this system is only secure if the proxy cannot act as a user. If he is able to, then he can use transformation keys to transform anything encrypted by the server's key into his own key, and thus

read messages that are routed through him, breaking the security of the system.

## 6.2 Spanning tree routing

Our second solution will route messages to all subscribers using per-subscriber spanning tree structures. This will be accomplished by providing an overlay network of the nodes, and then representing each spanning tree within the routing tables of the nodes in the overlay.

The list of nodes in the network will be registered in a global directory, which can be either a single server, or a DHT for greater scalability. Nodes will then use Bloom Filter indexes as routing tables to forward messages by checking their subjects against the indexes. A destination will be represented as a single Bloom Filter, storing subjects as elements in the filter. Subscriptions live as elements in these filters. We can thus support routing based on topic for any type of topic that can be represented in a Bloom Filter (i.e. exact strings, ranges, etc.). In order to prevent cycles, each message will carry a header with a Bloom Filter storing unique labels that nodes can check to see if they have already forwarded the same message. These will be randomly generated and updated on regular intervals.

All that remains now is to set the routing Bloom Filters such that all published messages will be received by all interested subscribers. To do this, each subscriber will construct a unique spanning tree of the network, rooted on himself. We assume the existence of an underlying anonymous communication network that allows both sending a message while protecting the identity of the sender, and providing a pseudonymous address by which other users can route messages to a recipient who wishes to protect their true identity. In our implementation, we use Tor [67] to provide these functionalities. Although Tor has many known limitations, it is efficient and used often in the real world.

A subscriber will then anonymously instruct all nodes in the network to add a routing entry for that subject to their parent in his uniquely constructed tree. Thus, any message anywhere in the network, when routed on this subject, will find its way

to every subscriber that is interested. Although expensive for subscription, this is fast for publication, and so is well suited for systems where publications dominate subscriptions in terms of network load. Unsubscription is a little trickier, and is not handled by our current implementation. We could handle this in the future either by allowing Bloom Filters to expire and requiring subscriptions to be updated on a regular basis, or by using counting Bloom Filters which will allow deletion of entries.

- `Subscribe(u,t)`*:* User $u$ looks up the node from directory $D$. As a constant parameter of the system, he assumes a routing chain length of $r$. He then constructs a random, balanced spanning tree of depth $r$ using all nodes in the network with himself as the root. For each node in the tree, he anonymously contacts that node, and instructs it to route all messages with subject $t$ to their parent in the tree. This is done more efficiently by forwarding instructions for each node through their parents with layered encryption in the same manner as an onion routing network. Thus, we multicast the subscription along the same structure as the tree itself.

- `Publish(m,t)`*:* The sender picks a random origin point in the network, and uses the underlying anonymous communication network to send his message to that node. That node routes his message to the nodes indicated by looking up the subject on its own Bloom Filter index. All other nodes will forward the message in similar fashion, except first checking their loop-detection label, then inserting it into the header of the message.

The system and message path is thus laid out as in Fig. 6-2.

In the absence of false positives within the routing Bloom Filters, the common case is that for each publication, there will be a randomly selected path of length $r$ from the initial node the publisher selects to each subscriber. If the number of nodes is significantly larger than the number of subscribers for a single topic, then in all likelihood, the initial node will have to multiply the message for each end subscriber.

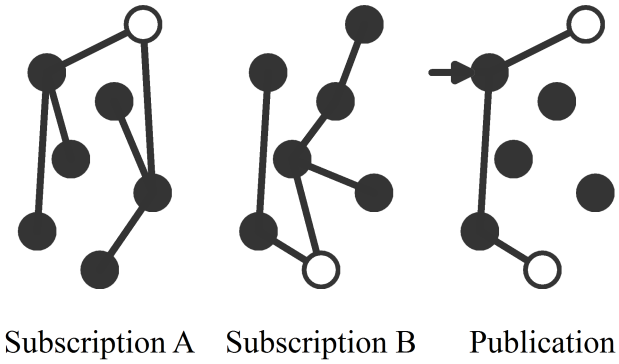



Figure 6-2: Spanning tree anonymous publish-subscribe system

However, this is still a benefit over the central server solution, since the central server solution uses a single node to multiply all traffic within the system, whereas with the spanning tree solution, each publication will use a different randomly selected initial point. This will better distribute load when there are a large number of publications going out simultaneously.

### 6.2.1 Efficiency comparison

We now compare the efficiency of our protocols for $n$ receivers, and a tree depth of $k$, and a load involving $p$ simultaneous publishers.

The primary tradeoffs are between the longest path, which affects the latency of message delivery, and the bottleneck branching point, which determines how scalable the protocol is. The central server solution will have a fixed longest path of four hops (from publisher, to proxy, to server, to proxy, to subscriber). So for low-load situations, we can expect the latency to be $O(1)$. Conversely, the spanning tree solution's path will scale with the depth of the spanning trees selected, and so has a worse $O(k)$ behavior. However, spanning tree depth is simply chosen to add layers of indirection and does not need to scale with the size of the system. So while the spanning tree has worse latency, it is boundable.

Both solutions involve a single point of multicast for each publication in the expected case. For the central server solution, this is the server. For the spanning tree

solution, if the subscribers are not a significant portion of the total userbase, then likely each has a unique path from the publisher, making the publisher the point of multicast. However, in the central server solution, all published messages share the same point, whereas in the spanning tree solution each has a unique one. Thus the central server solution faces a load of $\theta(kp)$ on the server, whereas the spanning tree solution faces a load no greater than $\theta(k)$ on any one node. Clearly, the spanning tree solution can handle multiple publisher load scaling better.

### 6.2.2 Security analysis

Our system provides *Completeness* and *Non-excessiveness* under the honest-but-curious model, that is when all parties perform the protocol correctly but may try to learn more than they should. If all nodes are forwarding correctly, messages are guaranteed to be routed to all interested subscribers, with a boundable false positive rate on loop detection.

**Claim 24.** *Assuming the security of the underlying anonymous communication system, our system achieves complete publisher anonymity.*

The message itself does not contain any information unique to the publisher. From the perspective of the initial receiving node, any message it receives is only visible as an output of the underlying anonymous communication system. Thus, if it can identify the origin, then that implies a failure of that system. From then on, clearly no other node in the network can do better in terms of identifying the publisher.

**Claim 25.** *Assuming our underlying TOR system is secure against identification attacks, our system achieves topic subscriber anonymity.*

For any given topic in the system, every node will have one or more nodes which it is expected to route matching messages towards. And if TOR protects the identities of its senders, then the subscription process itself will not reveal the subscriber identity. Thus, no node can distinguish between a neighbor who is an interested party, and a neighbor who is merely forwarding towards one. In order to identify a node as an

endpoint, an adversary would need to identify a publication originating from one of the leaves of its subscription tree, and then compromise all of the nodes on the path from publisher to subscriber, a requirement as stringent as for an adversary of TOR.

A more complicated issue is denial of service attacks. An attacker who was himself a subscriber could refuse delivery of messages to nodes he is intended to forward towards. However, since it is the subscribers who choose the routing tree that leads to them, an attacker would not be able to fully block a particular subscriber from receiving messages, nor fully block a publisher from disseminating them. Nor would he have any control over which particular publisher-subscriber relationships he could interfere with. Furthermore, if subscriptions are updated on a regular basis, his sphere of influence would be steadily changing. A system of checks wherein a subscriber occasionally publishes test messages and begins them at different points in the network could be implemented to specifically identify malicious nodes, however this remains to be further developed.

## 6.3 Performance

We implemented and tested both our central server and splay-tree based systems to observe scaling issues both in subscription and publication. Unfortunately, to our knowledge, there exist no other anonymous publish-subscribe systems to compare to, so we show only to demonstrate usability in comparison to normal network transactions, and to demonstrate efficiency differences between the two. To obtain a large number of nodes for scalability testing, we used the PlanetLab network [27]. Each participating node has at minimum 4x 2.4Ghz Intel cores, 4 GByte ram, and 500GB disk space. Nodes are distributed around the globe to provide a simulation of internet traffic. For our experiments, we used nodes with varying geographic locations contained within the US.

Figure 6-3 shows time to add subscriptions for a varying number of subscribers for the central server and spanning tree solutions. This was done for a system with a total of 500 participating nodes. Measurements for the central server solution were

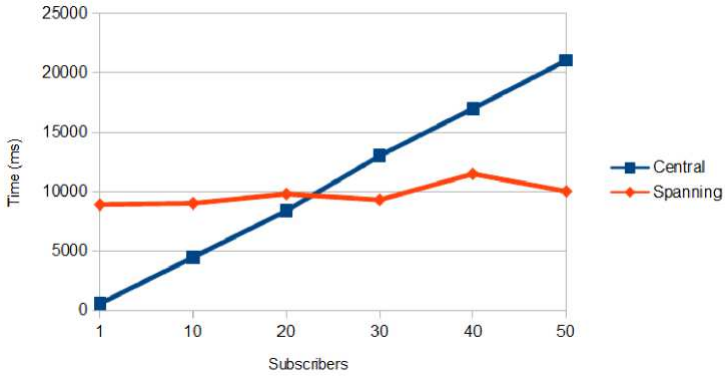



Figure 6-3: Subscription cost

taken using a one server and two proxy arrangement (one proxy for publishers and one for subscribers). This was measured from a start time when the subscription requests are queued into the systems, to the time when the last subscriber completes their request. Time scales roughly linearly for the central server system, because it is bottlenecked by the single point of connection and later subscriptions must wait for earlier ones to complete. The spanning tree solution performs worse at lower numbers of subscribers, due to its more involved protocol. However, it scales much better for larger numbers of subscribers as they can be handled concurrently. The growth is not entirely smooth, as the tree generation for each of the subscribers is random, and can cause more or less requests to be bottlenecked by various nodes depending on what kinds of overlap results.


Figure 6-4: Publication cost

Figure 6-4 shows time to deliver a publication. This was measured by observing a single node which subscribes to the same topic it is publishing on, and recording the time taken to receive its own message. Messages chosen were small text strings. This was measured from start time when the publisher initiated the publication to time when the last subscriber reported reception of the message. Again, time taken scales linearly with the number of subscribing nodes, again bottle-necking on the server which must duplicate the message once for each subscriber. In this test, only one publication is issued into the system at a time. Because of this, the spanning tree solution scales with similar behavior to the central server solution, but with a large constant overhead for the multiple hop message transmissions.

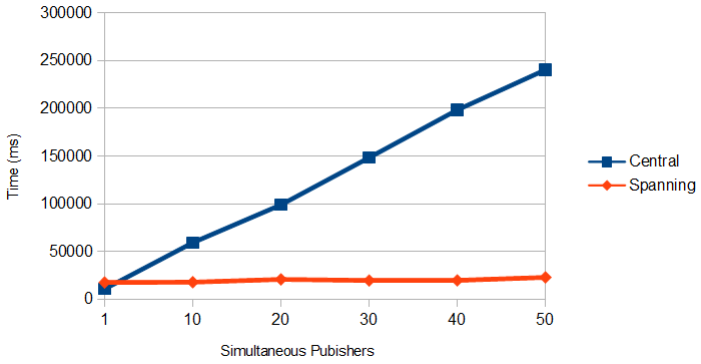



Figure 6-5: Publication cost in active system

Figure 6-5 shows the same measurements taken with increasing numbers of simultaneous publications issued into the system. The number of subscribers is kept constant at 50. The X axis shows the number of participating publishers, with each sending a publication at the same time. The Y axis shows the time taken for a single node which we are monitoring to receive a single publication it has itself sent into the system. In this case, we see that the spanning tree solution steadily outperforms the central server solution, demonstrating a greater ability to distribute the load in an active system with multiple publishers and subscribers.

# Chapter 7

# Identity Management

Master secret based identity systems allow users to obtain anonymous certificates from a central certificate authority. They can then use these certificates to register anonymously in various organizations and obtain pseudonymous membership credentials. The credentials issued are unforgeable and the anonymity they provide, conditional, i.e., misbehaving users are identified. Unfortunately, when applied in real world systems, previous anonymous credential systems have been lacking in many "real world" issues.

First of all, no *efficiency* optimization has been applied. User registration, credential issue procedures require a respectable amount of exponentiations, which in large scale systems would cause a serious bottleneck. In most credential systems there are no scalable *credential blacklistability* operations. In the large extent of identity theft cases, nowadays, *credential blacklistability* is a necessity when it comes to real time systems. Another issue is that, currently, there is no *variety* in organization registration policies. This becomes particularly important in "real world" systems as i.e., banks will probably require a lot more information to register a user than an online subscription service. However, the most important deficiency of current credential systems is the the lack of any sense of *secret information update* procedure. Master secrets may be compromised or stolen and, previously, there was no efficient way for a user to transfer his old registrations and credentials to his new account, while the attacker can make use of the old secret.

We will take in consideration all of these matters and design a privacy preserving and master secret-based identity management system, where every individual can generate a single master secret, with which he can

- demonstrate multiple times unlinkably and anonymously,
- use to register to organizations along with other credentials depending on each organization's policy
- change, when compromised; master secret update is done recursively for all the registrations a user has made through his previous master secret.
- recover, when lost or destroyed.

We need to emphasize on the fact that our system is deployable, i.e.,

- we make "real world" assumptions for our adversary's powers,
- we take in consideration "real world" settings for the various organizations
- it is computationally efficient.

## 7.1 System Architecture

### 7.1.1 Entities

The entities in our system are specified by our system's application. First of all, we identify users, who are the citizens of a country; they may be self-employed or employees of one or more companies, maintain — depending on their financial status — one or more bank accounts, receive and perform payments, use internet for online purchases and information retrieval, while taxed accordingly. Banks, Employers, Tax Authority TA, can, thus, be added to the set of our system's entities. All valid users and organizations of a country are registered to that country's registration authority RA. RA maintains information regarding each U's identification, i.e., U's birthday, birthplace, parents and may issue one or more certificates regarding U's identification attributes.

### 7.1.2 Operations

Based on our system's use, the main operations supported are the following:

1. Generate Master Identity, where Ugenerates his single master secret identity, i.e., $\mathsf{Ms}_{\mathcal{U}}$.

2. Register, where U registers to RA and validates $\mathsf{Ms}_{\mathcal{U}}$.

3. ShowMS, where U demonstrates knowledge of his RA-validated $\mathsf{Ms}_{\mathcal{U}}$; in our system, this procedure is extended to include ShowMSAttribute operations, where U proves that his $\mathsf{Ms}_{\mathcal{U}}$ has a particular attribute.

4. ObtainMembership, where U and Organization org collaborate for U to become a member of org. Depending on the type of org, ObtainMembership prerequisites may vary.

5. UpdateMS, where U and RA collaborate for U to obtain a new $\mathsf{Ms}_{\mathcal{U}}$; all past U's registrations through his ex-$\mathsf{Ms}_{\mathcal{U}}$ are also be updated.

6. RecoverMS, where the U-authorized people collaborate to recover $\mathsf{Ms}_{\mathcal{U}}$; this procedure only takes place in case of emergency.

### 7.1.3 Requirements

Our system, as mentioned before, is privacy preserving. In the context of an identity management system, privacy can be interpreted as the combination of *unlinkability* and *anonymity*, namely no one should be able to link a particular system activity to a different one as having originated by the same individual (*unlinkability*) or to a particular identity (*anonymity*). It is critical that privacy provisions are conditional: a misbehaving party will have both his identity and entire activity revealed.

Another fundamental requirement of our system is *deployability*. *Deployability* requires that we take in consideration the current settings of various organizations realized in our protocols: (a) their functionality and how "misbehavior" is defined in

them, i.e., under what conditions should their members be considered as malicious, (b) their participation, since we have to optimize our protocols' efficiency accordingly.

Master secret credential *unforgeability* and *forward secrecy* are two more of our requirements. In particular, we require that no credential can be created without the participation of the authorized authorities, such that the MSShow operation accepts(*unforgeability*). *Forward Secrecy* requires that no past activities of a particular individual are traced through the UpdateMS or RecoverMS procedures.

## 7.2 System Design

**1.** RA **Registration** (user U- RA interaction)

1. U $\rightarrow$ RA: identification credentials, i.e., id, birth certificates, etc.
2. U: generates $\mathsf{Ms}_{\mathcal{U}}$.
3. U $\leftrightarrow$ RA:
   - validate $\mathsf{Ms}_{\mathcal{U}}$ in a blind way towards RA.
   - issue two types of credentials for U:
     - perm-type of credentials, which can only be demonstrated to an organization once and are meant for users to obtain membership to these organizations. perm-type of credentials may be realized as accountable ecash coins ([50]).
     - cred -type of credentials, whose possession can be proven multiple times in ZK fashion to any service; these credentials are not transferable and may be implemented as plain ecash coins.
4. U: generates a recovery encryption key pair: $(\mathsf{ek}_{\mathsf{U}}, \mathsf{dk}_{\mathsf{U}})$. U then encrypts and stores the serials of the perm-type of credentials and some info (serials if ecash -type) for the cred -type ones to his RA-account. So, U creates

$$\mathsf{RegInfo}_{\mathsf{U}} = \{\mathsf{Ms}_{\mathcal{U}}, \text{ perm-serials, cred -serials, date}\}_{\mathsf{ek}_{\mathsf{U}}}$$

5. U→ RA: $\mathsf{RegInfo}_{\mathsf{U}}$

6. U ↔ RA: (for integrity purposes) exchange proofs of the final form of the encrypted value:

$$\mathsf{Sig}^{h_i}_{\mathsf{RA}}(\mathsf{RegInfo}_{\mathsf{U}}||\text{date}) \text{ and } \mathsf{Sig}^{h_i}_{\mathsf{U}}(\mathsf{RegInfo}_{\mathsf{U}}||\text{date})$$

7. U shares in a shared secret fashion his $\mathsf{Ms}_{\mathcal{U}}$ and $\mathsf{dk}_{\mathsf{U}}$.

**2. Organization Registration**(user U – Organization $\mathsf{org}_i$ interaction) It takes place between an Organization $\mathsf{org}_i$ and a user U for the latter to obtain membership in $\mathsf{org}_i$. Depending on its membership pre-requisites, $\mathsf{org}_i$ may require for a perm or cred type of credentials or even U's identity. (In fact, we assume that perm-credentials have the form of accountable ecash introduced in [50][1]. cred -credentials may have the form of a plain ecash scheme.)

1. U↔ $\mathsf{org}_i$:

   - U spends a credential to $\mathsf{org}_i$. If there is a restriction regarding how many registrations a user should maintain in $\mathsf{org}_i$, perm-credential should be used; otherwise, cred -credential may be used. Let $\mathsf{S}^{\mathsf{perm/cred}}_{\mathsf{U}\rightarrow\mathsf{org}_i}$ be credential's serial. $\mathsf{S}^{\mathsf{perm/cred}}_{\mathsf{U}\rightarrow\mathsf{org}_i}$ is stored in U's IDC and $\mathsf{org}_i$'s database $\mathsf{D}_{\mathsf{org}_i}$.
   - perform GS.Join or AnonCred.PseudoGen procedures or any other organization membership procedure. U obtains secret membership information $\mathsf{MemSec}^{\mathsf{org}_i}_{\mathsf{U}}$ and $\mathsf{org}_i$ the corresponding public information $\mathsf{MemPub}^{\mathsf{org}_i}_{\mathsf{U}}$. From now on, we will assume that U is known to $\mathsf{org}_i$ as $\mathsf{P}_{\mathsf{U}-\mathsf{org}_i}$.

2. U:

   - creates the recovery $\mathsf{org}_i$-related encryption key pair: ($\mathsf{ek}^{\mathsf{org}_i}_{\mathsf{U}}$, $\mathsf{dk}^{\mathsf{org}_i}_{\mathsf{U}}$)

[1]In accountable ecash, if the same ecoin is double spent or two ecoins are spent to the same merchant, they reveal the identity of the coin's owner.

- computes

$$\epsilon^{\mathsf{dk}}_{\mathsf{P_U-org}_i} = \mathsf{encdk_U ek_U dk_U^{org}}_i$$

- generates secret and computes $H_{\mathsf{org}_i}(\mathsf{secret})$, and

$$\epsilon^{\mathsf{sec}}_{\mathsf{P_U-org}_i} = \mathsf{encsecretek_U^{org}}_i,$$

$$\mathrm{sign}\mathsf{P_U-org}_i\mathsf{sec} = \mathsf{Sig}^{H\mathsf{org}_i}_{\mathsf{P_U-org}_i}(\mathsf{secret}),$$

where $H_{\mathsf{org}_i}$ is an $\mathsf{org}_i$-specific hash.

3. U→ $\mathsf{org}_i$: $\mathsf{RecData} = \{\epsilon^{\mathsf{dk}}_{\mathsf{P_U-org}_i}, \epsilon^{\mathsf{sec}}_{\mathsf{P_U-org}_i}, \mathrm{sign}\mathsf{P_U-org}_i\mathsf{sec}\}$. Notice that U is the only one knowing the secret.

4. orgi: creates U's entry in his $\mathsf{D}_{\mathsf{org}_i}$, where he stores the following:

$$\mathsf{Entry_U^{org}}_i = \{\mathsf{S}^{\mathsf{perm/cred}}_{\mathsf{U}\rightarrow\mathsf{org}_i}, \mathsf{MemPub_U^{org}}_i, \mathsf{RecData}\}.$$

**3. Proof of membership in** $\mathsf{org}_i$ This is the case where U proves his $\mathsf{org}_i$-membership in an anonymous and unlinkable way to a verifier. Both, U and verifier should make sure that he has updated $\mathsf{org}_i$'s public information.

**4. Compromise case - Credential Blacklisting** It involves two phases:

1. U-RA interaction

2. U-$\mathsf{org}_i$ interaction

*1.* U– RA*interaction*

1. U→ RA: birth certificate, U-identification, loss declaration.

2. U: recovers his $\mathsf{Ms}_{\mathcal{U}}$and $\mathsf{dk_U}$ in a shared secret way.

3. U$\leftrightarrow$ RA:

   - U recovers his serials from RA through $\mathsf{dk}_{\mathsf{U}}$ and $\mathsf{RegInfo}_{\mathsf{U}}$.
   - issue a new IDC validating a new $\mathsf{Ms}_{\mathcal{U}}$.
   - issue one time use revocation credentials, rev-credentials;

*2.* U– $\mathsf{org}_i$ *interaction* Assuming that somehow (we will deal with it later on) U knows with which serial he registered to which organization, the following series of procedures takes place:

- U contacts org and shows the rev-credential from the RA. (rev-credential is also unlinkable to U's identity, nevertheless only U can demonstrate possession of it). U shows the serial, which he used to register to $\mathsf{org}_i$.
- $\mathsf{org}_i$ looks in its database, and sends $\epsilon^{\mathsf{sec}}_{\mathsf{P}_{\mathsf{U}-\mathsf{org}_i}}$ to U
- U uses the recovered $\mathsf{dk}_{\mathsf{U}}$ to decrypt his secret and demonstrates its knowledge to $\mathsf{org}_i$. A malicious user cannot pass this test.
- $\mathsf{org}_i$ blacklists all the credentials issued for the $\mathsf{MemPub}^{\mathsf{org}}_{\mathsf{U}}$ — probably using a technique similar to [19] — and both parties collaborate to change U's account details using $\mathsf{Ms}_{\mathcal{U}}$'.

After phase 1, the credential-serials are only visible to U. Revealing the serials to the RA would enable the latter with the collaboration of all organizations to trace U's activities. On the other hand, there is no guarantee that U is not lying for his numbers. We need to find a way so that no user is DoS-ed and that traceability of U's past activities in not done. We will discuss the following section.

**User Registrations' Recovery** It includes two phases:

1. *Recovery preparation* stage, which takes place at the $\mathsf{org}_i$–U registration procedure:

- $\mathsf{U} \rightarrow \mathsf{org}_i$: $\epsilon^{\mathsf{ser}}_{\mathsf{U}-\mathsf{org}_i} = \mathsf{encS}^{\mathsf{perm/cred}}_{\mathsf{U}\rightarrow\mathsf{org}_i}\mathsf{ek}_{\mathsf{U}}\mathsf{H}_{\mathsf{serial}}$, where $\mathsf{H}_{\mathsf{serial}}$ is a serial-number specific hash.
- $\mathsf{org}_i$:
  - computes $\epsilon^{\mathsf{org}}_{\mathsf{org}_i} = \mathsf{encorg}_i\mathrm{F}(\mathsf{S}^{\mathsf{perm/cred}}_{\mathsf{U}\rightarrow\mathsf{org}_i})$.
  - logins anonymously but authoritatively in RA.
  - uploads
$$\epsilon^{\mathsf{ser}}_{\mathsf{U}-\mathsf{org}_i}, \epsilon^{\mathsf{org}}_{\mathsf{org}_i}.$$

2. *User Bloom filtering* phase, which takes place after the IDC loss declaration phase. In particular,
   - U logins anonymously to RA.
   - U shows rev-credential.
   - U computes $[\epsilon^{\mathsf{ser}}_{\mathsf{P}_{\mathsf{U}-\mathsf{org}_i}}$ and searches through bloom filters the uploaded serials.
   - U decrypts the corresponding $\epsilon^{\mathsf{org}}_{\mathsf{org}_i}$ to get the name of the $\mathsf{org}_i$.
   - $\mathsf{U} \leftrightarrow \mathsf{org}_i$: U validation phase, through RecData.

# Chapter 8

# Private, Distributed, and Searchable Content Providers

Although there is existing work on anonymous content providing, existing systems do not provide as comprehensive protection as might be desired. Systems like FreeNet and FreeHaven allow users to publish documents without revealing their identities, however they do not hide the identities of those who search for and retrieve their content. Although the receiver can protect their own identities by using other means of anonymous communication as a precursor to contacting the FreeNet or FreeHaven network, the queries themselves are still not protected. This can still be a leakage of sensitive information, even if the identity of the querier is protected. For example, if someone had an innovative idea and wanted to search for patents on related concepts, they might be concerned that the query terms themselves would leak his idea to other parties.

Let's say that a number of support groups for people with various sensitive medical conditions wanted to be able to share literature about an ongoing series of current developments relating to their specific issues. This is the type of application that existing anonymous document sharing systems were designed for. However, it may be the case that simply being able to seek specific documents anonymously is not sufficient; since the articles and events are ongoing and specialized, users would need to be able to privately search for articles of interest, and those queries themselves

would be considered sensitive, since the types of things a person looks for form a profile about them and which conditions they may have. Thus we discuss a system that addresses this scenario, where both the identities of providers and the identities of the queriers are anonymous, and both the content and the queries are private. The system thus needs to be able to scale to large numbers of users, even if the number of users interested in specific articles was limited. Furthermore, as these are sensitive topics and established subcommunities, they may need to contend with undesirables causing trouble or antagonizing legitimate users. Thus a good revokable anonymous credential system would be needed to regulate these problems.

We will aim to provide a more comprehensive system that protects the identities of all parties involved, and also hides the content of the queries from the data providers. To do this, we will extend our earlier work in Chapter 3 on private database search. Private database search already encapsulates the query protection we aim to provide: the data owner does not see the types of queries that it is processing, and the querier does not learn anything about the database except the results of their queries. To achieve our goals, we merely need to extend this system to handle multiple parties (on both ends: queriers and data owners), and to protect the identities of those parties.

To do so, we will instrument the SADS system, using our anonymous publish-subscribe systems to route messages between multiple queriers and multiple servers anonymously. The SADS system at its core meets our requirements of protecting query and data privacy, and it presents no complications with scaling up the number of users or servers. However, it does not by default protect the identities of participants when doing so.

To accomplish this, anonymous publish-subscribe routing can fill this role perfectly. To do so, we invert the concepts of publishing and subscribing. Instead of having users subscribe to topics that interest them, we will instead have them publish queries for terms they want to search for. Conversely, data owners will have ongoing subscriptions for all terms they are able to meet. Thus, queries which they may be able to satisfy will automatically be routed to them. We will use this paradigm as a sort of 'handshake' to connect querier to data owner. The querier will pair this pub-

lished handshake with a Hidden Service (as provided by anonymous communication systems like TOR [67]). Once its handshake finds suitable data owners, the remainder of the SADS protocol can be carried out via the anonymous circuit established through the hidden circuit.

## 8.1 System Architecture

Our solution combines the SADS private search system described in Chapter 4 with the Anonymous Publish-Subscribe spanning tree routing system described in Chapter 5. It thus operates with the following entities:

- *Querier:* There will be many queriers in the system. They will broadcast queries, and expect to receive in return handles corresponding to matching documents in the system. These handles correspond to a data owner and record index for that owner's database, however may not be used to identify that owner.

- *Owner:* There will be many owners in the system. Each contains a database of records they wish to make available, along with search terms corresponding to those records. They will register these terms in the system, which will then deliver them queries that match those terms. They will answer these queries via anonymous communication with the queriers.

- *Router:* These nodes form the spanning tree relay network used by the Publish-Subscribe system described in Chapter 5. As with in that system, they do not require a great amount of trust, so long as compromised nodes comprise a small subset of the system, since the queries will be routed through a large number of them.

- *Query-Proxy:* These nodes are analogous to the $TP$ nodes used by the SADS system in Chapter 4. They will be used during anonymous query resolution between Querier and Owner, and it is assumed that each party trusts them not

to maliciously collaborate with the other. These should thus be chosen publicly from a trusted set of entities.

These entities will then support the following operations:

- `Register(Q)`*:* User $Q$ wishes to be able to issue queries. To do so, he generates a re-routable encryption key for his own use.

- `Provide(O, d)`*:* Where $d$ is a set of $n$ records $r_1, ..., r_n$, data owner $O$ makes the full set of records available for search in the system. He will do so by executing the server preprocessing step of the SADS system, generating a Re-routable Encryption key. He then computes a union of all terms across the records in $d$ and executes a subscribe on the publish-subscribe network of every term in $d$.

- `Locate(Q, t)`*:* Where $Q$ is a querier interested in querying for term $t$. $Q$ will then set up a hidden service with identifier $h_Q$. He will then publish a message with $h_q$ on topic $t$. Each owner $O$ that has provided a record containing $t$ will receive this message. They will then anonymously contact $Q$ through his hidden service, choose a Query-Proxy $TP$ from the list of publicly trusted entities, and run the `RatioKeyGeneration` protocol of SADS with $Q$ and $TP$. $Q$ will now have a list of $m$ data owners $O_1, ...O_m$ who have records of interest to him.

- `Query(Q, O, t)`*:* Given a successful result of the `Locate` call, $Q$ now has complete parameters for the SADS protocol set with each resulting data owner. He may then issue a SADS query and retrieve appropriate documents from one or more of his choice.

The basic approach is to use Publish-Subscribe to locate content providers that have records of interest to a querier. To do so, we simply reverse the intuitions of publishing and subscribing. In normal applications, users would "subscribe" to topics they are interested in, and other users would "publish" messages on topics they have relevant content about. However, to create a persistent search system, we can instead have content providers subscribe to topics they *have* relevant content for, allowing

them to continually monitor the system for relevant queries. Queriers instead publish their *interest* in various topics, knowing their interest will reach all providers with content of interest. By using an anonymous publish-subscribe system, we can thus establish connections between queriers and providers with relevant content while protecting the identities of both. And by using hidden services, we can then allow them to create circuits for communication while continuing to remain anonymous towards each other. The SADS protocol can be run over such an anonymous communication network, so long as we have global trusted parties to participate. The trust requirements for these parties does not change from what it was in SADS: they should not collaborate with either the querier or the provider.

Considering our hypothetical system providing support groups for medical conditions, the users themselves would be acting both as queriers and data owners, as they exchanged articles and support discussion. Since we do not need to trust all routers, only a subset of them, the users themselves could also volunteer to act as routers, much in the same way TOR networks operate on an assumption that the majority of participating nodes can be trusted. The query-proxies involve a higher assumption of trust, and would be best provided by a commonly trusted organization, such as a medical or therapy organization.

One downside to this approach is that we cannot support more complex queries, only single term. Even though both the underlying SADS system and the underlying publish-subscribe system support boolean and ranged queries, because of our inversion of publish-subscribe we cannot make full use of this functionality. There isn't a simple way to publish a query such that a boolean logic on subscriptions would receive it. This remains as a direction of future work.

Our solution thus aggregates providing of content into a single `Provide` call. Further provision in the future can be handled by setting up additional providers. Querying and retrieval, on the other hand is separated into a `Locate` call and many `Query` calls across multiple providers.

The system and query path is laid out as in Fig. 8-1.

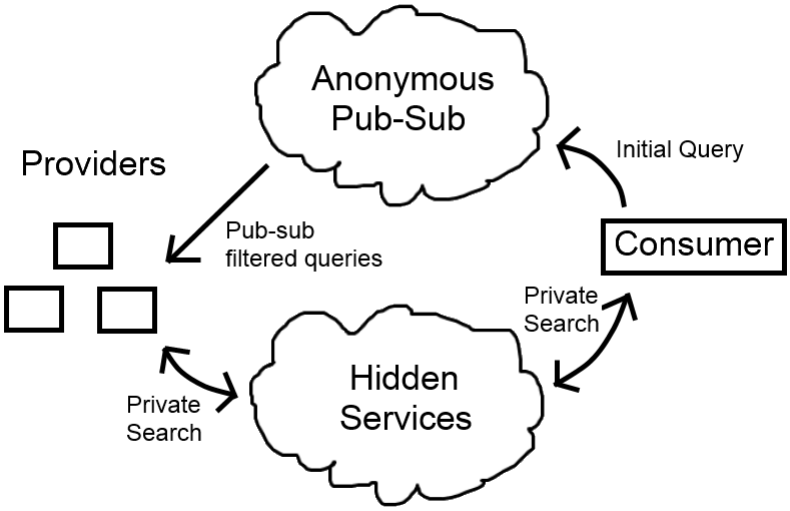



Figure 8-1: Anonymous and private content provider system

## 8.2 Security Analysis

We now discuss the anonymity and privacy provided by such a system. Our system is constructed from an underlying set of systems, and thus its security incorporates assumptions depending on what instantiations were used for those blocks. We have cited or invented systems demonstrating that each of these blocks can be built under a reasonable set of security assumptions throughout this work, and the choice of what subsystem to use for each of these requirements decides under what assumptions our system will remain secure.

**Claim 26.** *Assuming the security of the underlying anonymous communication system, anonymous publish-subscribe system, and SADS system, our system achieves complete provider anonymity.*

The `Provide` protocol involves only a localized setup of the SADS system and subscription to several terms under an anonymous publish-subscribe system. Since the

SADS setup is local, it is impossible for it to reveal the identity of the provider. An underlying anonymous publish-subscribe system guarantees subscriber anonymity, so assuming the security of the underlying system, the subscription phase will not reveal the identity of the provider either. The `Locate` protocol involves an anonymous publication by the querier, which is also guaranteed not to reveal the receiving subscribers. The provider then contacts the querier through a hidden service, which the underlying anonymous communication system guarantees will not reveal the identity of the initiator of the circuit. Finally, the `Query` protocol is a straightforward execution of SADS querying, which also does not reveal identity.

**Claim 27.** *Assuming the security of the underlying anonymous communication system, anonymous publish-subscribe system, and SADS system, and that the trusted party does not collaborate with any providers, our system achieves complete querier anonymity.*

As the querier does not participate in `Provide` protocol, this phase can not reveal their identity. The `Locate` protocol involves an anonymous publication by the querier, which is also guaranteed not to reveal his identity. The provider then contacts the querier through a hidden service, which the underlying anonymous communication system guarantees will not reveal the identity of the owner of the hidden service. Finally, the `Query` protocol is a straightforward execution of SADS querying, which also does not reveal identity so long as the trusted party behaves honestly.

**Claim 28.** *Assuming the security of the underlying anonymous communication system, anonymous spanning tree publish-subscribe system, and SADS system, and that the trusted party does not collaborate with any providers, our system achieves query privacy.*

A query goes through two phases: `Locate` and `Query`. As the second is identical to a standard SADS query, it provides privacy protection equal to SADS. The Locate phase will deliver to the data owner a publication of the querier's hidden service identifier $H_q$. This publication contains the message, which is a uniquely chosen hidden service ID that has no relation to the query content, and a topic. The topic

is stored as a set of bloom filter indices corresponding to the encrypted query term, and thus equivalent to what the owner would see in a SADS query. Our system thus protects query privacy as securely as SADS.

**Claim 29.** *Assuming the security of the underlying SADS system, our system achieves content privacy.*

A query goes through two phases: `Locate` and `Query`. As the second is identical to a standard SADS query, it provides privacy protection equal to SADS. The Locate phase does not deliver any content to the querier, the only information learned is whether or not a database has matching records, which is within the SADS definition of content privacy. Our system thus protects content privacy as securely as SADS.

## 8.3 Performance

We implemented our systems to test its performance. Unfortunately, to our knowledge, there exist no other distributed private search systems for comparison. We instead merely present performance numbers to demonstrate that it is practical for real-time use. We expect this to be used in scenarios where there are many users distributed globally. However, although there may be many total users in the system, specific topics are bounded to smaller sub-communities (within dozens or hundreds), and the number of users with results relevant to a specific query is a specific target subset of that group.

To obtain a large number of nodes for scalability testing, we used the PlanetLab network [27]. Each participating node has at minimum 4x 2.4Ghz Intel cores, 4 GByte ram, and 500GB disk space. Nodes are distributed around the globe to provide a simulation of internet traffic. For our experiments, we used nodes with varying geographic locations contained within the US.

Figure 8-2 shows time to issue a query, plotted against the number of providers with matching content. The system was instantiated with multiple providers, each of which had a database of 10000 records and an average of 100 matching records

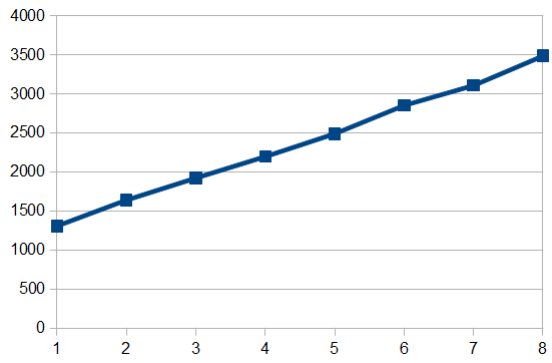



Figure 8-2: Query time(ms) against number of providers with results

from that set. As we can see, there is an expected linear growth with the number of providers, since the querier must independently contact each of them. At this time, we have not implemented parallel execution of these queries, so it is likely that performance can be further improved. We can also see that search times for small numbers of providers, which is also indicative of time to receive first results from a long ongoing search, are reasonable for real-time search, being limited to a few seconds. This does not include document transfer time.

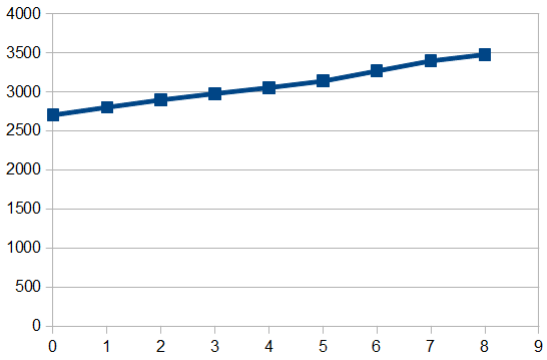



Figure 8-3: Boolean query time(ms) against false positive providers

Figure 8-3 shows time to run 2-term AND queries. While the system does not support boolean queries in the `Locate` phase, we can simply use a single term, and then run normal SADS boolean queries on the resulting list of providers. Since it is an AND query, this will have false positives, but no false negatives. We conversely

could not use this method to handle OR queries, since a single term would not find all matching providers. The system was instantiated with 5 matching providers, and plotted against the number of "extra" providers which have the single term but do not meet the full boolean query. As we can see, the performance hit is very minor, which is as expected since SADS handles zero result queries very efficiently.

# Chapter 9

# Conclusion

Systems that enable private and anonymous online interactions are an area of growing interest. Whereas there has been a large amount of work on anonymous communication and distribution of information, much of it either does not address the issue of how to establish relationships for communication and distribution, and/or comes with prohibitively expensive performance costs.

In this work, we have addressed these issues with new systems that make fewer assumptions about prior relationships between the participants, and using efficient data structures and algorithms to create systems that are practical for real-time use. Our work in private database search allows users to privately and anonymously determine if information they are interested in exists and how to get it. Furthermore, our work with anonymous publish-subscribe systems enables communication under a different paradigm that does not assume users know who they wish to communicate with. This is a natural improvement for anonymous communication. And we showed how anonymous credential systems can be created in an efficient and real-world deployable manner. Such systems are of value to other anonymous systems that need to provide different access levels and policies for multiple anonymous users. By combining these systems, we created systems that can create search between multiple anonymous parties.

Furthermore, by using efficient building blocks, we created systems that provide these functions with performance that is reasonable for real-time usage.

# Bibliography


[1] The porter stemming algorithm. `http://tartarus.org/martin/PorterStemmer/`.

[2] 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.

[3] 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.

[4] E. Anceaume, A.K. Datta, M. Gradinariu, and G. Simon. Publish/subscribe scheme for mobile networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, 2002.

[5] 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.

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

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

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

[9] 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.

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

[11] 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.

[12] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 535–554. Springer, February 2007.

[13] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting bloom filters. In *14th Annual European Symposium on Algorithms, LNCS 4168*, pages 684–695, 2006.

[14] S.A. Brands. Rethinking public key infrastructures and digital certificates: building in privacy. In *Cambridge, Mass. : MIT Press*, 2000.

[15] J. Broberg, R. Buyya, and Z. Tari. Metacdn: Harnessing storage clouds for high performance content delivery. In *Proceedings of ACM Conference on Data and Application Security and Privacy*, 2012.

[16] J. Camenisch and E. Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *CCS02: Proceedings of the 9th ACM conference on Computer and communications security*, 2002.

[17] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology - EUROCRYPT 2001, volume 2045 of Lecture Notes in Computer Science*, 2001.

[18] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, 2002.

[19] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *International Conference on Security in Communication Networks – SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Verlag, 2002.

[20] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373. Springer, August 2013.

[21] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM*, 1982.

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

[23] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, March 2012.

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

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

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

[27] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33:3–12, July 2003.

[28] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*, pages 46–66. Springer-Verlag New York, Inc., 2001.

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

[30] 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.

[31] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Cryptography ePrint archive, report 2006/210, http://eprint.iacr.org/2006/210*, June 2006.

[32] Y. Koglin D. Yao, E. Bertino, and R. Tamassia. Decentralized authorization and data security in web content delivery. In *Proceedings of the 22nd ACM Symposium on Applied Computing*, 2007.

[33] A.K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/subscribe in p2p networks. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, 2003.

[34] Emiliano De Cristofaro, Yanbin Lu, and Gene Tsudik. Efficient techniques for privacy-preserving sharing of sensitive information. In *TRUST'11*, pages 239–253, 2011.

[35] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, volume 2009 of *Lecture Notes in Computer Science*, pages 67–95. Springer, 2000.

[36] Brad Fitzpatrick and Brett Slatkin. Pubsubhubbub.

[37] O. Foundation and I.C.F. Openid foundation.

[38] T.E. Foundation. Higgins: Open source identity framework.

[39] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[40] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, August 2012.

[41] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *In Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, pages 803–815, 2005.

[42] Nick Mathewson George Danezis, Roger Dingledine. Mixminion: Design of a type iii anonymous remailer protocol. In *Security and Privacy*, 2003.

[43] 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.

[44] 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.

[45] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[46] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43:431–473, 1996.

[47] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM CCS 12*, pages 513–524, 2012.

[48] P.R. Group. Prime project.

[49] 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.

[50] Susan Hohenberger Jan Camenisch and Anna Lysyanskaya. Balancing accountability and privacy using e-cash (extended abstract). In *Security and Cryptography for Networks*, 2006.

[51] Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Outsourced symmetric private information retrieval. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 875–888. ACM Press, November 2013.

[52] Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 136–155. Springer, December 2005.

[53] Vladimir Kolesnikov and Ranjit Kumaresan. Improved secure two-party computation via information-theoretic garbled circuits. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 205–221. Springer, September 2012.

[54] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, July 2008.

[55] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In Gene Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 83–97. Springer, January 2008.

[56] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[57] Steve Lu and Rafail Ostrovsky. Distributed oblivious ram for secure two-party computation. In *TCC*, pages 377–396, 2013.

[58] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[59] Y. Manabe N. Akagi and T. Okamoto. An efficient anonymous credential system. In *Financial Cryptography and Data Security*, 2008.

[60] 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.

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

[62] Vasilis Pappas, Mariana Raykova, Binh Vo, Steven M. Bellovin, and Tal Malkin. Private search in the real world. In *ACSAC '11*, pages 83–92, 2011.

[63] A. Kapadia P.P. Tsang, M.H. Au and S. W. Smiths. Blacklistable anonymous credentials: blocking misbehaving users without ttps. In *CCS07: Proceedings of the 14th ACM Conference on Computer and communications security*, 2007.

[64] 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.

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

[66] Phillip Rogaway. *The round complexity of secure protocols.* PhD thesis, Massachusetts Institute of Technology, 1991.

[67] Paul Syverson Roger Dingledine, Nick Mathewson. Tor: The second-generation onion router. In *Usenix Security*, 2004.

[68] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy*, pages 350–364. IEEE Computer Society Press, May 2007.

[69] 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.

[70] TIBCO. Tib rendezvous. In *White Paper. TIBCO*, 1999.

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

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

[73] 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.

[74] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen. Towards end-to-end secure content storage and delivery with public cloud. In *Proceedings of ACM Conference on Data and Application Security and Privacy*, 2012.

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

[76] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

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

[78] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[79] Y.N. Li Z. Jiang, H. Luo and H.P. Icard - foundation for a new ubiquitous computing architecture. In *ICC03: Proceedings of the IEEE International Conference on Communications*, 2003.

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