Preserving Access Privacy Over Large Databases

Femi Olumofin
Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
fgolumof@cs.uwaterloo.ca

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
iang@cs.uwaterloo.ca

Abstract

Anonymity systems preserve the identities of users as they access Internet data sources. The security of many such systems, such as Tor, relies on a model where the adversary does not have a global view or control of the network. A different problem is that of preserving access privacy for users accessing a large database over the Internet in a model where the adversary has full control of the database. Private information retrieval (PIR) schemes are designed to prevent an adversary controlling the database from being able to learn any information about the access patterns of users. However, the state-of-the-art PIR schemes have a high computational overhead that makes them expensive for querying large databases.

In this paper, we develop an access privacy technique and system for querying large databases. Our technique explores constraint-based query transformations, offline data classification, and privacy-preserving queries to index structures much smaller than the databases. Our approach enables the querying of a large database by statically specifying or dynamically defining database portions on keys, possibly with high diversity in their range of values, thereby minimizing information leakage about the potential data items of interest to users. In addition, our approach requires minimal user intervention and allows users to specify descriptions of their privacy preferences and delay tolerances along with their input queries to derive transformed queries capable of satisfying the input constraints when executed. We evaluated the system using patent data made available by the United States Patent and Trademark Office through Google Patent; however, the approach has a much wider application and the system developed can be adapted and deployed for use with many user-centric privacy-preserving systems, thereby making access privacy obtainable for today's Internet users.

1 Introduction

Databases in the real world are often large and complex. The challenge of querying such databases in a timely fashion has been studied by the database, data mining and information retrieval communities, but rarely studied in the security and privacy domain. We are interested in the problem of preserving access privacy for users when querying large databases of several hundreds or thousands of gigabytes of data. This is a harder problem than in other domains because the textual contents of queries are themselves protected from the database server. Standard private information retrieval (PIR) schemes [12] solve this problem for smaller databases, but they are just too slow for large databases, making them inappropriate for meeting the expectations of today's Internet-savvy interactive users. This is because existing PIR schemes have an unavoidable computational cost linear in the database size n [5]. We note that even with faster algorithms and increased computing capacities are available, the overall per-query response time will still be slow for sufficiently large n.

In this paper, we justify the importance of providing access privacy over a reasonable subset of a large database, when absolute access privacy over the entire database is infeasible, and/or when users' privacy

preferences allow for some flexibility, but they cannot tolerate long delays in obtaining responses to their queries. We build upon and extend some earlier work [10, 30, 40] that supports performing PIR over a database subset as a useful practice. Our goal is to simplify how to query a subset of a database while minimizing the privacy lost and minimizing query response time. We provide a generalized approach for querying large databases. Our algorithm works in two stages: offline preprocessing and online query. During offline preprocessing, the database administrator classifies the data in the large database into smaller chunks (or *buckets*) and generates a number of indices to support keyword- and full-text-based search. The user specifies her online query either as array indices, one or more keywords, or in SQL. She also specifies her delay tolerances and privacy preferences to the algorithm or system. Delay tolerance indicates how long she is willing to wait (in seconds) for her query results, while her privacy preferences spell out constraints on the amount of information to leak about her query to the database. Our system derives and executes an alternative query that satisfies the input constraints and returns a correct response to the user. The query execution time is satisfactory to the user because her query runs against a subset of the database, unlike queries that run over the entire data set, which may be too slow.

Unlike previous work, our approach explores new ideas about classifying large databases into buckets, searching and retrieving data from large databases over index structures that are much smaller than the database, ensuring access privacy over the buckets while minimizing risks of correlated queries, and the development of a practical Patent Query System that demonstrates a privacy-friendly alternative to Google Patent Search.

Outline of the paper. We motivate the research in Section 2 and Section 3 reviews some related work. In Section 4, we provide the model we are proposing for preserving access privacy over large databases. In Section 5, we provide details of our approach for classifying, indexing, and querying a large database in such a way that the privacy and performance preferences of the user are considered. Section 6 describes our implementation and evaluation of the approach using patent data from the US Patent and Trademark Office (USPTO). Section 7 concludes the paper.

2 Motivation

We motivate the need for providing access privacy over large databases and highlight some drawbacks of the current state-of-the-art approach to the challenge of providing access privacy.

2.1 High Computational Costs for Large Databases

Existing PIR schemes are slow when used to query large databases. In a recent evaluation of some PIR implementations on current hardware [29], we found the round-trip response time of the fastest single-server PIR scheme [1] is approximately 3 minutes—the time to retrieve a meaningful piece of data (e.g., 4 KB) from a 1 GB database on a typical home user Internet connection of 9 Mbps download and 2 Mbps upload [31]. We found the fastest multi-server PIR schemes [12, 17] to have a better response time of between 0.5 s and 1.3 s. On the other hand, it takes approximately 15 minutes for a typical home user to trivially download the entire database. There are clear advantages from using PIR (even in comparison to trivial download) over databases of this size. However, for a larger database of 16 GB, the respective response times for single-server and multi-server PIR are 16 minutes and 7–21 s respectively, compared to 4 hours for trivial download. Again, there is an advantage in using PIR over the trivial solution; however, the response time will not be as tolerable for users as with the smaller 1 GB database. If we project the response times by assuming a 1 TB database that is not uncommon in the real world, the numbers look ridiculously expensive: 17 hours for single-server PIR, or 34–53 minutes for multi-server PIR. Although those numbers are far superior to the 11-day trivial download time, these times are still too slow for usable systems. We

note that it is unlikely for the response time to become affordable in the future because the multiplicative factor resulting from the scale of a large database will overwhelm almost any per-item computational cost, no matter how low or insignificant. Querying a large database by retrieving every block is not practical for interactive queries in the traditional information retrieval context, and so will it most probably remain in the PIR context.

2.2 Most Internet Users Have Low Network Bandwidth

The original motivation for studying PIR recognizes network bandwidth usage as the most expensive resources that should be minimized. As a result, most recent PIR schemes have near-optimal communication complexity. After almost two decades of work on PIR, most Internet users are still low-bandwidth users. The network bandwidth of the average home Internet user is improving at a rather slow pace [31]. The most valid source of data to show this trend is the recently available Internet speed database [31]. The average download rates of users in Canada and the US for 2008, 2009, and January 1 to May 30 of 2010 are 6, 7.79, and 9.23 Mbps. The average upload rates for those respective periods are 1.07, 1.69, and 1.94 Mbps. We can see that the growth rate is rather slow. In early November 2010, the average Internet download and upload rates for the G8 Nations was 9.55 Mbps and 2.32 Mbps respectively. This latter result was based on a survey of over 400,000 Internet users for a six-month period ending November 2, 2010. These averages more correctly reflect the improvements in Internet bandwidth than those predicted by Nielsen's Law [27]. Nielsen's Law specifically addresses the type of users described as normal "high-end" who can afford high-bandwidth Internet connections [27]. This class of users should be contrasted from "low-end" users [27] that the above bandwidth averages from the Internet speed data [31] include. Since most Internet users are at the low-end category, minimizing the communication between the user and the database is necessary for realizing access privacy.

2.3 Physical Limits of Hard Disks and Memory

Large databases cannot fit in available RAM in most situations. Consequently, the processing rate of typical PIR schemes will be disk bounded since every data item must first be read and then processed. Disk read latency also applies to the cases of trivial download and traditional non-private information retrieval. The latter avoids reading the entire database for each query being processed, since it will be impractical to do so in order to answer queries interactively.

2.4 "All-or-nothing" Access Privacy is Insufficient

PIR schemes are well suited for solving the access privacy problem because they provide a strong privacy guarantee against a powerful adversary (i.e., a database administrator or owner). However, having a new perspective on how to leverage PIR to address the access privacy problem for large databases is necessary. Today's online users have *reasonable* (but not *absolute*) means of protecting information about their identity (i.e., anonymity) using onion routers and mix networks. A system like Tor is quite successful in helping dissidents, citizens of oppressive regimes, military personnel, and many Internet users stay anonymous. Whereas anonymity systems in wide use today do not offer an absolute guarantee for privacy, they are nonetheless successfully deployed and are used by a wide population of users. On the other hand, the PIR community has largely continued to approach the case of protecting the content of the user's query with the notion of absolute or "all-or-nothing" privacy, with respect to PIR-processing every data item in the database. Since we are yet to find a better alternative, we reexamined the currently held rigid notion of privacy with PIR and substituted a somewhat more relaxed notion that allows users to enjoy "graduated" privacy instead. Such a change in perspective will help realize the fielding of user-centric systems with acceptable performance and access privacy protection.

We note that Asonov et al. [3] relaxed the strong privacy guarantee of PIR that reveals *no* information about a query to a weaker notion that reveals *not much* information. They referred to schemes exhibiting such property as repudiative information retrieval (RIR). The information revealed by an RIR scheme is

insufficient for anyone, even with the cooperation of the database, to say whether or not a particular item of the n items in a database is the one retrieved by a query. The work was motivated by work in secure-coprocessor-based PIR on reducing preprocessing complexity from $O(n \log n)$ to $O(\sqrt{n})$. An open problem from the work is how to achieve RIR without using a secure coprocessor.

3 Related Work

PIR has to date been the primary approach to the problem of preserving access privacy for Internet users. For an n-bit database X that is organized into r b-bit blocks, Beimel et al. [4, 5] shows that standard PIR schemes cannot avoid a computation cost that is *linear* in the database size because each query for block X_i must necessarily process all database blocks X_j, $j \in [r]$. They proposed a preprocessing model of PIR that computes and stores some *extra bits* of information, which is polynomial in the number of bits n of the database. Several hardware-assisted PIR schemes [2, 37, 41] rely on the preprocessing model. With the exception of [39], all secure coprocessor-based PIR schemes require periodic database reshuffles (i.e., repeats of the preprocessing stage). The reshuffling cost of [41], for example, is $O(\log^4(n))$, but when amortized, it is $O(\log^3(n))$ per query. Nonetheless, the paper [41] shows how to achieve improvements in the communication and computational complexity bounds of hardware-assisted PIR to $O(\log^2 n)$ per query, provided that a small amount of temporary storage, on the order of $O(\sqrt{n})$, is available on the secure coprocessor.

An initial suggestion to base PIR on a subset of a database as a means of reducing the high computational overhead in some application areas was made by Chor et al. [13]. A similar suggestion to improve the performance of PIR-based techniques for location-based services by basing PIR on a restricted subset of the data space was left as an open problem by Ghinita [16]. Olumofin et al. [30] addressed the open problem identified by Ghinita in the specific context of location-based services. In contrast to these prior works, this paper uniquely addresses the problem of preserving access privacy over a large database in a generic way and provides a concrete system for querying a large database.

Wang et al. [40] proposed a bounding-box PIR (bbPIR) which combines the concept of k-anonymity with the single-server computational PIR scheme by Kushilevitz and Ostrovsky [25] to allow users to define a "bounding box" portion of the database matrix and basing PIR on that smaller portion. Their extension also allows the user to specify both the privacy and the service charge budget for PIR queries. The bbPIR work overlaps our work in some areas, but there are several differences. First, bbPIR defines rectangular bounding boxes within a PIR matrix at runtime, whereas our work considers both runtime and offline approaches to defining database portions. The way we define database portions at runtime also differs from that of bbPIR; we consider the sensitive constants in the input query, statistical information on the distribution of the data, and past query disclosures, which allow for logical or non-contiguous database portions. This is unlike bbPIR, which is agnostic to logical data contents. Second, the bbPIR charge-budget model is based on the number of blocks retrieved (typically the square root of the bounding box area). We model the user's budget in terms of her delay tolerances, which has more generic interpretations (e.g., response time, number of blocks, computations). Third, bbPIR is restricted specifically to one particular PIR scheme [25], whereas our approach is generic and can use any underlying PIR scheme. Fourth, bbPIR is limited to the retrieval of numeric data by address or by key using a histogram, whereas we support retrieval using any of three data access models—by index, keyword, or SQL. Our approach also involves an explicit intermediate stage for transforming input query q to an equivalent privacy-preserving query Q and requires minimal user intervention.

Howe and Nissenbaum [22] developed a browser extension known as TrackMeNot which tries to solve the problem of preserving access privacy during web searches. TrackMeNot tries to hide a user's request to a search engine in a cloud of dummy queries that are made at specified time intervals. The privacy guarantee is not as strong as our technique that is based on PIR because the server is still able to observe the

content of every query made. TrackMeNot utilizes a significant amount of constant bandwidth for generating decoy queries, which can potentially slow down the network and the search engine or database server when deployed on a large scale. In addition, the adversary might be able to distinguish actual queries from dummy queries by considering their submission timings or other metainformation.

Domingo-Ferrer et al. [15] considered a scenario where the holders of a database (e.g., a search engine corpus) are uncooperative in allowing the user to obtain access privacy. In other words, the holders are unwilling to support any PIR protocol, and yet the user desires reasonable access privacy over the large data set. They proposed $h(k)$-PIR which embellishes the user's query keywords with some other $k-1$ bogus keywords. After the server returns a response, the client filters the response to remove items related to the bogus keywords, and finally displays the result to the user. They defined an access privacy scheme as satisfying $h(k)$-PIR if the adversary can only view the user's query as a random variable Q_0 satisfying $H(Q_0) \geq h(k)$, where $h(\cdot)$ is a function, k is a non-negative integer, and $H(Q_0)$ is the Shannon entropy of Q_0. The security of the scheme relies on using a large set of k bogus keywords with identical relative frequencies as the query keywords. However, the accuracy of the query result degenerates with higher values of k, which is their point of tradeoff, unlike our approach where the tradeoff is between privacy and computational efficiency. In addition, their approach relies on the availability of a public thesaurus of keywords and their relative frequencies. It is somewhat misleading for the label of PIR to be used for this approach as its privacy guarantee is not as strong as standard PIR; the adversary can still observe the content of every query made by users.

4 Model

We describe our model of preserving access privacy for users of large databases. We will begin by providing an intuition for the model. A naive approach for searching and retrieving data from a large database is to perform a linear scan through every data item in the database until the desired item is located. A more efficient approach first builds an index based on a particular attribute or a combination of attributes (i.e., a key) and performs the search over this index to locate, and then to retrieve the desired item. This latter approach saves time by minimizing the number of data blocks that are retrieved. Searching an index over a large database minimizes the amount of I/O needed to efficiently retrieve items from the database because indices are often structured as a tree or a hash table.

In the case of private information retrieval over a large database, the best privacy is obtained by reading and computing on every data block in the database. A significant amount of time is required both to perform a linear scan that reads every database block, and to compute on every block. The block-reading time is avoided in standard information retrieval and yet the problem of preserving access privacy with PIR has mostly been approached in the light of reading and computing on every block of data. The incompatibility of such an approach with the size of real-world databases and the need of today's interactive users led to our model.

Our model performs scanning and computation first on indices defined over the data set, and then on a suitably defined and privacy-maximizing subset of the database. We explore how to define subsets that are privacy-maximizing instead of the naive approach of using contiguous portions or randomly chosen subsets of the database. This approach systematically avoids querying excessively large databases for impatient users, while still allowing very patient users to be able to query a much larger subset or even the entire database in order to increase the privacy afforded to them.

By systematically building indices over a large database, and defining privacy-maximizing portions of the database, our model will enable development of techniques and systems that would allow application clients needing access privacy to perform tradeoffs between privacy and computational efficiency with *minimal user intervention*. This model generalizes and specifically builds upon previous work, as outlined

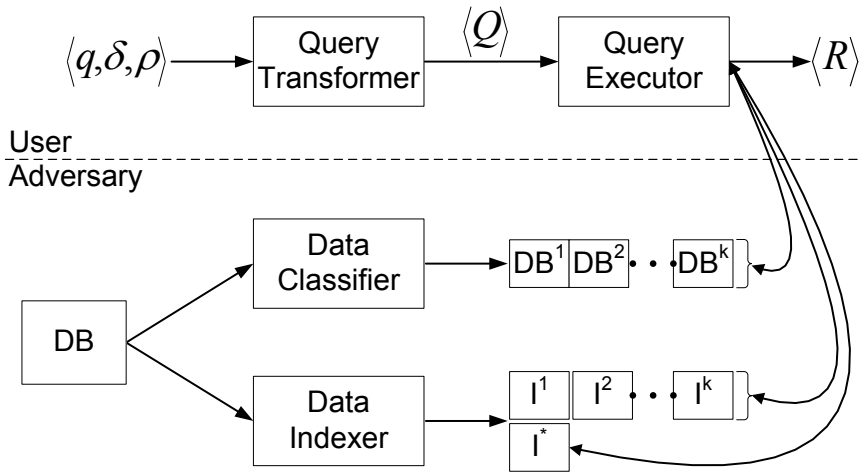

Figure 1: A model for preserving access privacy over large databases.

above.

4.1 Model Components

The model has algorithms and actors as components. The algorithms are the query transformer, query executor, data classifier, data indexer, and a PIR scheme. The actors are the users and the database administrator (the adversary). We illustrate the model components in Figure 1.

In the nomenclature from [14], our adversary is global, internal, and semi-active. Nevertheless, the adversary can inject his or her own queries in order to determine the content of queries from users. The adversary is global and internal because of the full control he or she has over the data being queried.

Before users can run their queries, the database administrator uses a classifier algorithm [7] to classify the data in the database into subsets or *buckets*. This process divides the large database into buckets using one or more attributes of the data as the key. The classification also returns an easily computable function that outputs a bucket number for an item of data when a key is passed as input. For example, we can compute a non-cryptographic hash of a key to determine the bucket number. We note that a tradeoff has to be made between an easily computable function that randomly partitions the data set and a more costly function that divides the data set into subsets that can provide the most access privacy when queried. There is information leakage when a client successively performs two or more queries relating to the user's specific interest over a single or multiple subsets of a large database. For example, the adversary might be able to easily determine that successive queries over database subset DB^1 indicate the user is interested in any one of these smaller subset of database items. However, such leakages can be minimized by systematically diversifying the distribution of the data by their keys over the subsets, thereby making it difficult for an adversary to correlate a user's interest to the set of queries he or she makes over any single bucket or multiple buckets. We leave for future work the problem of finding a way to classify a large data set into subsets that minimizes information leakage over multi-round search and retrieval. Nevertheless, as we will see later, our approach performs a PIR-based search and retrieval over an index built on the entire data set, and not a subset. It therefore forestalls any chance for inferring key relationships through queries made to indices built over the entire database.

The data classification algorithm does not have to be overly complex. We note that some hierarchical clustering algorithms in the unsupervised learning category [20] can probabilistically determine whether or not a data point or key is classified into some clusters (i.e., buckets) may be sufficient in some situations,

such as when query failures are allowed. However, it is unlikely for data in large publicly held databases, where access privacy is important, to require such complex clustering approaches that are typically studied in the data mining literature. Preprocessing with a classifier algorithm is not required if the user can suitably define data subsets dynamically at runtime by leaking some general information to the database for defining the subset. However, there might be performance penalties.

However, static buckets containing pre-classified data offer a number of benefits. First, there is a performance benefit because they are defined offline. Second, the user does not need to leak any information to the database at query time other than the bucket number. Third, bucket contents can be diversified as much as possible to make user queries more private. By this, we mean that each bucket can be striped with information from all major categories in a hierarchy of the data thereby minimizing the information gained by the adversary about the major category or subject of the user's query.

In addition, the database administrator builds two types of indices over the entire data set. The first type is a master index to be used for searches over every keyword in the entire database or selected attributes of the data. For example, we might want to build an index over the keywords in the title, abstract, claims, a combination of these, or the entire attributes of a data set of patent documents. This index also contains compressed summaries of the database items which will be displayed to the user during a search. We denote the master index by I^* in Figure 1. The second type of index is built over keys on specific attributes of the data and supports retrievals of full data items over that particular attribute. For example, indices might be built over the patent number, application date, issued date, title, inventor, or assignee fields of a patent document. We denote these second type of indices by I^1,I^2,$\cdots$,I^k in Figure 1.

Once the offline processing is completed, users can start querying the large database. The user request to the database is a three-tuple (q, δ, ρ), where q is the input query in the natural form; i.e., a database item index [12], a keyword [11], or a language-based [28] (e.g., SQL) query. An index-based query is the standard way to retrieve data from a PIR database. The user simply requests the data at index i (i.e., $X(i) =?$). Retrieval with keywords (i.e., $X(w^*) =?$, where w^* is a single or multiple keywords) is much more elaborate not only because the user is not required to know the address of the data item containing the keyword(s), but also that it enables the combination of two or more keywords using boolean operators (OR, AND, and NOT). Language-based PIR queries are the most advanced because they are much more expressive. Such queries are typically processed in two stages [28]. In the first stage, the input query is first desensitized into an alternative subquery and then sent to the PIR server. The PIR server executes the subquery and generates an appropriate index (i.e., B^+-trees, hash tables) over the subquery result. In the second stage, PIR queries are made to the PIR server over the indices generated from the previous stage and the results returned are filtered to obtain the final result for the query. δ is the cost tolerance of the user in seconds. Conversion to other dimensions of cost, such as dollars, is straightforward given the recent availability of pricing models [9] for cloud computing. ρ represents the privacy preferences of the user. For example, a user that knows that she is prone to privacy violation due to easy availability of additional information about her interests might opt for an option that does not disclose any partial or general information about her interests. The definitions of δ and ρ for a particular user only need to be completed once.

The user's request is input to the query transformer algorithm which will produce an equivalent query that can satisfy δ and ρ.

Definition 1. An algorithm Γ accepts query q, processing cost tolerance δ, and privacy preferences as inputs, and outputs either $\perp$ or a query Q that can be executed by some privacy-preserving system. Then Γ is a valid **query transformer** iff for all non-$\perp$ outputs, $result(execute(q)) = result(execute(Q))$, $cost(\Gamma) + cost(execute(Q)) + \Delta \leq \delta$, and $execute(Q)$ conforms to the given privacy preferences.

The functions $result(\cdot)$, $execute(\cdot)$, and $cost(\cdot)$ respectively denote the result of a query, the execution of a query, and the cost (time, computations and communications) required for executing an algorithm or a

query. Δ encompasses the variability in the costs of query transformation and query execution. It is often the case that $cost(execute(q)) \leq \delta$. The goal is to produce valid query transformers that output $\perp$ as little as possible.

The query executor runs the transformed query using a PIR client algorithm, performs additional processing on the data items retrieved with PIR, and returns the result to the client. A minimum of two rounds of PIR queries is required to retrieve information from the database. The first round of PIR queries is over one of the indices. For example, in a keyword search, the client will first determine the set of addresses (array indices) for the entries of an index matching the search terms. For a hash table index, the address can be computed from the perfect hash functions [6] for that index, whereas an oblivious traversal over a tree-based index is required to help locate the address in other instances. Oblivious traversal describes how a client can use PIR to search a tree-based index structure located at the server, without the server being able to learn any information about which nodes in the tree are included in the traversal path. The client algorithm for the PIR scheme is then used to encode these addresses into PIR queries, and the queries are sent to the PIR server(s). The server(s) uses the server algorithm for the PIR scheme (i.e., response encoding) over the index to generate a response, and forwards it to the client. The query executor locally ranks the data returned for the queries before displaying it to the user. The second round of PIR proceeds once the user selects the desired item for retrieval from the result. This time, the particular data item of interest is retrieved from a statically specified bucket or a dynamically defined portion of the large database. The retrieval procedure is akin to the standard search-and-retrieve model used for searching documents and retrieving one of the documents returned by following links on everyday search engines. In practice, however, more dummy queries may be needed to prevent the adversary from learning the number of terms in the user's query or the size of the data item that was eventually retrieved.

4.2 Quantifying the Level of Access Privacy

Applying PIR over an entire database offers a strong privacy guarantee. However, when PIR is applied to a subset of a large database, it is necessary to quantify the amount of information that is leaked to the adversary.

We assume the database X consists of r blocks, the database subset Y of X consists of t blocks $t \leq r$, and the user's query u is a discrete random variable, taking values in $[r]$. Let $u_i = P(u = i)$ be the adversary's prior probability distribution of the items the user may search for. This distribution may come from external knowledge about the popularity of items in the database, for example.

The information that is disclosed in defining a database subset Y affects the access privacy guarantee that is possible for that subset. This supplies the adversary with more information about the user's query.

We quantify the information content of u using Shannon entropy as $H_X(u) = -\sum_{i=1}^{r} u_i \log_2(u_i)$.

The information gained by the adversary when the user queries a database subset is simply the difference between the Shannon entropies of an entire database query $H_X(u)$ and a subset query $H_Y(u)$. We compute $H_Y(u) = -\frac{1}{p_Y} \sum_{i \in Y} u_i \log_2(\frac{u_i}{p_Y})$ where $p_Y = \sum_{i \in Y} u_i$. Then the amount of information leaked by the query is $\lambda = H_X(u) - H_Y(u)$. (Querying the entire database with PIR leaks no information to the adversary, for example, so $H_Y(u) = H_X(u)$, and $\lambda = 0$.)

Using techniques like those in [14, 35] for measuring the degree of anonymity provided by a system, we can define the degree of access privacy provided by any subset. We divide the equation above by $H_X(u)$ to obtain a bounded value in $[0, 1]$ for the information gained by adversary. We define the *degree of access privacy* for the subset Y as

$$\varrho = 1 - \frac{\lambda}{H_X(u)} = \frac{H_Y(u)}{H_X(u)} \tag{1}$$

5 Proposed Solution

We provide concrete details of our approach for providing access privacy over large databases in this section. Our solution allows the user to obtain reasonable access privacy when querying a large database by allowing the user to specify her delay tolerances and privacy preferences along with her query. For simple index-based retrieval, the query is first transformed at the client side to an equivalent query which can satisfy the delay tolerances and privacy preferences of the user. The query transformer is able to determine the size of the data subset to PIR-process because it has some knowledge about query response times for various database sizes and network bandwidths; we assume a prior calibration or training step that achieves this. The output query is a PIR-encoded query over some subset of data in the database. Data subsets that will be PIR-processed can be defined dynamically at runtime when the query transformer leaks some information about the query, provided that such leakages are allowed by the user's privacy preferences. Statically defined database subsets are used when the user query specifies which subset(s) or bucket(s) should be PIR-processed in order to service the user's query. The transformed query is subsequently forwarded to the PIR server, which processes it and returns a response to the query executor at the client. The client can perform further ranking or filtering before it finally displays the result to the user. We will describe in detail each of the steps from our model in Section 4 in two phases. In the first phase, the database administrator generates indices and establishes data subsets in buckets. During the second phase, the user runs her queries.

5.1 Defining Database Subsets

Since preserving absolute access privacy over a large database is hard to accomplish in near real time, data classification can be used to organize the data in such databases into smaller subsets. A subset can be defined dynamically at runtime or statically before the first query is run. We refer to dynamically defined subsets as *portions*, and statically defined subsets as *buckets*. The minimum size of a subset should be sufficiently small so that it can be queried within the smallest amount of delay a particular system supports. For example, if a system supports a minimum delay of 10 s, and given a PIR scheme like Goldberg's [17] which processes databases at 1.3 s/GB, then the size of each database subset should be approximately 5 GB, leaving some time for other processing activities and network delays.

5.1.1 Runtime definition

We note that it is not always the case that buckets must be defined for every practical situation. If it is sufficiently fast to define database portions at query time, then the user query may leak information to help the database server define the subset. Defining database subsets dynamically offers some flexibility that makes it suitable for any query type we support—index-, keyword-, and language-based queries. Besides, it groups data of similar kind in the target portion, thereby allowing range queries over the data in that portion. However, it has a number of drawbacks. First, the client software must explicitly leak information to the database about the interest of the user. Disclosing broad information about the user interests to the adversary might seem innocuous at first; however, if the adversary possesses some additional knowledge, he or she might be able to utilize this information to narrow down the search space for the user's interest. Second, it might be slow, because of the server-side requirement to retrieve the data for the portion before computing on the data using PIR. Third, the client would need information about the distribution of the data in the database (i.e., a histogram of the data) before it can have assurance that the portion it is defining is indeed safe or contains enough data. For example, revealing a substring of a user's query might just be enough for the adversary to identify the particular data targeted by the query. The client needs to be able to approximate the size of the resulting database portion from looking at her substrings. Data histograms are constructed easily by today's relational databases, but an extra step is required to construct it for unstructured and semi-

structured data. Finally, there are privacy concerns with respect to the popularity of defined portions. For example, if the data in the portions defined by a particular user is not popular, then the attacker might have higher probability of determining the user's interest. Conversely, better privacy will result if the data in the portions have high popularity.

Information that is leaked to define database portions dynamically depends on the type of query. For index-based queries, the client will identify the range of indices that contains the desired index. The client must maintain local state information to help it reuse the same range for queries on all indices in that range; otherwise, it might be vulnerable to intersection attacks. Defining database portions for keyword-based queries is difficult because a keyword can appear in any attribute of a data set. We will later describe our technique for realizing keyword-based queries. The client has some leverage in leaking information on language-based queries. In SQL for example, information might be leaked in several places, such as substrings of a predicate constant, whole constants, whole predicates or even all predicates in a query. As an example of leaking information about sensitive constant substrings, consider the SQL query in Listing 1.

Listing 1 Example SQL query to be transformed.

```
SELECT title, abstract, url, version, last_updated
FROM document
WHERE (last_updated > 20100101) AND
      (title = 'Practical Access Privacy')
```

Disclosing the prefix `'Practical'` of the sensitive constant `'Practical Access Privacy'` gives some privacy because several documents have title that begins with the keyword `'Practical'`. The really sensitive part of the title is still hidden from the database. The resulting subquery that leaks the title prefix is shown in Listing 2.

Listing 2 Transformed SQL query with some substring of the constant hidden.

```
SELECT title, abstract, url, version, last_updated
FROM document
WHERE (last_updated > 20100101) AND
      (title LIKE 'Practical%')
```

The corresponding PIR query for Listing 2 has the form $KeywordPIRQuery(\texttt{'Access Privacy'}, I)$, where I, based on the technique from Olumofin and Goldberg [28], is the index generated from the subquery result, and $KeywordPIRQuery(\cdot, \cdot)$ is a keyword-based PIR search over the index I. The result of the PIR query is the overall result for Q. The decision to leak information about constants `20100101` and `'Practical'` is best based on the statistics of past disclosures from other users of the system running this particular query. For example, if 90 percent of users leak information about the date `20100101` in the query in Listing 1, then it should not be a significant privacy loss to also leak information about that date in the current user's query. However, such leakages can result in much more efficient PIR because PIR will now be performed on a smaller portion of the database.

5.1.2 Offline Definition

Before the first query is made, the database is classified or clustered into some k buckets. We consider databases that contain information in hierarchically organized categories. For example, "Sports" might be a parent category for "Baseball" or "American League". If the database does not have a natural hierarchical structure, then decision-tree [32], neural-net [26], or Bayesian [20] classification techniques can be used for

most multi-attribute data sets with relevance to access privacy. There is a large body of work in the data mining literature about these areas and this work does not explore them further. A hierarchical structure is needed to support range queries within a particular parent category. Otherwise, data that should belong together are distributed to separate buckets, making range queries impossible. In addition, we require the classification approach to provide a function that accepts a data point and computes the bucket number containing that point. This requirement disqualifies many traditional clustering algorithms because removal of a single point may completely alter clusters. We note that this requirement is not difficult to realize in practice, because most real-world databases where access privacy is needed can be categorized using techniques like decision trees. The data in structured data sources like relational databases can always be classified by some attributes of the data. The classification can utilize histograms maintained by the database on the distribution of the data set. We note that histograms are widely used in relational databases for estimating selectivity during query optimization and for approximate query evaluation.

Once the data is classified, each of the k portions may simply be represented as pointers to items in the larger database or the data may be materialized from the larger databases and separately stored on disk. At query time, the client must evaluate the provided function in order to determine which bucket (or buckets, if the user can tolerate more delays) should be queried. Unlike when a client defines database portions at runtime, the client is not leaking any information *directly related to query contents* to the database. The only information being leaked is the bucket(s) to use, which can be made not to disclose any specific information related to the item of interest as explained next.

For the classification techniques for the static bucket definition approach to be maximally private, each bucket must be striped with information from many or all possible parent categories. In other words, every bucket should exhibit *high diversity in their information content*, such that the user's general interest cannot be localized to any one category, simply from learning the bucket number(s) used for the query. We defer to future work a way to derive optimally diverse buckets and how to guarantee such a property for any data set. In addition, mechanisms for ensuring that buckets have equal popularity among users are necessary especially if the database cannot be relied upon to provide information about the popularity of buckets. Mix-matching of buckets with disparate popularity increases the chances of the adversary guessing the item of interest to the user. One possible mechanism is for the servers in a multi-server PIR setting to compute the top-k popular items in the database and replicate these into a smaller database. We studied such a mechanism in previous work [21]. Note that in computing the top-k popular items, the servers do not learn any information about the queries of users; they only jointly learn the top-k popular items. The replication of the top-k items to smaller databases allows PIR to be performed in a more computationally efficient manner, in addition to enhancing the privacy of user queries.

Another possible mechanism is for users to jointly support a semi-trusted public forum to which they can periodically connect and upload the map of the database portions or bucket numbers that have been previously used for their queries. We note that disclosing the portion map does not help the database or a third party to violate the privacy of the user, since the data items of interest to the user will still remain hidden within the map. Such a forum serves as a repository to develop the statistics of the popularity of each database portion or bucket. The distribution of popular database items may not be as precise as the top-k mechanism above, but it is nonetheless useful in a single-server PIR setting or in a multi-server PIR setting where the servers cannot be relied upon to compute the top-k popular items. A user takes advantage of the public repository by retrieving a portion map that contains the data of interest from the repository and then sending the map along with her query to the PIR servers. A simple technique to ensure that queries have similar popularity is for users to volunteer by sending dummy queries to enhance the frequency of use of unpopular portions or buckets. Such public forums are not detrimental to users' privacy because it is the same information that users send to the database in the first place. However, there are performance implications from the use of dummy queries. The database can actually maintain the same information and even associate each map to the users that have used them. Making it public provides a means of sharing

maps among users in order for them to cooperate in protecting their queries. For example, many users may independently define a map containing a particular bucket, which happens to contain a data item about score information from the 2010 FIFA World Cup. Given the popularity of that data item in the months when the soccer games were played, the database might have a better chance at learning the target of such popular queries. On the other hand, if multiple buckets having the same popularity are used by several users, it will be harder for the database to learn which data item is responsible for the high traffic.

A related practice from major search engines is a recent feature advertising most popular search terms or hot topics to users. Microsoft Bing offers *xRank*. Google offers *Google Trends*, *Google Insights for Search*, and *Trends for Websites*. Yahoo! offers *Trending Now*. Some of these contain coarse-grained information about the location of the user that originated the search. Statistical information of this nature about previous searches accumulated for a particular system can be useful for making tradeoff decisions between performance and efficiency for applications using PIR schemes.

Some of the benefits of using static buckets include no explicit information leakage about the user's query, improved online query performance because data materialization for portions defined dynamically can be avoided, and flexible definition of sub-portions in a dynamic fashion over individual buckets. Some of the drawbacks are that it is best for data that has been or can be classified into some hierarchical structure and the need to cluster the data ahead of time.

5.2 Indexing Large Databases

The data indexer generates indices which will be used as supplementary data structures for privately querying the large database. We consider two types of index structures; a B^+ tree for non-unique keys for range-based queries and hashed indices based on perfect hash functions (PHF) [6] for point or single-value queries [28]. We note that the indices described in this section are intended to support queries over data in static buckets, and not on data over dynamically defined database portions. Indices to support queries over dynamically defined database portions must be built at runtime. We follow a similar technique to that of [28], where the database executes and index the result from a desensitized subquery sent by the client. For a tree-based index, the root is sent back to the client, which then performs an oblivious search over the index for the data item of interest. For a hash-based index, the metadata for computing the perfect hashes of data items is returned, and the client can compute the address of any data item that is to be retrieved with PIR using this function.

Several indices need to be built for querying a large database, which could either be used as a master index or as a secondary index. A master index is used to support keyword-based search over one or more attributes. For example, we might have a master index for all patent titles in the database, another one for both patent titles and abstracts, and a third one for the title, abstracts and claims in all patents. A secondary index can be built for each of the data attributes that will be used as keys for retrieval.

5.3 Generating an index to support keyword-based retrieval

We denote the range of integers between 1 and m as $[m]$. Given a list of attributes over a large data set to build an index on, the database follows these steps:

Step 1: Generate a PHF f over the list of m unique keywords (excluding stopwords) from the data in the attributes. Construction time for the state-of-the-art PHF [6] takes $O(m)$ time, and as low as 0.67 bits per key is stored. It takes under 30 s on commodity hardware to generate this PHF for $m = 2 \times 10^7$ (20 million unique words) and a representation size of under 2 MB.
Step 2: Create a table T of m rows, where each row T_i $(i \in [m])$ is the slot for the i^{th} unique word. More precisely, each row T_i consists of two fields: a counter field T_{i-size}, and a contiguous, fixed-size list

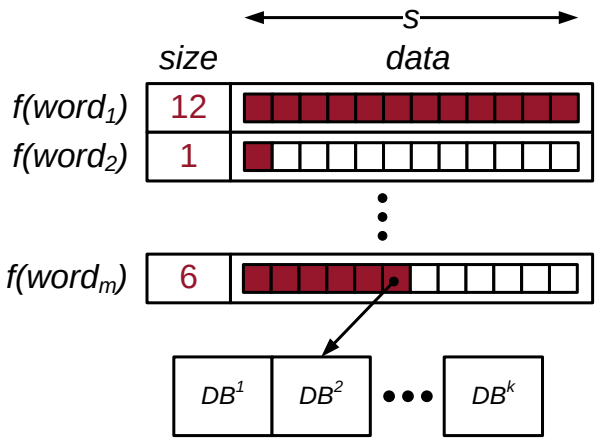

Figure 2: Index structure (i.e., table T) to support keyword-based retrieval over a large data set. $s = 12$ is the width of the index, $f(\cdot)$ is a PHF, $word_1, \cdots, word_m$ are keywords, and $DB^1, \cdots, DB^k$ are subsets of the database.

of length s of summary data T^j_{i-data} of data items that contain that keyword. The fixed-size summary of each data item should contain sufficient information to convey the content of the data item to the user and, possibly, for ranking of retrieved summaries by the client. Additionally, it should contain the offset of the data item in the bucket the data item is classified into. In a patent database, for example, things like title, date of application, patent number and an excerpt of the patent's abstract with the keyword in context would be appropriate. The value of s is carefully chosen to ensure that every data item in the database exists as T^j_{i-data} for some values of i and j. In other words, every data item can be searched and located by some keywords. On one hand, if the value of s is chosen too small, many summary data items might be excluded from an index, and could not be located during a search. On the other hand, if s is chosen to be too large, then query response over the index will suffer. Since the same number of patents are indexed for every word, an appropriate ranking strategy will help ensure that every patent appears in some T^j_{i-data}. Ranking may imply truncating the list of data items containing a particular keyword. Should there be fewer than s summaries for a particular word, the remaining slots are padded as appropriate. Similarly, some high-frequency words (i.e., words appearing in too many data items) or words that span too many unrelated buckets might be excluded entirely. Finally, initialize $T_{i-size} = 0$ $(i \in [m])$.

Step 3: For each data item, prepare a summary α and extract unique words (excluding stopwords) from the key attributes for the master index. In preparing α, a lookup to a secondary hash table might be necessary to obtain the offset for the data item which α is summarizing in the containing bucket. The hash table is easily generated during classification of data into buckets.

Step 4: For each word w use the PHF f to compute the table slot $k = f(w)$ and update $T^{\ell}_{k-data} = \alpha$, where $\ell = T_{k-size}$ or the ℓ^{th} slot in rank order of the list of summaries including α and the existing summaries in T_{k-data}. If $T_{k-size} = s$ (i.e., full), then we drop the lowest-ranked item. Otherwise, increment T_{k-size} by 1. Continue from Step 3 if there are more data items. The above step ensures that the highest-ranked summaries are placed on each T_{k-data} list. We illustrate the structure of T and the buckets in Figure 2.

We note that performing PIR over a master index is within reach, compared with performing PIR over the large database. The key benefit of the master index is that its size is much smaller than n. In environments with high transaction volume, the value of m will increase rather slowly, whereas the value of s or the ranking strategy for the patents containing a particular keyword w might need to be changed to ensure all data items are indexed, and also to minimize wasted space.

Secondary indices to support keyword-based PIR over any of the attributes of a data item can be gen-

erated in a similar manner; however, the words are now the values of the keys. We note that we describe a related approach in previous work [28] for the generation of hash-table and B^+-tree indices.

5.4 Privacy-Preserving Query Transformation

A query transformer follows some constraints to convert an input query into another query that may be executed by some privacy-preserving system. In the traditional relational database literature, query transformation or query rewriting is performed as part of the query optimization process. The goal is to find the most efficient plan for executing a query, which usually involves using some equivalent rules that define alternative execution strategies to expand input queries into many potential execution plans. The optimizer chooses the plan with the least cost, usually with the least amount of disk access [36].

Our notion of query transformation aims to derive an equivalent output query that will provide an acceptable level of access privacy, given the cost constraints and privacy preferences of the user. The concept of query transformation can be applied to the three data access and query models that have been proposed in the study of PIR (i.e., index-based [12], keyword-based [11] and language-based [28]). As an example, we will describe the transformation process for keyword-based queries, which is the most widely used online search model.

We use integer values for the user delay tolerance δ in seconds. Since there is no universal model of privacy preferences that can satisfy the needs of every user, system or privacy legislation [24], any number of parameters can be specified as privacy preferences ρ in a particular situation. The privacy preferences will constrain the information that is leaked about the buckets or database portions that will be used in answering the user's query. For example, a user might opt never to disclose any information about substring constants in an SQL query if the user considers it to be too risky.

An example of modeling user privacy preferences for an SQL query is available in Appendix A.

5.4.1 Transforming keyword-based queries

We take a k-keyword query as consisting of a nonempty set of keywords $\omega = \{w_1, w_2, \cdots, w_k\}$ combined by logical operators. The client computes the address of each of the keywords $i_j = f(w_j)$ for all $j \in [k]$ in the master index using the PHF f retrieved from the server. Then, it generates PIR queries for those addresses and forwards them to the PIR server(s). The PIR server(s) evaluate the queries over the master index, which is much smaller than the whole database. (It is also possible for each keyword to target a different index, but that requires some enhancements in the query syntax and semantics.) Let the result returned by the PIR server for the respective keywords be $r_1, r_2, \cdots, r_k$. Recall from the index generation steps above that each result $r_j \equiv T_{j-data}$. The client would then combine and rank the elements of T_{j-data} using the logical connectives. The ranked list of summaries $\alpha_1, \alpha_2, \cdots, \alpha_L$, $L \leq ks$, is now displayed to the user. We note that the database server cannot learn *any* information about ω since each PIR query is over the entire block of data for the master index or the indices used. As previously identified, dummy queries can further prevent the learning of the number of keywords in the user's query.

After the user responds by selecting an α_j, the query transformation process begins. Again, the description of ρ will help the query transformation process decide whether to base the retrieval upon the bucket specified in the description of α_j or to leak some information about the data item of interest using data contained in the summary. Recall that each summary contains the bucket number, index of the data item in the PIR data block for the bucket, and other identifying information. Should the client choose to define database portions for answering the queries, based on the user's privacy preferences, it will use information about the data statistics contained in the calibration metadata to determine whether the leakage is safe. For example, if the statistical information says that some $110,000$ patents were issued the first two quarters of the year 2009, the client might leak information about the issue dates of the patent as being January to June

of 2009 if the query input constraints are met. That is, it is also the case that $110,000$ patents is sufficient to satisfy the user's minimum access privacy level, the estimated execution time does not exceed δ, and the user privacy preferences allow disclosure of information about the year and/or month.

However, if the client chooses to query data from the buckets, it determines the bucket that contains the data item to be retrieved from the content of α. Using the parameters from calibration metadata (see Appendix A for a description), it checks if $\delta \leq (t_{tr} + t_{PIR} + \Delta)$ (Δ accounts for the variability in the timing for transformation and post-processing of PIR queries) and returns $\perp$ with a suggestion to the user about adjusting the value of δ; otherwise the transformation process proceeds. The client examines its archive of query maps and selects the one that contains the bucket of interest, if found. Otherwise, the user randomly selects some γ buckets in a list such that $\delta \leq (t_{rt} + (\gamma + 1) \cdot t_{PIR} + \Delta)$, adds the bucket containing the data item to the list, and permutes the modified list. The permuted list will be used to answer the query.

The output of query transformation over bucket(s) consists of a set of one or more bucket numbers and the addresses of the data items to be retrieved from the bucket(s). A similar output for queries over dynamically defined database portions consists of the identifying information to be leaked to the server and a *key* derived from α_j for locating the complete data for α_j from the portion.

5.5 Privacy-Preserving Query Execution

The query executor runs a transformed query over the data defined by the bucket or portion map to retrieve the data item of interest, while preserving the access privacy preferences and performance constraints of the user.

The process of running a transformed query over some bucket(s) begins with the client generating PIR queries to retrieve the data item at the specified address. The client determines the relative address of this item using the bucket offset and length information from the calibration metadata. In the case of dynamically defined portions, the query executor at the client requests the database to retrieve the data items related to the leaked information and to build an index over the retrieved data. The PIR server will forward some session metadata to the client which will allow it to perform keyword-based PIR [11, 28] over the index that was generated.

6 Implementation and Evaluation

The ability to query a patent database without giving away access privacy is one of the most well-cited application areas motivating the study of private information retrieval. We evaluated our approach using the patent database that has been made freely available through an agreement between Google and the USPTO.

6.1 Experimental Data Set

Bulk Downloads of US Patent Grants. In the past, organizations needing to do comprehensive analysis of patents issued by the USPTO could either request bulk downloads on digital media like CDs or DVDs for a fee of between $10,000 and $250,000 to cover USPTO expenses [19], or download the information on a file-by-file basis from the USPTO website. The recent agreement between Google and USPTO has enabled anyone to download the entire collection for free through Google. The bulk download data made available via Google are unchanged from their USPTO formats, although they have been repackaged as zip files.

Google Patent Search. In addition to making bulk patent data available, Google Patent Search enables users to perform full-text search and retrieval over the entire database of patent grants and applications. Google hosts and maintains a corpus containing all of these documents, which have been converted from the original USPTO images into a form that can be easily searched. As of November 2010, there are over 7

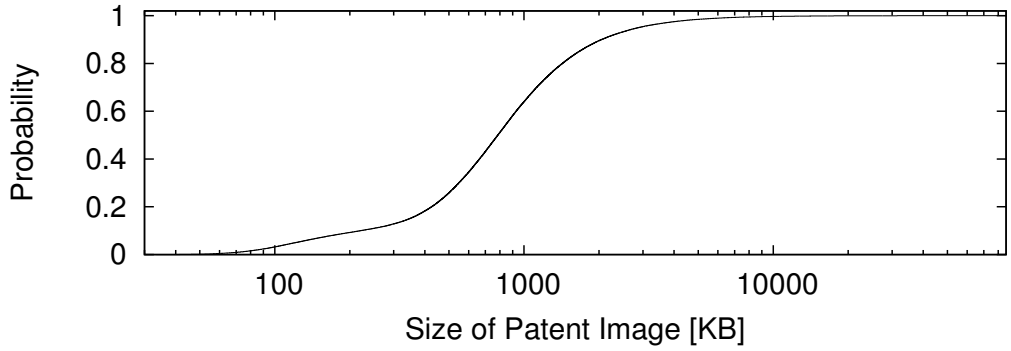

Figure 3: A CDF plot of our sample data patent image file sizes. The x-axis is on a log scale.

million patents and over a million patent applications. The entire collection of patents issued by the USPTO since the 1790's through those issued in the last few months, as well as patent applications that have been published, are available in the corpus. Users can either perform a full-text search over the corpus or search by attributes like patent number, classification, and issue date.

The USPTO grants approximately 4,000 patents weekly [19]. We downloaded zip files for the multi-page images of all patent grants for the year 2009, and used them as our experimental data set. The zip files contain images in Tagged Image File Format (TIFF) Revision 6.0. After extraction, we had a total of 209,960 individual patents in separate multi-page TIFF files (a total of 220 GB of data).

In addition, we downloaded zip files for the "US patent grants full text with embedded images" corpus for 2009. Each zip file contains the full-text, drawings, and secondary file formats like genetic sequence data and chemical structures. The full-text files are XML files that conform to the U.S. Patent Grant Version 2.4 Document Type Definition (DTD). We only extracted the full text XML files from the zip files; the secondary files were not extracted because they are already part of the patent TIFF images. Since XML files are more easily parsed and processed, we extracted data from the full-text XML and used it to index the patent TIFF images. The `doc-number` tag in each XML file served as a link to the image of the patent document in TIFF format since the patent number is used as the filename for each TIFF file. The full-text patent XML files resulted in an additional 16 GB of data.

The CDF plot of the file sizes for the TIFF images (in KB) is shown in Figure 3. The smallest patent file was 33.4 KB, and the largest was 155.8 MB. However, the next to the largest file was only 83.3 MB. From the CDF plot, about 99% of patents are 6 MB or smaller, which is reasonable. Padding 99% of the patent files to have equal length will not be a significant problem; however, the adversary might have an advantage identifying the remaining 1%. On the other hand, padding all patents to be 156 MB will offer the best privacy, but the user will have to perform more PIR requests than necessary 99.99% of the time. Although the padding strategy is an important security parameter, we deferred the tradeoff decision of choosing a strategy until actual system deployment time. We however pad each patent to have a size in multiples of the PIR block size. In addition, we were able to obtain some savings in the TIFF file sizes using compression.

6.2 Experimental Setup

We developed a C++ system based on our access privacy approach for searching and retrieving from the database of patent documents. We used RapidXml [23] to parse the patent XML files and the C Minimal

Perfect Hash (CMPH) Library [8] for building hash-based indices. We implemented the PIR aspects of our system using Percy++ [18]. Our hardware is an Intel Xeon E5420 server with two 2.50 GHz quad-core CPUs, 32 GB RAM, and Ubuntu Linux 10.04.1. The rest of this section describes how we built indices over the patent data in order to support keyword-based queries.

6.3 Classifying USPTO Patent Data

We leveraged the U.S. Patent Classification System (USPC) to classify the patents in our data set into buckets such that each bucket contained one or more collections that are identifiable by their alphanumeric subclass. We determined the number of buckets to use by considering the total size of our patent images (i.e., 220 GB) and the time it will take a low-budget user to successfully query a bucket. If the data in each bucket is too large, it might not be within the reach of low-budget users. On the other hand, if there are too many small buckets, then several of the buckets would need to be specified in the bucket map for a query, which adds to the bandwidth cost. With compression of the patent images, and exclusion of patent images of some full-text XML files that failed parsing with the RapidXml library, the total compressed data size we classified into buckets was 127 GB. Also, we decided to make each bucket be approximately 1 GB of compressed data. That informed our choice to set the number of buckets to 126. We simply took the hash of the alphanumeric subclass of each patent to determine the bucket number.

6.4 Generating Indices

We built indices over the experimental data to support keyword-based search. The process involves reading and parsing the XML files, extracting relevant attributes for creating the summary of the patent, distributing the patent TIFF image files into their respective buckets, and finally merging the buckets. We concatenated the TIFF images of the patents in each bucket because the PIR scheme used for the evaluation expects a PIR database to be a single block of data.

Again, we used a simple strategy to rank all patents containing a particular keyword. For a particular word, the patents containing that word are ranked using the reciprocal of the total number of words in each patent. We note that term frequency [33], inverse document frequency [38], and cosine similarity [34] are alternative ranking strategies explored in the data mining community.

Completeness of keyword-based indices. We evaluated the index for keyword-based search over the titles and abstracts in our data set to determine whether every patent appears as a summary item associated with some words. If some patents are missing, then they cannot be searched and retrieved with any keyword. Again we discovered that index completeness is a tradeoff decision driven by the ranking strategy of patents for a particular word and the width s of the index. We will only consider the latter.

For our experiment, when we set the value of s to 31 to align the index entry for each word to be in multiples of 4 KB, about 31% of the patents did not appear in the index. When we increased s to 63, still 24% of the patents were left out. However, when we increased s to 252 only about 0.2% of the patents were excluded.

The cumulative distribution function of words in the number of patents is shown in Figure 4, which further explains the reason for the behaviour. We only considered whether a word appeared in the key attributes (title and abstract) of a patent, and not how often it appeared. From the plot, 25% of the words appeared in exactly one patent, 65% of the words appeared in at most 10 patents, and 98% of the words appeared in up to 1000 patents. Recall that s constrains the width of an index. On one hand, if any patent

has all of its keywords being common (appearing in more than s patents), it may not be indexed at all. The smaller s is, the more likely this is to happen. On the other hand, the user might have to retrieve enormous amount of data in the display page of a search results if s is made too large.

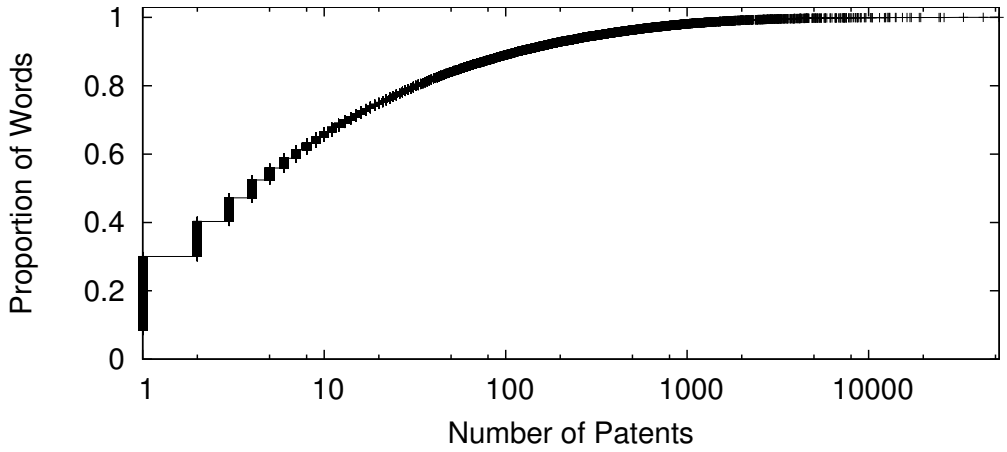

Figure 4: A CDF plot of our sample data showing the popularity of words in patents. The x-axis is on a log scale.

One solution to ensuring all patents are indexed is to use a better ranking strategy. Another solution is to set s to be the frequency of the word with the highest frequency of occurrence in patents. The T_{i-data} items would no longer be required to be of equal length. Each may, however, be padded to be in multiples of the PIR block size. At query time, the client will have the flexibility of choosing the number of summaries to retrieve for each T_{i-data}. This is achieved by specifying the beginning offset into T_{i-data} and the number of summaries to retrieve. Varying the size of the block does leak some information to the database. The client will have to be consistent in the size of the block used. A parallel to the above is when the results of a search engine are returned: sometimes thousands, sometimes few. Often, the user does not have to view the low-ranking entries at the bottom of the list before she finds the summary that interests her.

Patent class diversity of buckets. It is necessary for each bucket of patents to be diverse in terms of its USPC class representation. A highly diverse bucket decreases the chance of leaking information about the general interest of a user when the bucket number is disclosed to the PIR server. We examined the distribution of USPC classes into the various buckets, and found each bucket contains information from between 136 and 180 classes. There are 474 classes and a total of 158 514 unique subclasses in our data set.

Keyword diversity of buckets. We examined the distribution of the 44 600 keywords in our data set into buckets. If a keyword targets one or a few buckets, then the adversary might be able to infer the query keyword from the bucket number. We examined the frequency of keywords in each bucket for our experiment and found each bucket has between 5 862 and 10 137 keywords. The buckets defined exhibit high diversity in their keyword contents. Better privacy can be obtained from combining multiple buckets in a query.

6.5 Privacy-preserving Patent Database Query

We measured the query response time of our implementation for keyword-based query over our data set. The three timings reported in Table 1 represent the time to generate a master index over the entire data set ($Time_{gen}$), the time to query the index ($Time_{q_I}$), and the time to retrieve the patent image from the

Table 1: Measurements taken from executing keyword-based queries. s is the width of the index, $Size_I$ is the size of an index in MB, $Time_{gen}$, $Time_{q_I}$, $Time_{q_B}$, and $Time_{q_{DB}}$ are respectively the timings (seconds) for generating an index, querying the index using PIR, retrieving a patent from a bucket using PIR, and retrieving a patent from the entire database using PIR. $Download_B$ and $Download_{DB}$ are the respective timings (seconds) for downloading a bucket and downloading the entire database both over a 9 Mbps network bandwidth [31]. A bucket consists of 1.2 GB of data, while the database consists of 127 GB of data.

s	$Time_{gen}$ (secs)	$Size_I$ (MB)	$Time_{q_I}$ (secs)	$Time_{q_B}$ (secs)	$Download_B$ (secs)	$Time_{q_{DB}}$ (secs)	$Download_{DB}$ (secs)
31	536	348.27	0.6±0.0	42.7±0.1	1092.3	4519.1	115598.2
63	609	1393.09	2.4±0.2	42.7±0.1	1092.3	4519.1	115598.2
252	1600	5572.38	10.0±0.1	42.7±0.1	1092.3	4519.1	115598.2

corresponding bucket ($Time_{q_B}$). The size of the master indices for the parameter s is also shown ($Size_I$). For all values of s, the size of the bucket containing the patent retrieved was 1.2 GB. The time for generating an index increases linearly with s. The sizes of the indices also grow linearly in s, as does the time to query the index. The time to retrieve a patent TIFF image from the bucket is constant in all cases. It is not surprising that it takes longer to retrieve a patent image from a bucket than a single block containing patent summaries from an index, because the retrieval of patent images requires a larger block size and returns more data than the retrieval of summaries in an index. In this particular case, queries over the respective indices are fetching 8 KB, 32 KB, and 128 KB of data, whereas 768 KB of compressed patent images were retrieved from the bucket.

The performance benefits of our approach for privacy-preserving search and retrieval of patent images become more compelling when we compare the combined time to query the indices and the buckets with PIR to the time for downloading the entire 1.2 GB bucket. Our approach only requires 4% to 5% of the time for trivial download. Similarly, if we compare our approach with a naive approach that performs PIR over the entire 127 GB patent database, we see that query response times with our approach are only about 1% of the response time for the naive PIR approach. Besides, additional cost would be incurred from a necessary search that must first be performed over a secondary index structure built over the 127 GB database, since it is impossible to efficiently search and retrieve from a PIR database without using a secondary index data structure. The absolute privacy benefits of privately searching and retrieving from the entire database may be too expensive. Finally, if we compare our approach with the trivial download of the entire 127 GB database, we see that our approach requires between 0.04% and 0.05% of the time for trivial download. The main benefit of our approach is the feasibility of obtaining strong privacy over a reasonable subset of a large database, as determined by the user. All of the other approaches explored above are impractical today in terms of computation cost. With our approach, the user can privately search and retrieve from a database with reasonable privacy protection proportional to the size of a bucket.

6.6 Resistance to Correlation Attacks

Our approach is resistant to correlation attacks irrespective of the number of keywords used in a single query, the number of queries a user makes, or if queries are repeated. The reason behind the privacy guarantees of our approach is the completeness of the indices that users query; i.e., the indices were built over the entire data set of patents and not over a subset. Each T_{i-data} retrieved for each keyword does not leak any information about the keyword or its relationship with the other keywords. The ranking of the combined T_{i-data} summaries, and the selection of a single summary item from the combined summaries is performed at the client, and therefore leaks no information to the database servers. The summaries convey sufficient

information to help the user choose the exact summary of the patent of interest, which is retrieved at the second round of PIR queries. This second round of PIR is performed over a specific bucket, which leaks no additional information about the specific patent in the bucket that may be of interest to the user. Any information the server learns from the user performing successive PIR queries over the same or different buckets does not improve the adversary's chances of learning the data items of interest to a particular user.

7 Conclusions and Open Problems

We have proposed a practical approach for gaining reasonable access privacy when querying a large database. Our approach is based on classifying a large data set into buckets, dynamically defining portions of the database, generating indices over the larger database, performing keyword- and attribute-based queries over the indices, and retrieving data items from a suitably defined database portion or static buckets. We validated our approach experimentally by applying it to the query of patent data obtained from the agreement between USPTO and Google, and demonstrated that a privacy-friendly alternative to Google Patent is possible. Future work will further extend our approach using a larger data set and will explore using better classification and ranking algorithms in the classification and indexing of the data. It will be interesting to explore ways to classify any data set into optimal portions that minimize information leakage for user queries over the portions.

Acknowledgements

We thank Kevin Bauer, Tariq Elahi, Ryan Henry, Jean-Charles Grégoire, Rob Smits, and Angèle Hamel for their helpful comments on drafts of this paper. We gratefully acknowledge the funding support of NSERC, MITACS, and the Research In Motion Graduate Scholarship.

References

[1] C. Aguilar-Melchor, B. Crespin, P. Gaborit, V. Jolivet, and P. Rousseau. High-speed private information retrieval computation on GPU. In *SECURWARE'08: Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[2] D. Asonov. *Querying Databases Privately: A New Approach To Private Information Retrieval*. SpringerVerlag, 2004.

[3] D. Asonov and J.-C. Freytag. Repudiative information retrieval. In *WPES'02: Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, pages 32–40, New York, NY, USA, 2002.

[4] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, Lecture Notes in Computer Science, pages 55–73. Springer, 2000.

[5] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: Pir with preprocessing. *J. Cryptol.*, 17(2):125–151, 2004.

[6] D. Belazzougui, F. C. Botelho, and M. Dietzfelbinger. Hash, displace, and compress. In *ESA 2009: Proceedings of the 17th Annual European Symposium, September 7-9, 2009*, pages 682–693, 2009.

[7] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *CLOT '92*, pages 144–152, 1992.

[8] F. C. Botelho, D. Reis, and N. Ziviani. CMPH: C minimal perfect hashing library on sourceforge, Accessed June 2009. http://cmph.sourceforge.net/.

[9] Y. Chen and R. Sion. On securing untrusted clouds with cryptography. In *WPES '10*, pages 109–114, 2010.

[10] B. Chor and N. Gilboa. Computationally private information retrieval (extended abstract). In *STOC'97: Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 304–313, New York, NY, USA, 1997.

[11] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. Technical Report TR CS0917, Department of Computer Science, Technion, Israel, 1997.

[12] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *FOCS'95: Proceedings of the 36th Annual Symposium on the Foundations of Computer Science*, pages 41–50, Oct 1995.

[13] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[14] C. Díaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. In *PET'02*, pages 54–68, 2003.

[15] J. Domingo-Ferrer, M. Bras-Amorós, Q. Wu, and J. Manjón. User-private information retrieval based on a peer-to-peer community. *Data Knowl. Eng.*, 68:1237–1252, November 2009.

[16] G. Ghinita. Understanding the privacy-efficiency trade-off in location based queries. In *SPRINGL '08: Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS*, pages 1–5, New York, NY, USA, 2008.

[17] I. Goldberg. Improving the robustness of private information retrieval. In *IEEE Symposium on Security and Privacy*, pages 131–148, 2007.

[18] I. Goldberg. Percy++ project on SourceForge, June 2007. `http://percy.sourceforge.net/`.

[19] Google. About google patent search, Accessed November 2010. `http://www.google.com/googlepatents/about.html`.

[20] K. A. Heller and Z. Ghahramani. Bayesian hierarchical clustering. In *ICML '05*, pages 297–304, 2005.

[21] R. Henry, F. Olumofin, and I. Goldberg. Practical PIR for electronic commerce. Tech. Report CACR 2011-04, University of Waterloo, 2011.

[22] D. C. Howe and H. Nissenbaum. TrackMeNot: Resisting surveillance in web search. In I. Kerr, V. Steeves, and C. Lucock, editors, *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, chapter 23, pages 417–436. Oxford University Press, 2009.

[23] M. Kalicinski. RapidXml. http://rapidxml.sourceforge.net/, Accessed November 2009.

[24] A. Kobsa. Tailoring privacy to users' needs. In *UM '01*, pages 303–313, 2001.

[25] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *FOCS'97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, page 364, Washington, DC, USA, 1997.

[26] R. P. Lippmann. *An introduction to computing with neural nets*, pages 36–54. IEEE Computer Society Press, Los Alamitos, CA, USA, 1988.

[27] J. Nielsen. Nielsen's law of Internet bandwidth, April 1988. `http://www.useit.com/alertbox/980405.html`.

[28] F. Olumofin and I. Goldberg. Privacy-preserving queries over relational databases. In *PETS'10: Proceedings of the 10th Privacy Enhancing Technologies Symposium*, Berlin, 2010.

[29] F. Olumofin and I. Goldberg. Revisiting the computational practicality of private information retrieval. In *FC'11: Fifteenth International Conference on Financial Cryptography and Data Security*, St. Lucia, 2011.

[30] F. Olumofin, P. K. Tysowski, I. Goldberg, and U. Hengartner. Achieving efficient query privacy for location based services. In *PETS'10: Proceedings of the 10th Privacy Enhancing Technologies Symposium*, Berlin, 2010.

[31] Ookla. Netindex.com source data (Canada and US), Accessed July 2010. `http://www.netindex.com/source-data/`.

[32] S. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):660 –674, May 1991.

[33] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24:513–523, August 1988.

[34] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[35] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In *PET'02*, pages 41–53, 2003.

[36] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, Inc., New York, NY, USA, 5th edition, 2005.

[37] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Syst. J.*, 40(3):683–695,

2001.
[38] K. Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval*, pages 132–142. Taylor Graham Publishing, London, UK, UK, 1988.
[39] P. Wang, H. Wang, and J. Pieprzyk. Secure coprocessor-based private information retrieval without periodical preprocessing. In *AISC '10*, pages 5–11, 2010.
[40] S. Wang, D. Agrawal, and A. El Abbadi. Generalizing PIR for practical private retrieval of public data. In *DBSec'10*, pages 1–16, Rome, Italy, 2010.
[41] P. Williams and R. Sion. Usable PIR. In *NDSS'08: Proceedings of the 16th Annual Network and Distributed System Security Symposium*, 2008.

A Client calibration

Before a user runs the first query, the client needs to be calibrated. Calibration equips the client with some metadata for making correct estimates about the cost of executing a query. Some of the information in this metadata are the average duration for query transformation (t_{tr}), the average duration for retrieving a data item from a bucket (t_{PIR}), the PHF f that was generated when the data was indexed, statistical information about the distribution of data in the database, and a small table of information (T_B) about buckets; their numbers, beginning offsets and lengths in bytes. Some of the information might need to be refreshed periodically (e.g., statistical information on data distribution).