Practical Private Information Retrieval

by

Femi George Olumofin

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In recent years, the subject of online privacy has been attracting much interest, especially as more Internet users than ever are beginning to care about the privacy of their online activities. Privacy concerns are even prompting legislators in some countries to demand from service providers a more privacy-friendly Internet experience for their citizens. These are welcomed developments and in stark contrast to the practice of Internet censorship and surveillance that legislators in some nations have been known to promote. The development of Internet systems that are able to protect user privacy requires private information retrieval (PIR) schemes that are practical, because no other efficient techniques exist for preserving the confidentiality of the retrieval requests and responses of a user from an Internet system holding unencrypted data. This thesis studies how PIR schemes can be made more relevant and practical for the development of systems that are protective of users' privacy.

Private information retrieval schemes are cryptographic constructions for retrieving data from a database, without the database (or database administrator) being able to learn any information about the content of the query. PIR can be applied to preserve the confidentiality of queries to online data sources in many domains, such as online patents, real-time stock quotes, Internet domain names, location-based services, online behavioural profiling and advertising, search engines, and so on. Typically, the database consists of r blocks, and the user query is an index i between 1 and r. The client first encodes the index i into a PIR query, and then forwards it to the database. The database subsequently performs some computations linear in r, and returns the result to the client. The client finally decodes the database response to obtain the block at index i. The main parameters of interest are the number of bits communicated in the interaction between the client and the database, and the amount of computation — usually server computation.

In this thesis, we study private information retrieval and obtain results that seek to make PIR more relevant in practice than all previous treatments of the subject in the literature, which have been mostly theoretical. We also show that PIR is the most computationally efficient known technique for providing access privacy under realistic computation powers and network bandwidths. Our result covers all currently known varieties of PIR schemes. We provide a more detailed summary of our contributions below:

- Our first result addresses an existing question regarding the computational practicality of private information retrieval schemes. We show that, unlike previously argued, recent lattice-based computational PIR schemes and multi-server information-theoretic PIR schemes are much more computationally efficient than a trivial transfer

of the entire PIR database from the server to the client (i.e., trivial download). Our result shows the end-to-end response times of these schemes are *one to three orders of magnitude* (10–1000 times) smaller than the trivial download of the database for realistic computation powers and network bandwidths. This result extends and clarifies the well-known result of Sion and Carbunar on the computational practicality of PIR.

- Our second result is a novel approach for preserving the privacy of sensitive constants in an SQL query, which improves substantially upon the earlier work. Specifically, we provide an expressive data access model of SQL atop of the existing rudimentary index- and keyword-based data access models of PIR. The expressive SQL-based model developed results in between 7 and 480 times improvement in query throughput than previous work.

- We then provide a PIR-based approach for preserving access privacy over large databases. Unlike previously published access privacy approaches, we explore new ideas about privacy-preserving constraint-based query transformations, offline data classification, and privacy-preserving queries to index structures much smaller than the databases. This work addresses an important open problem about how real systems can systematically apply existing PIR schemes for querying large databases.

- In terms of applications, we apply PIR to solve user privacy problem in the domains of patent database query and location-based services, user and database privacy problems in the domain of the online sales of digital goods, and a scalability problem for the Tor anonymous communication network.

- We develop practical tools for most of our techniques, which can be useful for adding PIR support to existing and new Internet system designs.

Supervisor:
Ian Goldberg

Examiners:
Urs Hengartner, Doug Stinson, Alfred Menezes, and Ahmad-Reza Sadeghi

Acknowledgements

This thesis is a product of three years of work under the tutelage of my supervisor Ian Goldberg. He graciously accepted me as a transfer doctoral student from the University of Manitoba, even when I knew nothing about private information retrieval. Soon after my arrival to Waterloo, he exposed me to relevant literature in the areas of security, privacy, and applied cryptography, which eventually led to my interest in PIR. He listened to all my ideas and guided me in the pursuit of those that form the bulk of this thesis. An Oakland PIR paper [Gol07a] and its Percy++ open source implementation that Ian authored are the foundational work for this thesis. I count it a rare privilege to have worked with this brilliant, vast, highly motivated, kind, and skillful supervisor. Ian bought out my Teaching Assistantship duties to make more time available for my research, provided research assistantship support the moment my major scholarships got all used up, and gave many opportunities for professional development and networking via conference travels. For all these and many more, I am grateful.

I would like to thank the other members of my thesis examining committee for their great feedback amongst others. Thanks to Urs Hengartner for our work on location privacy, Doug Stinson for connecting me with my first community service opportunity, Alfred Menezes for attending and providing feedback to all my PhD Seminar talks, and Ahmad-Reza Sadeghi for coming all the way from Germany to sit as my external.

I have also enjoyed collaborations from some of the brightest people from within and outside Waterloo. In particular, I am grateful to Peter Tysowski, Ryan Henry, Prateek Mittal, Carmela Troncoso, and Nikita Borisov for fruitful collaborations. I have enjoyed encouragement and/or great feedback on draft papers and talks from a number of individuals that has made this thesis much better. They include current CrySP lab students and postdocs Mashael Alsabah, André Carrington, Tariq Elahi, Aleksander Essex, Kevin Henry, Ryan Henry, Atif Khan, Mehrdad Nojoumian, Sarah Pidcock, Rob Smits, Colleen Swanson, Jalaj Upadhyay, Andy Huang, Kevin Bauer, and Sherman Chow; former CrySP students Jeremy Clark, Greg Zaverucha, Qi Xie, Can Tang, Aniket Kate, Wanying Luo, Chris Alexander, Jiang Wu, Atefeh Mashatan, and Joel Reardon; friends and colleagues from the School of Computer Science John Akinyemi, Carol Fung, Jeff Pound, and Dave Loker; people from outside the University Jean-Charles Grégoire, and Angèle Hamel, conference shepherd Meredith Patterson; and the anonymous reviewers of our papers.

I am blessed to have an understanding wife, Favour, who has stuck with me all through my years of graduate school. Thank you for your support, care, patience, and faith in me.

I thank my dad, Folorunso Olumofin, for always encouraging me to press forward in my academic pursuits. I am grateful for the prayers and support of my mom, Esther

Olumofin, and my mother-in-law, Rachel Sani. Words cannot express my appreciation to my brother, Olabode Olumofin, for his surprise support gestures and my other siblings for their goodwill.

I am grateful to Steve and Beth Fleming and the entire body of Koinonia Christian Fellowship, for providing me with a safe place for worship, fellowship, service, and growth. I am also thankful for my spiritual dad, David Oyedepo, for being a blessing and an inspiration over the years.

The funding for this research was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) through a NSERC Postgraduate Scholarship, the University of Waterloo through a President's Graduate Scholarship, the Mprime research network (formerly MITACS) through Research Assistantships, Research In Motion (RIM) through a RIM Graduate Scholarship, and The Tor Project, Inc.

Dedication

This dissertation is dedicated to Favour, Grace, Faith, and Joy.

Table of Contents

List of Tables

List of Figures

List of Queries

Chapter 1

Introduction

1.1 Preamble

The advent of Internet-accessible data sources is revolutionizing the way service providers collect and use personal information about their users. The cleartext nature of queries submitted to these data sources greatly simplifies how these providers capture information about the interests, preferences, behaviours, and lifestyles of their users. While service providers embrace this surreptitious profiling ability as a way of improving the effectiveness and profitability of their online advertising efforts [Sol01], it provides new opportunities to embarrass, disrepute, or cause financial loss for the users so targeted and profiled.

This thesis deals with the problem of protecting sensitive information that may be contained in the queries that users submit to online databases. Such databases are found in several application domains, including location-based services, online behavioural advertising, search engines, online patent searches, real-time stock quotes, Internet domain registration and so on [Aso04, GKK+08, Jue01, SJ05]. We consider queries expressed in various formats, such as the indices or server's memory addresses of the data of interest to the user, search keywords, or structured queries expressed in some language, such as SQL. The main challenge is how to keep the sensitive information contained in the queries confidential, even without undermining the database server's ability to return correct responses. In other words, we are interested in preserving the user's *access privacy* — keeping both the queries and the responses to queries confidential from the database server and from other third parties. We differentiate access privacy from *anonymity*. The latter is about keeping the identity of a user from being disclosed. It usually involves a form of obfuscation of the link between a query and a user's digital identity, such as an IP address. Ensuring

a user remains anonymous does not guarantee access privacy for her queries. The mere knowledge of a query's content (even without a link to any user) is often enough to violate access privacy. For instance, a search through a patent database is required for every new patent to establish originality. An inventor or the filling attorney might hide their identity behind an anonymous overlay network like Tor, but that does not prevent a curious or malicious database administrator from learning the keywords used for the search (i.e., the query content). A malicious patent database administrator can preemptively file a patent with those keywords even before the search is complete.

The protection of the content of queries from online databases is important to the future of digital privacy on the Internet. The subject of online privacy has been attracting much interest in recent years, especially as more users are beginning to care about their privacy. In the UK for example, some citizens initially boycotted the 2011 UK census because of privacy worries.[1] This increasing privacy awareness is even prompting legislators in some countries to demand from service providers a more privacy-friendly Internet experience for their citizens. In Canada, there was a recent investigation of whether Facebook is breaching Canada's privacy law, with respect to how they share personal information of Canadian Facebook users.[2] In the U.S., the Energy and Commerce Joint Subcommittee of the U.S. House of Representatives held a joint hearing on the implications of location-based services on the privacy of consumers.[3] These are welcomed developments and in stark contrast to the practice of Internet censorship and surveillance that legislators in some nations have been known to promote. Thus, the problem addressed by this thesis can only grow in relevance in the future.

We focus on the technique of private information retrieval (PIR) [CGKS95] to realize our goal of preserving access privacy for Internet users. PIR provides a means for retrieving data from a database without the database (or database administrator) being able to learn any information about which particular item was retrieved. A trivial solution for the PIR problem is to transfer the entire database to the client, who then retrieves the items of interest from the downloaded database. Although the trivial solution offers perfect privacy protection, the communication overhead is impractical for large databases. PIR schemes are therefore required to have *sublinear* communication complexity.

PIR schemes are classified in terms of their privacy guarantees, the number of servers required for the protection they provide, and whether or not they require specialized hardware. The privacy guarantee of PIR schemes is either information theoretic, computa-

[1] `http://www.dw-world.de/dw/article/0,,15060513,00.html`

[2] `http://www.priv.gc.ca/media/nr-c/2009/nr-c_090827_e.cfm`

[3] `http://democrats.energycommerce.house.gov/index.php?q=hearing/the-collection-and-use-of-location-information-for-commercial-purposes`

tional, or a mix of both. *Information-theoretic PIR* (ITPIR) schemes guarantee query privacy irrespective of the computational capabilities of the database servers answering the user's query. In introducing the notion of PIR in 1995, Chor et al. [CGKS95] proved that the trivial scheme is optimal for information-theoretic privacy protection with a single server [CGKS95,CKGS98]. In other words, any ITPIR with sublinear communication complexity necessarily requires two or more non-colluding servers to participate in the query. Thus, all ITPIR schemes are *multi-server* PIR schemes; that is, the user needs to query multiple servers in parallel and combine their responses to obtain the result for her query. *Computational PIR* (CPIR) schemes, on the other hand, assume a computationally limited database server incapable of breaking a hard computational problem, such as the difficulty of factoring large integers. Therefore, all existing *single-server* PIR schemes are CPIR schemes. The non-collusion requirement of ITPIR is removed for CPIR, but at some cost to efficiency. Hardware-assisted PIR schemes require the support of a secure coprocessor (SC) at the database server host, which serves as the client proxy on the server hardware. The memory and computing space of the SC is isolated and protected from those of the host. We note that standard Trusted Platform Modules (TPMs) can be used to reduce the amount of computation and the number of rounds for interactive cryptographic protocols [GT08, TV09], including PIR schemes. Nevertheless, this thesis focus mainly on PIR schemes that do not require special hardware assistance.

Every PIR scheme consists of three basic algorithms: query generation, response encoding, and response decoding. For a given n-bit database X, organized into r b-bit blocks, a user intending to hide his access pattern to database block X_i uses the query generation algorithm to encode the input index i before sending it to the database. The database then uses the response encoding algorithm to combine each database entry X_j, $j \in \{1, ..., r\}$ with the query and returns an encoded response to the user. Finally, the user decodes the response received using the response decoding algorithm. In the multi-server information-theoretic setting where we assume two or more non-colluding database replicas, the user must interact with multiple replicas in a manner similar to the above. The combined computation cost of the client-side algorithms of query generation and response decoding is generally much cheaper than the server-side algorithm of response encoding. A PIR scheme is *correct* if it always returns the correct block X_i, *private* if it leaks no information to the database about i and X_i, and *non-trivial* if its communication complexity is sublinear in n [Cre06]. *Symmetric* PIR (SPIR) schemes have the additional property that the user only learns the block X_i, and learns no information about other database blocks X_j, for all $j \neq i$. In other words, SPIR schemes provide database privacy in addition to user privacy.

1.2 Open Problems

Most of the research efforts in PIR have been directed at lowering the number of bits transferred between the client and the server(s) — lowering the communication complexity bounds — because network bandwidth is considered to be the most expensive resource that should be minimized [Amb97, BS03, CGKS95, CKGS98, Efr09, GR05, KO97, Lip05, Yek08]. Some other bodies of work [Aso04, AF02, BS03, BS07, GIKM98a, Gol07a] have additionally addressed such problems as reducing server-side computational overheads via amortization and preprocessing, and improving query robustness, amongst others.

Given the importance of the PIR construction, and in spite of the progress made towards minimizing the communication in existing PIR schemes for nearly two decades, it has been difficult to apply PIR to solve real-world privacy problems. It has even been argued [SC07] that no single-server computational PIR scheme is as efficient as the trivial PIR scheme. While a few recent efforts have focused on the important problem of reducing PIR's computational costs, we argue in this thesis that other shortcomings persist with the current state-of-the-art approach to research and development aimed at making PIR more practical. Consequently, this thesis adopts a more detailed approach to address the main challenges hindering existing PIR schemes from being practical. We look beyond the existing literature and propose an extended definition of *practical private information retrieval*, which additionally considers other dimensions to the problem of making PIR more practical.

Definition 1 *A private information retrieval scheme is* practical *iff it satisfies the following five properties:*

Property 1 Reasonable communication. *The number of bits transferred between the client and the server must be sublinear in the database size n [CGKS95]. Most existing PIR schemes have practical communication complexity that is sublinear in the database size.*

Property 2 Fast computation. *It has been shown that an amount of computation linear in the database size on the server side is unavoidable for a typical PIR scheme [BIM00, BIM04], since each query for block X_i must necessarily process all database blocks X_j, $j \in \{1, ..., r\}$. Constructions that use cheap operations (such as XORs) [CGKS95, Gol07a] instead of modular multiplications, linear algebra [AMG07, CAMG08] instead of number theory, offline pre-computation [BIM00, BIM04] and amortization of computation costs over multiple queries [BIM00, IKOS04] offer improvements. Besides,*

the increasing pervasiveness of resource-constrained devices, which some see as the computing platform of the future [ALD08], and the growing awareness of users for privacy justifies the need for computationally efficient PIR schemes that are practical on such devices. In particular, resource-constrained hardware such as smartphones have limited processing power, memory, and wireless bandwidth. The model of PIR, where the majority of the computation is performed on the server side, and where the communication is minimized, fits the architecture of these devices very well. Therefore, any progress made in making PIR more practical is particularly advantageous for smartphone applications.

Property 3 Flexible data access model. *It is much more convenient to access data using expressive access models, such as with SQL and keywords over tree-based data structures, than with indices over single bits or blocks of bits. These advanced data models enable the integration of PIR schemes with today's programming languages and databases.*

Property 4 Transparent tradeoffs between privacy and efficiency. *The ability to perform tradeoffs between privacy and efficiency is an added convenience for users. The users should not be forced to settle for the current "all-or-nothing" approach to privacy with PIR schemes. There should be some middle ground for obtaining reasonable privacy with the available computation and communication capability.*

Property 5 Model solutions to real-world privacy problems. *PIR schemes should be readily applicable to solving real-world privacy problems. Developers of privacy-preserving systems should be provided access to APIs and model applications that use the PIR primitive.*

Thesis Statement. It is possible to build systems for protecting access privacy based on private information retrieval schemes that satisfy Definition 1.

1.3 Contributions

Our contributions cover most of the properties that we hypothesize could make PIR more practical (Definition 1). We illustrate with a concept map of the thesis in Figure 1.1. The main topic is at the root of the tree, and the main areas as described in our five properties are at the second level. We show how our main contributions relate to each of the main areas. We note that there has been extensive research on reducing the communication

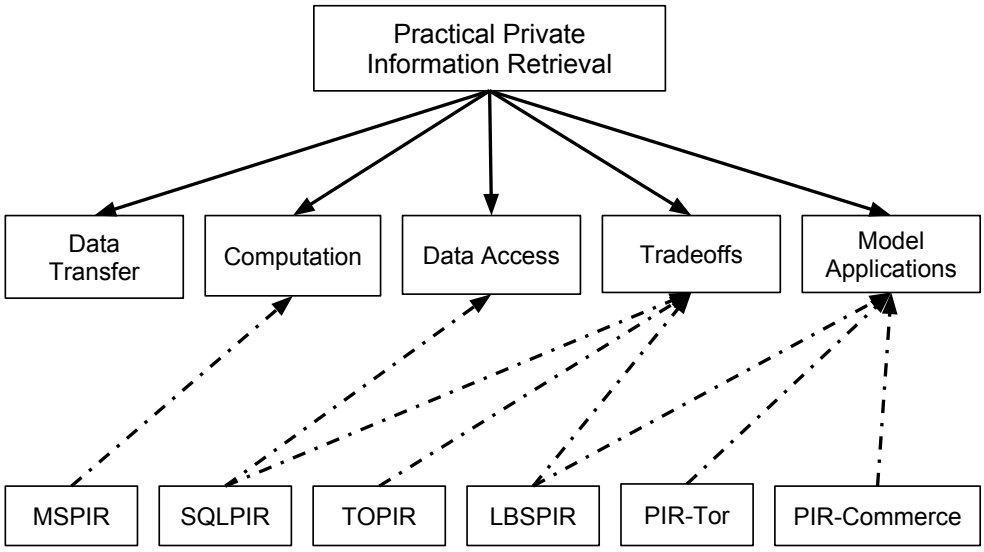


Figure 1.1: A concept map for this thesis showing contributions to the literature.

complexity of PIR schemes (property 1); our contributions cover the remaining properties. We made contributions to research and development in the areas of the computational practicality of non-trivial PIR schemes (MSPIR) [OG11] (property 2), a model of data access for PIR with SQL (SQLPIR) [OG10b] (property 3), a user-friendly approach to trade off privacy for computational performance for PIR schemes (TOPIR) [OG10a] (property 4), and application of the PIR primitive to solve the location privacy problem for resource-constrained devices (LBSPIR) [OTGH10], addressing a scalability problem for the Tor network (PIR-Tor) [MOT+11], and privacy-preserving electronic commerce to enable the sale of digital goods (PIR-Commerce) [HOG11] (property 5).

We summarize the main contributions of this thesis as follows:

- **MSPIR: Revisiting the computational practicality of private information retrieval.** We reexamine the computational practicality of PIR following the earlier work by Sion and Carbunar [SC07], which concluded that no existing PIR construction is as efficient as the server transferring its entire database to the client (i.e., trivial PIR). While often cited as evidence that PIR is impractical, Sion and Carbunar did not examine multi-server information-theoretic PIR schemes, and single-server PIR schemes that do not rely heavily on number theory, such as lattice-based schemes. We analytically and experimentally studied the single-server lattice-based PIR scheme by Aguilar-Melchor and Gaborit [AMG07], which has recently been introduced, as

well as two multi-server information-theoretic PIR schemes by Chor et al. [CGKS95] and by Goldberg [Gol07a]. We find the end-to-end response times of these schemes to be *one to three orders of magnitude* (10–1000 times) smaller than the trivial scheme for realistic computation powers and network bandwidths. Specifically, our results suggest (i) single-server PIR schemes based on modular multiplication cannot be faster than trivially downloading the entire database, (ii) single-server PIR schemes based on lattices are an order of magnitude (10 times) faster, (iii) multi-server PIR schemes are two to three orders of magnitude (100–1000 times) faster. Our results extend and clarify the conclusions of Sion and Carbunar for multi-server PIR schemes and single-server PIR schemes that do not rely heavily on number theory. We believe that many real-world situations that require privacy protection can obtain some insight from our work in deciding whether to use existing PIR schemes or the trivial download solution, based on their computing and networking constraints (property 2 of Definition 1).

- **SQLPIR: Private information retrieval from relational databases.** We present a novel approach that extends the rudimentary index-based retrieval with PIR to SQL-based retrieval. The approach shows how to retrieve data from a relational database while keeping the sensitive information in the predicates of the retrieval SQL query from being leaked to the PIR server and relational database system. Experimental results and microbenchmarking tests show our approach incurs reasonable storage overhead for the added privacy benefit, leverages database optimization opportunities to yield fewer PIR requests, and performs between 7 and 480 times faster than previous work.

  We believe this work addresses an important obstacle to deploying successful PIR-based systems, which is the development of a more general data access model for PIR from a relational database (property 3 of Definition 1). Prior PIR queries are specified in terms of database addresses [BS07, CGKS95, KO97] or by textual keywords [CGN97], and are therefore not suitable for retrieval from relational databases. We embed PIR schemes into the well-established context and organization of relational database systems, thereby making SQL relevant to PIR. This work also shows how to apply PIR schemes in realistic relational database scenarios to achieve both efficiency and query expressiveness. Since relational databases and SQL are the most influential of all database models and query languages, we believe that many realistic systems needing query privacy protection will find this result quite useful.

- **TOPIR: Large database access privacy and tradeoffs.** PIR schemes are typically designed to require PIR servers to compute on every item in the PIR database.

Doing this prevents the server from identifying the particular data item that is of interest to the user. However, such linear computational cost is impractical for large databases of several hundreds of gigabytes. This is irrespective of the negligibleness of the per-block computation cost, since the cost of disk/memory access alone (i.e., without computation) will overwhelm the server's capacity for sufficiently large databases. We develop an approach for large database access privacy to overcome this limitation of current PIR schemes. Our approach allows users to query a large database by statically specifying or dynamically defining database portions possibly with highly diverse domains of values, thereby minimizing information leakage about the data items sought by user queries. Our approach requires minimal user intervention and allows users to accompany their queries with a description of their privacy preferences and performance tolerances, and transform these inputs to an alternative query that can satisfy user constraints when run. Among the techniques explored are offline data classification, constrained privacy-preserving query transformations, and PIR over index structures much smaller than the databases. 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 in making access privacy obtainable on the Internet. Our results show how privacy and computational efficiency interplay in a large database query scenario, based on user-specified preferences (property 4 of Definition 1). It also shows how to adapt existing PIR schemes with the flexibility needed to meet the varying privacy needs of users. For example, some users may be comfortable with revealing the *shape* or textual content of their queries to the database, but not the *sensitive constants* in their queries; on the other hand, some other users may prefer better privacy protection that hides both the shape and constants of their queries.

We also demonstrate several results applying PIR to real-world problems to preserve access privacy (mostly property 5 of Definition 1, but others as well):

- **LBSPIR: Private information retrieval for location-based services (LBS).** The increasing popularity of mobile devices with positioning technology such as GPS or cell tower triangulation is fueling the growth of location-based services (LBS). Some examples of commercial LBS are Foursquare, Places, SCVNGR, and Loopt. We consider the problem of providing location privacy as mobile smartphone users search for nearby points of interest (POIs) from a location-based service in a way that preserves the privacy of the users' locations. We propose a novel PIR-based hybrid LBS technique [OTGH10] that achieves a good compromise between location

privacy and computational efficiency for resource-constrained devices, such as smartphones. Specifically, our approach uses a variable-sized cloaking region, which is a representation of a geographical city, province, or state, for greater location privacy than the traditional approach of a single cloaking region, while at the same time decreasing wireless data traffic usage from an amount proportional to the size of the cloaking region to a small fraction of that. Our proposal does not require the use of a trusted third-party component, and allows the user to choose various levels of privacy dynamically. We evaluated our approach with a proof-of-concept implementation over a commercial-grade database of points of interest. We also measured the performance of our query technique on a smartphone and wireless network. The results show that users can achieve a good compromise between privacy and computational efficiency with our technique, unlike all other existing LBS proposals for nearby point of interest search.

- **PIR-Tor: Scalable anonymous communication using private information retrieval.** Tor is a worldwide overlay network of servers, or *nodes*, that allows journalists, law enforcement workers, humanitarian workers, privacy-concerned citizens, and others to communicate over the Internet without disclosing their IP addresses. There are currently about 2000 nodes and 250,000 users [Tor11a, DMS04a]. The current Tor network requires all users to maintain an up-to-date list of all available nodes in the network before constructing a circuit for anonymous communication. As a result, Tor does not scale well. Current proposals for making Tor scale better rely on a peer-to-peer (P2P) approach, which has unfortunately provided new avenues to compromise anonymity. We propose PIR-Tor as a scalable architecture for the Tor network, whereby users no longer need to maintain a list of available nodes in order to construct circuits. In PIR-Tor, a user obtains information about only a few routers using PIR techniques, thereby preventing compromised directory servers from learning the user's choices and reducing the overall communication by at least an order of magnitude. Experimental results show that reasonable parameters of PIR-Tor maintain a level of security that is equivalent to that of the current Tor network, while allowing Tor to scale by two orders of magnitude.

  To the best of our knowledge, this is the first time PIR is applied to preserve access privacy for a real-world system. PIR minimizes bandwidth usage in PIR-Tor without compromising privacy guarantees.

- **PIR-Commerce: Practical private information retrieval for electronic commerce.** We provide an extension of Goldberg's multi-server information theoretic PIR scheme [Gol07a] to support privacy-preserving online sales of digital goods. Our

motivating examples are a pay-per-download music store [AIR01] where users must pay for each song they download, a pay-per-retrieval DNA database [CDN09], a stock-information database [GIKM98b], an e-book store, or a patent database [Aso04]. We first extend Goldberg's PIR scheme to a *symmetric* private information retrieval (SPIR) scheme to prevent a dishonest user from obtaining additional information beyond the database item sought, such as the record at index $j \neq i$, or the exclusive-or of some subset of records in the database [GIKM98b]. Afterwards, we extend the SPIR construction to a priced symmetric private information retrieval (PSPIR) scheme, which supports a tiered pricing model in which users purchase records from the PIR database in a privacy-preserving manner, and according to their price tier; for example, non-members may pay at a regular price, while members pay a discounted rate to purchase the same records. Our approach maintains query privacy, such that the database servers do not learn any information about the user's price tier, the index of the record purchased, the price paid, or the user's remaining balance. Via a simple extension, our construction allows the database to support access control at record-level granularity, which we believe to be of independent theoretical interest. We also provide a practical implementation and evaluate the performance of our constructions empirically.

Current priced and access-control-capable constructions lack the right combination of features necessary for their deployment in a practical setting. Some of these features are tiered pricing, sublinear communication complexity, access control, and availability of practical implementations. Our schemes improve upon a number of related constructions to PIR in the literature.

1.4 Thesis Organization

Chapter 2 begins with some motivations that users, service providers and privacy legislators have for using PIR. We then give some motivating application examples requiring practical PIR schemes, and describe various types of PIR schemes and establish their relationship with other cryptographic primitives. Chapter 3 discusses previous work related to the research described in this thesis. In particular, we cover topics like the computational practicality of PIR schemes, the models for data access with PIR, attempts at preserving access privacy over large databases, and application areas where attempts have been made to apply PIR for pseudonymous communication, location-based services, and electronic commerce. We cover some previous research activities in each area and discuss our contributions to research and development.

Chapter 4 describes how we revisited the computational practicality of PIR, where our results show that some non-trivial PIR schemes are more efficient than trivially downloading the PIR database, while Chapter 5 presents our model of data access for PIR with SQL. Afterwards, Chapter 6 describes our approach for preserving access privacy over large databases using PIR.

The next two chapters cover how we apply PIR to three different problem domains. First, we apply PIR to achieve location privacy when searching for nearby points of interest in Chapter 7. Then, we utilize PIR in Chapter 8 to solve a scalability problem in the Tor network, and we extend Goldberg's PIR construction to support electronic commerce. We end the thesis in Chapter 9 with conclusions and some more opportunities for further research.

Chapter 2

Preliminaries

2.1 Motivation

Research into how to make PIR more practical offers an attractive value proposition and benefits to users of privacy-preserving systems, service providers, and privacy legislators.

Users are increasingly aware of the problem of privacy and the need to maintain privacy in their online activities. The growing awareness is partly due to increased dependence on the Internet for performing daily activities — including online banking, microblogging, and social networking — and partly because of the rising trend of online privacy invasion. Privacy-conscious users will accept a service built on PIR for query privacy protection because no currently deployed security or privacy mechanism offers the needed protection; they will likely be willing to trade off query performance for query privacy and possibly even pay to subscribe to such a service.

Similarly, service providers may adopt such a system because of its potential for revenue generation through subscriptions and ad displays. As more Internet users value privacy, most online businesses would be motivated to embrace privacy-preserving technologies that can improve their competitiveness to win this growing user population. Since the protection of a user's identity is not a problem addressed by PIR, our proposal will not disable existing service models relying on service providers being able to identify a user for the purpose of targeted ads. In other words, protection of query privacy will provide additional revenue generation opportunities for these service providers, while still allowing for the utilization of information collected through other means to send targeted ads to the users. Thus, users and service providers have plausible incentives to use a PIR-based solution for maintaining

query privacy. We even anticipate that future targeted online advertising technologies would be enabled using techniques similar to PIR [GCF11, Jue01, TNB+10].

In addition, the very existence of a practical privacy-preserving database query technique could be enough to persuade privacy legislators that it is reasonable to demand that certain sorts of databases enforce privacy policies, since it is possible to deploy these techniques without severely limiting the utility of such databases.

2.2 Applications

The following interesting basic scenario motivates an application of PIR for user privacy protection [SJ05]:

> Picture the following scenario. Alice is looking for gold in California. What Alice does is look for a place with a little gold and follow the trace. Now, Alice wants to find gold in a place where no mining patent has been awarded, but many patents have been awarded in California during the gold rush. What Alice does is to walk around California with a GPS and a notebook computer. Whenever she finds a trace of gold she follows it querying if any patent has been awarded in that location. If she finds a trace of gold in a piece of land with no issued patent she can request the patent and start mining for gold.
>
> The problem is that she is worried that Bob's Mining Patents Inc., the service she queries the patents from, might cheat on her. Because Bob knows she is looking for gold in California (Alice said so when signing up for Bob's service), he knows that, if she queries from some location, then there is gold there. So, if she queries a location and there is no patent awarded, Bob may run to the patent office and get the mining patent for that location.
>
> Depending on privacy and economic constraints, a few solutions come to mind. Alice might buy from Bob the whole database for California. Alice then can make all the queries to her own database, and Bob will never find out where Alice is looking for gold. But this might be very expensive, because Bob charges per query; what he charges for the whole database will probably be more than what Alice is willing to pay. Alice can also, with each query, perform a collection of fake queries so that Bob can't figure out which is the real query (this leaks information unless she queries the whole database!), but that still makes Alice pay for more queries than she would like.

Private information retrieval schemes have applications in several problem domains including:

1. *Patent databases*: The filing process of a new invention necessarily requires the inventor to search the patent database to establish that no previous patent has a significant overlap with her invention. She would like to perform the search in a manner that does not leave her search terms on the query log of the patent database. Current patent database systems allow a curious or malicious database administrator to infer the user's interest either directly from the query log or in real time by following the query as it executes.

2. *Real-time stock quotes*: An investor interested in a particular stock would often need to monitor the stock price and market volume to know when to make a purchase decision. The investor might prefer a monitoring means that keeps her stock of interest confidential by not revealing any information about the stock.

3. *DNA databases*: Consider a pharmaceutical organization interested in purchasing information about particular genome sequences from a public DNA database [CDN10]. They may need the information to complete the manufacture of a new medication that is a closely kept trade secret.

4. *Registration of Internet domain names*: The current process for the registration of Internet domain names requires a user to first disclose the name for the new domain to an Internet domain registrar. However, the registrar could subsequently use this inside information to preemptively register the new domain and thereby deprive the user of the registration privilege for that domain. This practice is known as *front running* [ICA08]. The registrar is motivated to engage in front running because of the revenue to be derived from reselling the domain at an inflated price, and from placing ads on the domain's landing page. With PIR, users can search for new domain names without any fear of divulging the domain names to the registrar.

5. *Online behavioural analysis for ad networks*: Consider the challenge faced by ad networks when they serve ads to users based on online behavioural data aggregated about users from multiple websites. For example, if a user frequently visits some sports websites, then the user is targeted with ads about sports products. In a privacy-preserving situation, it is important to be able to target users with ads without the ad network being aware of the interests of or ads that are served to the user. With PIR, the user's ad client could privately retrieve ads based on the profiled online

behavioural data that is cached locally. The billing of content providers for ads displayed to users can also be performed in a privacy-preserving manner, such that users' interests are unknown to the ad network (see [BRT11,GCF11,GF10,Jue01,TNB+10]).

In all of the above application scenarios, the problem is that users find it unacceptable to disclose the sensitive information in the queries they send to the servers; they demand confidentiality for their queries, both from third parties and from the server holding the data of interest. Certainly, the abundance of potential applications for PIR is a justification for finding a means to make existing and new constructions more practical.

2.3 Single-server PIR Schemes

Several single-server PIR schemes followed the first single-database scheme by Kushilevitz and Ostrovsky [KO97]. Each of these later schemes made some intractability assumption based on the cryptographic primitive used. These include schemes based on group-homomorphic encryption [Ste98], length-flexible additive homomorphic public-key cryptosystems [Lip05], Φ-Hiding Assumption (ΦHA) trapdoor predicates (the hardness of deciding if a small prime is a factor of the totient of a composite integer of unknown factorization) [CMS99], and more recently, a lattice-based scheme by Aguilar-Melchor and Gaborit [AMG07], which is based on the hardness of a differential hidden lattice problem.

For surveys on single-server PIR schemes, the reader is referred to [AMD06, OS07b]. In spite of the convenience of fielding a single PIR database, the main problem of most single-server PIR schemes is their costly computations (modular multiplications or exponentiations), the number of which is linear in the database size. The state of the art in terms of the performance measure of computational complexity is the lattice-based scheme by Aguilar-Melchor and Gaborit [AMG07]. The authors introduced and based the security of the scheme on the differential hidden lattice problem, which they show is related to NP-complete coding theory problems [Wie06]. There are two main contributions of this work by Aguilar-Melchor and Gaborit. First, it shows that their scheme exhibits one order of magnitude speedup by using Graphics Processing Units (GPUs) instead of CPUs to do the bulk of the computation, and claims that other schemes will see the same speedup. Second, it shows that in GPU-based scenarios, linear algebra based single-server PIR schemes can be more efficient than trivial download; this attempts to dispel the question posed by Sion and Carbunar [SC07] with respect to the practicality of single-server PIR schemes. In comparison to other single-server PIR schemes [GR05, Lip05], this scheme gives a much better server-side processing rate both on CPUs and GPUs. A roundtrip response rate of

94 s/GB was measured in [OG11], which is far more practical than the 3.3×10^5 s/GB rate from the Kushilevitz and Ostrovsky scheme [KO97]. Aguilar-Melchor et al. [AMCG+08] demonstrated a further 10 times speedup in the server-side processing rate by running the PIR scheme on commodity GPUs, instead of on CPUs.

We present a note of caution, however, that although this PIR scheme resists known lattice-based attacks, it is still relatively new, and its security is not as well understood as those of the PIR schemes based on number theory.

2.4 Multi-server PIR Schemes

Gasarch [Gas04] provides an older survey of PIR schemes. A more recent primer on PIR by Beimel is also available [Bei08]. We provide an overview of two multi-server information-theoretic PIR schemes, from Chor et al. [CGKS95] and from Goldberg [Gol07a]. The Chor et al. [CGKS95] scheme is comparatively simpler, being the first PIR protocol invented. The Goldberg [Gol07a] scheme is slightly more elaborate and has available source code, which is useful for performance measurements [OG10b, OG11, OTGH10]. The implementation of [Gol07a], known as Percy++ [Gol07b], is an open-source project on SourceForge.

2.4.1 Chor et al. Scheme [CGKS95]

This description is based on the simple $O(\sqrt{n})$ protocol by Chor et al. The setup consists of ℓ servers each holding a copy of the database. The database D in each server is treated as an $r \times b$ matrix of bits (i.e., elements of $GF(2)$), where the k^{th} row of D is the k^{th} block of the database. During query generation, the client interested in block i of the database picks ℓ random bitstrings $\rho_1, \ldots, \rho_\ell$, each of length r, such that $\rho_1 \oplus \cdots \oplus \rho_\ell = e_i$, where e_i is the string of length r which is 0 everywhere except at position i, where it is 1. The client sends ρ_j to server j for each j.

During response encoding each server j computes $R_j = \rho_j \cdot D$, which is the XOR of those blocks k in the database for which the k^{th} bit of ρ_j is 1, and sends R_j back to the client.

Finally, during response decoding, the client computes $R_1 \oplus \cdots \oplus R_\ell = (\rho_1 \oplus \cdots \oplus \rho_\ell) \cdot D = e_i \cdot D$, which is the i^{th} block of the database. In the above, we ignore ℓ in the $O(\sqrt{n})$, because it is small.

2.4.2 Goldberg Scheme [Gol07a]

Goldberg's multi-server PIR scheme is similar, but more complex than the Chor et al. scheme. The similarity lies in its use of simple XOR operations to accomplish most of its server-side computations. However, it uses Shamir secret sharing [Sha79] to split the user's query vector e_i into ℓ shares which are then transmitted to the servers. The server database D is treated as an $r \times s$ matrix of w-bit words (i.e., elements of $GF(2^w)$), where again r is the number of blocks and s is the number of w-bit words per block. In addition, the elements of e_i, ρ_j, and R_j are elements of $GF(2^w)$, instead of single bits. These changes are necessary because the protocol addresses query robustness for Byzantine servers that may respond incorrectly or not respond at all. For simplicity, this thesis will only discuss honest servers, which respond correctly. We choose $w = 8$ to simplify the cost of computations; in $GF(2^8)$, elements are bytes, additions are XOR operations, and multiplications are lookup operations into a 64 KB table; all of these operations are very fast. These are the choices made by the open-source implementation of this protocol [Gol07b].

During query generation, the client encodes a query for database block i by first uniformly choosing ℓ random distinct non-zero indices $\alpha_1, \ldots, \alpha_\ell$ from $GF(2^8)$. Next, the client chooses r polynomials of degree t, one for each block in D. The coefficients of the non-constant terms for polynomial f_j are random elements of $GF(2^8)$, while those for the constant terms should be 1 if $i = j$ and 0 otherwise. Afterwards, the client hands out to each server j a vector ρ_j formed from evaluating all r polynomials at α_j; that is, $\rho_j = [f_1(\alpha_j), \ldots, f_r(\alpha_j)]$. (Note that each $f_k(\alpha_j)$ is an element of $GF(2^8)$ — a single byte.)

Response encoding and decoding: In a manner similar to the Chor et al. scheme, each server computes a response vector $R_j = \rho_j \cdot D$, where each of the s elements of vector $R_j = [r_{j1}, \ldots, r_{js}]$ is also a single byte. The servers send R_j to the client and the client computes the query result using Lagrange interpolation, which also amounts to simple arithmetic in $GF(2^8)$. Evaluating the resulting polynomial at a point $x = 0$ gives the desired database block. In particular, the client recieves $[r_{11}, \ldots, r_{1s}], \ldots, [r_{\ell 1}, \ldots, r_{\ell s}]$ from the ℓ servers. The database block i is computed by evaluating a vector of s unique polynomials, each of degree t, at a point $x = 0$; i.e., $[\phi_1(0), \ldots, \phi_s(0)]$. The client computes each unique polynomial ϕ_i from the ℓ points $(\alpha_1, r_{1i}), (\alpha_2, r_{2i}), \ldots, (\alpha_\ell, r_{\ell i})$ using Lagrange interpolation.

Goldberg's PIR scheme [Gol07a] provides good support for query robustness against colluding servers. It provides a t-private v-Byzantine-robust k-out-of-ℓ scheme for $0 < t < k \leq \ell$ and $v < k - \lfloor\sqrt{kt}\rfloor$. In other words, users submit their queries to at least k out of the ℓ servers, and the system can tolerate up to v servers being Byzantine (i.e., responding incorrectly) without inhibiting the ability of users to retrieve the correct record, and t servers

colluding without compromising users' query privacy. The scheme also optionally supports τ-independence [GIKM98a], a property that prevents the database servers from learning the contents of the database with information-theoretic protection against coalitions of up to τ servers.

2.5 Coprocessor-Assisted PIR Schemes

Coprocessor-assisted PIR utilizes a secure coprocessor (SC) at the server-side host [AKS08, Hen07] to create a computing space that is protected from outside adversaries and the host itself. There are two main approaches for coprocessor-assisted PIR schemes. In earlier schemes [SS01] where the focus was on reducing communication costs, the trusted hardware processes each query by reading every data item in the database and returning the requested item to the user. The server is oblivious of the particular item returned because it has no access to the private memory and computation of the trusted hardware. More recent schemes [IS04, IS05, WDDB06] facilitate faster online queries by performing most of the server-side reading and computations offline. However, depending on the amount of internal memory of the secure coprocessor, the external database requires reshuffling after every m queries ($m \ll n$). These latter schemes all have optimal $O(\log n)$ communication complexity, $O(1)$ online computational complexity and $O(n)$ offline computational complexity.

Asonov et al. [Aso04, AF02] propose a relaxation of the strong privacy requirement of PIR to reduce preprocessing complexity; their PIR also requires the support of a secure coprocessor (SC). This relaxation intends to replace the strong privacy requirement of *no* information about the user query being revealed to a weaker privacy notion of *not much* information about the user query being revealed. In order words, their construction offers repudiation instead of privacy. The user can deny querying a particular block and no one can prove the user wrong, even with the cooperation of the database. They gave the name Repudiative Information Retrieval (RIR) to this form of PIR with relaxed privacy. Their approach, however, reduces computation from $\Theta(n)$ to $\Theta(\sqrt{n})$.

The major drawbacks of these schemes are the requirements for specialized tamperproof hardware and the computationally expensive periodic reshuffling operation [AKS08, IS05]. The only SC scheme in the literature that does not require periodic reshuffling is recently proposed by Wang et al. [WWP10]. Their scheme works in two stages: offline shuffling and online retrieving. In the offline shuffling stage, the database records are doubly encrypted and permuted, while in the online retrieving stage, the SC reads two records from the

shuffled database and writes two records back to it. Reshuffling of the database is altogether avoided using the double encryption and twin-writing operations.

The state of the art for coprocessor-assisted PIR is the single-server PIR scheme by Williams and Sion [WS08], which offers an order of magnitude improvement in computational performance. This particular work describes how to realize single-server PIR using an efficient oblivious RAM protocol and a secure coprocessor. They achieved improvements in the communication and computational complexity bounds of hardware-assisted PIR to $O(\log^2 n)$ per query, given that a small amount of temporary secure storage, on the order of $\Theta(\sqrt{n})$, is available. Lu and Ostrovsky [LO11] recently introduced the concept of multi-server oblivious RAM, which makes a non-collusion assumption like multi-server PIR, but only provides computational privacy (unlike multi-server information theoretic PIR). They improved over existing ORAM protocols by eliminating the usually expensive oblivious sorting step of earlier protocols.

2.6 Relationship to other Cryptographic Primitives

Basic PIR schemes place no restriction on information leaked about other items in the database, which are not of interest to the user. However, an extension of PIR, known as *Symmetric* PIR (SPIR) [MS00], adds that restriction by insisting that a user learns *only* the result of her query. The restriction is crucial in situations where the database privacy is equally of concern.

Single-server SPIR schemes are closely related to several other cryptographic primitives, including the notion of *oblivious transfer* (OT) [NP99, NP01]. In OT, a database (or sender) transmits some of its items to a user (or chooser), in a manner that preserves their mutual privacy. The database has assurance that the user does not learn any information beyond what he or she is entitled to, and the user has assurance that the database is oblivious of his or her choice. A 1-out-of-2 OT scheme allows a database X consisting of two records and a user holding an index $i \in \{0, 1\}$ to run a protocol that results in the user learning the ith record and no information the $(1 - i)$th record, while the database learns nothing about i. Unlike PIR and SPIR, however, OT schemes have no sublinear communication requirements. Brassard et al. [BCR86] considered the more general notion of 1-out-of-n OT, where the database holds n records and the user learns the record at index i, and learns nothing about the remaining $n - 1$ records [OS07b]; the database still learns nothing about i. We can refer to OT and SPIR as generalizations of PIR, but they require extra computational cost. In addition, every 1-out-of-n OT scheme with a sublinear communication complexity is an SPIR scheme.

Also of independent interest is the intimate relationship between multi-server PIR and locally decodable codes (LDCs) [Yek07, Yek10]. LDCs are a class of error-correcting code with highly efficient "sublinear time" decoding procedures. LDCs provide efficient random-access retrieval and high noise resilience for data transmitted over noisy channels or stored on corrupted media. As a result, LDCs have longer codewords and are less efficient than traditional error-correcting codes. A (k, δ, ϵ)-LDC encodes an n-bit database X into a longer N-bit codeword Y, such that even after up to δN bits of the codeword Y are adversarially corrupted, it is still possible to use a randomized decoding algorithm that reads only k bits of Y, to retrieve bit X_i, $i \in \{1, ..., n\}$ with probability at least $1-\epsilon$ [Yek10]. The inefficient Hadamard code has $N = 2^n$ codeword length [Yek11]. For a codeword Y corrupted at $0.1N$ bits, it is possible to recover any bit X with probability 0.8 just by querying $k = 2$ bits of Y. The shortest codeword LDC by Yekhanin [Yek08] is based on a Mersenne prime $M_p = 2^p - 1$. We have $N = exp(n^{1/p})$, for every n, and require $k = 3$ queries. For the largest known M_p,[1] $p = 43\,112\,609$, and we have $N < exp(n^{1/10^8})$. It has been claimed [Yek10] that it is possible to derive all recent information-theoretic multi-server PIR schemes by first constructing LDCs, and then converting the LDCs to PIR schemes. Short LDCs would yield communication-efficient PIR schemes and vice versa. However, such PIR schemes are not efficient in terms of computation. We note, however, that most PIR schemes are constructed from the ground up, without any intermediate LDC construction step.

Private matching and private set intersection schemes [CZ09, FNP04, JL09] consider the problem of computing the intersection of two private sets from two users, such that each user only learns the sets' intersection. These schemes are more closely related to OT and SPIR than PIR because the privacy of both parties is considered. A modified private set intersection scheme, where one of the parties (the database) does not need to learn of the result of the intersection, is related to PIR, though it is much more efficient because it is a tailored (dedicated) scheme. Nevertheless, we can use PIR to solve the private matching problem, and additionally enjoy the flexibility of specifying queries in a more expressive language suitable for retrieving an arbitrary data.

A related problem to PIR is that of privately searching an unencrypted stream of documents [BSW09, OS07a]. In these schemes, the client selects some keywords, and then encrypts them before sending them to a server. The server performs a search using the keywords over a stream of unencrypted documents and returns the list of documents containing the keywords back to the client. The server remains oblivious of which particular document it returns, and the confidentiality of the keywords is preserved. Existing constructions are limited to returning documents that give exact matches on a keyword list,

[1] `http://www.mersenne.org/various/history.php`

or two keyword lists combined with logical "OR" or "AND". These types of queries are much simpler than a PIR-enabled relational database query, which may contain multiple operators — comparison, logical, and so on. In addition, range queries are not presently possible with private stream searching because exact keywords must be specified for the search.

We will consider some other related constructions in the next chapter.

2.7 Relation to Recent PhD Theses

There are a few recent PhD theses that have considered the practicality of cryptographic protocols. Geisler [Gei10], inspired by Yao's millionaires problem [Yao82], proposed a secure comparison protocol (secure two-party computation) based on homomorphic encryption. This work allows Alice and Bob to learn the result of comparing their respective private inputs, while neither learns any information about the input of the other party. A Java implementation of the protocol takes about 0.56 seconds to compare two 16-bit integers and 1.13 seconds to compare two 32-bit integers, both based on 1536-bit RSA for security. Geisler also presented a framework and a tool known as the Virtual Ideal Functionality Framework or VIFF for short. VIFF intends to simplify the building of secure multi-party computation systems and provides a high-level language for writing secure multi-party computation protocols. It works by compiling the high-level specification of a protocol into VIFF API calls, which are subsequently executed by a Python virtual machine. Unlike our contribution that seeks to make PIR more practical, this thesis contributes by simplifying the prototyping of secure multi-party computation protocols.

Also motivated by Yao's millionaires problem [Yao82], Schneider [Sch11] focused on algorithmic techniques to engineering two-party secure function evaluation (SFE) protocols. Two-party SFEs allow Alice and Bob holding respective inputs a and b to jointly compute a function f over their private inputs, without revealing any information other than the result $f(a, b)$ to both of them. The thesis presented results that leveraged TPMs to make the evaluation of SFEs more efficient and to make the communication costs of garbled circuit-based SFEs independent of the size of the evaluated function. In addition, the thesis developed a tool to generate cryptographic protocols such as zero-knowledge proofs from high-level descriptions. Again, this thesis leveraged TPMs to reduce the communication cost of a specific class of SFE and to make SFEs more efficient, which is a different problem from making PIR more practical. We note that we can view PIR as SFE with a constraint that only one of the parties learns the result of the computation. It is

also the case that TPMs can help minimize the computation and communication costs of PIR schemes [Aso04].

We also note that De Cristofaro [Cri11] very recently defended his thesis, which dealt with privacy-preserving ways for two parties to share sensitive information. The thesis, although not yet published, includes the work by De Cristofaro et al. [CLT11], which considers how two parties (user and database) can share information without either of them leaning more information beyond what they are entitled. They presented two protocols and modeled them in terms of simple database queries. The first protocol requires the user to download an encrypted database, from which he or she can selectively decrypt records. They addressed the high communication overhead of this first protocol in their second protocol, which introduces a third party "isolated box" assumed not to be colluding with either the user or database. As a result, the implementation of the second protocol added a small overhead of 10% to non-private MySQL database queries. In comparison, we emphasize in this thesis how to preserve access privacy over databases using PIR techniques. Our SPIR construction from Section 8.2 provides both user and database privacy, and equally makes a non-collusion assumption as the second protocol of De Cristofaro et al. [CLT11]. However, the non-collusion property of our schemes is configurable, with $t \geq 2$ (we used $t = 2$ in our experiments), whereas De Cristofaro et al. [CLT11] only supports $t = 1$. t is the maximum number of colluding parties that cannot compromise privacy guarantees.

Chapter 3

Related Work

This chapter covers previous work related to this thesis. The first three sections are mostly related to our MSPIR, SQLPIR and TOPIR contributions, while the last section covers prior attempts to apply PIR to realize access privacy in several problem domains. We begin by describing attempts in the literature to reduce the computational cost of PIR schemes. Afterwards, we give an overview of prior work aimed to make data access during PIR more expressive. Since computing on every PIR database block may be too slow for large databases, we present some work done to help users trade off privacy for computational efficiency. We briefly highlight a pseudonymous mailing system as one of the first systems that rely on PIR. Then we cover literature on location-based services and PIR. Finally, we describe some work on using cryptographic primitives like OT and SPIR for obtaining access privacy in the online sales of digital goods.

3.1 Multi-server and Single-server PIR

Chor et al., in defining the notion of PIR, proved that *the trivial PIR scheme* of transferring the entire database to the user and having him retrieve the desired item locally has optimal communication complexity for information-theoretic privacy protection with a single server. [CGKS95, CKGS98] However, more efficient information-theoretic solutions with sub-linear communication complexity were shown to exist if multiple, non-colluding servers hold copies of the database. They proposed a number of such multi-server information-theoretic PIR schemes [CGKS95, CKGS98], including a simple ℓ-server scheme transferring $O(\sqrt{n})$ bits, a 2-server scheme requiring $O(n^{1/3})$ bits transfer, a general ℓ-server scheme with $O(n^{1/\ell})$ bits transfer and a $(\frac{1}{3}\log_2 n + 1)$-server scheme trans-

ferring $\frac{1}{3}(1 + o(1)) \cdot \log_2^2 n \cdot \log_2 \log_2(2n)$ bits, where n is the size of the database in bits and $\ell \geq 2$ is the number of servers. Subsequent work has mostly focused on improving PIR's communication complexity bounds [CGKS95, CKGS98], while some others [Aso04, BS07, GIKM98a, Gol07a] have addressed such problems as using amortization and preprocessing to reduce server-side computational overheads and improving query robustness, amongst others.

Chor and Gilboa [CG97] were the first to relax the absolute privacy offered by multi-server information-theoretic PIR by using cryptographic primitives. They proposed a family of 2-server computationally private PIR schemes by making intractability assumptions on the existence of pseudorandom generators or one-way functions. Schemes in this family have a worst-case communication complexity of $O(n^\epsilon)$, for every $\epsilon > 0$. In the same year (1997), Kushilevitz and Ostrovsky [KO97] proposed the first single-server PIR scheme with a similar communication complexity by assuming quadratic residuosity decisions modulo a composite of unknown factorization are hard. Thus, the best protection offered by any non-trivial single-server PIR scheme is computational privacy, but database holders do not need to replicate their data to external servers. Several other single-server PIR schemes followed, each making some intractability assumption based on the cryptographic primitive used [AMG07, CMS99, Lip05]. These include schemes based on group-homomorphic encryption [Ste98], length-flexible additive homomorphic public-key cryptosystems [Lip05], Φ-Hiding Assumption (ΦHA) trapdoor predicates (the hardness of deciding if a small prime is a factor of the totient of a composite integer of unknown factorization) [CMS99], and more recently, a lattice-based scheme by Aguilar-Melchor and Gaborit [AMG07].

3.2 Computational Practicality of PIR

PIR has to date been the primary approach to the problem of preserving access privacy for Internet users and several attempts have been made to make PIR more practical. The general technique used to determine if a PIR scheme is practical is to compare its performance with that of the trivial scheme, given figures for network bandwidth and computational power. As observed previously, the most limiting of the barriers to making PIR more practical is the computational requirement of PIR schemes. The performance measure of a scheme in terms of its computational complexity has only received limited attention. The first of these is the work by Beimel et al. [BIM00, BIM04], which shows that, given an n-bit database X that is organized into r b-bit blocks, 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, $j \in \{1, \cdots, n\}$. They introduced

a model of PIR with preprocessing which requires each database to precompute and store some *extra bits* of information, which is polynomial in the number of bits n of the database, before a PIR scheme is run the first time. Subsequently, the databases can respond to users' queries in a less computationally expensive manner using the extra bits.

Asonov et al. [Aso04, AF03] and Smith et al. [SS01] similarly explore preprocessing for reducing server-side computation. However, the specialized hardware requirement at the server makes this solution less desirable. Several hardware-assisted PIR schemes [Aso04, SS01, WS08] rely on the preprocessing model. With the exception of Wang et al. [WWP10], all secure coprocessor-based PIR schemes require periodic database reshuffles (i.e., repeats of the preprocessing stage). The reshuffling cost of Williams and Sion [WS08], for example, is $O(\log^4(n))$, but when amortized, it is $O(\log^3(n))$ per query. Nonetheless, the paper 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 $\Theta(\sqrt{n})$, is available on the secure coprocessor.

In 2006, panelists from SECURECOMM [Cre06] came together to discuss how to achieve practical private information retrieval. The discussion covers several aspects of transitioning cryptographic primitives from theory to practice and the need for practical PIR implementations and benchmarks on real data. The panelists were optimistic about future PIR deployments and pointed to the need for finding PIR schemes that require cheaper operations or utilize secure hardware at the server side.

The paper by Sion and Carbunar [SC07] compares the bandwidth cost of trivial PIR to the computation and bandwidth cost of a single-server computational PIR scheme [KO97], which they considered to be the most efficient at that time. Their motivation was to stimulate practical PIR schemes; nevertheless, the result has been cited in the literature to promote the general idea that non-trivial PIR is always more costly than trivial download. A few attempts have been made [AMG07, CAMG08, TP11] to argue otherwise by introducing more efficient computational PIR schemes that may be practical to deploy. Our work (Chapter 4) extends Sion and Carbunar's work in important ways. First, their analysis was based on a number-theoretic computational PIR scheme [KO97], whereas we considered different varieties of computational PIR schemes: a number-theoretic scheme [KO97] and a lattice-based linear algebra scheme [AMG07]. A consideration of the state-of-the-art PIR schemes on the basis of their underlying mathematical assumptions is important because computational performance is currently a limiting factor to the practicality of PIR schemes. Second, we extend the analysis of practicality to multi-server PIR schemes, which have never been considered by any previous measurement study. Information-theoretic multi-server PIR schemes are especially important because they can offer a stronger privacy guarantee for non-colluding servers, unlike computational PIR schemes that require large

keys to protect against future powerful adversaries. Besides, multi-server PIR schemes give better performance and are directly deployable in domains where the databases are naturally distributed, such as Internet domain name registration [OG10b]. Even in domains where the database is not distributed, deployment is possible using servers containing random data [GIKM98a], which eliminates the need for an organization to replicate its data to foreign servers.

Aguilar-Melchor and Gaborit [AMG07] and Aguilar-Melchor et al. [AMCG$^+$08] explore linear algebra techniques using lattices to propose an efficient single-server PIR scheme. The security of the scheme is based on the hardness of the differential hidden lattice problem — a problem related to NP-complete coding theory problems [Wie06]. Aguilar-Melchor et al. [AMCG$^+$08] subsequently used commodity Graphics Processing Units (GPUs), which are highly parallelizable, to achieve a database processing rate of 2 Gb/s, which is about ten times faster than running the same PIR scheme on CPUs. However, the evaluation from Aguilar-Melchor et al. [AMCG$^+$08] considers a small experimental database consisting of twelve 3 MB files and they did not measure the total roundtrip response time for queries; they considered the server-side cost but ignored client-side computation and transfer costs. It is important to consider the total cost because their scheme is not as efficient in terms of communication complexity as other existing schemes, and roundtrip response time depends on both the communication and computational complexities of a scheme. In addition, the measurements for the single-server PIR schemes [GR05, Lip05] used for their comparison was based on estimates derived from *openssl speed rsa*, which is quite unlike our approach where the comparison is based on analytical expressions for query response times and experimental observations. Besides, they only considered single-server PIR schemes, whereas we also consider multi-server PIR schemes and the state-of-the-art single-server PIR schemes.

While the authors achieved a server-side processing rate of 2 Gb/s, there was no indication of what roundtrip query response time to expect. In addition, we would want to take measurements on our experimental hardware for fair comparison with the trivial PIR solution and multi-server PIR schemes. We note that the 2 Gb/s processing rate was obtained using commodity GPUs, and not on traditional CPUs. They show experimentally that their implementation has a better server-side processing throughput on GPUs than on CPUs (single-processor, dual-core and quad-core processors) by a factor of ten. In comparison to other single-server PIR schemes [GR05, Lip05], this scheme gives a much better server-side processing rate both on CPUs and GPUs.

Trostle and Parrish [TP11] proposed a single-server computational PIR scheme where server-side operations are simple modular additions. The scheme is based on a trapdoor group, which allows an entity with knowledge of the trapdoor to compute an inversion

problem efficiently (e.g., computing discrete logarithms). Without the knowledge, it is computationally hard to solve the problem. While the scheme gave better performance than similar single-server PIR schemes [GR05, Lip05], it is not quite as efficient as the scheme of Aguilar-Melchor and Gaborit [AMG07] and Aguilar-Melchor et al. [AMCG+08].

In the context of keyword search using PIR, Yoshida et al. [YCSI08] considered the practicality of a scheme proposed by Boneh et al. [BKOS07]. This public key encryption based keyword search protocol is essentially single-server PIR. Their investigations found the scheme to be costlier than the trivial PIR solution.

3.3 Data Access Models for PIR

Some research efforts have explored using expressive data access models for the databases used in PIR schemes. Most PIR proposals provide rudimentary data access models, limited to either retrieving a single bit or a block of bits [BS07, CGKS95, KO97] using data indices/addresses or textual keywords [CGN97]. These theoretical primitives are a limiting factor in deploying successful PIR-based systems. There is therefore a need for an extension to a more expressive data access model, and to a model that enables data retrieval from structured data sources, such as from relational databases. The main challenge in realizing the objective is the difficulty in finding a balance between keeping a query private and the need of a database to understand the query. A query expressed in SQL explicitly describes the result that the database is expected to understand and produce. Conversely, a PIR query is private, but not expressive; the PIR server can process a query without needing to understand its content. The goal therefore, will be to provide support for privacy, while preserving the expressiveness and benefits of SQL.

A common assumption for PIR schemes is that the user knows the index or address of the item to be retrieved. However, Chor et al. [CGN97] proposed a way to access data with PIR using keyword searches over three data structures: binary search tree, trie and perfect hashing. Our SQLPIR work in Chapter 5 extends keyword-based PIR to B^+ trees and minimal perfect hash functions. In addition, we provide an implemented system and combine the technique with the expressive SQL. The technique in [CGN97] neither explores B^+ trees nor considers executing SQL queries using keyword-based PIR.

Reardon et al. [RPG07] similarly explore using SQL for private information retrieval, and proposed the TransPIR prototype system. We will use this work as the basis for comparisons with our SQLPIR contribution (Chapter 5), since it is the closest. TransPIR performs traditional database functions (such as parsing and optimization) locally on the

client; it uses PIR for data block retrieval from the database server, whose function has been reduced to a block-serving PIR server. The benefit of TransPIR is that the database will not learn any information even about the textual content of the user's query. The drawbacks are poor query performance because the database is unable to perform any optimization, and the lack of interoperability with any existing relational database system.

Freedman et al. [FIPR05] provide a solution for database search with keywords in various settings including OT, using oblivious polynomial evaluation and homomorphic encryption. However, each database tuple, which they referred to as a payload, still needs to be tagged with an appropriate keyword. The key improvements over earlier results [NP99, NP01] is the preservation of privacy against a fixed number of queries after an initial setup, a fixed number of rounds for oblivious query evaluation, and the ability to deal with exponential domain sizes.

Recall that private matching and private set intersection schemes [CZ09, FNP04, JL09] consider the problem of computing the intersection of two private sets from two users, such that each user only learns the sets' intersection. Our SQLPIR work is significantly different from private intersection schemes because SQL queries are richer and more complex than simple set intersection. In addition, an SQL query describes the expected result of a query, which may not contain any itemized listing of the data, whereas private set intersection schemes require the exact data to be the input of a query.

A number of approaches for searching on encrypted data have been proposed [BSNS06, SWP00, YY08]. Shi et al. [SBC+07] consider the problem of storing encrypted data in an untrusted repository. To retrieve a subset of the encrypted data, the user must possess a key that will only decrypt the data matching some preauthorized attributes or keywords. They considered the encryption and auditing of network flows, but the approach is also applicable to financial audit logs, medical privacy, and identity-based biometric encryption systems. Our SQLPIR work is different from encrypted data search in three ways. First, we do not require encryption of the data, regardless of the assumption of the adversary being an insider to the database server. The privacy provided with PIR aims to hide the particular data that is of interest, in the midst of the entire unencrypted data set. Second, the type of query supported with our approach is much more extensive. Third, encrypted data search is typically performed on unstructured or textual data, whereas our approach deals with structured data in the context of relational databases.

A closely related research stream in the computing context of Database-as-a-Service [HILM02, HMT04] is the problem of privately searching an encrypted index over an outsourced database. Hacigümüş et al. [HILM02] present a technique for executing SQL queries over a user-encrypted database hosted in a service provider's server. The goal is to

protect the data from the service provider, but still enable the user to query the database. The context of use for the Database-as-a-service paradigm differs from that of PIR. The service provider typically owns the data that multiple users query with PIR, and the goal of PIR is not to hide data from the server, but to hide data access patterns, which could leak information about users' requests.

3.4 Large Database Access Privacy and Tradeoffs

Many public databases over which users' access privacy is important are large. Querying such databases with existing PIR schemes will always be too slow because it will take too long to read and compute on every database block. Traditional information retrieval even avoids reading every database block; the problem is even greater for PIR, which requires additional time to compute on every block. The reality of providing access privacy over large databases points to the need for a systematic way to query a reasonably large subset of the database in a timely fashion, in which case the user privacy is now with respect to the database subset. While prior work [CKGS98, Ghi08] suggested the possibility of tradeoffs, our LBSPIR work validates tradeoffs for PIR in an LBS context (Chapter 7). In contrast to these prior works, our TOPIR work (Chapter 6) uniquely addresses the problem of preserving access privacy over a large database in a generic way. For the rest of this section, we will present some related work on privacy-preserving queries over large data sets.

Wang et al. [WAEA10] proposed a bounding-box PIR (bbPIR) which combines the concept of k-anonymity with the single-server computational PIR scheme by Kushilevitz and Ostrovsky [KO97] 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 (Chapter 6) 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 his or 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 [KO97], 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 an input query q to an equivalent privacy-preserving query Q and requires minimal user intervention.

Howe and Nissenbaum [HN09] 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. Indeed, Peddinti and Saxena [PS10] have shown how a search engine can compromise the privacy guarantees of TrackMeNot using off-the-shelf machine learning classifiers.

Domingo-Ferrer et al. [DFBAWM09] 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 $h(.)$ 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 in Chapter 6 where the tradeoff is between privacy and computational efficiency. In addition, their approach relies on the availability of a public thesaurus of keywords with known 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.

3.5 Application Areas for PIR

This section covers some systems that have been described and built using the PIR primitive. We will describe each system and conclude with the contributions we have made to this area of research.

3.5.1 Pseudonymous Mailing System

An interesting description of a practical pseudonymous message retrieval system using the technique of PIR is presented by Sassaman et al. [SCM05]. The system, known as the Pynchon Gate, helps preserve the anonymity of users as they privately retrieve messages using pseudonyms from a centralized server. Unlike our use of PIR to preserve a user's query privacy, the goal of the Pynchon Gate is to maintain privacy for users' identities. It does this by ensuring the messages a user retrieves cannot be linked to his or her pseudonym. The construction resists traffic analysis, though users may need to perform some dummy PIR queries to prevent a passive observer from learning the number of messages she has received.

3.5.2 Location-Based Services and PIR

One of the motivations for developing useful and practical PIR schemes is the increasing pervasiveness of mobile devices with positioning capabilities. When a user queries a remote server from a stationary desktop, the information leaked by the query is often limited to the content of the query. However, when a user queries a location-based service, the longitude and latitude of the user's physical location are also being disclosed to the server. The problem of *location privacy* is how to keep users' location information private from an LBS while still ensuring the LBS is able to return responses that are precise; locations in close proximity for a nearby neighbour problem, for example.

We provide a brief overview of cloaking- and PIR-based approaches for location privacy. Solanas et al. [SDFMB08] provides a survey and classification of methods for location privacy in LBS. Similarly, in a position paper in 2008 [Ghi08], Ghinita introduced a taxonomy for LBS privacy techniques.

Location cloaking techniques

Location cloaking in general seeks to prevent an attacker from being able to compromise location privacy by linking queries made by a particular user to the user's exact location. The attacker may be in a position to observe traffic flowing through the network or even be situated at the LBS provider endpoint.

One popular cloaking technique is based on the principle of k-anonymity, where a user is hidden among $k-1$ other users. Queries from multiple users are typically aggregated at an anonymity server, which forms an intermediary between the user and the LBS provider. This central anonymity server can provide spatial and temporal cloaking functions, so that an attacker will encounter difficulty matching multiple queries that are observed with users at particular locations and at particular points in time. Many cloaking solutions for location privacy suggest either a central anonymity server as described [GG03,XC07], or other means such as decentralized trusted peers [CML06] or distributed k-anonymity [ZH09].

The chief problem is that the anonymity server must normally be part of the trusted computing environment and represents a single point of vulnerability. If it is successfully attacked, or collusion with the LBS server occurs, then the locations of all users may be divulged. It is also observed that although a cloaking technique by itself is advantageous in that it does not result in increased computational cost on the server, it can carry with it a high communication cost from the LBS provider to the client. This can mean a large and unacceptable penalty for mobile phone users. Finally, if a reduced sample population results from the number of active users in a particular geographic area, it may not suffice to satisfy the desired degree of anonymity. If the anonymity server delays execution of a request until the k-anonymity condition is satisfied, then this delay may prove to be unacceptable to the user from a feature interaction point of view.

PIR-based techniques

PIR has been applied to solving the problem of keeping a user's location private when retrieving location-based content from a PIR database. This content typically consists of points of interest (POIs), with each entry consisting of a description of a place of interest as well as its geographical location. The only work cited for PIR in the survey by Solanas et al. [SDFMB08] that does not utilize a third party is a 2008 paper by Ghinita [GKK$^+$08]. The key strengths of the solution of Ghinita are the nondisclosure of location information and the fact that it is resistant against correlation attacks for both stationary and highly mobile users.

Our LBSPIR work that leverages the PIR technique for location privacy (Chapter 7) differs from the PIR approach of Ghinita [GKK+08] in three important ways. First, the latter is specifically based on the 1997 computational PIR scheme by Kushilevitz et al. [KO97]. It would require considerable re-invention before it could be used with recent and more efficient PIR schemes. For instance, it re-organizes a POI database into a square matrix M despite the reduced communications costs attainable from using a rectangular matrix. On the other hand, our approach is flexible and supports any block-based PIR scheme. Second, the costs of computation and communication with the approach are $O(n)$ and $O(\sqrt{n})$, respectively, where n is the number of items, or POIs, in the database. The user has no flexibility for dealing with this linear computational cost for large n and it reveals too many POIs to the user; it is too costly for low-bandwidth devices. Our hybrid technique departs from this one-size-fits-all approach and enables users to negotiate their desired level of privacy and efficiency with LBS providers. Third, the scope of the approach did not consider privacy-preserving partitioning for the data set. It considers partitioning with kd-trees and R-trees in the general sense, without specific privacy considerations. On the other hand, our work uses a different method of partitioning of POI data that permits cloaking, and offers privacy protection when used in conjunction with PIR.

The common criticism against this PIR-based approach in the location privacy literature is that it is too costly to be practical [LBCP08], and that the computational overhead is unsuitable for resource-constrained hardware, such as smartphones [RPB08]. Most efforts to apply PIR in this domain rely on hardware-assisted variants, with a secure coprocessor (SC) at the LBS server host [AKS08, Hen07]. For example, Hengartner [Hen08] used trusted computing and secure logging to preserve location privacy. He addressed the inefficiency of PIR by adopting an alternative solution that requires a Trusted Platform Module (TPM) [Tru10] and cloaks a user's location before accessing the POI database. The service provider can learn the user's cloaking region, but not the exact location. The motivation for Hengartner's work was to avoid using PIR in such a way that it would have to process all of the POIs in the database in order to respond to a query, because the computational cost is linear in the size of the database.

The taxonomy for LBS privacy proposed by Ghinita [Ghi08] discusses how each technique realizes tradeoffs in privacy and efficiency. The taxonomy, in increasing order of privacy protection and decreasing order of performance, is: two-tier spatial transformations (e.g., SpaceTwist [YJHL08]), three-tier spatial transformations (e.g., Casper [MCA06]) and cryptographic transformations (e.g., the PIR approach by Ghinita et al. [GKK+08]). The paper defines the taxonomy using the architecture of the various techniques and the transformation of the user's location (i.e., through perturbation or encryption). Techniques based on the two-tier spatial transformation do not utilize an anonymity server and are

therefore vulnerable to background knowledge attacks — where the adversary has gathered some external knowledge about the user's location. The three-tier spatial transformation techniques employ an anonymity server and resist background knowledge attacks, but query performance is not as good as in the two-tier transformational techniques.

Hybrid techniques

Hybrid techniques [Ghi08] permit privacy-efficiency tradeoff decisions to be made by combining the benefits of cloaking- and PIR-based techniques. Chor et al. [CKGS98] conjectured a tradeoff between privacy and computational overhead as a means of reducing the high computational overhead for some application areas of PIR. Our work (Chapter 7) concretizes and validates their conjecture in the context of LBS, and also realizes the future work left open by Ghinita [Ghi08], which is to further reduce the performance overhead of PIR techniques. The optimization of PIR by Ghinita et al. [GKK$^+$08] reuses partial computation results (i.e., multiplications of large numbers) and parallelizes the computations. This optimization reduces the CPU cost by 40%, but the overall query response time is still impractical [LBCP08, RPB08]. Ghinita [Ghi08] suggests improving the performance of PIR-based techniques for LBS privacy through a hybrid method that includes a PIR phase on a restricted subset of the data space. Our work answers the open question of how to reduce the processing cost of PIR, without requiring the LBS to have multiple CPUs to take advantage of parallelization. Parallel processors are not typically found on smartphones, either.

3.5.3 Electronic Commerce

The related bodies of work to our PIR-Commerce contribution (Section 8.2) are symmetric private information retrieval (SPIR), oblivious transfer (OT), OT with access control (OTAC), and priced OT (POT).

Recall from Section 2.6 that OT schemes allow a database X consisting of two records and a user holding an index $i \in \{0, 1\}$ to run a protocol that results in the user learning the ith record and no information the $(1 - i)$th record, while the database learns nothing about i.

SPIR schemes [GIKM98b] address database privacy so that dishonest users cannot learn any information about other database records beyond the record retrieved. All existing communication-efficient 1-out-of-n OT schemes are essentially single-server SPIR, whereas all existing communication-efficient distributed 1-out-of-n OT schemes [GIKM98b]

(i.e., two or more servers) are essentially multi-server SPIR. The first work on preserving database privacy against dishonest users in a multi-server PIR setting was by Gertner et al. [GIKM98b] They propose a single-round ℓ-server SPIR scheme with communication complexity $O(\log n \cdot n^{1/(2\ell-1)})$ for $\ell \geq 2$ and a $O(\log n)$-server scheme with communication complexity $O(\log^2 n \cdot \log \log n)$. Kushilevitz and Ostrovsky [KO97] briefly discuss how to convert their single-server PIR into SPIR using general zero-knowledge proof techniques, however they propose no concrete constructions. No existing SPIR scheme simultaneously provides support for both access control and tiered pricing as in our PIR-Commerce work (Section 8.2).

Several OTAC schemes [CDN09, CGH09, ZAW+10] were recently proposed. As with our approach, these schemes typically consist of three parties: user, database, and issuer. The issuer provides users needing access to the database with credentials encoding the access rights of users as an attribute set. The database encrypts its content under an access policy specific to each record and makes the encrypted contents available to users for download. A user with a valid credential can run the OTAC protocol with the database to obtain a decryption key for a particular record. After the protocol, the database learns that a user with a valid credential has obtained a key, but learns nothing about the user's credential or the decryption key issued. Users download the entire encrypted database and use the key obtained to decrypt the desired record. Zhang et al. [ZAW+10] used attribute-based encryption to specify record-level access policies in disjunctive form without requiring duplication of database records. However, these schemes do not consider an economic model where users pay for each record and their high communication overhead makes them considerably more costly than SPIR.

POT schemes [AIR01, CDN10] were originally introduced by Aiello et al. [AIR01] to explore the difference between physical goods requiring close monitoring of inventory levels and digital goods that are essentially of unlimited supply (i.e., cannot be depleted by purchases). In their model, users first deposit some money with the database and then proceed to buy multiple digital goods from the database, such that the total price of purchased goods does not exceed the user's deposit/balance. The database does not learn which digital goods the user has purchased. However, since the database tracks the users' accounts, all queries by a single user are linkable. This enables the database server to deduce the number of digital goods a particular user has purchased, the average price of those purchases, and the user's spending pattern [CDN10]. Furthermore, the scheme provides no way for users to recharge their balance, which means that when a user's balance becomes lower than the price of any record, the remaining balance is rendered useless. Camenisch et al. [CDN10] address these problems by encoding users' wallets in an anonymous credential so that the database is no longer required to maintain user-specific

state information; as a result, user purchases become unlinkable. They also lay out an extension that makes use of a trusted third party to facilitate a *fair purchase* protocol; i.e., an optimistic fair exchange protocol to prevent the database server from cheating by not sending the correct decryption key (or wallet) to the user.

All of the above priced and access-control-capable OT and SPIR schemes lack some ingredients necessary for deployment in a practical setting. The foremost missing ingredient is the right combination of functionalities for access control, tiered pricing, sublinear communication complexity, and availability of practical implementations. The SPIR schemes [GIKM98b, KO97] provide no pricing or access control functions. OT schemes (i.e., 1-out-of-n) have prohibitively expensive communication costs and require a static encrypted database, which potentially breaks other applications using the same database. In particular, existing OTAC schemes [CDN09, CGH09, ZAW+10] do not provide pricing functions, while the POT schemes [AIR01, CDN10], on the other hand, provide no access control functions.

Chapter 4

MSPIR: Revisiting The Computational Practicality of PIR

This chapter is adapted from published work supervised by Ian Goldberg [OG11].

In this chapter, we report on a performance analysis of a single-server lattice-based scheme by Aguilar-Melchor and Gaborit, as well as two multi-server information-theoretic PIR schemes by Chor et al. and by Goldberg. Using analytical and experimental techniques, we find the end-to-end response times of these schemes to be *one to three orders of magnitude* (10–1000 times) smaller than the trivial scheme for realistic computation power and network bandwidth. Our results extend and clarify the conclusions of Sion and Carbunar for multi-server PIR schemes and single-server PIR schemes that do not rely heavily on number theory.

4.1 Introduction

In 2007, Sion and Carbunar [SC07] considered the practicality of single-server computational PIR schemes and concluded that PIR would likely remain several orders of magnitude slower than an entire database transfer — the trivial PIR scheme — for past, current, and future commodity general-purpose hardware and networks. They based their result on the cheaper cost of transferring one bit of data compared to the cost of PIR-processing that bit using modular multiplication on such hardware. The PIR scheme of Kushilevitz and Ostrovsky, which was used in their comparison, requires one modular multiplication per database bit. They projected future increases in computing performance and network

bandwidth using Moore's Law [Moo65] and Nielsen's Law [Nie88] respectively, and argued that improvements in computing performance would not result in significant improvements in the processing speed of PIR because of the need to use larger key sizes to maintain security. The significance of this work lies in establishing that any computational PIR scheme that requires one or more modular multiplications per database bit cannot be as efficient as the trivial PIR scheme.

However, it is not clear whether the conclusions of Sion and Carbunar [SC07] also apply to multi-server PIR schemes as well as single-server PIR schemes that do not rely heavily on number theory (i.e., modular multiplications). This is an important clarification to make because PIR-processing with most multi-server PIR schemes and some single-server PIR schemes [AMG07,TP11] costs much less than one modular multiplication per database bit. Besides, the projections by Sion and Carbunar [SC07] assume that all PIR schemes make intractability assumptions that would necessitate the use of larger keys to guarantee security and privacy when today's hardware and networks improve. However, multi-server PIR schemes offering information-theoretic privacy will continue to guarantee security and privacy without requiring key size changes irrespective of these improvements.

In this chapter, we revisit the computational practicality of PIR in general by extending and clarifying the results in [SC07]. First, we provide a detailed performance analysis of a recent single-server PIR scheme by Aguilar-Melchor and Gaborit [AMG07] and Aguilar-Melchor et al. [AMCG$^+$08,CAMG08], which has attempted to reduce the cost of processing each database bit by using cheaper operations than modular multiplications. Unlike previous schemes that rely heavily on number theory, this particular scheme is based on linear algebra, and in particular, lattices. The authors introduced and based the security of the scheme on the differential hidden lattice problem, which they show is related to NP-complete coding theory problems. [Wie06] They proposed and implemented the protocols, but their analysis was limited to server-side computations by the PIR server [AMCG$^+$08] on a small experimental database consisting of twelve 3 MB files. It is unclear how well the scheme compares against the trivial PIR scheme for realistic database sizes. Using the PIR scheme of Kushilevitz and Ostrovsky and updated parameters from [SC07], we first reestablished the result by Sion and Carbunar that this scheme is an order of magnitude more costly than the trivial PIR scheme. We also provide a new result that shows that the single-server PIR scheme in [AMG07] offers an order of magnitude smaller response time compared to the trivial scheme, thus extending the conclusions of Sion and Carbunar about computational PIR schemes.

Second, we explore the case of multi-server information-theoretic PIR, which is yet to be considered by any previous study. Considering multi-server PIR is important because such schemes do not require costly modular arithmetic, and hence will benefit immensely

from advances in computing and network trends. We derive upper-bound expressions for query round-trip response times for two multi-server information-theoretic PIR schemes by Chor et al. [CGKS95] and by Goldberg [Gol07a], which is novel to this thesis. Through analytical and experimental techniques we find that the end-to-end response times of multi-server PIR schemes to be two to three orders of magnitude (100–1000 times) smaller than the trivial scheme for realistic computation powers and network bandwidths.

4.1.1 Preliminaries

We begin by outlining a few building blocks, some of which are based on the work of Sion and Carbunar [SC07]. These include the hardware, network bandwidth between the user and the server, and execution time estimates for modular multiplication.

Hardware description. All but one of our experiments were performed on current server hardware with two quad-core 2.50 GHz Intel Xeon E5420 CPUs, 32 GB of 667 MHz DDR2 memory, 6144 KB cache per core, an Adaptec 51645 RAID controller with 16 1.5TB SATA disks, and running Ubuntu Linux 9.10. The memory bandwidth is 21.344 GB/s and the disk read bandwidth is at least 300 MB/s. We note that these machine characteristics are not unusual for database server hardware; this machine cost less than $8,000. We ran the GPU implementation of the Aguilar-Melchor et al. [AMG07] scheme on a machine with a Tesla C1060 GPU, 8 GB RAM, 116 MB/s disk bandwidth, and running Ubuntu Linux 9.10.

Network. Three types of network setups were considered [SC07]: average home-user last-mile connection [Inf09], Ethernet LAN, and commercial high-end inter-site connections [Int09, Spu00, Tan02]. Table 4.1 shows various network connection speeds (Mbps) since 1995, when PIR was introduced. The values up until 2006 are reused from [SC07], while we provided the subsequent values based on the capacity of today's network bandwidths.[1]

Modular multiplication. The work of Sion and Carbunar [SC07] uses Dhrystone MIPS ratings for Pentium 4 CPUs in order to estimate t_{mul}, the time it takes to compute a modular multiplication — the building block for the PIR scheme of Kushilevitz and Ostrovsky [KO97]. Such CPUs have long been retired by Intel and are no longer representative of

[1]The company Ookla Net Metrics launched Speedtest.net in 2007 and the Pingtest.net tool in 2009. Recently, it has made Internet bandwidth data based on tests from various locations around the world available to researchers for free download (`http://www.netindex.com`). The dataset covers 2008 to the present (Source: `http://www.itworld.com/networking/108922/internet-speedtest-results-going-public`).

Table 4.1: Bandwidth estimates (in Mbps) for 1995 to 2010. We adapted values up to 2007 from Sion and Carbunar [SC07] and those after 2007 are based on the Internet speed data for Canada and US from NetIndex.com [Ook10].

Network types	1995	1997	1998	1999	2001	2005	2006	2007	2008	2009	2010
End-user (B)	.028	.056		.768	1	4	6	6	6	8	9
Ethernet LAN (B_2)	10	100		1000		10000	10000	10000	10000	10000	10000
Commercial (B_3)	.256	.768	1	10	100	1000	1500	1500	1500	1500	1500

today's multi-core CPUs. In addition, the Dhrystone benchmark, which found widespread usage at the time it was introduced in 1984, is now outdated. According to Dhrystone benchmark author Reinhold P. Weicker, it can no longer be relied upon as a representative benchmark for modern CPUs and workloads [Wei02].

Instead, we measure the time directly. The current recommended key size for the security of the Kushilevitz and Ostrovsky scheme is 1536 bits, based on the key size schedule from RSA Labs [Kal03], NIST [Nat07], and NESSIE [New04]. We experimentally measured the value of t_{mul} on the server hardware described above. After repeated runs of the measurement code and averaging, we obtained $t_{mul} = 3.08 \pm 0.08$ μs.

Projections. Moore's Law [Moo65] has an annual growth rate of 60%, which surpasses the 50% growth rate of Nielsen's Law [Nie88]. While the faster growth rate of computing capabilities does not necessarily favour computational single-server PIR schemes, it does favour multi-server information-theoretic PIR schemes. Therefore, advances in computing and network trends will not outdate our result regarding the practicality of multi-server PIR schemes.

4.2 Efficient Single-server PIR (LPIR-A)

We experimentally evaluated an implementation of the Aguilar-Melchor et al. [AMCG+08] single-server PIR scheme. This is the most efficient known single-server PIR scheme, and has available source code both for CPUs and GPUs. We present a note of caution, however, that although this PIR scheme resists known lattice-based attacks, it is still relatively new, and its security is not as well understood as those of the PIR schemes that rely heavily on number theory.

4.2.1 Experiment

We obtained the source code [GPG09] for this scheme, removed interactivity, changed the default parameters to a set that guarantees security in a practical setting (complexity of over 2^{100} operations to break the scheme) [AMG07], and added instrumentation to the CPU and GPU code variants. The data set for our experiment consists of various databases of sizes between 1 GB and 28 GB, each containing random data. Bugs in the implementation [GPG09] prevented us from testing larger databases for the selected security parameters. We did not fix the bugs because doing so would require rewriting a significant portion of the code. Besides, 28 GB and smaller databases are sufficient for doing a fair comparison with the trivial PIR scheme. We ran the experiment on the server hardware described in Section 4.1.1 and measured the data transfer between the client and the server, running on the same machine. We ran queries to retrieve between 5 and 10 random blocks for each database size, and computed the average response time and standard deviation for each database size.

4.2.2 Result

Figure 4.1 shows the log-log plots of our results with breakdowns of the time for query generation and upload, response encoding and download, and response decoding, as well as the trivial download time for the different sizes of databases we tested. Plots (a), (b), and (c) respectively reflect bandwidth values typical of an Internet connection in the US and Canada, a 100 Mbps fast Ethernet, and a 1 Gbps gigabit Ethernet, all using the CPU for computation. Plot (d) represents a 100 Mbps fast Ethernet using the GPU hardware.

In plot (a), for example, the largest portion of the overall time is that of query upload; this is due to the comparatively low 2 Mbps upload bandwidth typical of a home Internet connection [Ook10]. On the other hand, the time to download the query result (at 9 Mbps) is much smaller. In general, the response time is proportional to n and the slope of the line is 1, as the computation costs, in particular server-side response encoding, dominate. When the database exceeds the available RAM size, further slowdowns are seen in the results.

The slope of the trivial PIR line is always 1, since the time is simply that of transferring the entire database. We observed that for tiny databases of a few megabytes (not shown on Figure 4.1), the trivial PIR scheme is faster, but depending on the bandwidth, there is a crossover point at which sending less data plus computing on every bit of the database becomes faster than sending the entire database. For the average home connection, for

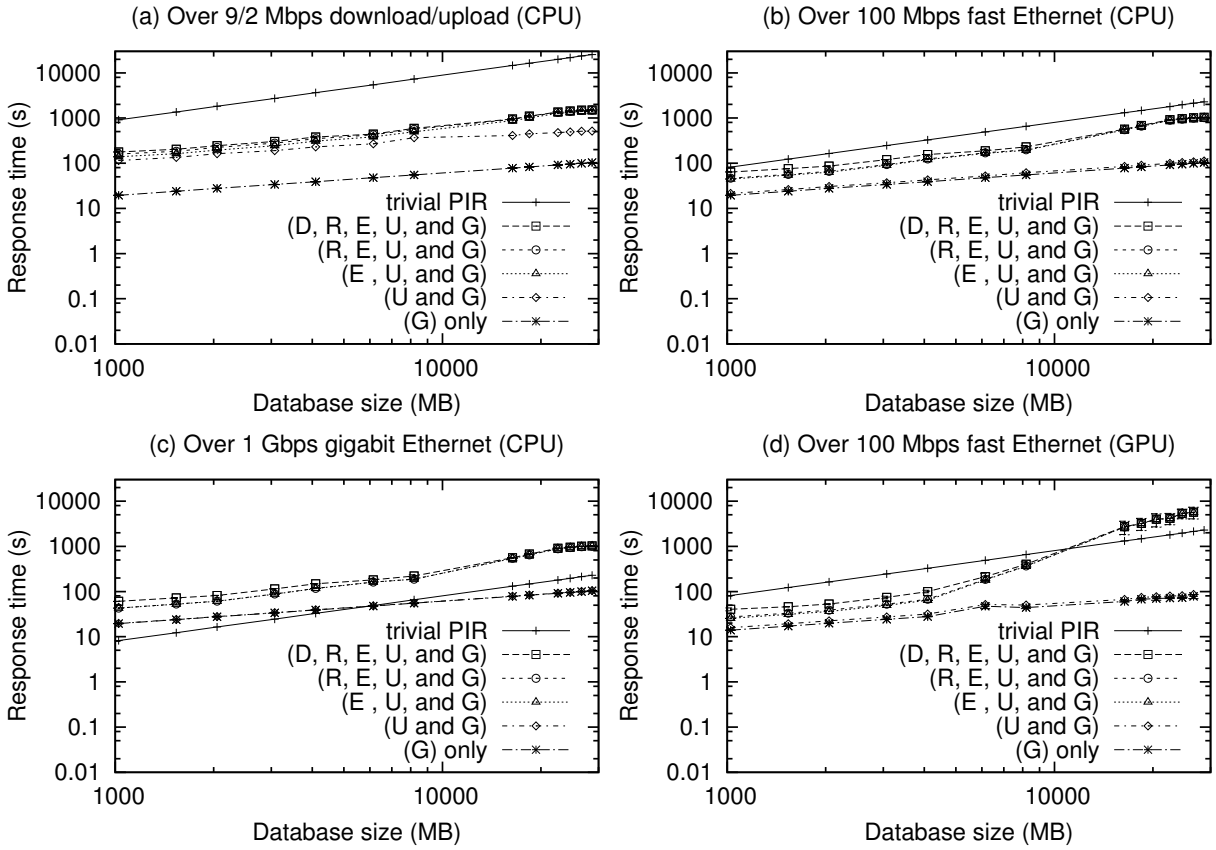


Figure 4.1: Logarithmic scale plots for query generation (G), query upload(U), response encoding (E), response download (R), and response decoding (D) times for the single-server PIR scheme [AMCG+08] and the trivial PIR scheme in different bandwidth scenarios.

example, we found this to occur at a very small database size (approximately 32 MB). For the 1 Gbps connection, the network is so fast that the entire database can be transferred in less time than it takes for the client to even generate its query, except for databases of 6 GB and larger. Even then, trivial transfer was much faster than the overall cost of this PIR scheme for such fast networks.

We note that plot (a) is the most representative of today's consumer bandwidth situation. Based on the recently available Internet speed database [Ook10], the average bandwidth for the Internet user is improving rather slowly, with average download rates of 6, 7.79, and 9.23 Mbps for Canada and the US for 2008, 2009, and January 1 to May 30 of 2010. The average upload rates for the respective periods are 1.07, 1.69, and 1.94 Mbps. We

note that Nielsen's Law specifically addresses the type of users described as normal "high-end" who can afford to pay a premium for high-bandwidth network connections. [Nie88] We contrast these users from "low-end" users [Nie88] that the above bandwidth averages from the Internet speed data [Ook10] include. Hence, the majority of Internet users are low-end users, and their bandwidth is much more limited than that predicted by Nielsen's Law.

In the plots and in the analysis above, we show changing bandwidths and assume that computing power stays the same. However, if we assume that processors improve at a faster rate than Internet bandwidth for high-end users due to Moore's Law and Nielsen's Law, then the crossover point will move down and the PIR scheme will become faster at smaller database sizes. From plot (d), the GPU run gives a better response time, in comparison to plot (b), for memory-bound databases (about 6 GB or less). For disk-bound databases, the response time degenerates due to the lower disk bandwidth of the GPU machine. We ran the same code on the CPU of the GPU hardware; using the GPU, we found about five times speedup in the server-side processing rate for memory-bound databases and no noticeable speedup for disk-bound databases. Our observed speedup is half the speedup reported by Aguilar-Melchor et al. [AMCG+08], but we used much larger databases.

4.3 Multi-server PIR

In this section, we provide detailed performance analyses of two multi-server information-theoretic PIR schemes, from Chor et al. [CGKS95] and from Goldberg [Gol07a]. We begin with an overview of these schemes and later show how they compare with the single server schemes [AMG07, KO97] and the trivial PIR scheme. The reason for choosing Chor et al. scheme is its simplicity, being the first PIR protocol invented. The reason for choosing Goldberg's scheme is its comprehensiveness and source code availability, which allows for easy experimental analysis. The implementation of the Goldberg's scheme, known as Percy++ [Gol07b], is an open-source project on SourceForge. We also reproduce and reuse the results by Sion et al. [SC07] for the purpose of comparison with the two multi-server PIR schemes. Their result features the single-server PIR scheme by Kushilevitz and Ostrovsky [KO97], which they argued to be the most efficient of existing single-server PIR schemes.

In order to maintain the user's privacy, it must be the case that not all (in the case of the Chor et al. protocol) or at most a configurable threshold number (in the case of the Goldberg protocol) of the database servers collude to unmask the user's query. This is sometimes brought forward as a problematic requirement of these schemes. We note

that, as discussed elsewhere [OG10b], there are reasonable scenarios — such as distributed databases like DNS or whois databases, where the copies of the database may be held by competing parties — in which the non-collusion requirement is acceptable. Further, other privacy-enhancing technologies, such as anonymous remailers [DDM03] and Tor [DMS04a], also make the assumption that not all of the servers involved are colluding against the user.

4.3.1 First Scheme (MPIR-C)

Sion and Carbunar [SC07] used a closed-form expression for the computation and communication cost of the PIR scheme in [KO97]. While we derive similar expressions for the multi-server schemes we studied, we note that such expressions will only approximate the cost because most modern x86 CPUs support hardware-level parallelism such as superscalar operations; single-cycle operations, such as XORs, are parallelized even within a single core. Hence, such expressions can be used to determine an *upper bound* on what response time to expect. We will later determine the exact response time for this PIR scheme through experiments.

Recall from Section 2.4.2 that in the simple $O(\sqrt{n})$ protocol by Chor et al, ℓ servers store a copy of the database D — an $r \times b$ matrix of bits. For optimal performance, we set $r = b = \sqrt{n}$. Hence, the upper bound for the client and server execution times for this protocol can respectively be computed as $2(\ell-1)\frac{\sqrt{n}}{m}t_{\oplus} + 2\ell\sqrt{n}t_t$ and $\frac{n}{m} \cdot (t_{\oplus} + 2t_{ac}) + n \cdot t_{ov}$, where $t_{\oplus}$ and t_t are respectively the execution times for one XOR operation and the transfer time for one bit of data between the client and the server; m is the machine word-size (e.g., 64 bits), n is the database size (in bits), ℓ is the number of servers, t_{ov} represents the amortized server overhead per bit of the database — this overhead is dominated by disk access costs, but also includes things like the time to execute looping instructions as a minor component — and t_{ac} denotes the time for one memory access. Note that the server execution time is the worst-case time because it assumes all the blocks in the database are XORed, whereas we only need to XOR blocks where the i^{th} bit of ρ_j is 1. The expression charges all of the data transfer to the client, since it needs to be serialized there, whereas the server processing is performed in parallel among the ℓ servers.

An upper bound on the query round-trip execution time for this multi-server PIR scheme is then $T_{MPIR-C} < (2(\ell-1)\sqrt{n}/m + n/m) \cdot t_{\oplus} + 2\ell\sqrt{n} \cdot t_t + 2n/m \cdot t_{ac} + n \cdot t_{ov}$. The most dominant term is $n \cdot \left(\frac{1}{m}t_{\oplus} + \frac{2}{m}t_{ac} + t_{ov}\right)$, which will suffice for the entire expression when the value of n is large. We note that the dominant term is independent of the number of servers ℓ. Nevertheless, it is better to use the entire equation to determine the upper bound on execution time for more objective comparison purposes.

Sion and Carbunar [SC07] denoted $t_t = \frac{1}{B}$, given that B is the bandwidth (in bps) of the network connection between the client and the server. $t_\oplus$ will be one cycle. (We indeed measured it to be 0.40 ± 0.01 ns, which is exactly as expected on our 2.50 GHz processor.) Similarly, we measured t_{ac} to be 1 cycle ($0.4000 \pm .0003$ ns). Using unrolling to minimize the overhead of loop instructions, t_{ov} will be dominated by the memory bandwidth if the database fits into memory, or by disk bandwidth otherwise. An upper bound for t_{ov} on our test machine is therefore 0.006 ns for in-memory databases and 0.417 ns for disk-bound databases, based on the numbers in Section 4.1.1.

4.3.2 Second Scheme (MPIR-G)

Recall that Goldberg's scheme [Gol07a] uses ℓ servers to treat the database D as an $r \times s$ matrix of w-bit words (i.e., elements of $GF(2^w)$), where again r is the number of blocks and s is the number of w-bit words per block. As before, we choose $r = s$, but now $r = s = \sqrt{n/w}$. We also choose $w = 8$ to simplify the cost of computations. Using the protocol description of this scheme and the source code [Gol07b], we counted each type of operation to derive upper bounds for the respective client and server execution times as $\ell(\ell-1)\sqrt{n/8}(t_\oplus + t_{ac}) + 2\ell\sqrt{8n}t_t + 3\ell(\ell+1)(t_\oplus + t_{ac})$, and $(n/8)(t_\oplus + 3t_{ac}) + n \cdot t_{ov}$, where the terms are as above. Again, note that we charge all of the communication to the client. The upper bound expression for the protocol's round-trip response time is then $T_{MPIR-G} < \left((\sqrt{n/8}+3)\ell^2 - (\sqrt{n/8}-3)\ell + n/8\right)(t_\oplus + 3t_{ac}) + 2\ell\sqrt{8n} \cdot t_t + n \cdot t_{ov}$.

Here, the dominant term is $n \cdot \left(\frac{1}{8}(t_\oplus + 3t_{ac}) + t_{ov}\right)$. Again, the dominant term is independent of ℓ.

4.3.3 Response Time Measurement Experiment

We measure the round-trip response times for the multi-server PIR schemes in this section. We first modified an implementation of MPIR-G (Percy++) [Gol07b] to use wider data types to enable support for larger databases. We then measured its performance over five different sets of databases, with databases in each set containing random data and ranging in size from 1 GB to 256 GB.

Next, we fetched 5 to 10 blocks from the PIR servers. On the first query, the database needs to be loaded into memory. The server software does this with `mmap()`; the effect is that blocks are read from disk as needed. We expect that the time to satisfy the first query will thus be noticeably longer than for subsequent queries (at least for databases

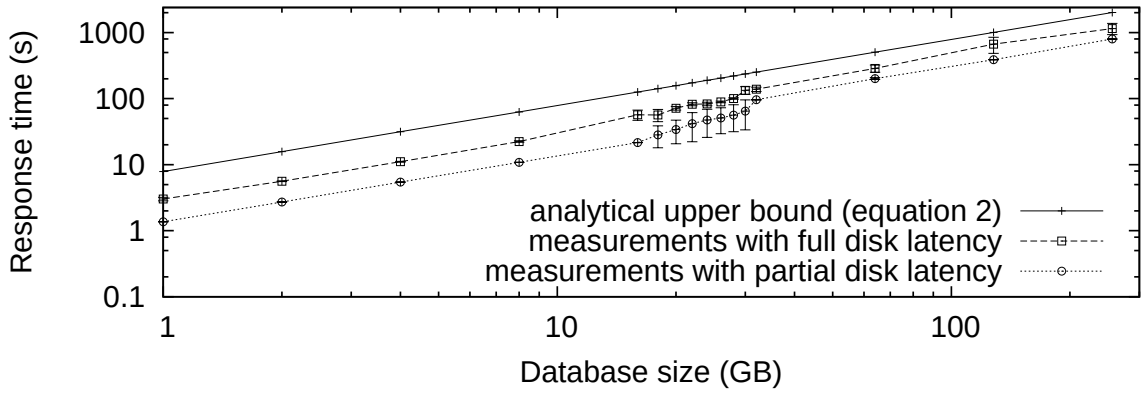


Figure 4.2: Analytical and experimental measurements of the response time of Goldberg's multi-server PIR scheme [Gol07a] (computations only). The upper line is derived from equation (2), but excluding time for communications. The middle line is the time for the first query, which includes startup overhead and reading the database from disk. The lower line is the time for subsequent queries, which only incur disk latencies once the database exceeds the available RAM size.

that fit into available memory), and indeed that is what we observe. For databases larger than available memory, we should not see as much of a difference between the first query and subsequent queries. We show in Figure 4.2 plots of the average response time with standard deviations for these two measurements (i.e., PIR response time for the first query, and for the second and subsequent queries). From the plot, the speed of 1.36 seconds per GB of data is consistent until the databases that are at least 16 GB in size are queried. Between 18 GB and 30 GB, the time per GB grew steadily until 32 GB. The threshold crossed at that range of database sizes is that the database size becomes larger than the available RAM (somewhat smaller than the total RAM size of 32 GB). As can be seen from the plot, the measured values for that range are especially noisy for the lower line. We designed our experiment to take measurements for more databases with size in that range; we surmise that the particulars of Linux's page-replacement strategy contribute a large variance when the database size is very near the available memory size. For even larger databases, PIR query response times consistently averaged 3.1 seconds per GB of data. This is because every query now bears the overhead of reading from the disk. In realistic deployment scenarios where the database fits into available memory, the overhead of disk reads is irrelevant to individual queries and is easily apportioned as part of the server's startup cost. Even when the database cannot fit in available memory, the bottleneck of disk read overheads could be somewhat mitigated by overlapping computation and disk reads;

we did not implement this optimization because the current performance was sufficient for head-to-head comparison with the trivial solution. Note that in practice, the disk read latency would equally come into play even for trivial PIR.

We made similar measurements for the Chor et al. [CGKS95] MPIR-C scheme using an implementation we developed. The implementation differed from the Percy++ [Gol07b] implementation in that it does XORs in units of 64-bit words, instead of in bytes. We obtained a speed of 0.5 seconds per GB (sometimes as fast as 0.26 seconds per GB) for small databases that fit in available memory and 1.0 seconds per GB for larger databases.

4.4 Trivial Download vs. Non-Trivial PIR Compared

We next compare the round-trip response rates for each of the PIR schemes already examined to the response rates of the trivial PIR scheme and the Kushilevitz and Ostrovsky [KO97] scheme. We note that for the non-trivial schemes, the amount of data transmitted is tiny compared to the size of the database, so the available bandwidth does not make much difference. To be as generous as possible to the trivial PIR scheme, we measure the non-trivial schemes with the home connection bandwidth B — 9 Mbps download and 2 Mbps upload. We provide comparisons to the trivial PIR scheme with bandwidths of B, B_2 — 10 Gbps Ethernet, and B_3 — 1.5 Gbps inter-site connections (see Table 4.1).

Figure 4.3 shows the log-log plot of the response times for the multi-server and lattice-based PIR schemes against the earlier results from Sion and Carbunar [SC07], which include the trivial scheme and the Kushilevitz and Ostrovsky scheme [KO97]. As in Sion and Carbunar [SC07], we give maximal benefit to the scheme of Kushilevitz and Ostrovsky [KO97] by ignoring all costs except those of modular multiplication for that scheme, using the value for t_{mul} given in Section 4.1.1. We point out that the values for the trivial scheme and the Kushilevitz and Ostrovsky scheme are computed lower bounds, while those for the LPIR-A, MPIR-G, and MPIR-C schemes are experimentally measured. The number of PIR servers for the multi-server schemes is $\ell = 2$.

We can see from the plot that, as reported in [SC07], the trivial PIR scheme vastly outperforms the computational PIR scheme of Kushilevitz and Ostrovsky, even at the typical home bandwidth. However, at that bandwidth, the lattice-based scheme of Aguilar-Melchor et al. is over 10 times faster than the trivial scheme. Further, both multi-server schemes are faster than the trivial scheme, even at the B_3 (1.5 Gbps) speeds; the MPIR-G scheme is about 4 times faster for databases that fit in RAM, and the MPIR-C scheme is over 10 times faster. For large databases, they are 1.7 and 5 times faster, respectively.

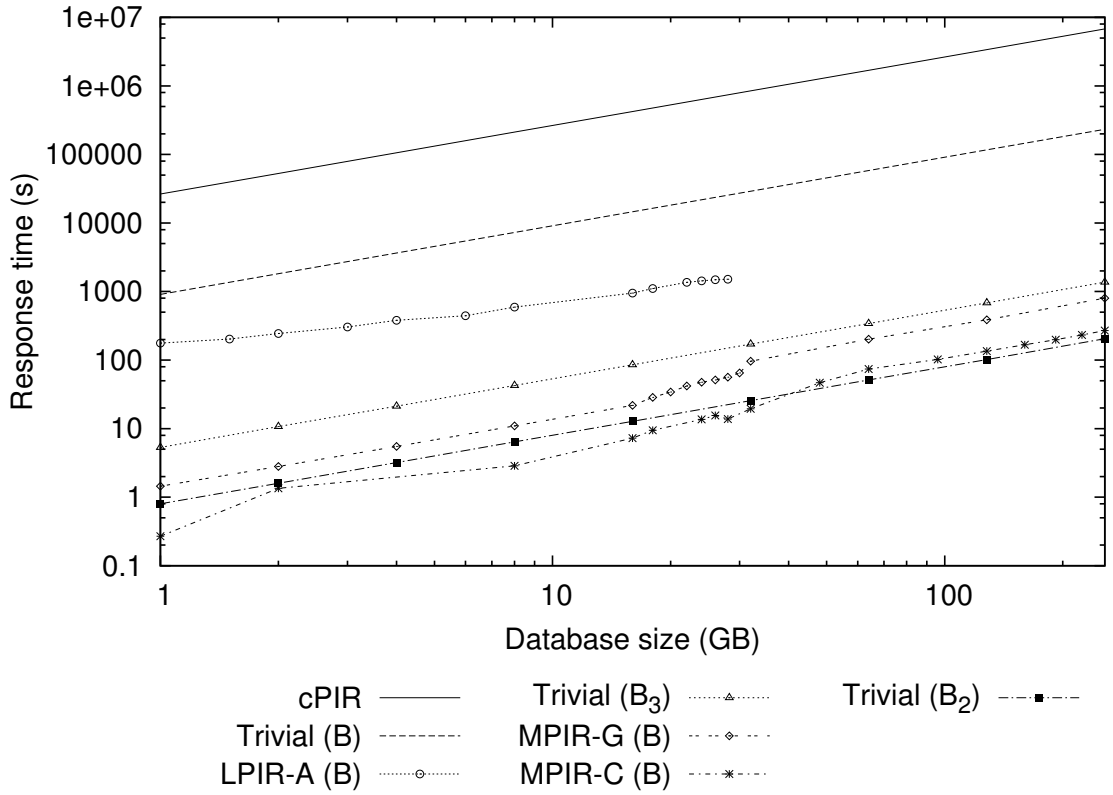


Figure 4.3: Comparing the response times of PIR schemes by Kushilevitz and Ostrovsky (cPIR) [KO97], Aguilar-Melchor [AMCG+08] (LPIR-A), Chor et al. [CGKS95] (MPIR-C), and Goldberg [Gol07a] (MPIR-G), as well as the trivial PIR scheme over three current network bandwidths using different database sizes. The bandwidth used for the non-trivial PIR schemes is B. $\ell = 2$ for the multi-server PIR schemes.

Only at B_2 Ethernet speeds of 10 Gbps does the trivial scheme beat the multi-server schemes, and even then, in-memory databases win for MPIR-C. The apparent advantage of the trivial scheme even at these very high bandwidths may, even so, be illusory, as we did not include the time to read the database from memory or disk in the trivial scheme's lower-bound cost, but we did for the LPIR and MPIR schemes.

One might try rescuing the trivial PIR scheme by observing that, having downloaded the data *once*, the client can perform *many* queries on it at minimal extra cost. This may indeed be true in some scenarios. However, if client storage is limited (such as on smartphones), or if the data is updated frequently, or if the database server wishes to more closely control the number of queries to the database — a pay-per-download music store,

for example — the trivial scheme loses this advantage, and possibly even the ability to be used at all.

To better see at what bandwidth the trivial scheme begins to outperform the others, we plot the response times vs. bandwidth for all five schemes in Figure 4.4. We include one plot for a database of 16 GB, which fits in RAM (a), and one for 28 GB, which does not (b). We see that the trivial scheme only outperforms LPIR-A at speeds above about 100 Mbps, and it outperforms the MPIR schemes only at speeds above 4 Gbps for large databases and above 8 Gbps for small databases. In addition, due to the faster growth rate of computing power as compared to network bandwidth, multi-server PIR schemes will become even faster over time relative to the trivial scheme, and that will increase the bandwidth crossover points for all database sizes.

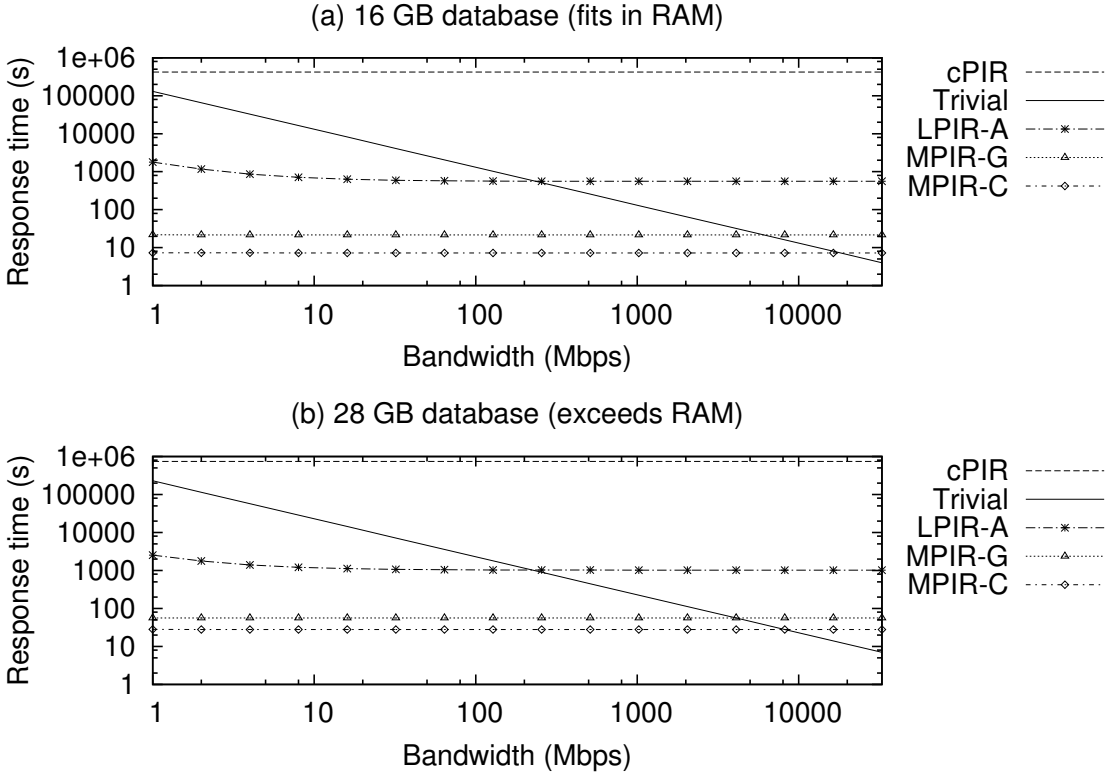


Figure 4.4: Plots of response time vs. bandwidth for the PIR schemes by Kushilevitz and Ostrovsky (cPIR) [KO97], Aguilar-Melchor [AMCG+08] (LPIR-A), Chor et al. [CGKS95] (MPIR-C) and Goldberg [Gol07a] (MPIR-G), as well as the trivial PIR scheme for database sizes that fit in RAM (16 GB) and that exceed RAM (28 GB).

It follows that the multi-server schemes are definitely more practical than the trivial scheme, and the lattice-based scheme is more practical than the trivial scheme for most reasonable bandwidths, including for home and mobile devices.

4.5 Conclusions

We reexamined the computational practicality of PIR following the earlier work by Sion and Carbunar [SC07]. Some interpret Sion and Carbunar as saying that no PIR scheme can be more efficient than the trivial PIR scheme of transmitting the entire database. While this claim holds for the number-theoretic single-database PIR scheme by Kushilevitz and Ostrovsky [KO97] because of its reliance on expensive modular multiplications, it does not hold for all PIR schemes. We performed an analysis of the recently proposed lattice-based PIR scheme by Aguilar-Melchor and Gaborit [AMG07] to determine its comparative benefit over the trivial PIR scheme, and found this scheme to be an order of magnitude more efficient than trivial PIR for situations that are most representative of today's average consumer Internet bandwidth. Next, we considered two multi-server PIR schemes, using both analytical and experimental techniques. We found multi-server PIR to be a *further* one to two orders of magnitude more efficient. We conclude that many real-world situations that require privacy protection can obtain some insight from our work in deciding whether to use existing PIR schemes or the trivial download solution, based on their computing and networking constraints.

Our work may be extended by exploring practical ways for deploying PIR to address real-world privacy problems. In addition, it would be interesting to explore practical and technical means to mitigate some of the assumptions for multi-server PIR schemes, such as preventing the collusion of the servers answering the queries of users.

Chapter 5

SQLPIR: Privacy-preserving Queries over Relational Databases

This chapter is adapted from published work supervised by Ian Goldberg [OG10b].

In this chapter, we explore how Private Information Retrieval (PIR) can help users keep their sensitive information from being leaked in an SQL query. We show how to retrieve data from a relational database with PIR by hiding sensitive constants contained in the predicates of a query. Experimental results and microbenchmarking tests show our approach incurs reasonable storage overhead for the added privacy benefit and performs between 7 and 480 times faster than previous work.

5.1 Introduction

Most software systems request sensitive information from users to construct a query, but privacy concerns can make a user unwilling to provide such information. For instance, the current process for Internet domain name registration requires a user to first disclose the name for the new domain to an Internet domain registrar. Subsequently, the registrar could then preemptively use this inside information to register the new domain and thereby deprive the user of the registration privilege for that domain. This practice is known as *front running* [ICA08]. The registrar is motivated to engage in front running because of the revenue to be derived from reselling the domain at an inflated price, and from placing ads on the domain's landing page. Many users, therefore, find it unacceptable to disclose the sensitive information contained in their queries by the simple act of querying a server.

Users' concern for query privacy and our proposed approach to address it are by no means limited to domain names; they apply to publicly accessible databases in several application domains where the user needs access privacy (see the preamble to this thesis in Section 1.1). Although ICANN claims the practice of domain front running has subsided [ICA08], we will, however, use the domain name example in this chapter to enable head-to-head performance comparisons with a similar approach by Reardon et al. [RPG07], which is based on this same example.

While today's most developed and deployed privacy techniques, such as onion routers and mix networks, offer anonymizing protection for users' identities, they cannot preserve the privacy of the users' queries. For the front running example, the user could tunnel the query through Tor [DMS04b] to preserve the privacy of his or her network address. Nevertheless, the server could still observe the user's desired domain name, and launch a successful front running attack.

The rudimentary data access models of PIR are one of the hindering factors in deploying successful PIR-based systems. These models are limited to retrieving a single bit or a block of bits [BS07, CGKS95, KO97] by an array index or by a textual keyword [CGN97]. These models have little use for deploying real-world systems relying on PIR. There is therefore a need for an extension of PIR's data access primitives to a more expressive model suitable for retrieval from structured data sources, such as from a relational database. We address this need by integrating PIR with the widely deployed SQL.

We begin with the concept of *dynamic SQL*, which is fundamental to modern applications relying on relational databases. *Dynamic SQL* is an incomplete SQL statement within a software system, meant to be fully constructed and executed at runtime [SKS05]; it is a flexible, efficient, and secure way of using SQL in software systems. The flexibility enables systems to construct and submit SQL queries to the database at runtime. Dynamic SQL is efficient because it requires only a single compilation that *prepares* the query for its subsequent executions. In addition, dynamic SQL is more secure because malicious SQL code injection is much more difficult. We observe that the *shape* or textual content of an SQL query prepared within a system is not private, but the constants the user supplies at runtime are private, and must be protected. For domain name registration, for example, the textual content of the query is exposed to the database, but only the textual keyword for the domain name is really private. For example, the shape of the dynamic query in Listing 1 is not private; the question mark `?` is used as a placeholder for a private value to be provided before the query is executed at runtime. Of note is the related observation made between parameterized SQL queries and parse tree validation [BWS05, HP05a]. In this context, runtime parse trees obtained from combining user inputs with parameterized queries are validated to ensure consistency with parse trees for programmer-specified

queries, thereby defeating SQL injection. Unlike valid inputs which only alter the semantics of a parse tree, SQL injection attempts to change both the syntax and semantics of a parse tree [HP05b].

Listing 1 An Example Dynamic SQL query (see Listing 6 in Appendix A for the corresponding database schema)

```
SELECT t1.domain, t1.expiry, t2.contact
FROM regdomains t1, registrar t2
WHERE (t1.reg_id = t2.reg_id) AND (t1.domain = ? )
```

Our approach to preserving query privacy over a relational database is based on hiding such private constants of a query. The client sends a *desensitized* version of the prepared SQL query appropriately modified to remove private information. The database executes this public SQL query, and generates appropriate cached indices to support further rounds of interaction with the client. The client subsequently performs a number of keyword-based PIR operations [CGN97] using the value for the placeholders against the indices to obtain the result for the query.

None of the existing proposals related to enabling privacy-preserving queries and robust data access models for private information retrieval makes the noted observation about the privacy of constants within an otherwise-public query. These include techniques that eliminate database optimization by localizing query processing to the user's computer [RPG07], problems on querying Database-as-a-Service [HMT04, HILM02], those that require an encrypted database before permitting private data access [SBC+07], and those restricted to simple keyword search on textual data sources [BSW09]. This observation is crucial for preserving the expressiveness and benefits of SQL, and for keeping the interface between a database and existing software systems from changing while building in support for user query privacy. Our approach improves over previous work with additional database optimization opportunities and fewer PIR operations needed to retrieve data. To the best of our knowledge, we are the first to propose a practical technique that leverages PIR to preserve the privacy of sensitive information in an SQL query over existing commercial and open-source relational database systems.

Our contributions. We address the problem of preserving the privacy of sensitive information within an SQL query using PIR. In doing this, we address two obstacles to deploying successful PIR-based systems. First, we develop a generic data access model for private information retrieval from a relational database using SQL. We show how to hide sensitive data within a query and how to use PIR to retrieve data from a relational

database. Second, we develop an approach for embedding PIR schemes into the well-established context and organization of relational database systems. It has been argued that performing a trivial PIR operation, which involves having a database send its entire data to the user, and having the user select the item of interest, is more efficient than running a computational PIR scheme [SC07]; however, as seen in Chapter 4, information-theoretic PIR schemes are much more efficient. We show how the latter PIR schemes can be applied in realistic scenarios, achieving both efficiency and query expressiveness. Since relational databases and SQL are the most influential of all database models and query languages, we argue that many realistic systems needing query privacy protection will find our approach quite useful.

The rest of this chapter is organized as follows: Section 5.2 provides some background on the relational model, SQL, and database indexing. Section 5.3 details the threat model, security, and assumptions for this work, while Section 5.4 provides a description of the approach for hiding sensitive constants within an SQL query. We provide detailed discussions of the algorithm in Section 5.5. Sections 5.6 and 5.7 present an overview of the prototype implementation, results of microbenchmarking and the experiments used to evaluate this prototype in greater depth. Section 5.8 concludes the chapter and suggests some future work.

5.2 Preliminaries

5.2.1 Indexing

A database index is a supplementary data structure used to access data from the database efficiently. Data is indexed either directly by the values of one or more attributes or by hashes (generally not cryptographic hashes) of those values. The attributes used to define an index form the *key*. Indices are typically organized into tree structures, such as B^+ trees where internal or non-leaf nodes do not contain data; they only maintain references to children or leaf nodes. Data is either stored in the leaf nodes, or the leaf nodes maintain references to the corresponding tuples (i.e., records) in the database. The number of nodes between the root and any leaf of a B^+ tree is constant, because the tree is balanced. Furthermore, the leaf nodes of B^+ trees may be linked together to enable sequential data access during range queries over the index; *range queries* return all data with key values in a specified range.

Hashed indices are specifically useful for *point queries*, which return a single data item for a given key. For many situations where efficient retrieval over a set of unique keys is

needed, hashed indices are preferred over B^+ tree indices. However, it is challenging to generate hash functions that will hash each key to a unique hash value. Many hashed indices used in commercial databases, for this reason, use data partitioning (bucketization) [HMT04] techniques to hash a range of values to a single bucket, instead of to individual buckets. Recent advances [BBdCR09, BZ07] in *perfect hash functions (PHF)* have produced a family of hash functions that can efficiently map a large set of n key values to a set of m integers without collisions, where n is less than or equal to m. A perfect hash function is *minimal* when $n = m$. These PHF can work with large sets of keys (on the order of billions), unlike earlier developments, such as gperf [Sch00], that can only manage small sets of keys.

5.3 Threat Model, Security and Assumptions

5.3.1 Security and Adversary Capabilities

Our main assumption is that the shape of SQL queries submitted by the users is public or known to the database administrator. Applicable practical scenarios include design-time specification of dynamic SQL by programmers, who expect the users to supply sensitive constants at runtime. Moreover, the database schema and all dynamic SQL queries expected to be submitted to, for example, a patent database, are not really hidden from the patent database administrator. Simultaneous protection of both the shape and constants of a query are outside of the scope of this work, and would likely require treating the database management system as other than a black box.

Our approach is sufficiently generic to allow an application to rely on any block-based PIR system, including single-server, multi-server, and coprocessor-assisted variants. We assume an adversary with the same capability as that assumed for the underlying PIR protocol. The two common adversary capabilities considered in theoretical private information retrieval schemes are the curious passive adversary and the Byzantine adversary [BS07, CGKS95]. Either of these adversaries can be a database administrator or any other insider to a PIR server.

A curious passive adversary can observe PIR-encoded queries, but should be incapable of decoding the content. In addition, it should not be possible to differentiate between queries or identify the data that makes up the result of a query. In our context, the information this adversary can observe is the desensitized SQL query from the client and the PIR queries. The information obtained from the desensitized query does not compromise

the privacy of the user's query, since it does not contain any private constants. Similarly, the adversary cannot obtain any information from the PIR queries because PIR protocols are designed to be resistant against an adversary of this capability.

A Byzantine adversary with additional capabilities is assumed for some multi-server PIR protocols [BS07, Gol07a]. In this model, the data in some of the servers could be outdated, or some of the servers could be down, malfunctioning or malicious. Nevertheless, the client is still able to compute the correct result and determine which servers misbehaved, and the servers are still unable to learn the client's query. Again, in our specific context, the adversary may compromise some of the servers in a multi-server PIR scenario by generating and obtaining the result for a substitute fake query or executing the original query on these servers, but modifying some of the tuples in the results arbitrarily. The adversary may respond to a PIR request with a corrupted query result or even desist from acting on the request. Nevertheless, all of these active attack scenarios can be effectively mitigated with a Byzantine-robust multi-server PIR scheme.

5.3.2 Data Size Assumptions

We service PIR requests using indexed data extracted from relational databases. The size of these data depends on the number of tuples resulting from the desensitized query. We note that even in the event that this *desensitized* query yields a small number of tuples (including just one), the privacy of the *sensitive part* of the SQL query *is not compromised*. The properties of PIR ensure that the adversary gains no information about the sensitive constants from observing the PIR protocol, over what he already knew by observing the desensitized query.

On the other hand, many database schemas are designed in a way that a number of relations will contain very few rows of data, all of which are meant to be retrieved and used by every user. Therefore, it is pointless to perform PIR operations on these items, since every user is expected to retrieve them all at some point. The adversary does not violate a user's query privacy by observing this public retrieval.

5.3.3 Avoiding Server Collusion

Information-theoretic PIR is generally more computationally efficient than computational PIR, but requires that the servers not collude if privacy is to be preserved; this is the same assumption commonly made in other privacy-preserving technologies, such as mix networks [Cha81] and Tor [DMS04b]. We present scenarios in which collusion among

servers is unlikely, yielding an opportunity to use the more efficient information-theoretic PIR.

The first scenario is when several independent service providers host a copy of the database. This applies to naturally distributed databases, such as Internet domain registries. In this particular instance, the problem of colluding servers is mitigated by practical business concerns. Realistically, the Internet domain database is maintained by different geographically dispersed organizations that are independent of the registrars that a user may query. However, different registrars would be responsible for the content's distribution to end users as well as integration of partners through banner ads and promotions. Since the registrars are operating in the same line of business where they compete to win users and deliver domain registry services, as well as having their own advertising models to reap economic benefits, there is no real incentive to collude in order to break the privacy of any user. In this model, it is feasible that a user would perform a domain name registration query on multiple registrars' servers concurrently. The user would then combine the results, without fear of the queries revealing its content. Additionally, individual service agreements can foreclose any chance of collusion with a third party on legal grounds. Users then enjoy greater confidence in using the service, and the registrars in turn can capitalize on revenue generation opportunities such as pay-per-use subscriptions and revenue-sharing ad opportunities.

The second scenario that offers less danger of collusion is when the query needs to be private only for a short time. In this case, the user may be comfortable with knowing that by the time the servers collude in order to learn her query, the query's privacy is no longer required.

Note that even in scenarios where collusion cannot be forestalled, our system can still use any computational PIR protocol; recent such protocols [AMG07] offer considerable efficiency improvements over previous work in the area, as seen in Chapter 4.

Listing 2 Example query with a WHERE clause featuring sensitive constants.

```
SELECT t1.contact, t1.email, t2.created, t2.expiry
FROM registrar t1, regdomains t2
WHERE (t1.reg_id = t2.reg_id) AND (t2.created > 20090101) AND
      (t2.domain = 'anydomain.com')
```

5.4 Hiding Sensitive Constants

5.4.1 Overview

Our approach is to preserve the privacy of sensitive data within the WHERE and HAVING predicates of an SQL query. For brevity, we will focus on the WHERE clause; a similar processing procedure applies to the HAVING clause. This may require the user (or application) to specify the constants that may be sensitive. For the example query in Listing 2, the domain name and the creation date may be sensitive.

Our approach splits the processing of SQL queries containing sensitive data into two stages. In the first stage, the client computes a public subquery, which is simply the original query that has been stripped of the predicate conditions containing sensitive data. The client sends this subquery to the server, and the server executes it to obtain a result for the subquery. The desired result for the original query is contained within the subquery result, but the database is not aware of the particular tuples that are of interest.

In the second stage, the client performs PIR operations to retrieve the tuples of interest from the subquery result. To enable this, the database creates a cached index on the subquery result and sends metadata for querying the index to the client. The client subsequently performs PIR retrievals on the index and finally combines the retrieved items to build the result for the original query. An alternative approach to storing materialized tuples or subquery results in an index is to maintain index entries as references to actual database tuples. In other words, each index entry will simply store keys and reference database tuples. We can consider an index built using this approach as maintaining a 'view' of the subquery result (i.e., no data materialization). The approach saves storage space, but will incur considerable performance overhead. PIR queries over such indices necessarily require individual fetching of all tuples in the original subquery result (at worst), or systematic range-based fetches (at best). These operations will be slow and much more complex to implement. For these reasons, our approach explores indices built on materialized data.

The important benefits of this approach as compared with the previous approach by Reardon et al. [RPG07] are the optimizations realizable from having the database execute the non-private subquery, and the fewer number of PIR operations required to retrieve the data of interest. In addition, the PIR operations are performed against a cached index, which will usually be smaller than the complete database. This is particularly true if there are joins and non-private conditions in the WHERE clause that constrain the tuples in the query result. In particular, a single PIR query is needed for point queries on hash

table indices, while range queries on B^+ tree indices are performed on fewer data blocks. Figure 5.1 illustrates the sequence of events during a query evaluation.

We note that often, the non-private subqueries will be common to many users, and the database does not need to execute them every time a user makes a request. Nevertheless, our algorithm details, presented next in Section 5.4.2, show the steps for processing a subquery and generating indices. Such details are useful in an *ad hoc* environment, where the shape of a query is unknown to the database *a priori*; each user writes his or her own query as needed. Our assumption is that revealing the shape of a query will not violate users' privacy (see Section 5.3).

5.4.2 Algorithm

We describe our algorithm with an example by assuming an information-theoretic PIR setup with two replicated servers. We focus on hiding sensitive constants in the predicates of the WHERE clause. The algorithm details for the SELECT query in Listing 2 follow. We assume the date `20090101` and the domain `anydomain.com` are private.

Step 1: The client builds an attribute list, a constraint list, and a desensitized SELECT query, using the attribute names and the WHERE conditions of the input query. We refer to the desensitized query as a *subquery*.

To begin, initialize the attribute list to the attribute names in the query's SELECT clause, the constraint list to be empty, and the subquery to the SELECT and FROM clauses of the original query.

- *Attribute list:* {`t1.contact`, `t1.email`, `t2.created`, `t2.expiry`}
- *Constraint list:* {}
- *Subquery:*
```
SELECT t1.contact, t1.email, t2.created, t2.expiry
FROM registrar t1, regdomains t2
```

Next, consider each WHERE condition in turn. If a condition features a private constant, then add the attribute name to the *attribute list* (if not already in the list), and add (attribute name, constant value, operator) to the *constraint list*. Otherwise, add the condition to the subquery.

On completing the above steps, the attribute list, constraint list, and subquery with reduced conditions for the input query become:

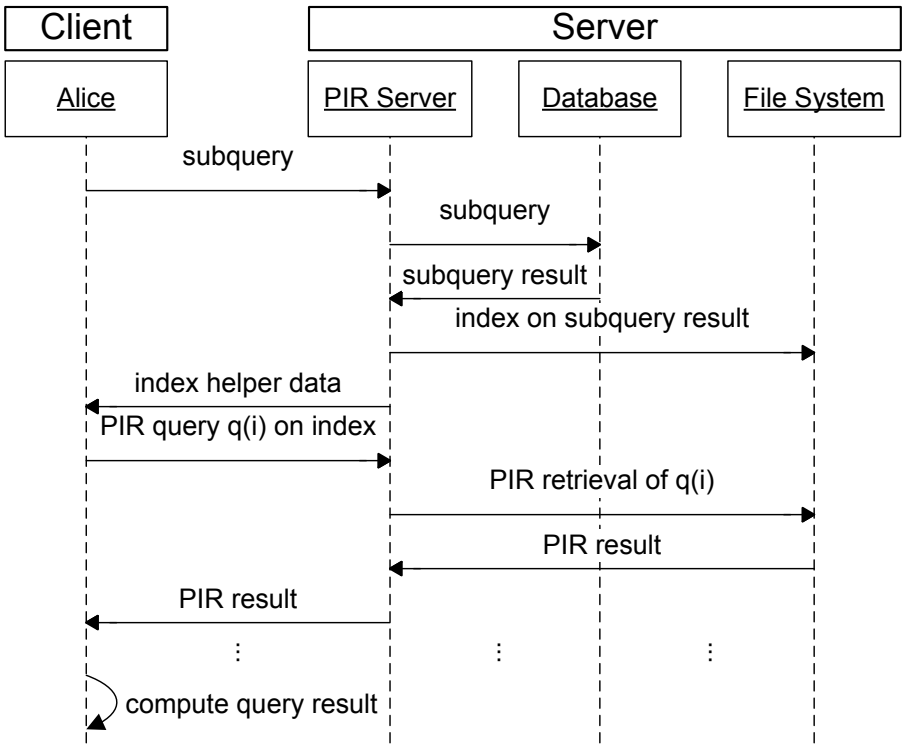


Figure 5.1: A sequence diagram for evaluating Alice's private SQL query using PIR.

- *Attribute list:* {`t1.contact`, `t1.email`, `t2.created`, `t2.expiry`, `t2.domain`}
- *Constraint list:* {`(t2.created,20090101,>)`,`(t2.domain,'anydomain.com',=)`}
- *Subquery:*
  ```
  SELECT t1.contact,t1.email,t2.created,t2.expiry,t2.domain
  FROM registrar t1, regdomains t2 WHERE (t1.reg_id = t2.reg_id)
  ```

Step 2: The client sends the subquery, a key attribute name, and an index file type to each server.

The key attribute name is selected from the attribute names in the constraint list — `t2.created, t2.domain` in our example. The choice may either be random, made by the application designer, or determined by a client optimizer component with some domain knowledge that could enable it to make an optimal choice. One way to make a good choice is to consider the *selectivity* — the ratio of the number of distinct values taken to the total

number of tuples — expected for each constraint list attribute, and then choose the one that is most selective. This ensures the selection of attributes with unique key values before less selective attributes. For example, in a patent database, the patent number is a better choice for a key than the author's gender. A poor choice of key can lead to more rounds of PIR queries than necessary. Point queries on a unique key attribute can be completed with a single PIR query. Similarly, a good choice of key will reduce the number of PIR queries for range queries. For the example query, we choose `t2.domain` as the key attribute name.

For the index file type, either a PHF or a B^+ tree index type is specified. Other index structures may be possible, with additional investigation, but these are the ones we currently support. More details on the selection of index types is provided below.

Step 3: Each server executes the subquery on its relational database, generates a cached index of the specified type on the subquery result, using the key attribute name, and returns metadata for searching the indices to the client.

The server computes the size of the subquery result. If it can send the entire result more cheaply than performing PIR operations on it, it does so. Otherwise, it proceeds with the index generation. For hash table indices, the server first computes the perfect hash functions for the key attribute values. Then it evaluates each key and inserts each tuple into a hash table. The metadata that is returned to the client for hash-based indices consists of the PHF parameters, the count of tuples in the hash table, and some PIR-specific initialization parameters.

For B^+ tree indices, the server bulk inserts the subquery result into a new B^+ tree index file. B^+ tree bulk insertion algorithms provide a high-speed technique for building a tree from existing data [APV02]. The server also returns metadata to the client, including the size of the tree and its first data block (the root). Generated indices are stored in a disk cache external to the database.

Step 4: The client receives the responses from the servers and verifies they are of the appropriate length. For a Byzantine-robust multi-server PIR, a client may choose to proceed in spite of errors resulting from non-responding servers or from responses that are of inconsistent length.

Next, the client performs one or more keyword-based PIR queries, using the value associated with the key attribute name from the constraint list, and builds the desired query result from the data retrieved with PIR.

The encoding of a private constant in a PIR query proceeds as follows. For PIR queries over a hash-based index, the client computes the hash for the private constant using the

PHF functions derived from the metadata[1]. This hash is also the block number in the hash table index on the servers. This block number is input to the PIR scheme to compute the PIR query for each server. For a B^+ tree index, the user compares the private value for the key attribute with the values in the root of the tree. The root of the tree is extracted from the metadata it receives from the server. Each key value in this root maintains block numbers for the children blocks or nodes. The block number corresponding to the appropriate child node will be the input to the PIR scheme.

For hash-based indices, a single PIR query is sufficient to retrieve the block containing the data of interest from the hash table. For B^+ tree indices, however, the client uses PIR to traverse the tree. Each block can hold some number m of keys, and at a block level, the B^+ tree can be considered an m-ary tree. The client has already been sent the root block of the tree, which contains the top m keys. Using this information, the client can perform a single PIR block query to fetch one of the m blocks so referenced. It repeats this process until it reaches the leaves of the tree, at which point it fetches the required data with further PIR queries. The actual number of PIR queries depends on the height of the (balanced) tree, and the number of tuples in the result set. Traversals of B^+ tree indices with our approach are oblivious in that they leak no information about the node access pattern; we realize retrieval of a node's data as a PIR operation over the data set of all nodes in the tree. In other words, it does not matter which particular branch of a B^+ tree is the location for the next block to be retrieved. We do not restrict PIR operations to the subset of blocks in the subtree rooted at that branch. Instead, each PIR operation considers the set of blocks in the entire B^+ tree. Range queries that retrieve data from different subtrees leak no information about to which subtree a particular piece of data belongs. The only information the server learns is the number of blocks retrieved by such a query. Therefore, specific implementations may utilize dummy queries to prevent the server from leaning the amount of useful data retrieved by a query [SCM05].

To compute the final query result, the client applies the other private conditions in the constraint list to the result obtained with PIR. For the example query, the client filters out all tuples with `t2.created` not greater than `20090101` from the tuple data returned with PIR. The remaining tuples give the final query result.

Capabilities for dealing with complex queries can be built into the client. For example, it may be more efficient to request a single index keyed on the concatenation of two attributes than separate indices. If the client requests separate indices, it will subsequently perform PIR queries on each of those indices, using the private value associated with each

[1]Using the CMPH Library [BBdCR09] for example, the client saves the PHF data from the metadata into a file. It reopens this file and uses it to compute a hash by following appropriate API call sequences.

attribute from the constraint list. Finally, the client combines the partial results obtained from the queries with set operations (union, intersection), and performs local filtering on the combined result, using private constant values for any remaining conditions in the constraint list to compute the final query result. The client thus needs query-optimization capabilities in addition to the regular query optimization performed by the server. This is an open area of work closely related to database optimization.

5.5 Discussion

In this section, we discuss important architectural components and design decisions related to the algorithm presented in Section 5.4.

5.5.1 Parsing SQL Queries

The algorithm parses an input query — the WHERE and HAVING clauses in particular. Other subclauses of the SELECT statements, such as GROUP BY and ORDER BY, can either be processed as part of a subquery or applied on the result obtained with PIR. Specific implementations can adopt the mature parsers developed with open source and commercial databases.

The expression tree provides an easy way to construct the desensitized query and the constraint list. The parsing process builds an expression tree representation for the WHERE clause conditions. The internal nodes of this expression tree typically contain arithmetic, relational, and logical operators, while the leaf nodes consist of attribute names and constants. Any WHERE clause predicate expression can be a join, a non-private condition or a private condition. The latter contains a sensitive constant value, whereas the former two do not. Our parser allows the user to tag sensitive constants with the symbol "#" to differentiate them from public constants. For example, the sensitive constant `'20090511'` is tagged in this query: `SELECT * FROM table WHERE n = 20090605 AND p = #20090511`. Each WHERE clause condition is related to another condition with the logical AND. Logical OR conditions are not considered as expression delimiters, but disjunct multiple subexpressions in the same condition. Typically, relational databases convert the WHERE clause conditions in the input query to an equivalent set of conditions in the conjunctive normal form, to facilitate query optimization.

For an example of AND and OR, consider the two SELECT queries below, which differ only in their WHERE clause conditions.

(i) `SELECT * FROM table WHERE a = 'SQL' AND b = 'LEX'`

(ii) `SELECT * FROM table WHERE a = 'SQL' OR b = 'LEX'`

The client can compute the result for (i) using either one or two indices, whereas it requires two indices to compute the result for (ii). To compute the result for (i) with a single index, the client requests an index for `a` or `b` because both of the conditions in the WHERE clause can only be true if one of them is true. If it requests an index for `a`, it will first perform keyword-based PIR using the literal `'SQL'` over this index, and then filter the result obtained with the second condition `b = 'LEX'`. To compute either (i) or (ii) with two indices, the client requests indices for both `a` and `b`, and then performs two keyword-based PIR searches using the string literals `'SQL'` and `'LEX'` over the respective indices. Finally, the client computes the intersection of the tuples in the two PIR results to obtain the result for (i), or it computes the union to obtain the result for (ii).

We note that a worst-case query scenario having several private conditions combined with an OR operator will have storage and computational costs linear in the number of unique attribute names used with the private conditions. In certain circumstances, it may be possible to eliminate the storage cost by maintaining references to the tuple data in the database rather than maintaining a materialized copy in an index.

Currently, logical NOT conditions cannot be processed with PIR. We are unable to find any practical PIR scenario to justify its use. For example, performing PIR queries on a patent database will generally not require a NOT operator. We prescribe client-side processing for NOTs, after using PIR to retrieve the data required for evaluating the condition.

This expression tree is traversed twice. The first traversal lists the desensitized query's WHERE conditions, which includes all joins and all non-private conditions. The logical AND operator combines the joins and the non-private conditions. The boolean true value can serve as a placeholder for every private condition. For example, the actual WHERE clause for the subquery in Listing 3 can be `WHERE (t1.reg_id = t2.reg_id) AND true AND true`, which can be subsequently optimized. The second traversal lists the private conditions, which are used to build the constraint list. The above two traversals can be combined in a single pass to output the public WHERE conditions and the private conditions.

Listing 3 Example subquery from the query of Listing 2.

```
SELECT t1.contact, t1.email, t2.created, t2.expiry, t2.domain
FROM registrar t1, regdomains t2
WHERE (t1.reg_id = t2.reg_id)
```

5.5.2 Indexing Subquery Results

For many general purposes, it may be impractical to execute the desensitized query and generate an index on the query result for every request. The use of an index cache addresses some of the cost, because the database can use the same cached index to serve multiple PIR queries (with the same private attributes, though not necessarily the same private constants) from multiple users. This mitigates the computational costs for generating indices. An exception for the use of a cache is when the shape of the input query is unpredictable, especially in an environment where the users make *ad hoc* queries. In this case, a separate index must be generated for each unique query.

5.5.3 Database Servers

Practical implementations could use any commercial or open-source database server to execute the desensitized query. The client does not need to install database client programs to query the database server in the privacy-friendly manner we describe; however, the client will need an installation of the private SQL client that implements the client-side logic of the algorithm. Similarly, a program that implements the server-side logic of the algorithm must be installed at the server.

5.5.4 Processing Specific Conditions

We provide an overview on how to deal with private constants in specific conditions of the WHERE clause. In particular, we consider simple conditions, as well as specialized conditions such as BETWEEN, LIKE, and IN.

A simple WHERE clause condition consists of the general form *column relop literal* or *literal relop column*, where *relop* is a relational operator, such as $=, <>, <, >, <=$, and $>=$. If the *column* is used to index a query result, then the *literal* will be used as input to the keyword-based PIR. The operator "=" indicates a point query. If the key attribute *column* is unique, then a single result is expected; either a hash or a B^+ tree index is

appropriate. On the other hand, a B^+ tree is preferred for non-unique key values, since there may be multiple tuples in the query result. The other operators, which imply range queries, require B^+ tree indices. The *literal* or its next or previous neighbours from the domain of values for the data type, in sorted or lexicographical order, provide one of the values for the range search. The other value is determined from the smallest or largest value in the domain for the data type. The input values for range search for the condition `t2.created > 20090101`, for example, are `(20090102, 99991231)`.

A BETWEEN condition has the general form *column* $BETWEEN$ $literal_1$ AND $literal_2$, which is equivalent to the condition *column* $>=$ $literal_1$ AND *column* $<=$ $literal_2$. This condition is processed as a range query on the two literal values.

A LIKE condition has the form *column* $LIKE$ *literal*, where *literal* is a search condition that involves one or more wildcards, such as % and _. The _ allows for the matching of a single character, while the % allows for matching strings of any length, including zero-length strings. Prefix-based conditions, such as `domain LIKE 'some%'`, and suffix-based ones, such as `domain LIKE '%main.com'` can easily be processed with a B^+ tree index over the attribute, or the reverse of the attribute, respectively. Other variants are more easily processed in the client; the client would first retrieve the data from the server with PIR, and then perform a more sophisticated filtering on the result using the wildcard expression.

An IN condition has the general form *column* IN ($literal_1$,$literal_2$, ...). If the attribute *column* has unique values, then the tuple associated with each literal can be retrieved with a point query on the same index over the *column* attribute. Some PIR implementations, such as Percy++ [Gol07b], can simultaneously retrieve multiple blocks for a set of literal values in a single query. Otherwise, a combination of range and point queries will be required. The client optimizer can be built to intelligently combine literal values to reduce the overall number of PIR queries.

Client-side support for database function evaluation is required when private constants are used as function parameters in a WHERE clause expression. Such functions can be evaluated before the data required are retrieved with PIR, or afterwards. The latter follows for functions that take private constants *and* attribute names as parameters.

We note that special WHERE clause conditions, such is IS $NULL$ and IS NOT $NULL$, do not require any private constants. It would suffice to include them in the desensitized query in many situations. Alternatively, they could be processed locally, especially for *ad hoc* queries, if they are considered to reveal sensitive information about the tuples of interest.

Finally, an implementation may decide to localize the processing of all the above conditions, as well as other conditions of the WHERE clause. The approach to adopt depends

on the amount of optimization the client is capable of performing and the requirements of the application domain.

5.6 Implementation and Microbenchmarks

5.6.1 Implementation

We developed a prototype implementation of our algorithm to hide the sensitive portions of SQL queries using generally available open-source C++ libraries and databases. We developed a command-line tool to act as the client, and a server-side database adapter to provide the functions of a PIR server. For the PIR functions, we used the Percy++ PIR Library [Gol07b, Gol07a], which offers three varieties of privacy protection: computational, information theoretic and hybrid (a combination of both). We extended Percy++ to support keyword-based PIR. For generating hash table indices for point queries, we used the C Minimal Perfect Hash (CMPH) Library [BBdCR09, BZ07], version 0.9. We used the API for CMPH to generate minimal perfect hash functions for large data sets from query results; these perfect hash functions require small amounts of disk storage per key. For building B^+ tree indices for range queries on large data sets, we used the Transparent Parallel I/O Environment (TPIE) Library [Dep09, VV95]. Finally, we base the implementation on the MySQL [Sun09] relational database, version 5.1.37-1ubuntu5.1.

5.6.2 Experimental Setup

We began evaluating our prototype implementation using a set of six whois-style queries from Reardon et al. [RPG07], which is the most appropriate existing microbenchmark for our approach. We explored tests using industry-standard database benchmarks, such as the Transaction Processing Performance Council (TPC) [Tra09] benchmarks, and open-source benchmarking kits such as Open Source Development Labs Database Test Suite (OSDL DTS) [WT09], but none of the tests from these benchmarks is suitable for evaluating our prototype, as their test databases cannot be readily fitted into a scenario that would make applying PIR meaningful. For example, a database schema that is based on completing online orders will only serve very limited purpose to our goal of protecting the privacy of sensitive information within a query.

We ran the microbenchmark tests using two whois-style data sets, similar to those generated for the evaluation of TransPIR [RPG07] (we provide a brief discussion of the

TransPIR solution in Section 3.3). The smaller data set consisted of 10^6 domain name registration tuples, and 0.75×10^6 registrar and registrant contact information tuples. The second data set similarly consisted of 4×10^6 and 3×10^6 tuples respectively. We describe the evaluation queries and the two database relations respectively in Listing 8 and Listing 7 of Appendix A. We chose the predicate parameters for the benchmark queries to ensure query selectivity values (ratio of the number of matching tuples in the query result — not the subquery result — to the total number of tuples in the database) similar to those used in the original benchmarking of TransPIR [RPG07]. The respective values for benchmark queries Q1 through Q6 for the small data set were 1.00×10^{-6}, 2.00×10^{-5}, 4.20×10^{-5}, 5.90×10^{-5}, 1.33×10^{-6}, and 3.87×10^{-2}. For the large data set they were 2.50×10^{-7}, 2.00×10^{-5}, 4.20×10^{-5}, 5.90×10^{-5}, 2.50×10^{-7}, and 4.20×10^{-5}.

The measurements for all test queries are based on the default behaviour of the TPIE Library with respect to determining the branching factor λ for B^+ tree indices. The following expression shows the computation of branching factor with this default configuration:

$$\lambda = \left\lfloor \frac{\gamma \times size(os\_block) - size(BID) - size(size\_t)}{size(Key) + size(BID)} \right\rfloor$$

Where γ, os_block, BID, $size\_t$, and Key are respectively the data logical blocking factor, operating system block, block ID, C++ $size\_t$ data type, and the key. $size(x)$ is the size of x in bytes. Specifically for our experimental setup for the large data set, the branching factor for indices over integer keys is 2730, and it is 409 for indices over character keys. For the small data set, these values are respectively 1634 and 215. Our implementation stores integer keys in 8 bytes, and character keys in 72 bytes. The branching factor values are based on a block size of 32 KB ($\gamma = 8$), and 16 KB ($\gamma = 4$), where $size(os\_block) = 4096$ bytes. The actual fill factor (again, the default for the TPIE Library) is 0.6 for internal B^+ tree nodes. Later in our evaluation, we report the disk sizes for indices built for our experiment on the queries with complex conditions (Table 5.2).

We ran all experiments on a test machine with two quad-core 2.50 GHz Intel Xeon E5420 CPUs, 8 GB RAM, and running Ubuntu Linux 9.10. We used the information-theoretic PIR support of Percy++, with two database replicas. The server also runs a local installation of a MySQL database.

5.6.3 Result Overview

The results from the benchmark tests indicate that while our current prototype incurs some storage and computational costs over non-private queries, the costs seem entirely

acceptable for the added privacy benefit (see Table 5.1 and Figure 5.2 later in this section and Table 5.2 and Figure 5.3 in Section 5.7). In addition to being able to deal with complex queries and leverage database optimization opportunities, our prototype performs much better than the TransPIR prototype from Reardon et al. [RPG07] — between 7 and 480 times faster for equivalent data sets.

The most indicative factor of performance improvements with our prototype is the reduction in the number of PIR queries in most cases. Other factors that may affect the validity of the result, such as variations in implementation libraries, are assumed to have negligible impact on performance. Our work is based on the same PIR library as that of Reardon et al. [RPG07]. Our comparison is based on the measurements we took by compiling and running the code for TransPIR on the same experimental hardware platform as our prototype. We initially attempted to run the microbenchmarking tests for the larger data with TransPIR on the development hardware platform for our prototype, but was limited by this commodity hardware because TransPIR requires a 64-bit processor and a minimum of 6 GB RAM to index or preprocess the larger data set. The development hardware (not the test machine above) had a 32-bit processor and only 3 GB RAM.

5.6.4 Microbenchmark Experiment

We executed the six whois-style benchmark queries over the data sets and obtained measurements for the time to execute the private query, the number of PIR queries performed, the number of tuples in the query results, the time to execute the subquery and generate the cached index, and the total data transfer between the client and the two PIR servers. Table 5.1 shows the results of the experiment (a plot of the same result for the large database is shown in Figure 5.2). The cost of indexing (QI) can be amortized over multiple queries. The indexing measurements for BTREE (and HASH) consist of the time spent retrieving data from the database (subquery execution), writing the data (subquery result) to a file and building an index from this file. Since TransPIR is not integrated with any relational database, it does not incur the same database retrieval and file writing costs. However, TransPIR incurs a one-time preprocessing cost (QI) which prepares the database for subsequent query runs. Comparing this cost to its indexing counterpart with our BTREE and HASH prototypes shows that our methods are over an order of magnitude faster.

Table 5.1: Experimental results for microbenchmark tests compared with those of Reardon et al. [RPG07]. **BTREE**, **HASH**, and **TransPIR** are respectively the timing result for our B^+ tree prototype, our hash table prototype, and from TransPIR [RPG07]; **Time** = time to evaluate private query, **PIRs** = number of PIR operations performed, **Tuples** = count of rows in query result (not subquery result), **QI** = timing for subquery execution and index generation, **Xfer** = total data transfer between the client and the two PIR servers.

Small database (.75 M contacts, 1 M registrations, and block size of 16 KB)

Query	Approach	Time (s)	PIRs	Tuples	QI (s)	Xfer (KB)
Q1	**HASH**	0	1	1	4	64
	BTREE	6	4	1	9	256
	TransPIR	7	2	1	120	128
Q2	**BTREE**	3	3	20	7	192
	TransPIR	76	23	20	120	1,472
Q3	**BTREE**	3	3	42	7	192
	TransPIR	149	45	42	120	2,880
Q4	**BTREE**	13	3	59	8	256
	TransPIR	217	62	59	120	3,968
Q5	**BTREE**	5	4	1	13	256
	TransPIR	10	3	1	120	192
Q6‡	**BTREE**	5	3	29	13	192
	TransPIR	558	111	42	—	7,104

Large database (3 M contacts, 4 M registrations, and block size of 32 KB)

Query	Approach	Time (s)	PIRs	Tuples	QI (s)	Xfer (KB)
Q1	**HASH**	2	1	1	16	128
	BTREE	4	3	1	38	384
	TransPIR	25	2	1	1,017	256
Q2	**BTREE**	5	4	80	32	512
	TransPIR	999	83	80	1,017	10,624
Q3	**BTREE**	5	4	168	32	512
	TransPIR	2,055	171	168	1,017	21,888
Q4	**BTREE**	6	5	236	37	640
	TransPIR	2,885	240	236	1,017	30,720
Q5	**BTREE**	5	3	1	67	384
	TransPIR	37	3	1	1,017	384
Q6‡	**BTREE**	5	4	168	66	512
	TransPIR	3,087	253	127	—	32,384

‡We reproduced TransPIR's measurements from [RPG07] for query Q6 because we could not get TransPIR to run Q6 due to program errors. The '—' under QI indicates measurement missing from [RPG07].

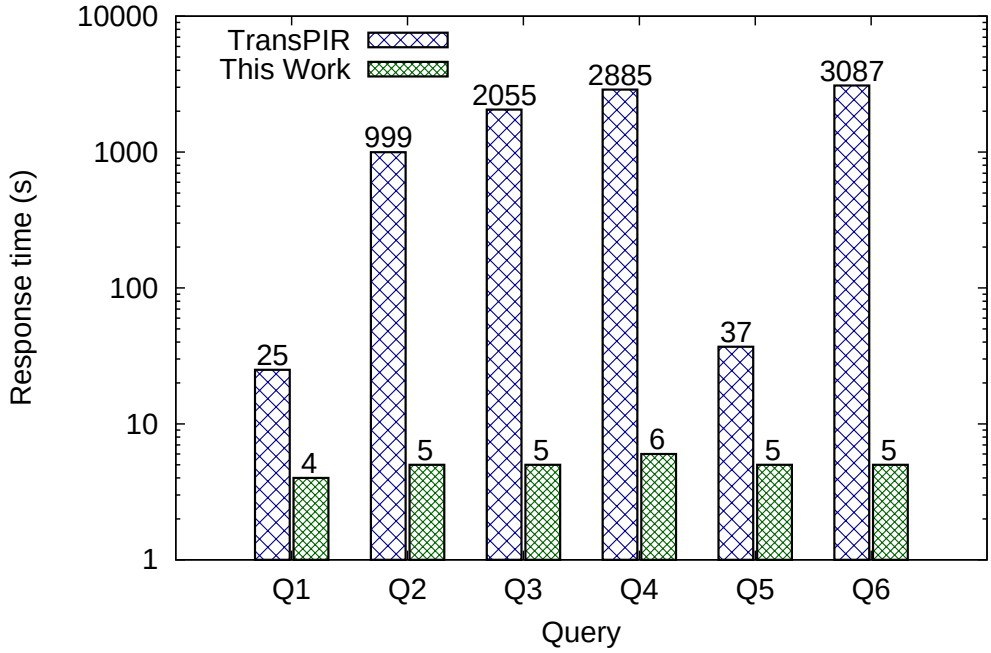


Figure 5.2: Comparing microbenchmarking results for the large data set. The y-axis is on a logarithmic scale.

5.6.5 Discussion

The empirical results for the benchmark tests reflect the benefit of our approach. For all of the tests, we mostly base our comparison on the timing for query evaluation with PIR (Time), and sometimes on the index generation timing (QI). The time to transfer data between the client and the servers is directly proportional to the amount of data (Xfer), but we will not use it for comparison purposes because the test queries were not run over a network.

Our hash index (HASH) prototype performs the best for query Q1 on both data sets, followed by our B^+ tree (BTREE) prototype; it achieves better performance for the large set. The query of Q1 is a point query having a single condition on the domain name attribute.

Query Q2 is a point query on the `expiry_date` attribute, with the query result expected to have multiple tuples. Again, our BTREE prototype outperforms TransPIR by a significant margin for both data sets; the improvement is most noticeable for the large data set. The number of PIR queries required to evaluate Q2 with BTREE is 5% of the

number required by TransPIR. A similar trend is repeated for Q3, Q4 and Q6. Note that the HASH prototype could not be used for Q2 because hash indices accept unique keys only; it can only return a single tuple in its query result.

Query Q3 is a range query on the `expiry_date` attribute. Our BTREE prototype respectively was approximately 50 and 411 times faster than TransPIR for the small and large data sets. Of note is the large number of PIR queries that TransPIR needs to evaluate the query; for the large data set, our BTREE prototype requires only 2% of that number. We observed a similar trend for Q4, where BTREE was 17 and 480 times faster for the small and large sets respectively. This query features two conditions in the SQL WHERE clause. The combined measured time for BTREE — the time taken to both build an index to support the query and to run the query itself — is still 10 and 67 times faster than the time it takes TransPIR to execute the query alone.

Query Q5 is a point query with a single join. For the large data set, it took BTREE only about 14% of the time it took TransPIR. We observed the time our BTREE spent in executing the subquery to dominate; only a small fraction of the time is spent building the B^+ tree index.

Our BTREE prototype similarly performs faster for Q6, with an order of magnitude similar to Q2, Q3, and Q4.

In all of the benchmark queries, the proposed approach performs better than TransPIR because it leverages database optimization opportunities, such as for the processing of subqueries. In contrast, TransPIR assumes a type of block-serving database that cannot give any optimization opportunity. Therefore, in our system the client is relieved from having to perform many traditional database functions, such as query processing, in addition to its regular PIR client functions.

5.7 Complex Query Evaluation

In addition to the above microbenchmarks, we performed two other experiments to evaluate our prototype. The first of these studies the behaviour of our prototype on complex input queries, such as aggregate queries, BETWEEN and LIKE queries, and queries with multiple WHERE clause conditions and joins. Each of these complex queries has varying privacy requirements for its sensitive constants. The second experiment tests whether our prototype leverages database optimization. We run the first experiment on the same hardware configuration as the microbenchmark tests, and the second experiment on the developmental hardware platform. For the test data, query selectivity for complex queries

CQ1 through CQ5 are 3.70×10^{-6}, 5.05×10^{-2}, 2.11×10^{-6}, 1.30×10^{-3}, and 5.34×10^{-6} for the small data set. Similarly for the large data set, the values are 5.70×10^{-7}, 5.12×10^{-2}, 1.58×10^{-6}, 9.52×10^{-7}, and 1.50×10^{-6}.

5.7.1 Result Overview

The results obtained from the experiments demonstrate the benefits of our approach for dealing with complex queries. While the storage and computational costs (over non-private querying) remain, the overall performance and resource requirements are still reasonable for the added privacy benefit. The prototype requires additional storage for two types of indices used for PIR operations. The first type of index is generated for a particular shape of query, over one or more key attributes or combinations of attributes. These types of indices need permanent storage in the same manner as native indices for relational databases. The second type of index is used in an *ad hoc* environment, where the tuples in a subquery result can be constrained in an unpredictable manner, with one or more WHERE clause conditions and joins. These latter indices must be generated as needed. In most practical situations, it should be possible for indices to be based on the former type, just like most software systems rely on indices prebuilt by the database to efficiently run their queries.

5.7.2 Experiments on Queries with Complex Conditions

We describe and present the results of experiments that examined the behaviour of our prototype when supplied with SQL queries that are more complex than the above microbenchmarks. We provide a number of synthetic query scenarios having different requirements for privacy, the corresponding SQL queries with appropriate tagging for the condition involving sensitive data, and the measurements. As mentioned above, our SQL parser uses the "#" character to tag private conditions; we include that tag in the SQL queries we present in Listing 9 of Appendix A. We used the same database schema (see Listing 7 of Appendix A) as the microbenchmarks. The measurements show execution duration for the original query without privacy provision over the MySQL database, the same query after removal of conditions with sensitive information over the MySQL database, and several other measurements taken from within our prototype using a B^+ tree index. All of the measurements are reported in Table 5.2. We show a breakdown of the timings for the large database in Figure 5.3.

Private point query (CQ1). The task is to obtain a domain name record from the whois server without revealing the sensitive domain name information.

Private range query (CQ2). The ICANN Security and Stability Advisory Committee may be interested in performing an investigation on some registrars, with IDs ranging from 198542 to 749999. The task is to privately obtain some domain name information without revealing the range of IDs for the registrars. We show the query to obtain registration records with status in the set (1, 4, 5, 7, 9), and expiration dates between 20090101 and 20091031, without revealing the registrar ID range.

Private aggregate point query (CQ3). The task is to compute the total number of registrations sponsored by a particular registrar in a privacy-preserving manner. The registrar ID is sensitive.

Non-private LIKE query (CQ4). The task is to efficiently retrieve a single domain name record from a whois server with some amount of privacy. In other words, a user wants to reveal a prefix of the domain name to improve performance, while still preventing the adversary from learning the exact textual domain name. Since many long domain names have a common prefix, the user intends to leverage that knowledge to improve query performance.

Private LIKE query (CQ5). The task is to retrieve registration records from a whois server without revealing the LIKE wildcard.

Results. We see from Table 5.2 that in most cases, the cost to evaluate the subquery and create the index dominates the total time to privately evaluate the query (BTREE), while the time to evaluate the query on the already-built index (Time) is minor. An exception is CQ2, which has a relatively small subquery result (rTuples), while having to do dozens of (consequently smaller) PIR operations to return thousands of results to the overall range query. Note that in all but CQ2, the time to privately evaluate the query on the already-built index is at most a few seconds longer than performing the query with no privacy at all; this underscores the advantage of using cached indices.

We note from our results that it is much more costly to have the client simply download the cached indices. We observe, for example, that it will take about 5 times as long, for a user with 10 Mbps download bandwidth, to download the index for CQ5 on the large data set. Figure 5.4 compares the roundtrip time for queries with our approach to the time it takes to download the database over a 9 Mbps connection, which is representative of the average bandwidth of Internet users from Canada and the US in 2010 [Ook10]. Moreover, this trivial download of data is impractical for devices with low bandwidth and storage (e.g., mobile devices).

Table 5.2: Measurements taken from executing five complex SQL queries with varying requirements for privacy. **oQm** = timing for executing original SQL query directly against a MySQL database, **BTREE** = overall timing for meeting privacy requirements with our B^+ tree prototype,**Time** = time to evaluate private query within BTREE, **PIRs** = number of PIR operations performed, **Tuples** = number of records in final query result, **rTuples** = number of indexed records in subquery result, **Xfer** = total data transfer between the client and the two PIR servers, **Size** = temporary storage space for cached index.

Small database with .75 M contact records, 1 M registration records

Query	oQm (s)	BTREE (s)	Time (s)	PIRs	Tuples	rTuples	Xfer (KB)	Size (MB)
CQ1	**0**	**8**	**2**	3	1	328,805	192	110.57
CQ2	**0**	**2**	**2**	17	686	13,594	1,088	4.57
CQ3	**0**	**13**	**2**	3	1	473,646	192	157.82
CQ4	**0**	**0**	**0**	2	1	768	128	0.32
CQ5	**0**	**17**	**3**	4	4	749,472	256	251.82

Large database with 3 M contact records, 4 M registration records

Query	oQm (s)	BTREE (s)	Time (s)	PIRs	Tuples	rTuples	Xfer (KB)	Size (MB)
CQ1	**2**	**31**	**2**	3	1	1,753,144	384	579.63
CQ2	**1**	**15**	**13**	41	3,716	72,568	5,248	25.13
CQ3	**0**	**80**	**3**	3	1	631,806	384	209.38
CQ4	**2**	**25**	**5**	3	1	1,050,300	384	348.63
CQ5	**2**	**69**	**3**	3	6	4,000,000	256	1,324.13

5.7.3 Database Optimization Experiments

We studied the overall response of our prototype to determine the benefits accrued from database optimization. The experimental MySQL database runs mostly with the default settings. The only change made was reducing the default number of user connections to free up memory for other processes running on the machine. In other words, we did not tune the database for optimal performance in the course of our previous experiments.

Most databases cache query plans and small-sized query results when a query is executed for the very first time. Subsequent executions of the same query will be more responsive by reusing the cached plan and result. For this experiment, we disabled cache usage by flushing the relations in our database before running each query. Flushing relations in MySQL closes the open relations and flushes the query cache. This ensures the

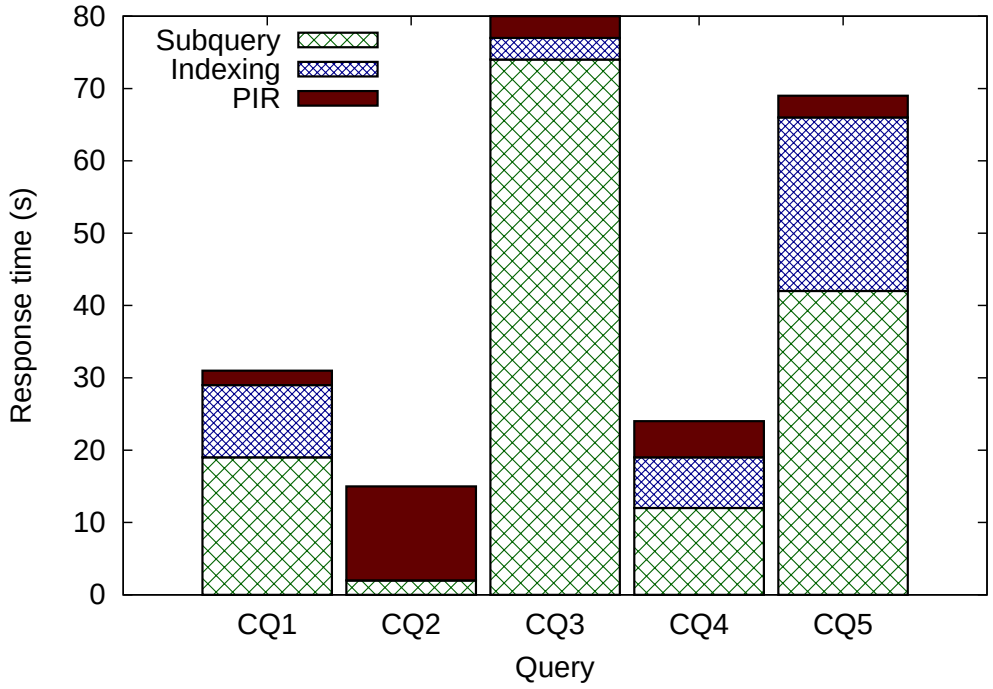


Figure 5.3: Breakdown of result for the complex queries experiment for the large database.

Table 5.3: Effects of database optimization on query responsiveness, over the large data set. **BTREE** = overall timing for meeting privacy requirements with our B^+ tree prototype, **rQp** = subquery execution duration within BTREE, **cI** = timing for generating cached index within BTREE, **Time** = time to evaluate private query within BTREE.

Query	**With optimization: default database settings** BTREE = (s)	rQp + (s)	cI + (s)	PIR (s)	**Without optimization: query cache disabled** BTREE = (s)	rQp + (s)	cI + (s)	PIR) (s)
CQ1	104	**50**	52	2	122	**69**	50	3
CQ2	22	**3**	1	18	29	**4**	2	23
CQ3	375	**347**	22	5	454	**434**	15	6
CQ4	66	**25**	32	9	66	**26**	29	11
CQ5	214	**90**	118	6	436	**310**	120	6

database obtains a fresh query plan and result set every time.

We ran this experiment on the less powerful hardware platform we used to develop the

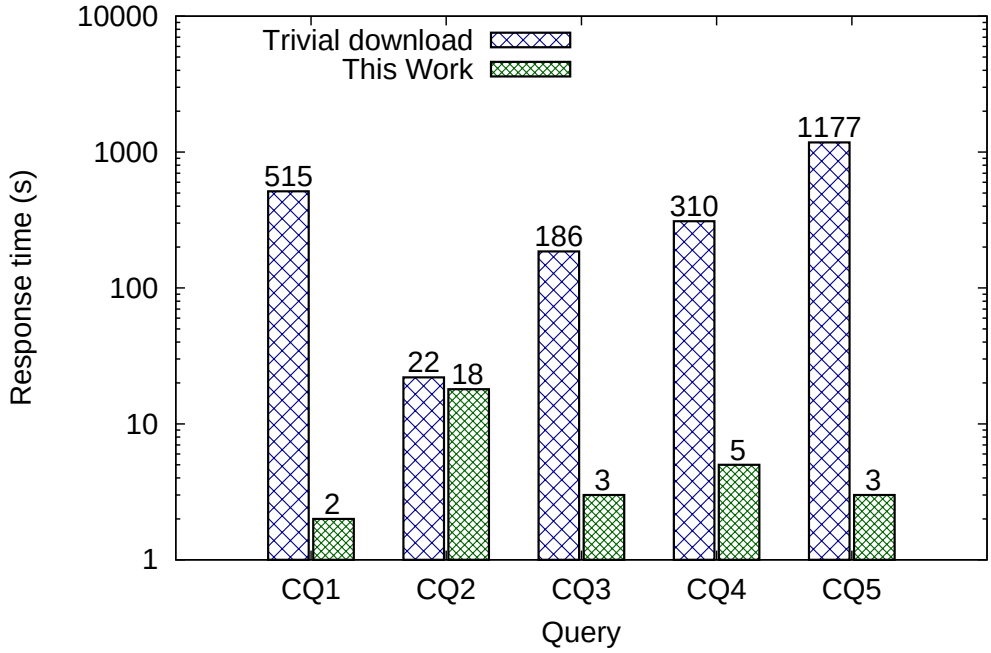


Figure 5.4: Comparing this work to trivial download over a 9 Mbps connection. The y-axis is on logarithmic scale.

prototypes, because the time to build a fresh query plan does not take a significant time for the more powerful hardware platform we used for running the previous tests.

Table 5.3 presents the measurements taken for CQ1 through CQ5 over the large data set under default database behaviour, and the measurements taken when the relations of the database are flushed. The result obtained for these queries validates the claim that our approach leverages database optimization to improve performance. The most interesting measurements taken in this experiment are the subquery execution durations (rQp). For CQ1, CQ3, and CQ5, the difference in measurements is obvious. However, the effect is not quite obvious for CQ2 and CQ4. For the latter pair, the fraction of the overall timing spent for PIR queries is nonnegligible: 79% and 17% respectively. For CQ1, CQ3, and CQ5, the portion of the time spent for PIR is respectively 2%, 1%, and 1%. The results for CQ1, CQ3, and CQ5 clearly indicate the contributions of database optimization to query responsiveness with our approach.

5.7.4 Improving Performance by Revealing Keyword Prefixes

One way to improve query performance is by revealing a prefix or suffix of the sensitive keyword in a query. Revealing a substring of a keyword helps to constrain the result set that will be indexed and retrieved with PIR. We have demonstrated the feasibility of this technique with complex query CQ4 (Listing 9 and Table 5.2). While this technique may be infeasible in some application domains, due to the sensitive nature of the keyword, it does improve performance in others. This technique does, of course, trade off improved performance for some loss of privacy, though it is in fact the user (who can best make this trade-off decision) who can decide to what extent to use it. Making this trade-off decision in a privacy-friendly manner necessarily requires some knowledge of the data distribution in terms of the number of tuples there are for each value in the domain of values for a sensitive constant. This information can be included in the metadata a server sends to the client and the client can make this trade-off decision on behalf of the user based on the user's preset preferences. We consider this extension in Chapter 6.

The processing of queries that allow users to reveal either a prefix or suffix of their private constant will proceed as follows on a prebuilt index: A user would first request the root to a particular subtree in a prebuilt B^+ tree index (indexed either on the attribute or the reverse of the attribute, as above), by supplying a substring for that root. The server would search for and return the requested root, without PIR. Subsequent PIR queries by the user will be based on the subtree with the retrieved root, instead of the entire B^+ tree. In other words, revealing a substring of a user's private keyword reveals the portion of the data that is of interest to the user. However, the level of privacy protection may still be sufficient for many user and application purposes.

The only realistic situations where performance cannot be easily improved with this technique are when users must make *ad hoc* queries that are unknown to the server before a system is deployed. In such situations, it is difficult to make a single index generic enough to serve the diversity of the constraints in an *ad hoc* query.

5.7.5 Limitations

Our approach can preserve the privacy of sensitive data within the WHERE and HAVING clauses of an SQL query, with the exception of complex LIKE query expressions, negated conditions with sensitive constants, and SELECT nested queries within a WHERE clause. Complex search strings for LIKE queries, such as `(LIKE 'do%abs%.c%m')`, are beyond the current capability of keyword-based PIR. Similarly, negated WHERE clause conditions,

such as `(NOT registrant = 45444)`, are infeasible to compute with keyword-based PIR. Our solution to dealing with these conditions in a privacy-friendly manner is to compute them on the client, after the data for the computation has been retrieved with PIR; converting NOT = queries into their equivalent range queries is generally less efficient than our proposed client-based evaluation method. In addition, our prototype cannot process a nested query within a WHERE clause. We propose that the same processing described for a general SQL query be recursively applied for nested queries in the WHERE clause. The result obtained from a nested query will become an input to the client optimizer, for recursively computing the enclosing query for the next round. There is need for further investigation of the approach for nested queries returning large result sets and for deeply nested queries.

5.8 Conclusion and Future Work

We have provided a privacy mechanism that leverages private information retrieval to preserve the privacy of sensitive constants in an SQL query. We described techniques to hide sensitive constants found in the WHERE clause of an SQL query, and to retrieve data from hash tables and B^+ tree indices using a private information retrieval scheme. We developed a prototype privacy mechanism for our approach offering practical keyword-based PIR and enabled a practical transition from bit- and block-based PIR to SQL-enabled PIR. We evaluated the feasibility of our approach with experiments. The results of the experiments indicate our approach incurs reasonable performance and storage demands, considering the added advantage of being able to perform private SQL queries. We hope that our work will provide valuable insight on how to preserve the privacy of sensitive information for many existing and future database applications.

Future work can improve on some limitations of our prototype, such as the processing of nested queries and enhancing the client to use statistical information on the data distribution to enhance privacy. The same technique proposed in this chapter can be extended to preserve the privacy of sensitive information for other query systems, such as URL query, XQuery, SPARQL and LINQ. Private information retrieval is only the first step for preserving a user's query privacy. An extension to this work can explore private information storage (PIS) [OS97], and how to use it for augmenting the privacy of users in real-world scenarios. An interesting focus would be to extend PIS to SQL in the manner of this work, in order to preserve the privacy of sensitive data within SQL INSERT, UPDATE and DELETE data manipulation statements.

Chapter 6

TOPIR: Preserving Access Privacy Over Large Databases

We develop a technique to address the problem of preserving access privacy for users accessing a *large database* over the Internet. 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 in making access privacy obtainable for today's Internet users.

6.1 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 PIR schemes [CGKS95] 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 [BIM04]. We note that even when faster algorithms and increased computing capacities are available, the overall per-query response time will still be slow for sufficiently large n.

In this chapter, 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 [CG97, OTGH10, WAEA10] 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 chapter. We motivate the research in Section 6.2. In Section 6.3, we provide the model we are proposing for preserving access privacy over large databases. In Section 6.4, 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 6.5 describes our implementation and evaluation of the approach using

patent data from the US Patent and Trademark Office (USPTO). Section 6.6 concludes the chapter.

6.2 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.

6.2.1 High Computational Costs for Large Databases

Existing PIR schemes are slow when used to query large databases. We found from our evaluation in Chapter 4 that the round-trip response time of the fastest single-server PIR scheme [AMCG+08] 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 [Ook10]. We found the fastest multi-server PIR schemes [CGKS95,Gol07a] 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.

6.2.2 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 [Ook10]. As mentioned in Chapter 4, the most valid source of data to show this trend is the recently available Internet speed database [Ook10]. 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 [Nie88]. Again, as noted in Chapter 4, Nielsen's Law specifically addresses the type of users described as normal "high-end" who can afford high-bandwidth Internet connections [Nie88]. This class of users should be contrasted from "low-end" users [Nie88] that the above bandwidth averages from the Internet speed data [Ook10] 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.

6.2.3 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.

6.2.4 "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. [AF02] 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.

6.3 Model

Our model consists of the user and the database interacting using four algorithms: query transformer, query executor, data classifier, and data indexer. The query executor encapsulates the query generation and response decoding algorithms for some PIR scheme. We illustrate the components of our model in Figure 6.1.

Before users can run their queries, the database administrator uses a classifier algorithm [BGV92] 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

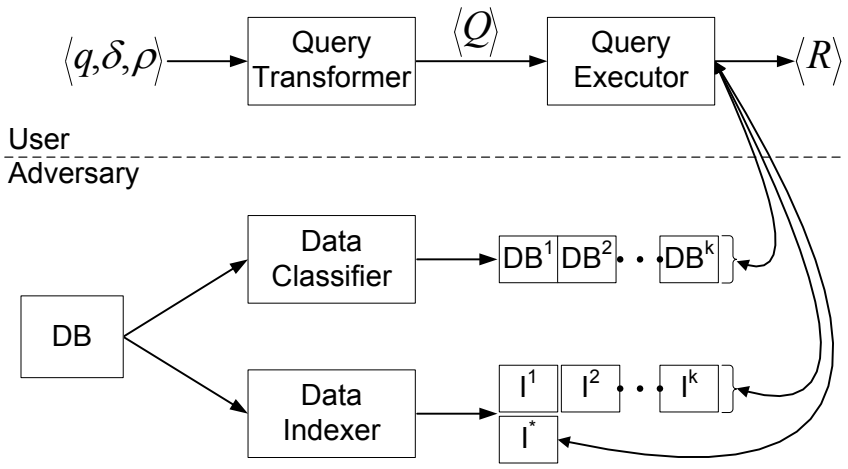


Figure 6.1: A model for preserving access privacy over large databases.

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 [HG05] 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.

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 6.1. 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 6.1.

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 [CGKS95], a keyword [CGN97], or a language-based [OG10b] (e.g., SQL) 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 [CS10] 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 2 *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 [BBD09] for that index, whereas an oblivious traversal over a tree-based index is required to help locate the address in other instances. From Chapter 5, 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.

6.4 Proposed Solution

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. We will describe in detail each of the steps from our model in Section 6.3 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.

6.4.1 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 [Gol07a] 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.

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 4.

Listing 4 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 title `'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 5.

Listing 5 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 5 has the form $KeywordPIRQuery($`'Access Privacy'`$, I)$, where I, based on the technique from Chapter 5, 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 4, 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.

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 [SL91], neural-net [Lip88], or Bayesian [HG05] 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

our original paper [HOG11], which is related to our PIR-Commerce contribution (later in Chapter 8). However, it is not a direct contribution from this thesis, and will therefore not be discussed further. 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 list of the database portions or bucket numbers that have been previously used for their queries. A list of bucket numbers or database portions of a user forms an access map. We note that disclosing a user's access 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 an access 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.

6.4.2 Indexing Large Databases

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 *perfect hash function* (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 [BBD09] 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 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

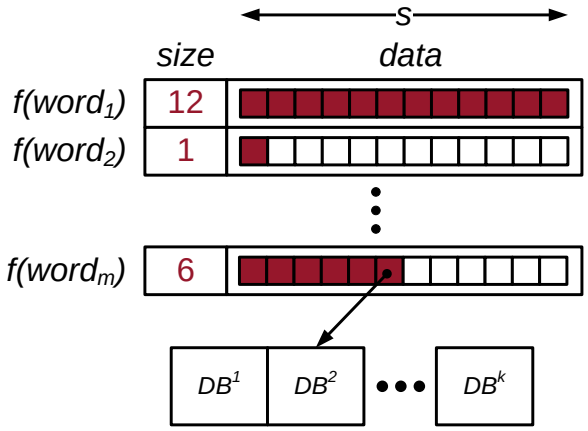


Figure 6.2: Index structure (i.e., table T) to support keyword-based retrieval over a large data set. $s = 12$ is the width of the index, $f(\cdot)$ is a PHF, $word_1, \cdots, word_m$ are keywords, and $DB^1, \cdots, DB^k$ are subsets of the database.

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 6.2.

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 generated in a similar manner; however, the words are now the values of the keys. We note that we describe a related approach in Chapter 5 for the generation of hash-table and B^+-tree indices.

6.4.3 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. 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 [CGKS95], keyword-based [CGN97] and language-based [OG10b]). 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 [Kob01], 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 B.

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 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 B 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.

6.4.4 Privacy-Preserving Query Execution

The query executor runs a transformed query over the data defined by the access 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 [CGN97, OG10b] over the index that was generated.

6.5 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.

6.5.1 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 [Goo10], 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. Besides 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 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 [Goo10]. 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 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 6.3. 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.

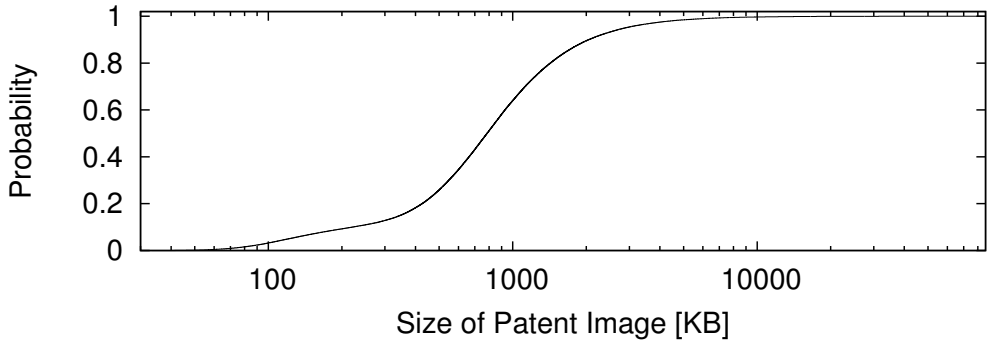


Figure 6.3: A CDF plot of our sample data patent image file sizes. The x-axis is on a log scale.

6.5.2 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 [Kal09] to parse the patent XML files and the C Minimal Perfect Hash (CMPH) Library [BBdCR09] for building hash-based indices. We implemented the PIR aspects of our system using Percy++ [Gol07b]. 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.

6.5.3 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 users would need to specify several buckets for each query, which may add 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. A better way would be to divide individual patents into buckets by considering the keywords that relate to them. When patents containing a particular keyword are evenly distributed among the buckets, there will be less information leakage from an adversary observing the retrieval pattern of a user over multiple buckets.

6.5.4 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 [SB88], inverse document frequency [SJ88], and cosine similarity [SM86] 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 6.4, 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.

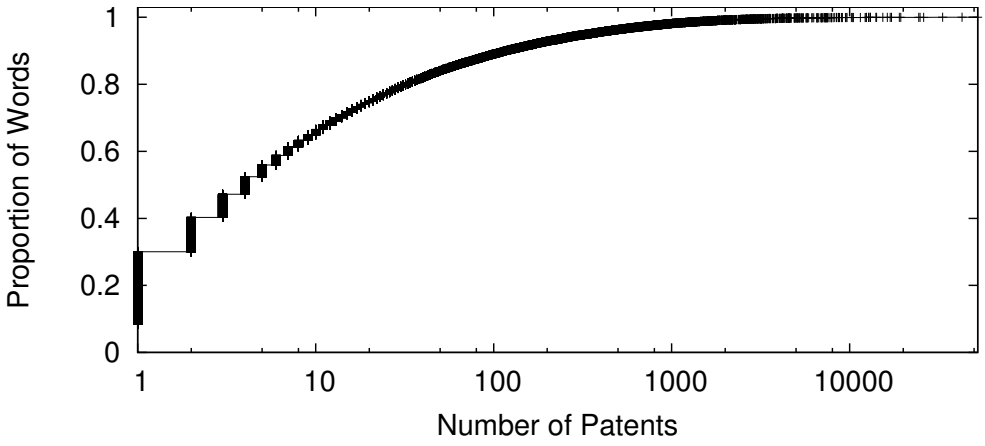


Figure 6.4: A CDF plot of our sample data showing the popularity of words in patents. The x-axis is on a log scale.

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,

Table 6.1: 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 [Ook10]. 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

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.

6.5.5 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 6.1 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 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.

6.5.6 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, provided each bucket is striped with patents containing all possible keywords. In other words, if the data classification results in buckets that are well diversified in their patent contents (i.e., for each search keyword, one can find a patent containing that keyword in every bucket), our approach does not leak extra information.

6.6 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.

Chapter 7

LBSPIR: Achieving Efficient Location Privacy

This chapter is based on published work co-authored with Piotr K. Tysowski, and supervised by Ian Goldberg and Urs Hengartner [OTGH10]. The results in Section 7.4 improve upon the earlier result in the paper. We provide more details on the improvements later in this chapter.

We present a technique for private information retrieval that allows a mobile device user to retrieve information from a database server without revealing what is actually being retrieved from the server. We perform the retrieval operation in a computationally efficient manner to make it practical for resource-constrained hardware such as smartphones, which have limited processing power, memory, and wireless bandwidth. In particular, our algorithm makes use of a variable-sized cloaking region that increases the location privacy of the user at the cost of additional computation, but maintains the same traffic cost. Our proposal does not require the use of a trusted third-party component, and ensures that we find a good compromise between user privacy and computational efficiency. We evaluated our approach with a proof-of-concept implementation over a commercial-grade database of points of interest. We also measured the performance of our query technique on a smartphone and wireless network.

7.1 Introduction

The increasing popularity of mobile devices with positioning technology such as GPS or cell tower triangulation is fueling the growth of location-based services (LBS). We consider

a particular LBS that allows users of mobile devices to find Points Of Interest (POIs), such as restaurants, hotels, or gas stations, in close proximity to their current locations. LBS providers such as Google, Yahoo!, and Microsoft typically store a collection of POIs in databases, which can be queried by the company's own mobile client applications or licensed to third party independent software vendors. A user queries an LBS by first establishing his or her current location using the positioning technology supported by his or her device, and then use it as the origin for the search. The problem is that if the user's actual location is provided as the origin to the LBS, which performs the lookup of the POIs, then the LBS will learn that location. In addition, a history of locations visited may be recorded and could potentially be used to target the user with unexpected content such as local advertisements, or worse, used to track him or her. The user's identity may be divulged through the inclusion of the originating dynamic IP address, e-mail address, or phone number in requests to the LBS server so that the results of an LBS query can be routed back to the correct user via a TCP data connection, e-mail reply, or SMS reply, respectively. If a location can always be correlated to each request, then the user's current pattern of activity and even personal safety is being entrusted to a third party, potentially of unknown origin and intent. Although search engines routinely cache portions of previous queries in order to deliver results that are more relevant in the future, we are concerned when the user's exact location history is tracked, and not just the key words used in the search.

For many users, this constitutes an unacceptable violation of privacy, and efforts should be made to avoid it. As location technology becomes commonplace, users will become increasingly aware of and concerned about location privacy. Not only are privacy and personal safety important considerations, but recent advances in mobile advertising have even opened the possibility of location-based spam. Our challenge has been to design a system whereby a user can retrieve useful POI information without having to disclose his or her exact location to a third party such as the LBS server. The user should also not have to reveal what particular POIs were searched for and found, as each POI record typically includes precise location coordinates. Thus, the server will be unable to infer the user's current location or likely destination, or accumulate a history of requests made for profiling purposes. Generally speaking, a user will typically be comfortable with a certain degree of privacy, meaning that the user could be expected to be anywhere within a certain geographic area, such as a city or neighbourhood, without fear of discovery.

Today's smartphones have high-performing processors which are suitable for cryptographic operations that can enable location privacy. For instance, the Apple iPhone 3GS contains a Samsung ARM 833 MHz CPU, while the BlackBerry Storm 2 contains a Qualcomm 528 MHz CPU. However, these devices have limited memory and bandwidth. For

instance, the iPhone and Storm are both limited to 256 MB of dynamic RAM, 32 GB of flash memory, and operate on 3G wireless networks no faster than the (theoretical) 7.2 Mbps HSDPA network. Consider these data limits with respect to a typical commercial POI database for the U.S. and Canada, which can contain 6 to 12 million entries and require 1 to 2 GB or more of flash data storage [GPS09b]. Requiring that the smartphone download the entire database for *each* request so as not to provide information about its current location is clearly not practical; nor is requiring that it periodically download just the updated data to ensure accuracy of results, given the practical bandwidth limits, data usage limits, and associated overage charges (penalties for exceeding the limits) of smartphone data plans. Thus, it is desirable to provide a cryptographic way for a mobile user to request local information while preserving location privacy. Although extra server-side processing demands must be anticipated on a privacy-enhanced LBS server, it may easily be scaled to multiple computers in a distributed fashion, which is a reasonable tradeoff.

7.1.1 Requirements and Assumptions

We are interested in a nearby POIs (i.e., nearest neighbour) LBS that meets the following requirements:

- The LBS server must not learn the user's exact location. It may only identify a general region that is large enough, in terms of area and the number of POIs it contains, to confer a sufficient level of privacy to the user's satisfaction.
- There must be no third parties, trusted or otherwise, in the protocol between the user and the server.
- The implementation must be computationally efficient on hardware, such as smartphones, which are resource constrained. A user may be expected to tolerate a delay of no more than several seconds for any kind of query.
- The approach cannot rely on a secure processor that is not typically found on a commercial smartphone.

Our proposed solution is sufficiently generic to allow an application to rely on any PIR scheme. We make the same assumptions as that of the underlying PIR scheme, where retrieval is either by object index or keyword [CGN97]. We describe a server that can find the relevant POI entries based on the user's location of interest, without learning that

location; this is possible through the use of PIR because the entries in the POI database are indexed by their location.

Although PIR satisfies our baseline privacy constraints, current implementations of it fail to satisfy our third condition, which is usable performance on modern smartphone hardware. Our challenge has been to complement PIR with a new algorithmic approach that effectively reduces the amount of computations without significantly sacrificing the user's location privacy.

Note that we make no effort to hide the user's identity from the location-based service. We assume that it is acceptable to reveal the user's identity for the purpose of routing the response to a location-based request, and for offering a customized LBS experience. A user that also wishes to hide his or her identity to some extent may wish to make use of an onion router, such as Tor [DMS04a]. However, we note that there are application domains where the protection of a user's location using our proposed technique is superior to anonymizing the user's identity. For example, it is easy to try to identify a user who made a query with a particular geographical coordinate, simply by looking up the user who lives at the corresponding residential address and assuming the request did not originate elsewhere. On the other hand, our proposed technique hides query contents from the LBS, and leaves no useful clues for determining the user's current location.

When a typical mobile phone accesses a third-party LBS provider through a wireless 3G data connection, we assume that it reveals only its identity and the query itself to the provider. Unavoidably, a mobile communications carrier is always aware of the user's location based on the cell towers in contact, and so it must not collude with the LBS provider. Our assumption relies on the LBS provider not being integrated into the carrier's infrastructure, such as a traffic reporting service using cell tower data that discovers a user's location passively. Our assumption is valid for the vast majority of LBS applications, which are unaffiliated with the carrier; these include search portals, social applications, travel guides, and many other types. When communicating with such an application, the mobile user's IP address is of no help in determining the user's physical location, as it is dynamically assigned independent of location. Only a central gateway that is administered by the telecommunications carrier will be identified. We assume that no other information will be gleaned by the LBS provider. In the case where a mobile user utilizes Wi-Fi instead, the user will be assigned an address that points to the nearby access point, however, and may need to employ other techniques, such as Tor, to mask the address.

7.1.2 Our Results

We propose a novel hybrid LBS technique that integrates location cloaking and private information retrieval. We have also implemented and evaluated our proposal to determine its practicality on resource-constrained hardware. The results show that users can achieve a good compromise between privacy and computational efficiency with our technique, unlike all other existing LBS proposals.

7.2 Our Tradeoff Solution

We have developed a hybrid solution that uses PIR to achieve query privacy in the context of a location-based service, and a cloaking technique to reduce the computational cost of PIR to a feasible level. Our technique essentially describes how the user creates a cloaking region around his or her true location, and performs a PIR query on the contents of the cloaking region only. The benefits are numerous: the user's location is kept hidden from the server to an acceptable degree regardless of the number of other users in the area; there is no intermediary server that is responsible for cloaking and that would need to be trusted; and the computational cost of the cryptographic algorithms employed is still practical. We ensure that the user downloads only the POIs that are of interest to the smartphone, keeping wireless traffic to a minimum to reduce costs and conserve the battery. We describe our solution in this section.

The approach that we propose entails two phases. First, there is a pre-processing phase in which the system is set up for use. The pre-processing operation must be carried out whenever significant changes are made to the POI database on the server. In practice, it can occur every few months during a period of low usage on the server such as nighttime maintenance activities. Second, there is an execution phase, in which the LBS server responds to queries for POIs from users. At a high level, the pre-processing phase consists of the following steps:

1. A geographic region is projected onto a two-dimensional plane.
2. A suitable grid is formed on the plane.
3. A collection of POIs is saved in a database such that each row corresponds to one POI.
4. Each cell of the grid is mapped to a portion of the database, i.e., a particular set of database rows (each containing a POI).

5. The grid structure is transmitted and saved on the client device in a local mapping database so that it can be referenced in a subsequent query.

The execution phase, in which a query is made for a set of nearby POIs, consists of the following steps:

1. The user determines the area of interest, either based on the current physical position as determined through GPS, or some other arbitrary area that the user may be traveling to in the future.
2. The user chooses a desirable level of privacy.
3. The client creates a cloaking region corresponding to this level of privacy, which will enclose the area of interest.
4. The client sends the cloaking region to the server. Also, the client identifies which portion of the cloaking region contains the area of interest, in a way that is hidden from the server.
5. The server receives the request, and finds the database portion corresponding to the cloaking region. A block of rows is retrieved from this portion based on the user's specified location of interest. The POIs present in these rows are transmitted back to the client. Note that this step is done privately using PIR, so that the server does not learn anything about the block retrieved.
6. The client decodes the result, and automatically finds the nearest neighbour POI, or presents the full list of POIs returned to the user to choose amongst.

7.2.1 Level of Privacy for the PIR Query

To defeat a server's ability to narrow down the search space for the item of interest to the user, PIR protocols typically process every item, or POI, in the PIR database. This results in a computational complexity that is linear in the PIR database size n.

We propose a tradeoff, in the tradition of PIR development over the years, to make PIR-based solutions practical. For example, information theoretic privacy necessitates replacing a single database with at least two replicated databases; another option is to compromise information theoretic privacy for lower privacy (i.e., attain computational privacy). Our proposal is to offer users the choice of trading off privacy for better query performance,

by specifying the levels of privacy that they want for their queries. A level of privacy for the query determines the number of items that the PIR server must process in order to provide a response. Setting levels of privacy is a common practice in several domains where privacy is important (e.g., web browsers). In the specific case of location privacy, we argue that resource-constrained device users are willing to trade off privacy to obtain reasonable performance. On the other hand, such users are equally willing to trade off some level of performance to gain some level of privacy support.

A user sets the desired privacy level by specifying the size and position of the cloaking region. The ratio of the number of POIs inside this region to the number of POIs in the entire POI database defines the level of privacy. The privacy level can be specified in terms of cities/towns (city level), states/provinces (provincial level), and so on, to enhance user-friendliness. Thus, a privacy level value of 1 indicates that the user desires query privacy at the same level as that offered by a typical PIR protocol. Similarly, if a user sets the query privacy level to 0.6, the PIR query will execute faster. Although the cost is still linear in the number of items in terms of computational complexity, the constant coefficient is modified, leading to significant performance gains. At the same time, it will be disclosed to the server that a particular amount of $0.4n$ items are not of interest to the user; this leakage of information does not necessarily constitute a significant breach of location privacy.

The cloaking region is thus identified as a subset of the entire world described by the database. If we imagine that the world is mapped as a grid of so-called geographic grid cells that are equally distributed, then one of these cells will be chosen to comprise the cloaking region. If a higher privacy level is desired, then the cloaking region may be expanded to include multiple geographic grid cells, and thus a larger portion of the database that describes the world. It is sufficient to identify each grid cell by its cell number if the mapping is static and published. The process of mapping the world to a geographic grid occurs during the pre-processing phase, is described next.

7.2.2 Pre-processing and Location Cloaking

The first step in the pre-processing phase is to represent a geographic area such as the United States and Canada on a two-dimensional plane using a map projection method such as the commonly used Miller cylindrical projection [Sny93]. Once that is done, the user's location of interest may be found on this plane. It is necessary to obscure the user's location by creating a cloaking region around the user's true position or area of interest. POIs will be found anywhere by the LBS server within this cloaking region. The region

must be sufficiently large in order to achieve sufficient privacy for the user, but at the same time it must be sufficiently small to minimize the amount of computation required on the user's mobile device to process the query results, as well as to constrain the amount of wireless data traffic required to transport them.

Several techniques allow POIs to be mapped to a cloaking region. One technique is quad-tree mapping [GG03], but it has the disadvantage (from its use in Casper [MCA06]) of forming an unnecessarily large cloaking region, which can impair performance [BLPW08]. Another technique is called VHC (Various-size-grid Hilbert Curve) mapping [PYZ+09], which suits our purpose. In particular, it solves the problem of the density of POIs varying by geographic area. If the density of POIs is significantly higher for a given region (such as a city), then a higher data traffic cost will result if the size of the cloaking region remains constant, and the query will be much slower. If on the other hand, the density becomes significantly lower (such as in a sparsely populated region like the countryside), then the result size may be so minimal that the server may guess the user's likely destination with a high degree of confidence, leading to loss of privacy. VHC solves this problem by creating variable-sized regions that can be used for cloaking, based on the density of the POIs in the geographic area.

Essentially, VHC maps the two-dimensional geographic grid to a one-dimensional space such that it has equal POI density everywhere (see Figure 7.1a). Assume that a typical POI database that covers the regions of Canada and the U.S. will have 6 million POIs. If each VHC cell must contain the same number of POIs, such as 60, then there will be a total of 100,000 VHC cells that will cover this geographic region. Suppose that the lowest POI density found in the database is 60 POIs per 40,000 km^2. Thus, the maximum size of a VHC cell will be 40,000 km^2.

Now, we create a geographic grid overlaying the U.S. and Canada regions with fixed-size square cells that are 200 km in length (the area of each is 40,000 km^2). This corresponds to the maximum size of a single VHC cell as described above. Each geographic grid cell, however, may contain any number of smaller-sized VHC cells if the POI density of the region is greater (see Figure 7.1 (b)).

Finally, the client determines a cloaking region based on a particular privacy level, which will dictate the number of geographic grid cells to include inside the cloaking region. Suppose that the client chooses a privacy level such that the cloaking region consists of four geographic grid cells. The user's true location is in one of these grid cells. Inside of the geographic grid cell, there is a set of variable-sized VHC cells according to the distribution of the POIs in the geographic grid cell. The user's area of interest, in which POIs will be searched, will be the single current VHC cell found inside the geographic grid cell. The

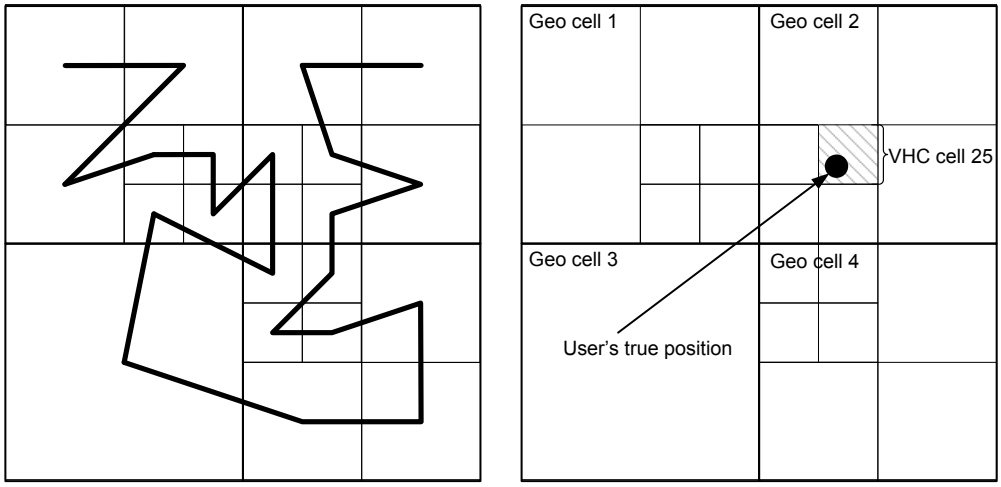


Figure 7.1: (a) A Various-size-grid Hilbert Curve (VHC) mapping with uniform POI density. (b) A user's true position inside VHC cell 25 (shaded) and within a cloaking region bounded by the single geographical grid cell 2. The POI results for VHC cell 25 only will be returned in a query. If a larger cloaking region consisting of geographic grid cells 1 to 4 was specified (for privacy), the same POI results would still be returned.

number of POIs per VHC cell is known, and in our case, it is 60. Thus, the user will initiate a request that will publicly reference the cloaking region, as well as privately referencing the specific VHC cell in which the user is located or interested in. The user will receive a set of 60 POIs that are found in his or her current VHC cell only. The server will only know that the location of interest is somewhere within the cloaking region defined by the geographic grid cells. In other words, the server learns of the larger geographical area defining the cloaking region, but does not learn of the smaller area defining the VHC cell that contains the 60 POIs.

The geographic grid is useful in specifying the size of the cloaking region and for identifying which VHC cells will comprise the cloaking region. The level of privacy, defined from 0 to 1, establishes the size of the cloaking region. The client then sends this cloaking region to the server, by identifying the bounding coordinates (i.e., the longitude and latitude of the top-left and bottom-right corners). The server will then be able to identify which VHC cells belong to this cloaking region, and therefore which portion of the database must be read. The client must also encode the VHC cell containing the area of interest inside a PIR query. (Each VHC cell in the system is uniquely identified by a numeric value.) Figure 7.2

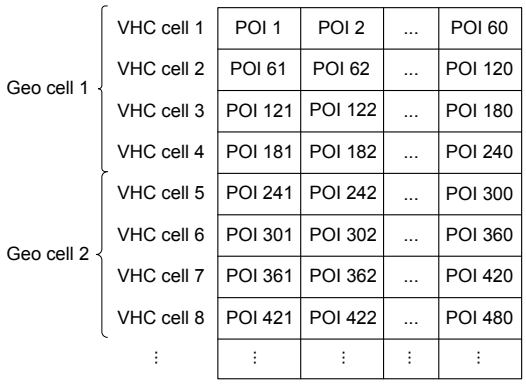


Figure 7.2: Illustration of the relationship between geographical grid cells, VHC cells, and POIs as stored in database rows.

further illustrates the relationships among a geographical grid, VHC cells and POIs.

Thus, our cloaking technique provides a way of reducing the search space of the POI database by employing multiple levels of database segmentation. The cloaking region itself is described as a rectangular area corresponding to a single, or multiple, geographic grid cell or cells. Inside each geographic grid cell are found one or multiple VHC cells, the number depending on the POI density. The user's true location is inside one of these VHC cells, and the user retrieves POIs corresponding to that VHC cell only. As far as the LBS server is concerned, though, the user could be located anywhere within the larger geographic grid cell.

The geographic grid is fixed. The initial grid cell dimensions are configured based on the maximum size of each VHC cell, but once established, will not need to change. Both the client and server must have the same knowledge of the geographic grid. It can be distributed offline (along with the software for the user's smartphone). A simple approach to determining grid cell dimensions is to use a geographic coordinate system such as Degrees-Minutes-Seconds (DMS) [KK00]. For instance, each grid cell may be two latitude degrees in length, which roughly equates to 200 km. A population of tens of thousands to millions of users may typically inhabit and stay within the bounds of a grid cell that is 40,000 km^2 in size, leading to excellent privacy. Cells of larger size will afford province- and state-level privacy if desired.

Both the client and server must agree on the same VHC mapping, and this mapping must be done off-line in advance. Because it is dependent on population density, it will remain relatively static over time even as the population grows, and can be dynamically updated on the client if necessary. In order to contain knowledge of the mapping to define

the cloaking region, the user may make use of a pre-computed map file that is stored locally on the device. This mapping technique is a replacement for a cloaking region that is simply based on cells of constant size, and ensures that a constant and predictable number of results are returned for the user's grid cell.

The idea of using VHC to address the general problem of location privacy was proposed by Pingley et al. [PYZ+09], but in a way that is very different from ours. Specifically, VHC was used to map the user's current location to a 1-dimensional space. Random perturbation was then applied on the 1-dimensional value, which was then mapped back to 2-dimensional space according to the VHC mapping, to represent the user's true location. In essence, the random perturbation was applied to create confusion for an attacker about the user's true location. Our technique differs in that VHC is used for a different purpose; it defines the storage of POI entries of interest within a geographic cell, which comprises the cloaking region, in a way that allows proximate POIs to be stored as adjacent database entries. We then utilize this cloaking region within the context of a privacy-preserving PIR protocol. We do not perform perturbation of the location, which we argue would result in decreased privacy. Indeed, a non-stationary user whose true location is randomly perturbed is still subject to correlation attacks. In our approach, we will demonstrate that the cost of computational and communication overhead through our use of PIR is acceptable, as we provide a method for retrieving only a subset of entries of the entire POI database for each query. Our technique is also impervious to correlation attacks.

The device must store a copy of the VHC map in local non-volatile memory, but the storage requirements are very reasonable. The current geographic grid cell encapsulating the user can be derived from the user's current latitude and longitude coordinate, if the mapping convention is known. A single coordinate for the intersection point of each VHC cell inside (i.e. one of its corners) can then be recorded. Hence, a single coordinate would suffice to store each VHC cell in device memory. For quick lookup and to minimize storage requirements, the coordinates of all VHC cells only in the current geographic cell could be stored. Assuming that the smallest VHC cell size is 1 km^2 in size, then the worst case is that 40,000 coordinates will need to be stored to account for all VHCs. Two bytes will be sufficient to store each VHC coordinate, because the origin of the geographic grid cell is known, so that the total cost will be approximately 80,000 bytes to store all VHC cells. This is the worst theoretical case; in practice, small VHC cells will only be encountered in very dense metropolitan areas, and they will not occupy an entire geographic cell.

7.2.3 Variable Level of Privacy

The size of the cloaking region and the performance of a query depend on the user's specified level of privacy. If the user wishes to obtain a higher level of privacy, then the size of the cloaking region can be defined to be larger, and to encompass a larger number of geographic grid cells (and thus VHC cells), but the amount of computation on the server will increase accordingly, delaying the response. Nevertheless, the chief benefit is that the processing time of the query on the server is predictable, because each VHC cell in each request contains the same number of POIs. The key fact is that the amount of data transmitted will be roughly proportional to the number of POIs in a single VHC cell (depending on the details of the PIR scheme being employed), but the server will only learn the client's location to the resolution of the cloaking region. The amount of variation allowed in the size of the cloaking region should be kept to a minimum, as this variable may be used to form part of a fingerprint of a target in a correlation attack. Allowing a one-cell or two-by-two-cell region only may be a good compromise. The latter could be employed by the user on a permanent basis to avoid the threat of inter-cell movement being discovered.

Our proposed algorithms for privacy-preserving queries, which allow the user to specify a level of privacy, are explained in detail in Appendix C.

7.3 Security Analysis

In this section, we qualitatively analyze the security guarantees of our approach. We note that the location of interest for a nearby POI search may be either the user's true, physical location, or some other location that must be kept private. For the purpose of this analysis, however, we will simplify the job of a malicious LBS server by assuming the location of interest is the user's actual location. Without this assumption, it will be more difficult for the server to infer a user's actual location because the user may issue a query for a geographical area without physically residing in that area.

7.3.1 Collusion Prevention for PIR

The approach we have presented is sufficiently generic to allow an application to rely on any existing computational or information-theoretic PIR (single-server, multi-server or hardware-assisted). We present an approach for preventing collusion between servers in the

case of information-theoretic multi-server PIR. This concept is important to our discussion because information-theoretic PIR makes the usual assumption that the PIR servers that implement it are not colluding to violate the privacy of the users.

The problem of colluding servers is mitigated by practical business concerns. Realistically, a single POI database would be maintained by an organization that is independent of the LBS providers that a user may query. For instance, LBS providers such as Google and Microsoft may contract the use of a POI source such as the Yellow Pages, an organization that is responsible for its own content that it updates regularly. However, Google and Microsoft would be responsible for the content's distribution to end users as well as integration of partners through banner ads and promotions. Since the LBS providers are operating in the same or similar line of business where they compete to win users and deliver their own advertising models to reap economic benefits, there is no real incentive to collude in order to break the privacy of any user. In this model, it it conceivable that a user would perform a location-based query and would invoke it on the multiple LBS providers concurrently, and combine the results, without fear of the queries divulging the user's precise location. Users then enjoy greater confidence in usage of the service, and the LBS providers in turn can capitalize on revenue generation opportunities such as pay-per-use subscriptions and revenue-sharing ad opportunities.

7.3.2 Subscription Service and Privacy

There is still an open question concerning the effect of a subscription service on the privacy of users. If the model is structured as one that is pay-per-use, then the LBS provider must be provided with a means of identifying the user making a request for billing purposes. Similarly, in order to deliver a customized experience to a user based on personal preferences or history of usage, identification of the user must occur. This practical and reasonable requirement breaks the anonymizing component in proposals such as that of Pingley et al [PYZ+09]. In contrast, this limitation does not lead to the loss of privacy in our approach because it is focused on preserving the privacy of the location rather than the identity of the user. The LBS provider may have multiple ways of determining which user it was that issued a particular query. Nevertheless, that correlation will not constitute a loss of access privacy. If a requirement exists to keep the user's identity secret for any reason, then there is a need to employ cryptographic schemes based on private credentials and anonymous e-cash as described by Camenisch et al. [CLM07]. As we will see later in Chapter 8, we extend PIR to a model that allows the user to purchase digital goods in a manner that preserves both her access privacy and her anonymity (when the transaction is made over an anonymous communication network, such as Tor).

7.3.3 Privacy and Size of the Cloaking Region

Unlike location privacy solutions based on k-anonymity [CML06, GG03, XC07], our solution preserves the privacy of the user's location irrespective of the number of other users initiating queries for the same location. The server can infer only the user's location based on the cloaking region. The user may adjust the size of the cloaking region based on his or her personal preferences (i.e., the desired level of privacy, query performance, and cost), because a larger region will entail more computation.

The size of the cloaking region is based on a particular size of geographic area and does not need to be adjusted based on the known distribution of POIs within the region. The user only establishes a reasonable level of privacy based on the number of geographic grid cells that define a geographic area. The boundary of the cloaking region utilized in a request is established by the user and is based on the geographic cell map contained on the user's device and the level of privacy parameter. The size of the cloaking region and its boundaries are not controlled by the server.

In the extreme case, the user may choose a maximum level of privacy, despite the computational costs entailed. In this case, the user will define a cloaking region that includes the entire geographic region that can be queried; e.g., all of North America. The server would execute its query on all the rows of its database, and so there will be a significant computation cost. However, because only the coordinates of the cloaking region are sent in our protocol, and because only the POIs for the user's current VHC cell will be returned, the additional bandwidth cost in this scenario is limited to the cost of sending a larger PIR query, which is now over the entire database of POIs. The level of privacy will be absolute, in that no information about the user's location will be leaked to the server.

In typical usage, however, the user is expected to define a cloaking region that bounds one or more geographic grid cells, depending on the desired level of privacy. For example, a matrix of 1-by-1 geographic grid cell, 2-by-2 cells, or 3-by-3 cells, etc. This user can always adjust the level of privacy for each request. This is useful in the case where the user is traveling between VHC cells. For example, if the user is traveling by car on a highway, and may cross the current geographic grid cell before the next query is issued, then a larger cloaking region formed from the neighbouring grid cells may be appropriate to request.

7.3.4 Passive Attacks

The cloaking region is selected based on the user's location. Let us suppose that the user's preferred level of privacy will be satisfied with a cloaking region that is defined by a matrix

of 2-by-2 geographic grid cells. The user will be located in one of these geographic grid cells. Three additional geographic cells must be chosen to form the cloaking region. If the user proceeds to pick these geographic cells randomly from the surrounding set of geographic cells for each request, then each request may send a different cloaking region. After a number of these requests, and if the user remains stationary inside one of these geographic grid cells, then the server will be able to correlate these requests, and determine the overlapping grid cell which likely contains the user.

In our design, we have elected to specify fixed groups of geographic grid cells at various levels of privacy. In other words, if the user remains stationary, then for each request, the user, depending on the privacy level, will select the same 2-by-2 matrix, or 3-by-3 matrix, etc. Therefore, the correlation attack described above is impossible because the client will send the same cloaking region for all queries for a given privacy level.

Next, consider the case of a mobile user who is physically moving between VHC cells. As long as the user does not move outside of the cloaking region, then the same cloaking region will be transmitted for all queries, and the user will not be subject to a correlation attack, even if the user moves between VHC cells. The same is true if the user moves between geographic grid cells, but still within the same cloaking region defined by the user's level of privacy.

If the user moves to a VHC cell that is outside of the original cloaking region, then a new cloaking region must be formed. The user will now occupy a new geographic grid cell that will define a new cloaking region. The server could observe that requests are being made for neighbouring cloaking regions, and could infer that the user is somewhere close to the edge of these regions, or is traveling between them. This is a consequence of the user having to create cloaking regions such that there is never any overlap between them. The assignment of geographic grid cells to cloaking regions must satisfy this requirement, and is done based on their consecutive ordering on the grid. To avoid the possibility of the server detecting movement between cloaking regions, it is recommended that the user instead increases the privacy level so that a greater number of geographic grid cells will define the cloaking region. This enlarged cloaking region should contain the original cloaking region. Now, if the user moves between grid cells, the same cloaking region will still cover them, and a correlation attack will be unsuccessful. Thus, the user should be aware of his or her likely movement in the future and choose a privacy level such that the cloaking region will contain all current and likely future locations. If the size of the cloaking region is constantly adjusted up and down, based on movement and stationary states, then the server may be able to reconstruct a history of the user's movement patterns. Note that in the general proposed scheme, it is not strictly required for the user to be able to predict the path of future travel for all requests. It simply confers an optional improvement to

privacy if the user can encapsulate an intended path of travel through a more informed choice of the cloaking region boundaries.

In addition, if an attacker observes the communication between the client and the server, then only the client's identity will be disclosed. Neither the contents of the query nor the results can be decoded if end-to-end encryption, such as Transport Layer Security (TLS), is utilized for the communications link. The server must be able to identify the address of a user's mobile device in order to route its response. The address itself will not give away the user's position, however.

7.3.5 Active Attacks

Replay Attack

If an attacker observes a request from the client and launches a replay attack against the server, then the server will respond to the request, but neither the attacker nor the server will receive any additional information on the user's location. TLS can of course also mitigate this attack if necessary.

Result Tampering Attack

When our approach is used with a single-server PIR scheme, a malicious server can return POIs that are not found in the cloaking region, but the user can filter the results out based on their coordinates. If the server returns false POIs, the user will not be able to verify and detect them. This is unlikely, as the server would need to replace all the rows in the portion of the database (corresponding to the cloaking region) with false information. If the server returns an empty result instead of the actual content in its database portion, then the user may expand the search by increasing the level of privacy and increasing the size of the cloaking region or by exploring neighbouring VHC cells within the cloaking region, but the server will fail to learn anything additional about the user's location.

For information-theoretic PIR, this attack is impossible unless the servers are colluding. At best, one of the servers could deny service for the query by returning garbage or by not responding. Such an attack can be prevented by multi-server PIR schemes with support for Byzantine robustness [BS07, Gol07a].

7.4 Experimental Evaluation

7.4.1 Implementations

We developed a C++ prototype and a Java prototype for our proposal. The evaluation of our approach in terms of feasibility and scalability is based on the C++ prototype. The point of the Java prototype is to demonstrate the successful porting of our implementation to a smartphone platform.

The C++ prototype is based on the Percy++ [Gol07b] implementation of Goldberg's PIR scheme [Gol07a]. We modified Percy++ to support our proposal for allowing PIR queries to be based on a database portion defined by the cloaking region and added code for instrumentation. We measured the computational performance of the PIR algorithm when it does take into account the query level of privacy, and when it does not take it into account. We ran the PIR implementation against a database of 6 million synthetic POIs, the typical number of POIs in a commercial POI database for the U.S. and Canada [GPS09a, GPS09b]. We note that a comparable set of experiments by Ghinita et al. [GKK+08] considers a much smaller database: only 10,000 and 100,000 POIs. A head-to-head comparison with their work is infeasible because we used different PIR implementations and test data. Each POI consists of 256 bytes that we generated randomly. Again, this size is a conservative representation of practical POI sizes. In comparison, the POIs from the experiments of Ghinita et al. [GKK+08] are only 64 bits in length.

For the Java prototype, we re-implemented the open source C++ client code for Percy++ in Java and ported the implementation to Google's Android smartphone platform, which supports the Java programming language. Note that we did not need to rewrite the Percy++ server code. In the original paper [OTGH10], we based the Java prototype on a computational SPIR protocol implementation [SJ05], which is the only publicly available Java implementation to our knowledge. This SPIR protocol was derived from the oblivious transfer protocol by Naor and Pinkas [NP99]. This earlier Java prototype development consists of both a server component and a client component that we deployed on the Android smartphone platform. In spite of the extensive modification of the code to get it working on Android (e.g., replacing RMI mechanism with HTTP socket communication), the resulting implementation is quite slow. In this thesis, we address the performance problem with the Percy++ Java implementation. We also repeated the tests for the C++ prototype using an optimum word size setting for Percy++ and obtained better numbers.

7.4.2 Results and Discussion

We measured query roundtrip times for the C++ prototype on a machine with a 2.91 GHz dual-core AMD CPU, 3 GB RAM, and running Ubuntu Linux. Since the Percy++ PIR uses replicated databases, we set the number of databases to 2, and the word size to the optimal setting of 8 bits per word [Gol07a]. Figure 7.3 shows query roundtrip times and levels of privacy for queries returning various numbers of POIs. A similar plot for total data transfer per server is shown in Figure 7.4. The number of POIs returned for each query is equivalent to the number of POIs in a VHC cell. Similarly, the number of POIs returned by a query is equivalent of the number of blocks (in bytes), that a traditional PIR query returns. A block of 10 POIs is equivalent to 2560 bytes of data (each POI consists of 256 bytes).

For various levels of privacy (0.01, 0.09, $\cdots$, 1), the communication-optimal set of block sizes (in number of POIs) is respectively ζ = (15, 46, 63, 77, 88, 98, 107, 116, 123, 131, 138, 144, 151, 153). The query roundtrip or response times at query level of privacy 1 for block sizes 10, 50, 100, 250, 500, and 153 (element of ζ) are between 3.4 and 6.7 seconds. This is because each PIR request runs against the entire database of 6 million synthetic POIs. However, the query roundtrip time improves with lower levels of privacy. For example, the query response times for the above block sizes 10, 50, 100, 250, 500, and 63 (element of ζ) at a privacy level of 0.17 are between 0.6 and 3.9 seconds. One must observe that setting the query level of privacy to 0.17 is equivalent to privately querying a block of POIs from a portion of the database consisting of 1.02 million POIs. If we assume the number of POIs in each of the territories, provinces, and states of Canada and US is proportional to their population, a level of privacy set to 0.17 implies a cloaking region that covers approximately all ten provinces and all three territories of Canada, as well as the US states of New York and Connecticut combined. Similarly, a user who intends to hide his or her query in a cloaking region that covers the US state of Washington will set his or her query level of privacy to a much lower value 0.02, while a user who intends to hide his or her query in a cloaking region that covers the Canadian city of Toronto will simply set his or her query level of privacy to an even lower value of 0.01. The query response times for Washington State and Toronto are respectively 0.10 ± 0.00 seconds and 0.05 ± 0.01 seconds for an optimal block size, which in our testing configuration consists of 22 and 15 POIs respectively. It is easy to observe from Figure 7.4 that the communication-optimal block sizes set ζ results in the least amount of data transfer (query and response size). While it does not appear to be the most computationally efficient, from Figure 7.3, it results in the transfer of the most number of POIs, with the least per query rate of transfer. Furthermore, the worst-performing block size is the one consisting of 10 POIs

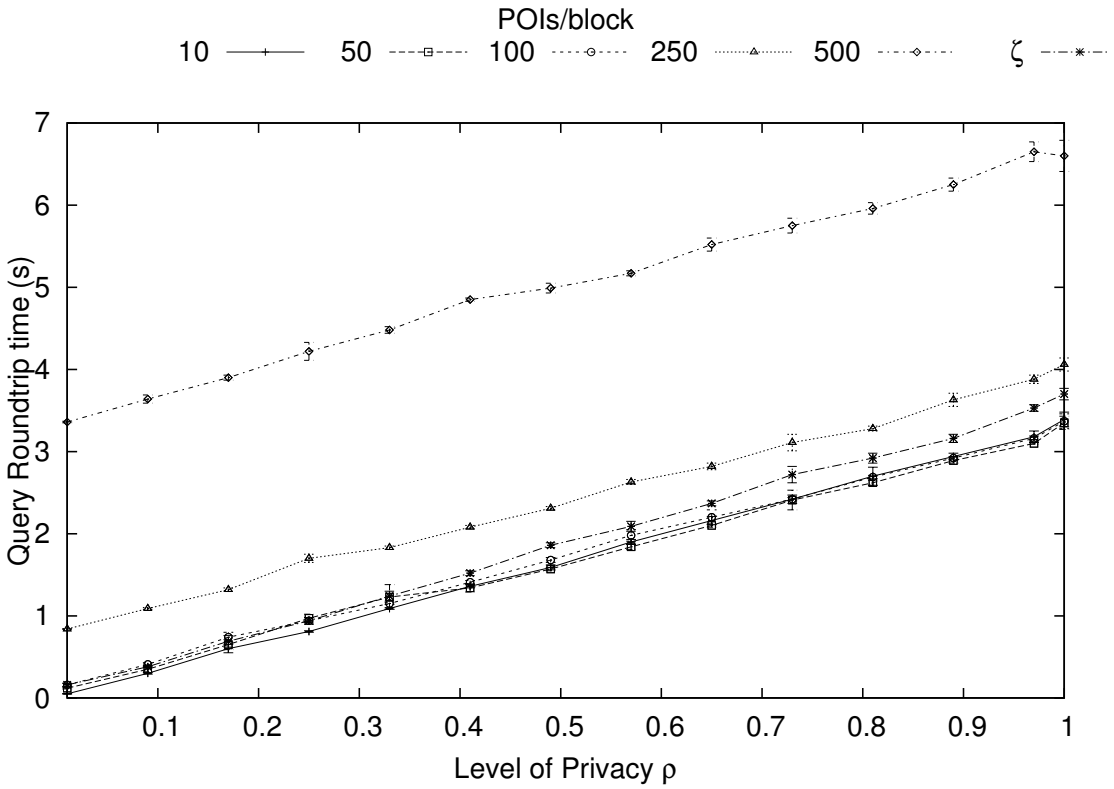


Figure 7.3: Query roundtrip time and level of privacy for various numbers of POIs returned per query. ζ represents the communication-optimal set of block sizes for the levels of privacy. Error bars are plotted for all data points, but some of these may be difficult to see.

(Figure 7.4). Again, that might not seem obvious if one considers just the roundtrip query time without the cost of data transfer. On the other hand, large block sizes, such as 500, carry performance penalties and overheads, which depend on the characteristics of the underlying PIR scheme, and on the resource constraints of the runtime hardware (e.g., RAM, disk and memory cache sizes, and network bandwidth).

We also installed the client for the Java prototype on a Nexus One Android smartphone, which features a 1 GHz Qualcomm ARM processor and 512 MB memory. We ran our tests over a WiFi connection. Figure 7.5 shows the overall roundtrip query time (including data transfer) for the Java prototype. For a province or state granularity for level of privacy (value of 0.02), the roundtrip response time for an optimal block setting is about 1 second, which is obviously practical. We note that the Android smartphone runs out of heap

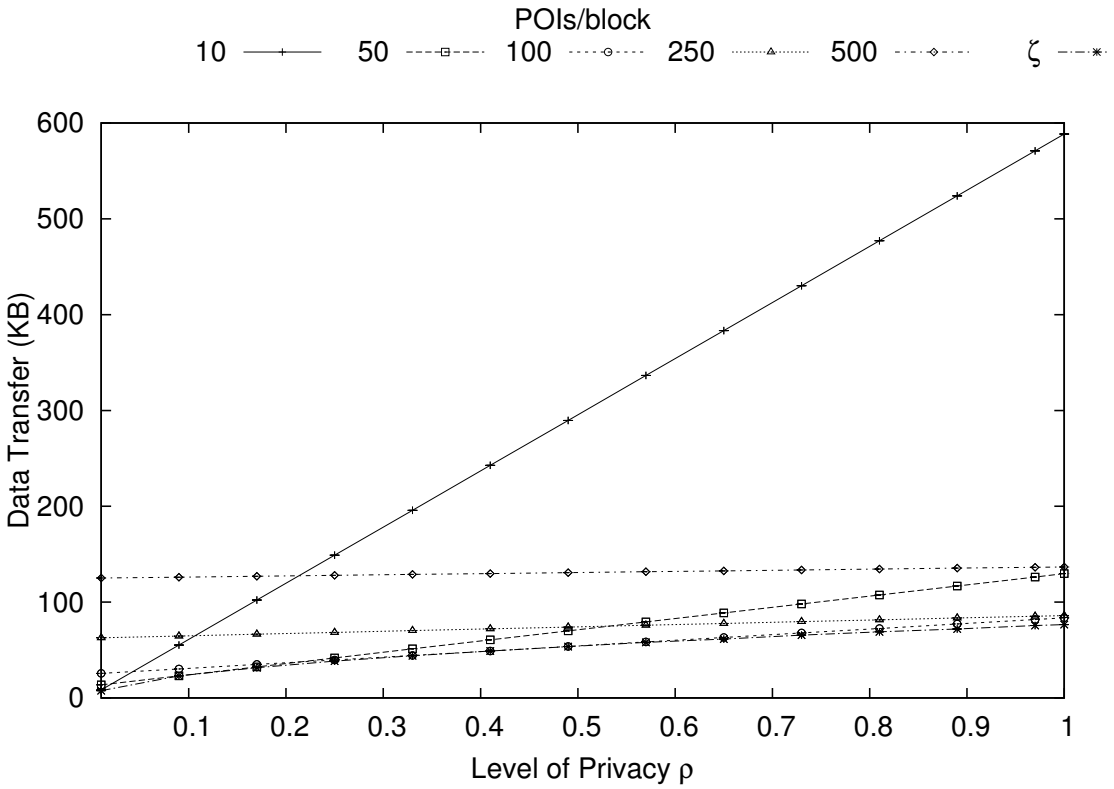


Figure 7.4: Data transfer per server and level of privacy for various numbers of POIs returned per query. ζ represents the communication-optimal set of block sizes for the levels of privacy.

memory when we set the level of privacy higher than 0.73 for most of the non-optimal block size settings. For example, the two best-performing non-optimal block settings from Figure 7.5 are the 50 and 100 POIs/block. Unfortunately, the highest level of privacy a user can set for these without exhausting available heap size is about 0.73. At higher values, the size of the query computed by the smartphone can no longer fit in available heap memory. This behaviour further justifies the importance of our approach of using level of privacy to scale the query size and the amount of computation. Note that the communication-optimal set of block sizes, ζ, allows the user to set up to the maximum level of privacy. Increasing available heap memory for the Nexus One Android smartphone requires recompiling and reinstalling the smartphone OS.

In our earlier work [OTGH10], we installed the client for the SPIR Java prototype on a G1 Android smartphone from T-Mobile, which features a Qualcomm ARM processor

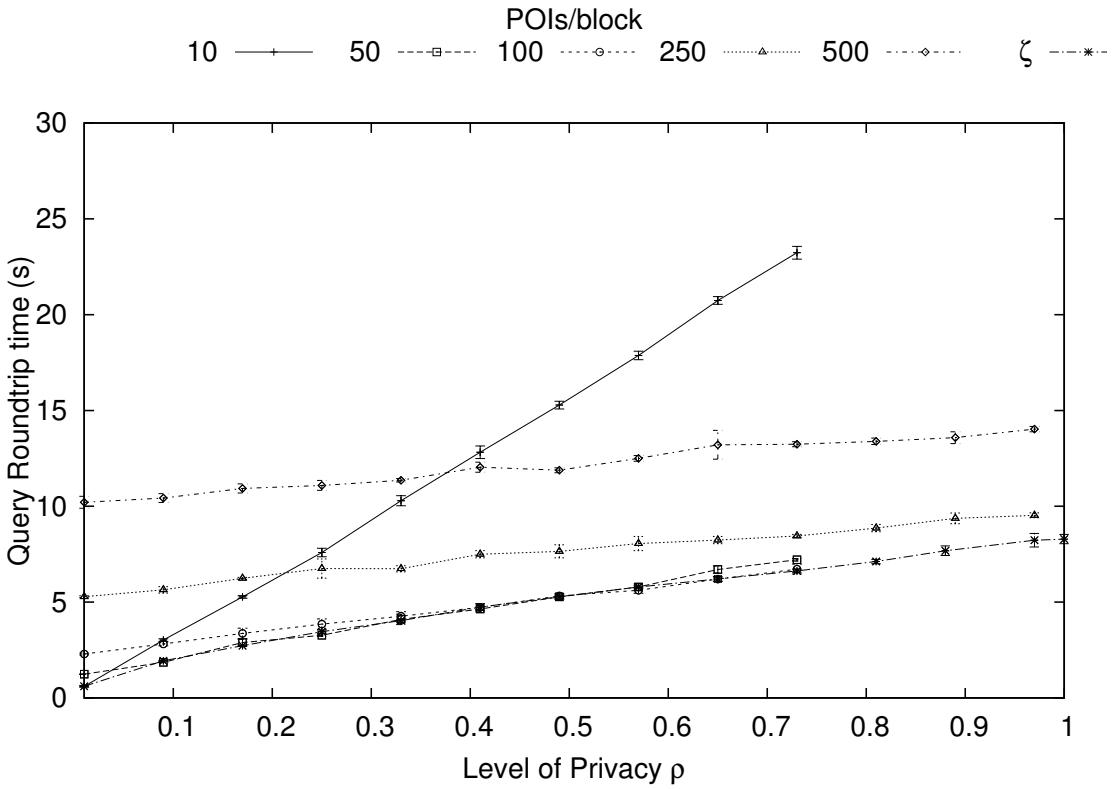


Figure 7.5: Query roundtrip time and level of privacy for various numbers of POIs returned per query for the Percy++ Java prototype. The cost of data transfer is included in the roundtrip time. Error bars are plotted for all data points, but some of these may be difficult to see.

running at 528 MHz, and includes 192 MB DDR SDRAM, and 256 MB flash memory. Although our locked smartphone was capable of running on T-Mobile's 3G network in the U.S., it did not support the 3G frequency bands in operation in Canada. We ran our tests using the Rogers EDGE network, which is slower by up to a factor of ten. We created an Android application with a user interface that allows the user to specify the server address and query parameters such as the size of the cloaking region and the size of the portion of the cloaking region to fetch.

For a cloaking region consisting of 400 cells (i.e., database rows), and a POI result size of 32 bits (i.e., size of a database row), the following performance was measured on the actual Android smartphone: query generation required 2.04 s, result processing required 8.02 s, and the total roundtrip time (including transmission over the wireless EDGE packet

data network) required 207.49 s. In comparison, when the cloaking region was reduced to 100 cells, with the same POI result size, the performance measurements were: query generation of 4.40 s, result processing of 3.42 s, and a roundtrip time of 62.55 s. We observed little variance in the query encoding time, but the decoding time varied nearly proportionally with the size of the cloaking region, as one would expect. More detail on our implementation is available in the original paper [OTGH10].

Overall, we are positive that the Percy++ Java implementation will be usable on live mobile networks. Moreover, there should be a noticeable speedup when server hardware is used to run the PIR server; our experiment uses a commodity desktop computer.

7.5 Conclusions

In this chapter, we have proposed an algorithm for privately querying location-based services that achieves a good compromise between user location privacy and computational efficiency. We have implemented and evaluated our algorithm and shown that it is practical on resource-constrained hardware. Our approach of using a variable-sized cloaking region divided into VHC cells results in greater location privacy than the traditional approach of a single cloaking region, while at the same time decreasing wireless data traffic usage from an amount proportional to the size of the cloaking region to an amount proportional to the size of a VHC cell. It also allows the user to dynamically choose various levels of privacy. Although increasing the size of the cloaking region does result in higher computation in processing the query, we believe that this tradeoff is very reasonable, given that the processing power of today's smartphones is still less of a concern than the speed and cost of wireless network connectivity.

Our work could be extended by improving on our general scenario where the user retrieves all of the POIs that belong to the VHC cell of interest. It is possible that the user will not find a suitable POI within this set, and will wish to search further in neighbouring VHC cells. This may be the case when the result set does not contain a desired POI. Therefore, the user may wish to expand the search by searching in a broader geographical area.

Chapter 8

PIR-Tor and PIR-Commerce: Real-World Applications of PIR

This chapter covers two areas where we applied PIR to solve practical problems. First, we demonstrate practical examples of applying computational and information-theoretic PIR schemes to make the Tor anonymous communication network scale better and examine the resulting design tradeoffs. Second, we show how to tailor and extend the cryptography of PIR to solve the real-world problem of preserving access privacy in an electronic commerce transaction.

8.1 PIR-Tor: Scalable Anonymous Communication

This section is adapted from published work co-authored with Prateek Mittal, Carmela Troncoso, Nikita Borisov, and Ian Goldberg [MOT+11].

Existing anonymous communication systems like Tor require all clients to maintain an up-to-date list of every available node in the network before they can construct a *circuit* for anonymous communication. As a result, Tor does not scale well. Current proposals for making Tor scale better rely on a peer-to-peer (P2P) approach, which has unfortunately provided new avenues to compromise anonymity. In this section, we step away from P2P approaches and propose a client-server architecture that relies on PIR. We propose PIR-Tor as a scalable architecture for the Tor network, whereby users no longer need to maintain a list of available nodes in order to construct circuits. In PIR-Tor, a client obtains information about only a few nodes using PIR techniques, thereby preventing compromised directory

servers from learning the user's choices and reducing the overall communication by at least an order of magnitude. Experimental results show that reasonable parameters of PIR-Tor maintain a level of security that is equivalent to that of the current Tor network, while allowing Tor to scale by two orders of magnitude.

8.1.1 Introduction

Tor is a worldwide overlay network of servers, or *nodes*, that allows journalists, law enforcement workers, humanitarian workers, privacy-concerned citizens, and others to communicate over the Internet without disclosing their IP addresses [DMS04a, Tor11b]. As of June 2011, there are over 2000 nodes which serve hundreds of thousands of users daily [Tor11a, Loe09]. In order to build circuits for anonymous communication, a Tor client would need to learn of available Tor nodes. It does this by downloading two pieces of information from special Tor nodes, known as *directory servers*. First, some trusted *directory authorities* create a *network consensus* file, which is a signed list of all available nodes and their network addresses, and distribute the list to clients through the directory servers. The second piece of information to download from directory servers is the *node descriptors* file, which contains detailed information about each node. A client constructs a circuit using three nodes. The first node is known as an *entry guard* or simply a *guard*. The first time a client connects to Tor, it selects three nodes to be its entry guards. These nodes will continue to be used for all circuit construction by the same client as long as they remain available. The rationale for choosing three fixed nodes as entry guards is to protect against certain long-term attacks on anonymous communication [ØS06]. Next, the client picks an *exit node* from the list of available nodes. Each exit node has an *exit policy*, which specifies ports the node is willing to use for forwarding traffic to the Internet. The client's desired exit port must match the exit policy of the selected exit node. Finally, the client picks any other node as a *middle node*. Since nodes in Tor have different bandwidths, clients select nodes with a probability that is proportional to their bandwidth, in order to achieve load balancing on the network.

The high cost of bandwidth required for downloading the network consensus and node descriptors is a major hindrance to Tor's scalability. While a client is able to multiplex multiple TCP/IP streams over a single circuit, it needs to download an up-to-date network view and reestablish a new circuit (with nodes likely different from the prior circuit) quite frequently. In particular, it must download a fresh network consensus every 3 hours, download a fresh node descriptors file every 18 hours, and rebuild a new circuit every 10 minutes. Thus, if there are n clients, r nodes, and the consensus size is S, the overhead of fetching the network consensus will be nrS data transfer every 3 hours. As shown by

McLachlan et al. [MTHK09], the current Tor architecture could soon lead to a situation where more bandwidth is used to help users download an up-to-date view of the network than for anonymous communication.

Current proposals for making Tor scale better rely on a peer-to-peer (P2P) approach, which has unfortunately provided new avenues to compromise anonymity. The dynamism and complexity of many such P2P-based designs make it very hard for the authors to do a thorough analysis of their design and provide a rigorous proof of security. As a result, the security community has been quite successful at breaking the state-of-art peer-to-peer anonymity designs [BDMT07, DC06, DS08, MB08, WMB10, THK09].

Consequently, we step away from P2P-based approaches and propose a client-server architecture that relies on PIR. Our new architecture is called PIR-Tor. Our key observation for PIR-Tor is that while Tor clients require as few as 3 nodes to construct circuits for anonymous communication, the current architecture still requires them to download the entire database of nodes. We use PIR to download the few nodes required for circuit construction, thereby preventing compromised directory servers from learning the particular choice of nodes selected by a client for circuit construction, and thereby mitigating route fingerprinting attacks [DC06, DS08]. A route fingerprinting attack is one where an adversary is able to observe the subset of nodes learned by a client, and uses this partial knowledge the client has about the network view to identify the traffic from the client.

We consider two architectures for PIR-Tor, based on the use of computational PIR (CPIR-Tor) or information-theoretic PIR (ITPIR-Tor), and evaluate their performance and security. We find that for a scenario where clients create a single circuit, CPIR-Tor provides an order of magnitude improvement over a full download of all descriptors, while ITPIR-Tor provides a two order of magnitude improvement. However, for scenarios where clients wish to build multiple circuits, several PIR queries must be performed, and the communication overhead of CPIR-Tor quickly approaches that of a full download. We therefore propose to perform a few PIR queries and reuse their results for creating multiple circuits, and highlight the security implications of the same. In the case of ITPIR-Tor, we find that the communication overhead for creating multiple circuits is at least an order of magnitude smaller than that of a full download. The client can therefore perform new PIR queries for each circuit construction. We show that, subject to certain constraints, ITPIR-Tor has security equivalent to the current Tor network. Our proposed architectures will allow the current Tor to easily sustain a 10-fold increase in both the number of nodes and of clients, and reasonably sustain a 100-fold increase. It also supports a scenario where all clients can be used as middle-only nodes [DM06] to improve the security and performance of the Tor network.

The remainder of this section is organized as follows. We briefly highlight related work in Subsection 8.1.2. We give details of our proposals in Subsection 8.1.3 and a sketch of the traffic analysis implications of our architecture in Subsection 8.1.4. Subsection 8.1.5 sketches our performance evaluation for the computational and information-theoretic PIR proposals and provides some discussion. Finally, we conclude the section in Subsection 8.1.6.

8.1.2 Related Work

Peer-to-peer approaches proposed for scalable anonymous communication systems can be categorized into architectures that rely on distributed hash tables (DHT), and those that rely on random walks on unstructured or structured topologies. Besides our work that does not rely on P2P approaches, Mittal et al. [MBTR10] briefly considered the idea of using PIR to make anonymous communication systems scale better. However, their description is incomplete and their evaluation is very preliminary; they considered an architecture based on an inefficient CPIR scheme [KO97]. We build upon their work and present a complete PIR-based architecture that considers both efficient CPIR and ITPIR schemes. We show that an ITPIR-based architecture outperforms a CPIR-based architecture in many scaling scenarios, discuss the privacy implications of our architecture, and show that reasonable parameters of PIR-Tor provide equivalent security to Tor.

Distributed hash tables, also known as structured P2P topologies, assign neighbour relationships using a peseudorandom function based on the IP addresses or public keys of nodes. For example, Salsa [NW06], by Nambiar and Wright, is built on top of a DHT, and uses a specially designed secure lookup operation to select random nodes in the network. The secure lookups use redundant checks to mitigate attacks that try to bias the result of the lookup. Other examples of DHT-based P2P anonymous communication architectures include NISAN [PRR09] by Panchenko et al. and Torsk [MTHK09] by McLachlan et al. However, Mittal and Borisov [MB08] showed that Salsa leaks information that helps an attacker to observe a large fraction of node lookups in the system; node selection is not private. In addition, Salsa is vulnerable to a selective denial-of-service attack, where nodes break circuits that they cannot compromise [BDMT07, THK09]. In a similar manner, NISAN and Torsk have been broken by Wang et al. [WMB10].

Architectures that rely on P2P random walks include MorphMix [RP02] and ShadowWalker [MB09]. For example, MorphMix [RP02] alleviates the scalability problem in Tor by organizing nodes in an unstructured peer-to-peer overlay, where each node has knowledge of only a few other nodes in the system. A client constructs a circuit using

recommendations from other nodes in the system. It performs a random walk by first selecting a random neighbour and building an onion routing circuit to it. Next, the client queries the neighbour for its list of neighbours, selects a random peer, and then extends the onion routing circuit to it. This process is repeated to construct a circuit of any length. Just as for the DHT-based P2P architectures, MorphMix has been proven as insecure by Tabriz and Borisov [TB06], and Schuchard et al. [SDH+10] have found a way to attack ShadowWalker.

We note that all the peer-to-peer designs are quite complex, which makes analyzing their security and privacy guarantees quite difficult. It is therefore no wonder that the security community has been quite successful at breaking the state-of-art designs.

8.1.3 PIR-Tor System Details

We are interested in an architecture that meets the following requirements:

- The bandwidth requirement to maintain an up-to-date network view by each client should be small and sublinear in the size of the network consensus and node descriptors. Such a design will improve performance, allow the number network to scale better, and improve users' anonymity [DM06].
- The security properties must be well understood. In other words, it must leverage security mechanisms that are easy to analyze.
- It must not impose additional latency on circuit creation time. In other words, the wait time for circuit creation must be the same as for the existing Tor network.
- Deployment of the architecture must require minimal or no changes to the current Tor architecture to make for easy transition. We note that a peer-to-peer system would require substantial engineering effort to deploy.
- The constraints that the existing Tor network places on circuit creation, such as the requirement for clients to comply with the policies of exit nodes, and the selection of trusted nodes as entry guards, must be preserved. Some prior work like ShadowWalker [MB09] and Salsa [NW06] did not address these issues.

System architecture

Our key observation when designing PIR-Tor is that the client-server model of Tor can be preserved while simultaneously improving scalability by having clients download descriptors only for a few nodes, instead of downloading the entire database of descriptors. Unfortunately, doing this in a naive way breaks the client's privacy, as compromised directory servers learn which nodes the client knows about. We therefore leverage PIR for downloading a few descriptors from the descriptors database, without letting the database know what was downloaded. Note that a client connecting to Tor for the first time should still do a one-time download of the network consensus and node descriptors database, which does not affect the scalability of our proposal.

Recall that we base the two flavours of our architecture on CPIR and ITPIR. While clients can use either of these, the underlying techniques have different threat models, resulting in slightly different architectures and security guarantees. While our architecture is compatible with all existing CPIR and ITPIR schemes, we base CPIR-Tor on the single-server lattice-based scheme by Aguilar-Melchor et al. [AMG07], and ITPIR-Tor on the multi-server scheme by Goldberg [Gol07a]. The lattice-based CPIR scheme is the computationally fastest single-server scheme available [OG11], and both schemes are available as open-source libraries [Gol07b, GPG09].

CPIR-Tor architecture. Since Tor directory servers already maintain a copy of the network consensus and node descriptors, we propose to use them as the PIR database. Note that the current Tor architecture uses directory servers as download servers for the network view. For CPIR-Tor, clients can use a CPIR protocol to query a single directory server and obtain the identities of random nodes in the system. The directory authorities sign each block of the consensus and descriptors lists to ensure the integrity of the network view.

ITPIR-Tor architecture. Since Tor already places significant trust in guard nodes, we use a client's three guard nodes as the servers for ITPIR. Unless all three guard nodes are compromised (a low-probability event), they cannot learn of the descriptors retrieved by the clients. Just as in the existing Tor network, a compromise of all three guard nodes allows the traffic from the client to be linked. When all three guard nodes *are* compromised, they still cannot actively modify entries in the PIR database because they are individually signed by the directory authorities. However, they can only learn which exit node descriptors were downloaded by the client. (In Tor, guards always know the

identities of the middle nodes in circuits through them.) If the exit node in a circuit is honest, then guard nodes cannot break user anonymity. On the other hand, if the exit node used is malicious, then user anonymity is trivially broken [DC06]; the adversary could have performed end-to-end timing analysis anyway [BMG+07].

We discarded two other alternative architectures for ITPIR-Tor that were not as attractive. The first is to have clients query three directory servers using PIR. However, this gives many opportunities to break clients' anonymity as the current Tor architecture does not trust directory servers. The second alternative is for clients to perform PIR over some threshold of directory authorities. This approach is somewhat secure because Tor already trusts that a majority of the directory authorities are honest. However, the computational overhead from all clients querying a fixed number of directory authorities will become a source of performance bottleneck and a target for DDoS attacks as the network grows.

Database organization and formatting. We propose to organize nodes in the database in line with the constraints on the existing Tor network. In particular, we propose to use three databases: one each for the entry guards, middle nodes, and exit nodes. Nodes that can serve both as entry guard and as exit are duplicated in the respective databases. We took a snapshot of the current Tor network and found 222 unique exit policies from 953 exits. Of these, 471 are standard exits and the remaining 482 share 221 non-standard exit policies. Had the number of non-standard exits been few, we would have recommended for clients to download them entirely and use PIR to fetch from the standard exit policies, but this was not the case.

In order to load balance the network, Tor clients select nodes with a probability that depends on their bandwidth capacity; that is, higher capacity nodes are selected with higher probability. We therefore require descriptors in the exit database to be grouped by exit policy, and then sorted in descending order of their bandwidth capacity. Similarly, descriptors in the middle database are sorted solely by bandwidth. To pick a high-bandwidth index for PIR queries, the client selects a low-value index. In order to make appropriate choices for bandwidth-weighted selection of nodes, we propose that each Tor client download a bandwidth distribution synopsis from the directory servers. An alternative is for them to use the Snader-Borisov [SB08] criterion for node selection, which utilizes only the relative *rank* of each node in terms of its bandwidth, and not the bandwidth values themselves.

The PIR block size should be large enough to fit at least one descriptor; however, more may fit in a single block. Padding should be used to ensure that descriptors do not cross block boundaries. We propose to use the threshold BLS signature scheme [BLS01] to sign each block since the signature is short (a single group element of 23 bytes, for 80-bit

security), regardless of the number of directory authorities issuing signatures.

Querying directory servers. The client queries directory servers with PIR to retrieve descriptors well in advance of circuit construction. Note that the client may be unable to determine in advance the exit policies required for circuit construction. The client can overcome this by retrieving blocks with descriptors that support all standard exit policies. In addition, the user may download non-standard exit policies periodically (e.g., every 3 hours) when they are few.

A client initiates a query by connecting to one of its directory servers, which responds with timestamped and signed meta-information about each PIR database, such as the number of blocks in the database, the block size, the distribution of exit policies, and a bandwidth synopsis. Afterwards, the client generates PIR queries based on the information to retrieve descriptors.

A client can optimize node selection for ITPIR-Tor by not retrieving a middle node with PIR. Instead, the client retrieves (non-privately) a single middle node descriptor from one of its guard nodes. This optimization is possible because ITPIR-Tor uses guard nodes as PIR servers, and uses them as the first node in every circuit creation. The middle node so retrieved should only be used for circuits originating for the same guard node; otherwise, even a single compromised guard node would be able to perform fingerprinting attacks [DC06].

Creating circuits. The circuit creation mechanism for ITPIR-Tor and CPIR-Tor remains the same as with the current Tor network. In the current Tor network, a client recreates circuits every 10 minutes. We find the cost of all Tor clients performing one PIR query every 10 minutes manageable for ITPIR-Tor, whereas the cost is higher for CPIR-Tor. Therefore, we propose to perform fewer PIR queries for CPIR-Tor and reuse descriptors within each 3-hour window. We provide more details in Subsection 8.1.5.

8.1.4 Security and Privacy Implications (sketch)

We provide a sketch of the security and privacy implications of PIR-Tor. The full details are available in the original published work [MOT+11].

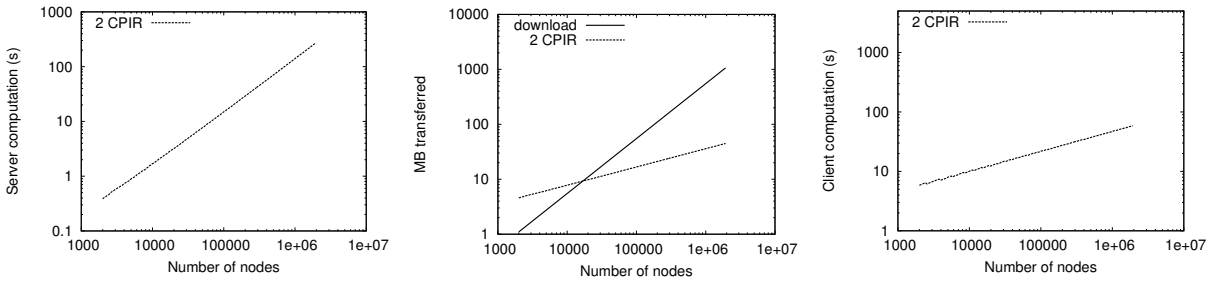
We analyzed how resistant our designs are to traffic analysis by considering a realistic adversary who can:

- Control a fraction of the Tor network, either by compromising existing nodes or by introducing new nodes, where she can observe traffic.
- Create, modify, delete, or delay traffic
- Observe clients' PIR queries to directory servers and know that a client only learns a fixed number of descriptors per query.

Prior work [DC06, DS08] has shown that the success of *route fingerprinting* attacks increases as clients' knowledge about the total number of nodes in an anonymity network becomes limited. These papers assumed node discovery by clients is not anonymous; that is, the adversary can learn which subset of the network a user knows about. This knowledge can help build a mapping between the profiles of users and their potential choices of nodes for building circuits. In cases where the set of nodes discovered by a user is disjoint (or unique) from those discovered by other anonymous users, it is trivial for the user's circuit to be "fingerprinted" or mapped to her identity. The current Tor network is not prone to route fingerprinting attacks because clients maintain an up-to-date view of the entire network.

In the case of PIR-Tor, the threat model is slightly different. The use of PIR completely hides node discovery information from the adversary. What the adversary learns from a PIR query is that a client has retrieved a block (potentially containing one or more descriptors), but is unsure of which particular block. This makes it impossible to localize the knowledge of a client to any specific partial view of the network. However, if the adversary controls the entry guard and the exit node of a circuit constructed by a client, then the adversary is able to use this additional knowledge to link a client with the destination of her traffic and perform a traffic confirmation attack [Ray00]. This latter attack will occur with equal chances in PIR-Tor as in the current Tor network.

Clients for our ITPIR-Tor architecture can request new descriptors for each circuit creation because the computation and communication cost of ITPIR is small. As a result, the adversary must assume that the client may know any node, and ITPIR-Tor fingerprinting resistance is equivalent to the current Tor network. In the case of CPIR-Tor, the higher computation and communication cost of CPIR demands for the client to reuse descriptors. Unfortunately, reusing descriptors for creating multiple circuits improves the chances the adversary has in aggregating profiles of nodes in the connection made by a client. Thus, CPIR-Tor does not break anonymity, but breaks unlinkability of circuits. However, the window of linkability is limited to 3 hours, after which the client retrieves fresh descriptors and creates new circuits. If a client retrieves $b > 1$ blocks every three hours, and rebuilds



(a) Server computation (b) Total communication (c) Client computation

Figure 8.1: Computation and data transfer costs for a single CPIR-Tor circuit (one PIR query for exit node and another query for middle nodes).

a new circuit every 10 minutes, then the adversary is limited to linking data for only $36/b$ circuits (the client requires 2 descriptors for each circuit creation).

8.1.5 Performance Evaluation

We experimentally evaluated our CPIR-Tor and ITPIR-Tor architectures on a dual Intel Xeon E5420 2.50 GHz quad-core machine running Ubuntu Linux 10.04.1. We base our results on numbers for a single core, and assume a bandwidth of 9 Mbps, being the standard home Internet connection for Canada and the US [Ook10]. We measured the client/server computation cost and communication cost by running an implementation [GPG09] of the CPIR scheme of Aguilar-Melchor et al. [AMG07], and the Percy++ implementation [Gol07b] of Goldberg's multi-server PIR scheme [Gol07a]. We chose the standard CPIR security parameters [AMG07] ($\ell_0 = 19$ and $N = 50$), set the number of ITPIR servers to 3, and set the size of a descriptor to 2 100 bytes (the maximum descriptor size we measured from the current Tor network). In addition, we varied the number of nodes in a PIR database and set the middle database to be twice the size of the exit database [Tor11a]

Reducing ITPIR-CPIR communication cost. Unlike existing CPIR schemes that rely heavily on number theory, the lattice-based CPIR scheme [AMG07] transmits more data per query, but has the best overall response time due to its lower computational cost (see Chapter 4). Data transfer for this scheme (as well as other CPIR schemes) can be reduced using the recursive construction by Kushilevitz and Ostrovsky [KO97] without much increase in computational cost; this recursion can be implemented in a single round of

interaction between the client and the server. We denote the recursion parameter in CPIR using R. If we denote the number of nodes in the database by n, then the communication cost of CPIR in our architecture is proportional to $8^R \cdot n^{1/(R+1)}$. We explored various values of the recursive parameter R and found that $R = 2$ gives the best overall performance. For example, when $R = 3$, the CPIR scheme requires about the same amount of computation, whereas $R = 2$ transmits less data. Beyond $R = 3$ the communication cost becomes higher because the term 8^R begins to dominate in the expression.

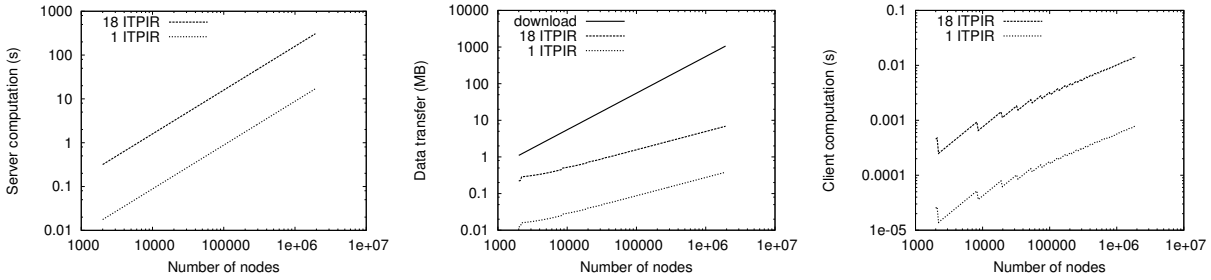
Results and Analysis

Figure 8.1 shows the server computation, data transfer, and client computation costs for two CPIR queries; one for a middle node and another one for an exit node. In terms of data transfer, performing two CPIR queries is more expensive for the current network size. However, the client needs to create 18 circuits every 3 hours (rebuild a circuit every 10 minutes). Tor clients are able to amortize the cost of a single download over the 3 hours period. However, performing 36 CPIR, 18 for each of middle and exit nodes, will be more expensive than downloading the complete databases unless the number of nodes in the Tor network grows to over 40 000. We instead propose to perform $b < 36$ CPIR queries for both middle and exit nodes and reuse nodes within each 3-hour window. As noted previously, reusing nodes for rebuilding circuits does not affect anonymity, but breaks unlinkability of multiple circuits.

Similarly, we plot the server computation, data transfer, and client computation cost for ITPIR-Tor in Figure 8.2. We observe that the cost of performing a single ITPIR query is at least 2 orders of magnitude smaller than the cost of trivial download. The cost of PIR queries to build 18 circuits every 3 hours is even at least an order of magnitude smaller. Thus, the client does not have to reuse nodes for rebuilding circuits, resulting in security that is equivalent to the current Tor network. The regular vertical drop in client compute time in Figure 8.2(c), the zigzag pattern, results from fitting more node descriptors into a PIR block. As the PIR block size increases gradually, it becomes large enough to fit an additional descriptor, and the number of blocks in the database jumps down.

Growth Projections

We project several growth scenarios for the Tor network to evaluate how well CPIR-Tor and ITPIR-Tor would scale in comparison to the Tor network. We base our projection on the data transfer cost and the number of cores it would take to compute the required



(a) Server computation (b) Total communication (c) Client computation

Figure 8.2: 3-server ITPIR cost.

number of PIR queries, and set $b = 1$ block. We consider the following growth scenarios 1) the current Tor network size of 2 000 nodes and 250 000 clients; 2) increasing the number of clients by a constant factor; 3) increasing the number of nodes by a constant factor; 4) increasing both the number of nodes and clients by a constant factor; and 5) Allowing all clients to become middle-only nodes.

At the current Tor network size of 2 000 nodes and 250 000 clients, the server compute time for 2 000 node descriptors for CPIR-Tor is 0.2 s. The number of exit nodes is about half the total number of nodes; i.e., 1 000, with 0.1 s server compute time. For CPIR-Tor, the time to retrieve an exit node and a middle node combined is therefore 0.3 s. Therefore a directory server core can support 36 000 clients $\left(\frac{3 \cdot 60 \cdot 60}{0.3}\right)$ every 3 hours, and we would need 7 cores to support 250 000 clients (at less than 10% utilization since there are more than 100 cores). As of February 2011, the network consensus size is 560 KB, while the node descriptor size is 3.3 MB. In a 3-hour period the current Tor client would download 1.1 MB (the entire network consensus and a sixth of the size of node descriptors), whereas CPIR-Tor client would download 2 MB. We can see that CPIR-Tor is not a better option for the current Tor capacity. For ITPIR-Tor, the total compute time to support a single circuit is 0.005 s. Each guard node core can support 36 000 clients. Each of the current 500 guard nodes would service PIR requests for an average of 1 500 clients (three guard nodes per request), which amounts to 0.4% of one core. The communication cost of building a single circuit in this case is 12 KB, and the cost of building 18 circuits is 216 KB. Thus, ITPIR-Tor transmits less data even on the current Tor network size.

For other growth scenarios, the data transfer cost and number of cores required to support CPIR-Tor scales sublinearly in the number of nodes, whereas the Tor network would scale linearly in the number of nodes. For example, when the number of nodes becomes 20 000, the total data transfer and cores needed for CPIR-Tor every 3 hours is

respectively 4 MB and 59, whereas Tor would require 11 MB download. As the number of nodes increases, it becomes more and more advantageous to use CPIR-Tor in the place of Tor. When all clients are made middle-only nodes, the server compute time for middle nodes increases to 20 s, while the server compute time for exits remains the same (i.e., 0.1 s). Therefore, a total of 466 cores would be required to support CPIR-Tor. In terms of data transfer, CPIR-Tor costs just 8 MB every 3 hours, compared to 111 MB for Tor.

ITPIR-Tor maintains its advantages for all of the growth scenarios considered. The data transfer cost is between 0.2 MB to 0.5 MB, while the core utilization is between 0.4% and 5% for network growth of up to 10 times. Finally, converting all clients to middle-only nodes would not increase the cores utilization of our ITPIR-Tor architecture because we retrieve middle nodes non-privately (without PIR).

The CPIR-Tor architecture is easier to deploy because only a fraction of the directory servers would need to maintain an up-to-date view of the network to be able to support PIR requests. It is advantageous in situations where a client's browsing time is small, or when the client is not concerned about linkability of circuits. In the case of ITPIR-Tor, all guard nodes must be converted to directory servers and would need to maintain a view of the network for it to be deployed. However, the overall data transfer by client is minimized and the architecture scales better.

Limitations

The Tor network runs on bandwidth donated by volunteer nodes. We have proposed an architecture that trades off bandwidth for computation at the directory servers, and thus directory servers are required to volunteer some extra computational resources. We find in our performance evaluation that directory servers need to volunteer only a small fraction of their CPU resources, especially in the case of ITPIR-Tor. In addition, the communication overhead in our architecture is at least an order of magnitude smaller than that of Tor for all growth scenarios. Overall, we argue that PIR-Tor offers a good compromise between bandwidth and computational resources, and results in an overall reduction in resource consumption at volunteer nodes. Secondly, PIR-Tor is not as scalable as alternate peer-to-peer approaches, which can scale to tens of million nodes. However, our design provides improved security properties over prior work. In particular, reasonable parameters of ITPIR-Tor provide equivalent security to that of the Tor network. The security of our architecture mostly depends on the security of PIR schemes, which are well understood and relatively easy to analyze, as opposed to peer-to-peer designs that require analyzing extremely complex and dynamic systems. Finally, PIR-Tor assumes that the exit database

contains a small set of standard exit policies, though it can tolerate a few outliers by downloading their information in their entirety.

8.1.6 Conclusion

We have presented CPIR-Tor and ITPIR-Tor as a two PIR-based architectures that could allow Tor to scale better. Our architectures eliminate the need for clients to maintain an up-to-date view of network nodes, while preventing compromised Tor directory servers from learning the choice of nodes the user selects for creating circuits. We experimentally evaluated the two flavours of our architecture and found that they reduce data transfer cost in Tor by at least an order of magnitude. For CPIR-Tor, the client retrieves only a few blocks and reuses them for multiple circuit creation. We show that reusing circuits breaks linkability, but anonymity of the communication is preserved. In the case of ITPIR-Tor, a client does not have to reuse circuits, thereby offering a security that is equivalent to the current Tor network. Finally, ITPIR-Tor may be more difficult to deploy because it requires all guard nodes to become PIR servers.

8.2 PIR-Commerce: Privacy-preserving e-Commerce

This section is adapted from published work co-authored with Ryan Henry and Ian Goldberg [HOG11].

We extend Goldberg's multi-server information-theoretic PIR scheme to enable privacy-preserving online purchase of digital goods. Our first extension is a symmetric PIR (SPIR) scheme, which adds support for database privacy under computational assumptions (the user privacy is still information-theoretically protected). We then extend the SPIR to a priced symmetric private information retrieval (PSPIR) scheme, which allows for tiered pricing, such that users can purchase records from the database without the database being able to learn any information about the index or the price paid for the record. Tiered pricing allows the database to set prices based on the status of users. For example, platinum members pay at the most discounted price, while gold members pay a higher price, and other members pay full price. When a user buys a record, the membership status of the user is not revealed by the transaction. Finally, we provide an extension to PSPIR to support group-based access control with record-level granularity, such that the database is able to restrict the purchase of some records according to membership status. There are no other PSPIR or oblivious transfer protocols that support tiered pricing and access control

lists, while preserving the sublinear communication complexity requirement of PIR. We also implemented our protocols as an add-on to Percy++ [Gol07b]. Measurements indicate that our protocols are practical for deployment in real-word e-commerce systems.

8.2.1 Introduction

In multi-server PIR, ℓ database servers each possess a replica of the database and a user submits his query to some size-k (or larger) subset of these servers in such a way that no server (or coalition of servers up to some threshold t) can learn the user's query. One can view the database X as consisting of n bits organized into r records, each of size $b = \frac{n}{r}$ bits. We follow the usual convention of specifying a PIR query by the index i of interest. Thus, in a PIR query, the user retrieves the record at index i without the servers learning any information about i. We note that our SQLPIR contribution (Chapter 5) allows one to build SQL-based queries atop this basic index-based retrieval model. Existing multi-server PIR schemes offer information-theoretic privacy protection for the user's query, but they allow a dishonest user to obtain additional information, such as the record at index $j \neq i$, or the exclusive-or of some subset of records in the database [GIKM98a]. However, for many real-world applications, protecting database privacy by preventing dishonest users from learning extra information about the database is advantageous. Examples abound in online sales of digital goods, such as a pay-per-download music store [AIR01] where users must pay for each song they download, a pay-per-retrieval DNA database [CDN09], a stock-information database [GIKM98a], or a patent database [Aso04]. In all of these practical situations, it is necessary to guarantee the seller of these digital goods that users learn exactly the database record of interest and nothing more. In some scenarios it may even be desirable to sell database records according to a *tiered pricing plan* whereby different users pay different prices for each record depending on, e.g., their membership status or geographic location.

Symmetric private information retrieval [GIKM98a] adds an additional restriction to PIR that prevents the user from learning information about any records except for the one he requested, thus addressing the need for simultaneous user and database privacy; however, no existing SPIR scheme supports both (tiered) record-level pricing and access control lists. Some oblivious transfer (OT) schemes [AIR01, CDN09, CDN10, CDNZ11] offer one or the other of these functions, but no scheme in the literature provides them both. Moreover, OT schemes generally have no requirement for sublinear communication complexity, which renders them useless for online sales of some types of digital goods, such as multimedia data, where the bandwidth requirement is high. Some schemes even require the user to download an encrypted copy of the *entire database* (e.g., [CDN09, CDN10,

CDNZ11]) and later purchase decryption keys for individual encrypted records. This allows one to amortize the cost of many transactions, but renders the scheme unsuitable for applications in which the contents of the database change frequently. Storing the database in an encrypted format also limits the usefulness of the database for other applications that need ready access to the cleartext data. Other OT-based schemes [AIR01] require the database servers to store state information, such as the number of purchases made by a user, or his remaining balance, which might leak information about the user's queries or enable the server to link his purchases.

In this section, we present a protocol that extends the open-source PIR scheme by Goldberg [Gol07a] first to an SPIR scheme, and then to a priced symmetric private information retrieval scheme offering tiered pricing with record-level granularity. Our initial PSPIR construction allows a *content provider* or merchant to sell digital goods through a distributed database and collect all proceeds from these sales. We then extend this simple scheme to enable the database servers to control access to individual records by implementing group-centric access control lists. These enhancements provide a stronger and more realistic model of private information retrieval that enables strong privacy protection for e-commerce.

In our model, users belong to different pricing tiers and pay (perhaps different amounts) for each record; moreover, the database may require users to have explicit authorization to access some or all of the records in the database. In particular, tiered pricing logically groups users into different price tiers and allows the database to set the price and availability of each record with respect to each tier (a price tier is then roughly analogous to a *group* in the context of access control). Our approach enforces these constraints without revealing the user's price tier to the servers during a protocol run. Thus, when run over an anonymous communications channel, our protocols maintain user anonymity in addition to access privacy; that is, the database servers do not learn any information about the identity nor the query of the user. More specifically, queries do not leak information about the index or price of the purchased record, the price tier according to which the user pays, the user's remaining balance, or even whether the user has ever queried the database before.

Outline.

The reminder of this section is organized as follows: Subsection 8.2.2 gives our system model and threat model. Subsection 8.2.3 gives sketches of our SPIR and PSPIR constructions as well as the extension of PSPIR to support group-centric access control. In Subsection 8.2.4, we discuss our implementation of the protocols and our result. We conclude this section in Subsection 8.2.5.

8.2.2 System Model, Threat Model, and Use Case

Our system model consists of three players: the *merchant*, the *bank*, and the *user*. The merchant is a seller holding a database of several tens of gigabytes of digital goods, such as MP3 files and e-books, divided into r records (or files). Our system relies on multi-server PIR which implies ℓ non-colluding servers hold a copy of the database, and the user must submit query to a subset of $k > \ell/2$ of them. We note that our scheme can easily use a random server model of PIR [GIKM98a], where the merchant can replicate a random-looking database to multiple non-colluding servers and the user purchases records by querying the merchant's server (containing the real database) as well as a subset of $k > \ell/2$ of the random servers. We associate T price tiers to the database, such that each price tier encodes the prices of all r records as nonnegative integers or $\perp$ to indicate that a record is unavailable for that price plan. We do not make any assumption about the size of T. Pricing tiers are usually not many; under 10. A large T, such as over 1000, may impair performance.

The bank is a credential issuer of digital *wallets* to users. We did not discuss the semantics of the bank, but any non-rerandomizable (one-show) credential issuing system, such as that of Brands [Bra95] or Au et al. [ASM06] should suffice. Each wallet encodes a balance (0 when initially issued) and the index of the user's price tier. A user loads funds in her wallet using, for example, prepaid credit cards purchased via cash transactions from a grocery store.

The user is any online shopper interested in privacy-preserving purchase of digital goods from the merchant using a *wallet* that encodes a price tier and some balance. A user buys a record by proving to the merchant's servers that her wallet encodes sufficient funds for the price of that record in her price tier. She does this by sending her wallet to some $k > \ell/2$ servers as well as her query. The servers check that the wallet has not been spent before and then issue a cryptographically signed *receipt* that encodes the price of the record and the wallet used to make the payment. The user then uses the receipt to obtain a new wallet, unlinkable from her old one, which encodes here remaining balance (if any). Note that the purchase and wallet refreshing processes do not reveal any information to the merchant or bank about the user's price tier, the record purchases, the price paid for the record, the remaining balance, or any information that could be used to link the user's old and new wallets.

As a motivating use case, we consider the online purchase of e-books, such as from Amazon's Kindle Store. Interestingly, Amazon announced in May 2011 that in the last year, they have sold more e-books than paperbacks and hardbacks combined [Ama11]. The growing popularity of e-books suggests the deployment of a way to purchase e-books

that does not reveal the user's reading preferences will only grow in relevance. The recent lawsuit by Amazon against the state of North Carolina to prevent the disclosure of customer purchase records [McC10] further justifies why a privacy-preserving alternative to purchase Amazon's e-books will be an atractive offering to privacy-conscious customers.

Our threat model considers users that are malicious, and merchants (and database servers) that are honest-but-curious. However, Goldberg's PIR scheme — and by extension, our proposed scheme — is robust against some threshold of malicious database servers through the Byzantine robustness property as well.

8.2.3 Constructions (sketch)

We provide highlights of our constructions in this subsection. Complete details and proof of security are available in the full paper [HOG11]. We begin with an extension of Goldberg's PIR scheme to an SPIR scheme, and later extend the SPIR to support our model of tiered pricing.

SPIR Construction

We converted Goldberg's PIR scheme into Symmetric PIR to ensure that no single query will ever reveal information beyond a single database record. Recall from Section 2.4.2 that the constant terms of the random polynomials a client selects when generating a query for Goldberg's PIR scheme are 0 everywhere except at the position of the record sought (i.e., i), where it is 1. A dishonest client that chooses 1 in more than a single position would be able to retrieve partial information about more than a single database record. Running malformed queries of these type multiple times and aggregating their results by interpolation will help the user learn about more records than the total number of queries sent. Thus, the client needs to convince the servers that *exactly one* of the polynomials has a non-zero constant term.

We accomplish the SPIR construction using the $PolyCommit_{DL}$ polynomial commitment construction of Kate et al. [KZG10]. Polynomial commitments allow a client to create constant-sized commitments to polynomials, which the servers can use at a later time to verify evaluation of the polynomials to some point, even while remaining oblivious of the polynomials. We call the commitment a client creates and sends to a server to confirm the evaluation of a polynomial at some point a *witness*. Recall from Section 2.4.2 that the client generates r polynomials. Our extension requires the client to commit to these polynomials, provide witnesses to the evaluation of the polynomials at the servers' respective

indices, and prove in zero knowledge that the polynomials evaluate to a standard basis vector (i.e., the client's query vector e_i) at a point $x = 0$. We also use the threshold BLS signature [BLS04] in such a way that each server signs and forwards a signature on the commitment it receives from the client. The client combines these signatures into a single group signature and forwards it the servers, which can then verify that, indeed, they are all seeing the same commitments. This defeats attacks from a dishonest user that rely on sending commitments to different polynomials to each server.

Further, a dishonest user may choose her polynomial in such a way that it reveals information about some other database record when interpolated at a point $x = a$. This is not surprising because Gertner et al. [GIKM98a] has proved that information-theoretic SPIR is impossible to accomplish without some interactions among the servers or a shared secret among them. We address this problem by having the servers share a secret key sk that is unknown to the users. This is a reasonable assumption to make since the servers are already sharing the same copy of the database. Each server j uses $F_{sk}(\vec{C})$ to seed a pseudorandom generator (PRG) and generate a common ephemeral $(r+1)^{th}$ database record for the current query, where F is a pseudorandom function family and $\vec{C}$ is a vercor of the commitments. It also generates a polynomial f_{r+1} of degree t with constant term 0, and non-constant terms drawn pseudorandomly from $GF(2^8)$, and appends $f_{r+1}(j)$ to the ρ_j it receives from the client. Afterwards, the server encodes a response to the query exactly as in Goldberg's original scheme (see Section 2.4.2).

Our SPIR construction provides query privacy information theoretically against up to $t-1$, and computationally (under the Discrete Logarithm assumption [KZG10]) against t, colluding servers. The database privacy is protected computationally under the t-Strong Diffie-Hellman assumption [KZG10]. Full protocol details and proof are available in the original paper [HOG11]. In addition, our construction preserves the t-private v-Byzantine-robust k-out-of-ℓ properties of Goldberg's scheme [Gol07a], but not the τ-independence property. The rerandomization of the user's query with the ephemeral database record breaks the latter property.

PSPIR Construction

We extend our SPIR construction to a priced symmetric PIR as follows. First, the user and the servers independently compute a receipt as a commitment to the price of the record for purchase, under each of the price tiers. We take advantage of the homomorphic property of polynomial commitments: given two commitments $C_1 = g^{\phi_1(\alpha)}$ and $C_2 = g^{\phi_2(\alpha)}$ to two polynomials ϕ_1 and ϕ_2, the product of the commitments C_1C_2 is a commitment to their

sum $\phi_1 + \phi_2$. [KZG10] We have the user prove in zero knowledge that a receipt encodes the right price for the record according to her price tier, and that her wallet has enough balance to complete the purchase.

Once convinced of the proof, each server issues a BLS signature on the receipt and the user's wallet, and then proceeds to process the query as in the SPIR construction.

If the user wishes to make another purchase at a future date, then she must run a credential issuing protocol with the bank using the BLS signature, receipt and wallet. The bank verifies the signature is indeed valid and the user obtains a new wallet encoding her new balance. Note that the old and new wallet are unlinkable.

PSPIR and access control

We make a simple modification to the PSPIR construction to implement access control lists. The idea is to impose a maximum balance $b_{\max}$, and have them prove that their new balance does not exceed $b_{\max}$ each time they refresh their wallets with the bank. The bank will not issue a new wallet to any user that is unable to pass the proof. The rest of the protocol description remains the same with the exception that we now interpret $\perp$ in the price tiers to mean $b_{\max} + 1$. These simple changes ensure that no user can ever purchase records marked with $\perp$, for which they are not allowed.

8.2.4 Implementation and Evaluation

We implemented the protocols using Ben Lynn's PBC (Pairing-Based Cryptography) library [Lyn11] with Aniket Kate's PBCWrapper package [Kat11] for C++ Wrapper classes, Victor Shoup's NTL [Sho11] with the GNU Multi-Precision Arithmetic Library [Fre11] for multi-precision arithmetic, and OpenSSL [Pro11] for hash functions (we use SHA-256). Our PSPIR implementation is built atop Ian Goldberg's implementation of his PIR scheme, Percy++ [Gol07b]. For our evaluation, we implemented the protocol as a standalone add-on to Percy++, but we will later integrate it with the Percy++ library. All measurements were taken in Ubuntu Linux 10.04.1 LTS running on a machine with Dual Intel Xeon E5420 2.50 GHz CPUs and 32 GB memory. The value of the modulus for the polynomial operations was 160 bits long.

We measured the performance of the implementation of our PSPIR protocol for various values of the PIR parameters n (the size of the database), b (the size of each record in the database), k (the number of servers participating in each query), and t (the number of servers that can collude without affecting query privacy).

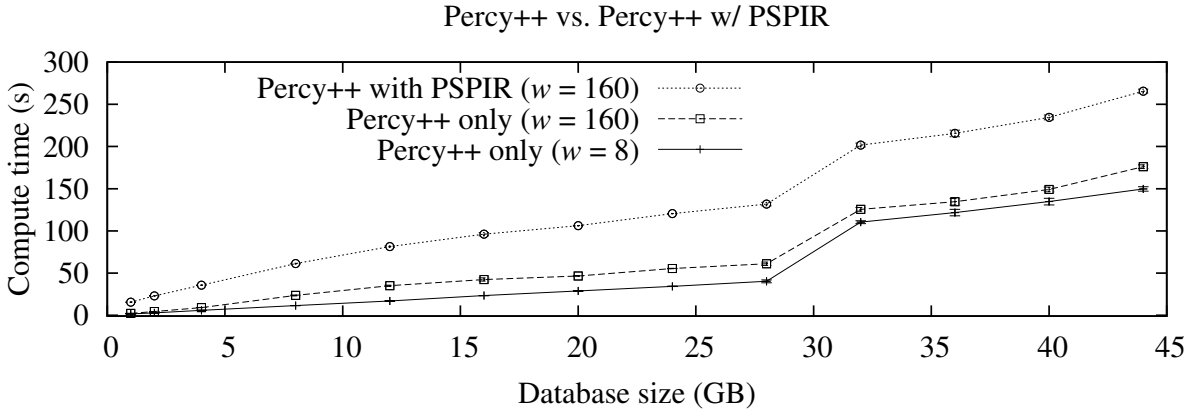


Figure 8.3: Query execution time for Percy++ with and without PSPIR ($k = 4$, $t = 2$). The percent compute time attributable to the PSPIR enhancements decreases monotonically from $\approx 86\%$ for a 1 GB database down to $\approx 33\%$ for a 44 GB database. Percy++ starts carrying the extra overhead of disk reads after a 28 GB database, which exceeds available RAM. The $w = 8$ plot shows the execution time for Percy++ using its performance-optimal parameter choices, whereas the $w = 160$ plot shows Percy++ with parameters needed to ensure the security of PSPIR. The Percy++ with PSPIR plot shows the combined cost of Percy++ with $w = 160$ and the PSPIR enhancements. Error bars are plotted for all data points, but are small and may therefore be difficult to see. For comparison, downloading a 44 GB database in OT-based schemes takes over 11 hours at 9 Mbps, which is the average Internet bandwidth in Canada and the US [Ook10].

Our experiment measures the computational overhead added to Percy++ by the PSPIR enhancements. We generated databases of sizes ranging from 1 GB to 44 GB containing random data, and took measurements for both Percy++ and Percy++ with PSPIR. We set $k = 4$, $t = 2$, and $b = \sqrt{160n}$, which is the communication-optimal record size for this PIR scheme. Figure 8.3 shows the plots of the measurements for Percy++ with and without PSPIR. We observe that PSPIR results in a moderate increase in compute times, with the percent compute time attributable to the PSPIR enhancements decreasing monotonically from about 86% for a 1 GB database down to just 33% for a 44 GB database. The upward bump just before 30 GB marks the point after which the database no longer fits in available memory. From that point on, every query bears more overhead from disk reads. In terms of communication overhead, PSPIR increases Percy++'s query size, which is itself just k times the size of the retrieved record, by a multiplicative factor of about 5. However, it increases each server's response by only 46 bytes, which corresponds to two BLS signature

shares.

We observe, however, that the use of 160-bit words in the database to guarantee security for the commitments used in PSPIR slightly degrades Percy++'s performance from its optimal setting of 8-bit words.

8.2.5 Conclusion

We have extended Goldberg's multi-server information-theoretic PIR with a suite of protocols for privacy-preserving e-commerce. Our protocols add support for symmetric PIR, priced symmetric PIR with tiered pricing, and group-based access control lists with record-level granularity, while preserving the sublinear communication complexity of PIR. We have implemented our PSPIR protocol atop Percy++, an open-source implementation of Goldberg's PIR scheme, and evaluated its performance empirically. Our measurements indicate the performance of our protocols is reasonable, making them acceptable for deployment in real-world e-commerce systems. Furthermore, the right mix of features, the fact that our SPIR construction preserves PIR's sublinear communication complexity, and the ability to apply it over unencrypted databases, makes it more practical than competing OT-based schemes. For future work, we intend to optimize our implementation to incorporate our protocols into Percy++.

Chapter 9

Conclusions

The main goal of this thesis is to show that current private information retrieval schemes can be usefully engaged to preserve access privacy in many real-world practical situations. It is our desire that the results discussed will stimulate further research and practical deployment of PIR-based systems.

While it may be infeasible to obtain absolute access privacy protection in some problem domains due to the scale of the databases, we are quite convinced that current PIR schemes are able to deliver absolute access privacy in many problem areas. In situations where absolute privacy cannot be attained, our approach to preserving access privacy over very large databases can help end users trade off access privacy for computational performance and *vice versa*. We encourage system designers to adopt PIR, even when it can only deliver access privacy on a subset of a large data set. As research and development continues to improve, PIR will become more and more practical and relevant to people's transactions over the Internet.

We already highlighted some future work related to our contributions. Several other open problems remain.

In order to maintain the user's privacy with information-theoretic PIR, it must be the case that not all [CGKS95] or at most a configurable threshold number [Gol07a] of the database servers collude to unmask the user's query. This is sometimes brought forward as a problematic requirement of these schemes. As we noted in Chapter 5, there are reasonable scenarios — such as distributed databases like DNS or whois databases, where the copies of the database may be held by competing parties — in which the non-collusion requirement is acceptable. Further, other privacy-enhancing technologies, such as anonymous remailers [DDM03] and Tor [DMS04a], also make the assumption that not all of

the servers involved are colluding against the user. The above considerations apparently limit the application of multi-server PIR in environments where the database is not naturally distributed. There is need for an exploration of practical means that will reduce the likelihood of collusion to a similar extent as Tor.

Along similar lines, there is need for research on deploying PIR in a cloud computing environment where each PIR server is located on a cloud belonging to different geographical regions of the world, managed by different infrastructure providers, and controlled by different entities. It is important for there to be a practical means to detect and prevent situations with high likelihood of collusion, such as when some supposedly independent servers are colocated on the same cloud or are running on different clouds but by the same holding organization. Cloud computing provides an "unlimited" amount of computation power for making PIR queries. It is especially relevant to PIR because of the possibility to configure the servers to dynamically scale up or scale down the amount of computation power for servicing user queries. Dynamic scaling of available computation power is not usually available for locally hosted solutions.

Closely following the above is another interesting future work that explores the random server model of PIR [GGM98] to make PIR more practical. The goal is to make it possible for users to interact with various PIR servers hosted on the cloud to answer their queries. The same set of PIR servers can be used to answer the query of users for different databases (multiple organizations providing PIR-based services). The PIR server itself will not contain any information about the individual data of the organizations. The random server model of PIR makes it possible for database owners to use PIR without physically distributing their database to untrusted holders.

The current random server model scheme [GIKM98a], however, does not guarantee privacy for repeated queries because the adversary is able to observe when queries are repeated. They proposed a buffering technique to keep track of query-response pairs (j, X_j) such that an X_j is retrieved from the buffer if it has been requested before, otherwise it is retrieved from the PIR server (using their basic scheme). A random element of X_i is also retrieved if X_j is already in the buffer. However, the technique adds $m \log n$ communication complexity to the scheme for a buffer size of size m and requires the scheme to be reinitialized (database setup) after every m queries.

The work may begin with adapting Goldberg's scheme and its implementation library to use the random server model on cloud computing infrastructures to protect the content of a PIR database. The extension would not require each server to hold a separately precompiled database and would protect the privacy of repeated queries without requiring reinitialization of the PIR database. In this manner, a single server can use its independent

database to service the retrieval requests of users from multiple PIR-based systems.

With regards to improving the computational performance of multi-server PIR schemes, it has been claimed that PIR schemes generally exhibit one order of magnitude speedup by using GPUs instead of CPUs to do the bulk of the computation [AMCG+08]. An interesting future work is to study to what extent the computational performance of Goldberg's multi-server scheme can be enhanced by such parallelization. The database of the PIR server can be shared between different read devices with each portion processed with different cores or GPUs. Research in this direction is important to our goal of making PIR more practical because the implementation of this PIR scheme [Gol07b, Gol07a] is the only available open-source multi-server PIR library. A parallelized version of the library with better performance will encourage application developers to consider using this PIR scheme in their systems.

References

[AF02] Dmitri Asonov and Johann-Christoph 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.

[AF03] Dmitri Asonov and Johann-Christoph Freytag. Almost optimal private information retrieval. In *PET'02: Proceedings of the 2nd International Workshop on Privacy Enhancing Technologies*, pages 209–223, 2003.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Proceedings of EUROCRYPT 2001*, Innsbruck, Austria, May 2001.

[AKS08] Houtan Shirani-Mehr Ali Khoshgozaran and Cyrus Shahabi. SPIRAL, a scalable private information retrieval approach to location privacy. In *Proceedings of the 2nd International Workshop on Privacy-Aware Location-based Mobile Services (PALMS)*, 2008.

[ALD08] Niwaer Ai, Ying Lu, and Jitender Deogun. The smart phones of tomorrow. *SIGBED Rev.*, 5(1):1–2, 2008.

[Ama11] Amazon.com. News release: Now selling more kindle books than print books, Accessed May 2011.

[Amb97] Andris Ambainis. Upper bound on communication complexity of private information retrieval. In *ICALP'97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 401–407, London, UK, 1997.

[AMCG+08] Carlos Aguilar-Melchor, Benoit Crespin, Philippe Gaborit, Vincent Jolivet, and Pierre 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.

[AMD06] Carlos Aguilar-Melchor and Yves Deswarte. Single-database private information retrieval schemes : Overview, performance study, and usage with statistical databases. In *Privacy in Statistical Databases, CENEX-SDC Project International Conference, PSD 2006, Rome, Italy, December 13-15, 2006*, Lecture Notes in Computer Science, pages 257–265, 2006.

[AMG07] Carlos Aguilar-Melchor and Philippe Gaborit. A lattice-based computationally-efficient private information retrieval protocol. In *WEWORC*, July 2007.

[APV02] Lars Arge, Octavian Procopiuc, and Jeffrey Scott Vitter. Implementing I/O-efficient data structures using TPIE. In *ESA'02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 88–100, London, UK, 2002.

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In *Proceedings of SCN 2006*, Maiori, Italy, September 2006.

[Aso04] Dmitri Asonov. *Querying Databases Privately: A New Approach To Private Information Retrieval*. SpringerVerlag, 2004.

[BBD09] Djamal Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *ESA 2009: Proceedings of the 17th Annual European Symposium, September 7-9, 2009*, pages 682–693, 2009.

[BBdCR09] Fabiano Cupertino Botelho, Djamel Belazzougui, and Davi de Castro Reis. CMPH: C minimal perfect hashing library on sourceforge, Accessed June 2009. http://cmph.sourceforge.net/.

[BCR86] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *Proceedings of CRYPTO 1986*, Santa Barbara, CA, 1986.

[BDMT07] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In Sabrina De Capitani di Vimercati and Paul Syverson, editors, *ACM Conference on Computer and Communications Security (CCS 2007)*, pages 92–102, 2007.

[Bei08] Amos Beimel. Private information retrieval: A primer. Department of Computer Science, Ben-Gurion University (Submitted for Publication), Jan 2008.

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *CLOT '92*, pages 144–152, 1992.

[BIM00] Amos Beimel, Yuval Ishai, and Tal 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, 2000.

[BIM04] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: Pir with preprocessing. *J. Cryptol.*, 17(2):125–151, 2004.

[BJSW08] Claudio Bettini, Sushil Jajodia, Pierangela Samarati, and Xiaoyang Sean Wang, editors. *Proceedings of the 1st International Workshop on Privacy in Location-Based Applications, Malaga, Spain, October 9, 2008*, volume 397 of *CEUR Workshop Proceedings*, 2008.

[BKOS07] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith, III. Public key encryption that allows PIR queries. In *CRYPTO'07: Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, pages 50–67, 2007.

[BLPW08] Bhuvan Bamba, Ling Liu, Peter Pesti, and Ting Wang. Supporting anonymous location queries in mobile environments with privacygrid. In *Proceeding of the 17th international conference on World Wide Web*, pages 237–246, New York, NY, USA, 2008.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *7th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2001)*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, 2001.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, January 2004.

[BMG+07] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against Tor. In Ting Yu, editor, *ACM Workshop on Privacy in the Electronic Society (WPES 2007)*, pages 11–20, 2007.

[Bra95] Stefan Brands. Restrictive blinding of secret-key certificates. In *Proceedings of EUROCRYPT 1995*, Saint-Malo, France, May 1995.

[BRT11] Mikhail Bilenko, Matthew Richardson, and Janice Y. Tsai. Targeted, not tracked: Client-side solutions for privacy-friendly behavioral advertising. In *4th Hot Topics in Privacy Enhancing Technologies (HotPETs 2011)*, pages 29–40, 2011.

[BS03] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. In *SCN'02: Proceedings of the 3rd International Conference on Security in Communication Networks*, pages 326–341, 2003.

[BS07] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. *J. Cryptol.*, 20(3):295–321, 2007.

[BSNS06] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. On the integration of public key data encryption and public key encryption with keyword search. In *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, Lecture Notes in Computer Science, pages 217–232, 2006.

[BSW09] John Bethencourt, Dawn Song, and Brent Waters. New techniques for private stream searching. *ACM Trans. Inf. Syst. Secur.*, 12(3):1–32, 2009.

[BWS05] Gregory Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti. Using parse tree validation to prevent SQL injection attacks. In *SEM*, pages 106–113, 2005.

[BZ07] Fabiano C. Botelho and Nivio Ziviani. External perfect hashing for very large key sets. In *CIKM'07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 653–662, New York, NY, USA, 2007.

[CAMG08] Carlos Carlos Aguilar-Melchor and Philippe Gaborit. A fast private information retrieval protocol. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 1848 –1852, 6-11 2008.

[CDN09] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Oblivious transfer with access control. In *Proceedings of ACM CCS 2009*, Chicago, IL, November 2009.

[CDN10] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Unlinkable priced oblivious transfer with rechargeable wallets. In *Proceedings of FC 2010*, Tenerife, Canary Islands, January 2010.

[CDNZ11] Jan Camenisch, Maria Dubovitskaya, Gregory Neven, and Gregory M. Zaverucha. Oblivious transfer with hidden access control lists. In *Proceedings of PKC 2011*, Taormina, Italy, March 2011.

[CG97] Benny Chor and Niv 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.

[CGH09] Scott E. Coull, Matthew Green, and Susan Hohenberger. Controlling access to an oblivious database using stateful anonymous credentials. In *Proceedings of PKC 2009*, Irvine, CA, March 2009.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS'95: Proceedings of the 36th Annual Symposium on the Foundations of Computer Science*, pages 41–50, Oct 1995.

[CGN97] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Technical Report TR CS0917, Department of Computer Science, Technion, Israel, 1997.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[CLM07] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *SP'07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 101–115, Washington, DC, USA, 2007.

[CLT11] Emiliano De Cristofaro, Yanbin Lu, and Gene Tsudik. Efficient techniques for privacy-preserving sharing of sensitive information. In Jonathan M. McCune, Boris Balacheff, Adrian Perrig, Ahmad-Reza Sadeghi, Angela Sasse, and Yolanta Beres, editors, *TRUST*, volume 6740 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 2011.

[CML06] Chi-Yin Chow, Mohamed F. Mokbel, and Xuan Liu. A peer-to-peer spatial cloaking algorithm for anonymous location-based service. In *Proceedings of the 14th Annual ACM international Symposium on Advances in Geographic information Systems*, pages 171–178, New York, NY, USA, 2006.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT'99: Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, pages 402–414, 1999.

[Cre06] Giovanni Di Crescenzo. Towards practical private information retrieval. Achieving Practical Private Information Retrieval Panel, SecureComm 2006, Aug. 2006.

[Cri11] Emiliano De Cristofaro. *Privacy-Preserving Sharing of Sensitive Information (Working Title)*. PhD thesis, University of California, Irvine, California, USA, August 2011.

[CS10] Yao Chen and Radu Sion. On securing untrusted clouds with cryptography. In *WPES '10*, pages 109–114, 2010.

[CZ09] Jan Camenisch and Gregory M. Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security: 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, pages 108–127, 2009.

[DC06] George Danezis and Richard Clayton. Route fingerprinting in anonymous communications. In A. Montresor, A. Wierzbicki, and N. Shahmehri, editors, *International Conference on Peer-to-Peer Computing (P2P 2006)*, pages 69–72, 2006.

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *SP'03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, Oakland, CA, May 2003.

[Dep09] Department of Computer Science at Duke University. The TPIE (templated portable I/O environment), Accessed July 2009. `http://madalgo.au.dk/Trac-tpie/`.

[DFBAWM09] Josep Domingo-Ferrer, Maria Bras-Amorós, Qianhong Wu, and Jesús Manjón. User-private information retrieval based on a peer-to-peer community. *Data Knowl. Eng.*, 68:1237–1252, November 2009.

[DM06] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In *5th Workshop on the Economics of Information Security (WEIS 2006)*, 2006.

[DMS04a] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *SEC'04: Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, August 2004.

[DMS04b] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 21–21, 2004.

[DS08] George Danezis and Paul F. Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In N. Borisov and I. Goldberg, editors, *8th Privacy Enhancing Technologies Symposium (PETS 2008)*, volume 5134 of *Lecture Notes in Computer Science*, pages 151–166, 2008.

[Efr09] Klim Efremenko. 3-query locally decodable codes of subexponential length. In *STOC'09: Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 39–44, New York, NY, USA, 2009.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324, 2005.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19, 2004.

[Fre11] Free Software Foundation. The GNU multiple precision (GMP) arithmetic library, Accessed March 2011. Version 5.0.1.

[Gas04] William I. Gasarch. A survey on private information retrieval (column: Computational complexity). *Bulletin of the EATCS*, 82:72–107, 2004.

[GCF11] Saikat Guha, Bin Cheng, and Paul Francis. Privad: Practical privacy in online advertising. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 13–13, 2011.

[Gei10] Martin Geisler. *Cryptographic Protocols: Theory and Implementation*. PhD thesis, Aarhus University, Aarhus, Denmark, 2010.

[GF10] Saikat Guha and Paul Francis. Privacy Analysis of the Privad Privacy-preserving Advertising System. Technical report, MPI-SWS-2010-002, Max Planck Institute for Software Systems, 2010. `http://adresearch.mpi-sws.org/privad-privacy.pdf`.

[GG03] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 31–42, New York, NY, USA, 2003.

[GGM98] Yael Gertner, Shafi Goldwasser, and Tal Malkin. A random server model for private information retrieval or how to achieve information theoretic PIR avoiding database replication. In *RANDOM'98: Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 200–217, London, UK, 1998.

[Ghi08] Gabriel 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.

[GIKM98a] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *STOC'98: Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 151–160, New York, NY, USA, 1998.

[GIKM98b] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of STOC 1998*, Dallas, TX, May 1998.

[GKK+08] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. Private queries in location based services: Anonymizers are not necessary. In *SIGMOD'08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 121–132, 2008.

[Gol07a] Ian Goldberg. Improving the robustness of private information retrieval. In *IEEE Symposium on Security and Privacy*, pages 131–148, 2007.

[Gol07b] Ian Goldberg. Percy++ project on SourceForge, June 2007. `http://percy.sourceforge.net/`.

[Goo10] Google. About google patent search, Accessed November 2010. `http://www.google.com/googlepatents/about.html`.

[GPG09] GPGPU Team. High-speed pir computation on GPU on Assembla, Accessed June 2009. `http://www.assembla.com/spaces/pir_gpgpu/`.

[GPS09a] GPSmagazine. Garmin nüvi 780 GPS review, Accessed April 2009. `http://www.gpsmagazine.com/2008/04/garmin_nuvi_780_gps_review.php`.

[GPS09b] GPSreview.net. POI – points of interest, Accessed April 2009. http://www.gpsreview.net/pois/.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *ICALP'05: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, pages 803–815, 2005.

[GT08] Vandana Gunupudi and Stephen R. Tate. Generalized non-interactive oblivious transfer using count-limited objects with applications to secure mobile agents. In *FC'08: Twelfth International Conference on Financial Cryptography and Data Security*, Cozumel, Mexico, 2008.

[Hen07] Urs Hengartner. Hiding location information from location-based services. In *Mobile Data Management, 2007 International Conference on*, pages 268–272, May 2007.

[Hen08] Urs Hengartner. Location privacy based on trusted computing and secure logging. In *SecureComm '08: Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pages 1–8, New York, NY, USA, 2008.

[HG05] Katherine A. Heller and Zoubin Ghahramani. Bayesian hierarchical clustering. In *ICML '05*, pages 297–304, 2005.

[HILM02] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *ACM SIGMOD*, pages 216–227, 2002.

[HMT04] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *VLDB*, pages 720–731, 2004.

[HN09] Daniel C. Howe and Helen Nissenbaum. TrackMeNot: Resisting surveillance in web search. In Ian Kerr, Valerie Steeves, and Carole Lucock, editors, *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, chapter 23, pages 417–436. Oxford University Press, 2009.

[HOG11] Ryan Henry, Femi Olumofin, and Ian Goldberg. Practical PIR for electronic commerce. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, Chicago, IL, October 2011.

[HP05a] Robert J. Hansen and Meredith L. Patterson. Guns and butter: Towards formal axioms of input validation. In *Black Hat USA 2005*, Las Vegas, July 2005.

[HP05b] Robert J. Hansen and Meredith L. Patterson. Stopping injection attacks with computational theory. In *Black Hat USA 2005*, Las Vegas, July 2005.

[ICA08] ICANN Security and Stability Advisory Committee (SSAC). Report on domain name front running, February 2008.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 262–271, 2004.

[Inf09] InformIT. Introduction to DSL, Accessed July 2009. `http://www.informit.com/articles/article.aspx?p=31699&seqNum=3`.

[Int09] Intel. Intel and ethernet, Accessed July 2009. `http://www.intel.com/standards/case/case_ethernet.htm`.

[IS04] Alexander Iliev and Sean Smith. Private information storage with logarithmic-space secure hardware. In *In I-NetSec 04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*, pages 201–216, 2004.

[IS05] Alexander Iliev and Sean W. Smith. Protecting client privacy with trusted computing at the server. *IEEE Security and Privacy*, 3(2):20–28, 2005.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC'09: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, pages 577–594, Berlin, Heidelberg, 2009.

[Jue01] Ari Juels. Targeted advertising ... and privacy too. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 408–424, London, UK, 2001.

[Kal03] Burt Kaliski. TWIRL and RSA key size, 2003. `http://www.rsa.com/rsalabs/node.asp?id=2004`.

[Kal09] Marcin Kalicinski. RapidXml. http://rapidxml.sourceforge.net/, Accessed November 2009.

[Kat11] Aniket Kate. PBCWrapper: C++ wrapper classes for the pairing-based cryptography library, Accessed March 2011. Version 0.8.0.

[KK00] Melita Kennedy and Steve Kopp. *Understanding Map Projections*. ESRI (Environmental Systems Research Institute) press, 2000.

[KO97] Eyal Kushilevitz and Rafail 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.

[Kob01] Alfred Kobsa. Tailoring privacy to users' needs. In *UM '01*, pages 303–313, 2001.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194, 2010.

[LBCP08] Dan Lin, Elisa Bertino, Reynold Cheng, and Sunil Prabhakar. Position transformation: a location privacy protection method for moving objects. In *SPRINGL '08: Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS*, pages 62–71, New York, NY, USA, 2008.

[Lip88] Richard P. Lippmann. An introduction to computing with neural nets. In *Artificial neural networks: theoretical concepts*, pages 36–54, Los Alamitos, CA, USA, 1988.

[Lip05] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328, 2005.

[LO11] Steve Lu and Rafail Ostrovsky. Multi-server oblivious ram. Cryptology ePrint Archive, Report 2011/384, 2011. `http://eprint.iacr.org/`.

[Loe09] Karsten Loesing. Measuring the Tor Network: Evaluation of Client Requests to the Directories. Technical report, The Tor project, 2009.

[Lyn11] Ben Lynn. PBC library: The pairing-based cryptography library, Accessed March 2011. Version 0.5.11.

[MB08] Prateek Mittal and Nikita Borisov. Information leaks in structured peer-to-peer anonymous communication systems. In P. F. Syverson and S. Jha, editors, *ACM Conference on Computer and Communications Security (CCS 2008)*, pages 267–278, 2008.

[MB09] Prateek Mittal and Nikita Borisov. ShadowWalker: peer-to-peer anonymous communication using redundant structured topologies. In S. Jha and A. D. Keromytis, editors, *ACM Conference on Computer and Communications Security (CCS 2009)*, pages 161–172, 2009.

[MBTR10] Prateek Mittal, Nikita Borisov, Carmela Troncoso, and Alfredo Rial. Scalable anonymous communication with provable security. In *5th USENIX conference on Hot topics in security (HotSec'10)*, pages 1–16, 2010.

[MCA06] Mohamed F. Mokbel, Chi-Yin Chow, and Walid G. Aref. The new Casper: query processing for location services without compromising privacy. In

VLDB '06: Proceedings of the 32nd international conference on Very large data bases, pages 763–774, 2006.

[McC10] Declan McCullagh. Amazon Fights Demand for Customer Records, April 2010. `http://news.cnet.com/8301-13578_3-20002870-38.html`.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8), 1965.

[MOT+11] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In *Proceedings of the 20th USENIX Security Symposium*, San Diego, CA, August 2011.

[MS00] Sanjeev Kumar Mishra and Palash Sarkar. Symmetrically private information retrieval. In *INDOCRYPT'00: Proceedings of the First International Conference on Progress in Cryptology*, pages 225–236, London, UK, 2000.

[MTHK09] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with Torsk. In Somesh Jha and Angelos D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 590–599, 2009.

[Nat07] National Institute of Standards and Technology. Key management guideline, 2007. `http://csrc.nist.gov/groups/ST/toolkit/index.html`.

[New04] New European Schemes for Signatures, Integrity, and Encryption. Final report of European Project IST-1999-12324, 2004. `https://www.cosic.esat.kuleuven.be/nessie/`.

[Nie88] Jakob Nielsen. Nielsen's law of Internet bandwidth, April 1988. `http://www.useit.com/alertbox/980405.html`.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *ACM Symposium on Theory of Computing*, pages 245–254, 1999.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA'01: Proceedings of the 12th annual ACM-SIAM Symposium On Discrete Algorithms*, pages 448–457, Philadelphia, PA, USA, 2001.

[NW06] Arjun Nambiar and Matthew Wright. Salsa: a structured approach to large-scale anonymity. In R. N. Wright and S. De Capitani di Vimercati, editors, *13th ACM Conference on Computer and Communications Security (CCS 2006)*, pages 17–26, 2006.

[OG10a] Femi Olumofin and Ian Goldberg. Preserving access privacy over large databases. Technical report, CACR 2010-33, University of Waterloo, 2010. `http://www.cacr.math.uwaterloo.ca/techreports/2010/cacr2010-33.pdf`.

[OG10b] Femi Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *PETS'10: Proceedings of the 10th Privacy Enhancing Technologies Symposium*, Berlin, 2010.

[OG11] Femi Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *FC'11: Fifteenth International Conference on Financial Cryptography and Data Security*, St. Lucia, 2011.

[Ook10] Ookla. Netindex.com source data (Canada and US), Accessed July 2010. `http://www.netindex.com/source-data/`.

[OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC'97: Proceedings of the twenty-ninth annual ACM Symposium on Theory of Computing*, pages 294–303, New York, NY, USA, 1997.

[ØS06] Lasse Øverlier and Paul Syverson. Locating hidden servers. In *IEEE Symposium on Security and Privacy (S&P 2006)*, 2006.

[OS07a] Rafail Ostrovsky and William E. Skeith, III. Private searching on streaming data. *J. Cryptol.*, 20(4):397–430, 2007.

[OS07b] Rafail Ostrovsky and William E. Skeith, III. A survey of single-database private information retrieval: Techniques and applications. In *PKC'07: Proceedings of the 10th international conference on Practice and theory in public-key cryptography*, pages 393–411, 2007.

[OTGH10] Femi Olumofin, Piotr K. Tysowski, Ian Goldberg, and Urs Hengartner. Achieving efficient query privacy for location based services. In *PETS'10: Proceedings of the 10th Privacy Enhancing Technologies Symposium*, Berlin, 2010.

[Pro11] OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS, Accessed February 2011. Version 1.0.0.

[PRR09] Andriy Panchenko, Stefan Richter, and Arne Rache. NISAN: network information service for anonymization networks. In S. Jha and A. D. Keromytis, editors, *ACM Conference on Computer and Communications Security (CCS 2009)*, pages 141–150, 2009.

[PS10] Sai Teja Peddinti and Nitesh Saxena. On the privacy of web search based on query obfuscation: a case study of TrackMeNot. In *Proceedings of the 10th Privacy Enhancing Technologies Symposium*, PETS'10, pages 19–37, 2010.

[PYZ+09] A. Pingley, W. Yu, N. Zhang, X. Fu, and W. Zhao. CAP: A context-aware privacy protection system for location-based services. In *29th IEEE International Conference on Distributed Computing Systems*, Jun 2009.

[Ray00] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 10–29, 2000.

[RP02] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-peer based anonymous Internet usage with collusion detection. In Sushil Jajodia and Pierangela Samarati, editors, *ACM Workshop on Privacy in the Electronic Society (WPES 2002)*, pages 91–102, 2002.

[RPB08] Daniele Riboni, Linda Pareschi, and Claudio Bettini. Privacy in georeferenced context-aware services: A survey. In Bettini et al. [BJSW08].

[RPG07] Joel Reardon, Jeffrey Pound, and Ian Goldberg. Relational-complete private information retrieval. Technical report, Technical Report CACR 2007-34, University of Waterloo, 2007.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24:513–523, August 1988.

[SB08] Robin Snader and Nikita Borisov. A tune-up for Tor: Improving security and performance in the Tor network. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2008)*, 2008.

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

[SC07] Radu Sion and Bogdan Carbunar. On the computational practicality of private information retrieval. In *Network and Distributed Systems Security Symposium*, 2007.

[Sch00] Douglas C. Schmidt. *More C++ gems*, chapter GPERF: a perfect hash function generator, pages 461–491. Cambridge University Press, New York, NY, USA, 2000.

[Sch11] Thomas Schneider. *Engineering Secure Two-Party Computation Protocols: Advances in Design, Optimization, and Applications of Efficient Secure Function Evaluation*. PhD thesis, Ruhr-University Bochum, Bochum, Germany, February 2011.

[SCM05] Len Sassaman, Bram Cohen, and Nick Mathewson. The Pynchon Gate: a secure method of pseudonymous mail retrieval. In *WPES'05: Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 1–9, New York, NY, USA, 2005.

[SDFMB08] Agusti Solanas, Josep Domingo-Ferrer, and Antoni Martínez-Ballesté. Location privacy in location-based services: Beyond TTP-based schemes. In Bettini et al. [BJSW08].

[SDH+10] Max Schuchard, Alexander W. Dean, Victor Heorhiadi, Nicholas Hopper, and Yongdae Kim. Balancing the shadows. In K. Frikken, editor, *9th ACM workshop on Privacy in the electronic society (WPES 2010)*, pages 1–10, 2010.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Sho11] Victor Shoup. NTL: A library for doing number theory, Accessed March 2011. Version 5.5.2.

[SJ88] Karen Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval*, pages 132–142. Taylor Graham Publishing, London, UK, UK, 1988.

[SJ05] Felipe Saint-Jean. Java implementation of a single-database computationally symmetric private information retrieval (cSPIR) protocol. Technical Report YALEU/DCS/TR-1333A, Yale University, New Haven, CT, USA, 2005.

[SKS05] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, Inc., New York, NY, USA, 5th edition, 2005.

[SL91] Seyed Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):660 –674, May 1991.

[SM86] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[Sny93] John P. Snyder. *Flattening the Earth, two thousand years of map projections*. University of Chicago Press, 1993.

[Sol01] Daniel J. Solove. Privacy and power: Computer databases and metaphors for information privacy. *Stanford Law Review*, 53(6):1393–1462, 2001.

[Spu00] Charles E. Spurgeon. *Ethernet: The Definitive Guide*. O'Reilly & Associates, Inc., USA, 2000.

[SS01] Sean William Smith and David Safford. Practical server privacy with secure coprocessors. *IBM Syst. J.*, 40(3):683–695, 2001.

[Ste98] Julien P. Stern. A new efficient all-or-nothing disclosure of secrets protocol. In *ASIACRYPT'98: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 357–371, London, UK, 1998.

[Sun09] Sun Microsystems. MySQL, Accessed July 2009. http://www.mysql.com/.

[SWP00] 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.

[Tan02] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 2002.

[TB06] Parisa Tabriz and Nikita Borisov. Breaking the collusion detection mechanism of MorphMix. In George Danezis and Phillipe Golle, editors, *Privacy Enhancing Technologies (PETS 2006)*, volume 4258 of *Lecture Notes in Computer Science*, pages 368–383, 2006.

[THK09] Andrew Tran, Nicholas Hopper, and Yongdae Kim. Hashing it out in public: common failure modes of DHT-based anonymity schemes. In S. Paraboschi, editor, *8th ACM Workshop on Privacy in the Electronic Society (WPES 2009)*, pages 71–80, 2009.

[TNB+10] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. Adnostic: Privacy preserving targeted advertising. In *Network and Distributed Systems Security Symposium*, 2010.

[Tor11a] The Tor Project. Tor metrics portal, June 2011. http://metrics.torproject.org/.

[Tor11b] The Tor Project. Who uses Tor. `http://www.torproject.org/about/torusers.html.en`, Accessed June 2011.

[TP11] Jonathan Trostle and Andy Parrish. Efficient computationally private information retrieval from anonymity or trapdoor groups. In *Proceedings of the 13th international conference on Information security*, ISC'10, pages 114–128, 2011.

[Tra09] Transaction Processing Performance Council. TPC benchmark C, Accessed July 2009. `http://www.tpc.org/`.

[Tru10] Trusted Computing Group. Trusted platform module. `http://www.trustedcomputinggroup.org/developers/trusted_platform_module`, Accessed July 2010.

[TV09] Stephen R. Tate and Roopa Vishwanathan. Improving cut-and-choose in verifiable encryption and fair exchange protocols using trusted computing technology. In *Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security XXIII*, pages 252–267, 2009.

[VV95] Darren Erik Vengroff and Jeffrey Scott Vitter. Supporting I/O-efficient scientific computation in TPIE. In *SPDP'95: Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, page 74, Washington, DC, USA, 1995.

[WAEA10] Shiyuan Wang, Divyakant Agrawal, and Amr El Abbadi. Generalizing PIR for practical private retrieval of public data. In *DBSec'10*, pages 1–16, Rome, Italy, 2010.

[WDDB06] Shuhong Wang, Xuhua Ding, Robert H. Deng, and Feng Bao. Private information retrieval using trusted hardware. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 49–64, 2006.

[Wei02] Alan R. Weiss. Dhrystone benchmark: History, analysis, scores and recommendations. *EEMBC White Paper*, 2002.

[Wie06] Christian Wieschebrink. Two NP-complete problems in coding theory with an application in code based cryptography. In *2006 IEEE International Symposium on Information Theory*, pages 1733–1737, July 2006.

[WMB10] Qiyan Wang, Prateek Mittal, and Nikita Borisov. In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. In Angelos D. Keromytis and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security (CCS 2010)*, pages 308–318, 2010.

[WS08] Peter Williams and Radu Sion. Usable PIR. In *NDSS'08: Proceedings of the 16th Annual Network and Distributed System Security Symposium*, 2008.

[WT09] Mark Wong and Craig Thomas. Database test suite project on SourceForge, Accessed July 2009. `http://osdldbt.sourceforge.net/`.

[WWP10] Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. Secure coprocessor-based private information retrieval without periodical preprocessing. In *AISC '10*, pages 5–11, 2010.

[XC07] Toby Xu and Ying Cai. Location anonymity in continuous location-based services. In *Proceedings of the 15th Annual ACM international Symposium on Advances in Geographic information Systems*, pages 1–8, New York, NY, USA, 2007.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, FOCS '82, pages 160–164, 1982.

[YCSI08] Rei Yoshida, Yang Cui, Rie Shigetomi, and Hideki Imai. The practicality of the keyword search using PIR. In *ISITA'08: Proceedings of the 2008 International Symposium on Information Theory and its Applications*, pages 1–6, December 2008.

[Yek07] Sergey Yekhanin. *Locally decodable codes and private information retrieval schemes*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007.

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1–16, 2008.

[Yek10] Sergey Yekhanin. Private information retrieval. *Commun. ACM*, 53(4):68–73, 2010.

[Yek11] Sergey Yekhanin. Locally decodable codes: a brief survey. In *Proceedings of the 3rd International Workshop on Coding and Cryptography (IWCC), 2011*, Qingdao, China, May/June 2011.

[YJHL08] Man Lung Yiu, C.S. Jensen, Xuegang Huang, and Hua Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 366–375, April 2008.

[YY08] Stephen S. Yau and Yin Yin. Controlled privacy preserving keyword search. In *ASIACCS'08: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, pages 321–324, New York, NY, USA, 2008.

[ZAW+10] Ye Zhang, Man Ho Au, Duncan S. Wong, Qiong Huang, Nikos Mamoulis, David W. Cheung, and Siu-Ming Yiu. Oblivious transfer with access control : Realizing disjunction without duplication. In *Proceedings of Pairing 2010*, Yamanaka Hot Spring, Japan, December 2010.

[ZH09] Ge Zhong and Urs Hengartner. A distributed k-anonymity protocol for location privacy. In *Proceedings of Seventh IEEE International Conference on Pervasive Computing and Communication (PerCom 2009). Galveston, TX*, pages 253–262, 2009.

APPENDICES

Appendix A

Database Schema and Queries for our SQLPIR Contribution

This appendix lists the database schemas and the queries relating to the examples and experiments in Chapter 5. Listing 6 shows the database schema for the examples, Listing 7 shows the schema for the microbenchmarks and experiments, Listing 8 shows the microbenchmark queries from Reardon et al. [RPG07], and Listing 9 shows the experimental queries with complex conditions.

Listing 6 Database schema for examples.

```
CREATE TABLE registrar (
  reg_id int(11) NOT NULL,
  contact char(60) default NULL,
  phone char(80) default NULL,
  address char(80) default NULL,
  email char(60) default NULL,
  PRIMARY KEY (reg_id));

CREATE TABLE regdomains (
  id int(11) NOT NULL,
  domain char(80) default NULL,
  created int(8) default NULL,
  expiry int(8) default NULL,
  reg_id int(11) NOT NULL,
  status varchar(2) default NULL,
  PRIMARY KEY (id));
```

Listing 7 Database schema for microbenchmarks and experiments.

```
CREATE TABLE contact (
  contact_id int(11) NOT NULL,
  name char(60) default NULL,
  address char(80) default NULL,
  email char(60) default NULL,
  PRIMARY KEY  (contact_id));

CREATE TABLE registration (
  reg_id int(11) NOT NULL,
  domain char(80) default NULL,
  expiry_date int(8) default NULL,
  reg_date int(8) default NULL,
  registrant int(11) default NULL,
  registrar int(11) default NULL,
  status varchar(2) default NULL,
  PRIMARY KEY  (reg_id));

ADD FOREIGN KEY fk_registrant (registrant) REFERENCES contact(contact_id);
ADD FOREIGN KEY fk_registrar (registrar) REFERENCES contact(contact_id);
```

Listing 8 Microbenchmark queries from Reardon et al. [RPG07].

Q1 – Point query with single result

```
SELECT domain, reg_date
FROM registration
WHERE domain = ?
```

Q2 – Point query with multiple results

```
SELECT domain
FROM registration
WHERE expiry_date = ?
```

Q3 – Range query with single condition

```
SELECT domain, status
FROM registration
WHERE expiry_date > ?
```

Q4 – Range query with multiple conditions

```
SELECT *
FROM registration
WHERE expiry_date > ? AND
      reg_date < ?
```

Q5 – Point query with join

```
SELECT domain, name, email
FROM contact, registration
WHERE domain = ? AND
      registrant = contact_id
```

Q6 – Range query with join

```
SELECT *
FROM contact, registration
WHERE expiry_date > ? AND
      registrar=contact_id
```

Listing 9 Experimental SQL queries CQ1–CQ5. "#" indicates a private constant or condition. Each private constant or condition features a sensitive constant.

CQ1 – Private point query on a range with sensitive domain name information (private constant).

```
SELECT domain, name, address, email, reg_date, expiry_date
FROM registration, contact
WHERE (contact_id = registrant) AND
      (reg_date > 20090501) AND
      (domain = #'somedomain.org')
```

CQ2 – Private range query with sensitive registrar ID range (private condition).

```
SELECT domain, name, address, email, expiry_date
FROM registration, contact
WHERE (contact_id = registrant) AND
      (status IN (1,4,5,7,9)) AND
      (registrar #BETWEEN 198542 AND 749999) AND
      (expiry_date BETWEEN 20090101 AND 20091031)
```

CQ3 – Private aggregate point query with sensitive registrar ID value (private constant).

```
SELECT registrar, count(domain)
FROM registration, contact
WHERE (contact_id = registrar) AND
      (registrar = #635393)
GROUP BY registrar
HAVING count(domain) > 0
ORDER BY registrar ASC
```

CQ4 – Non-private LIKE query revealing only the prefix of a domain name (private condition).

```
SELECT domain, name, address, email, reg_date, expiry_date
FROM registration, contact
WHERE (contact_id = registrant) AND
      (domain LIKE 'some%') AND
      (domain = #'somedomain.com')
```

CQ5 – Private LIKE query with domain name prefix as wildcard (private condition).

```
SELECT domain, name, address, email, reg_date, expiry_date
FROM registration, contact
WHERE (contact_id = registrant) AND
      (domain #LIKE 'some%')
```

Appendix B

Client Calibration, Map Tracking, and User Privacy Preferences for our TOPIR Contribution

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 is being 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).

Tracking Buckets and Portions Map

The client needs to maintain maps of portions and buckets for all her queries. A map describes either the buckets or the identifying information submitted to the database for defining the portion for answering a query. The main idea is that the client will reuse the same map for repeated queries. If the client does not reuse existing maps, intersection attacks by the adversary can help isolate the bucket of interest or the data sub-portion of

Table B.1: Modeling privacy preferences for SQL-based queries.

```
[1] subset      := bucket    | ; Use static buckets only
                   portion   | ; Use dynamic portions only
                   any         ; Use either

[2] disclosures := query     | ; Entire query (no access privacy)
                   substring | ; All but the substrings of constants
                   constant  | ; All but constants in predicates
                   predicate | ; All but predicates having constants
                   none        ; Hide query shape, predicates and constants

[3] attributes  := title     | ; Allow disclosures of patent titles
                   abstract    ; Allow disclosures of abstracts

[4] mindegree   := real_number ; Minimum degree of access privacy
```

interest, thereby decreasing the level of access privacy. Whenever a map is used for the first time, it is appended to a list of maps and its popularity or frequency of use is set to 1. Subsequently, the client retrieves and reuses maps from the list if a query can be answered using an existing map; otherwise it creates a new map. The client also updates the popularity of a reused map. In an honest-but-curious adversarial model, the database can be relied upon to publish maps of portions and buckets queried by all users. A user can reuse maps predefined in the database in that case. However, if the database is not relied upon to publish correct maps, a user can share maps with other users by periodically uploading their map data to a third-party system through an anonymous channel, and the system will use the information to update the popularity of the maps.

An Example User Privacy Preferences Modeling

An example of modeling user privacy preferences for an SQL query is shown in Table B.1. The `attributes` element in the definition specifies the list of attribute names containing data that may be disclosed to define a database portion. The client software should not disclose values for attributes names not listed (e.g., patent assignees, inventing parties, date of issue, etc.).

Appendix C

Level of Privacy Algorithm for our LBSPIR Contribution

Algorithms

In this appendix, we present the algorithms that implement our proposal to allow a user to set his or her level of query privacy (equivalent to the size of the cloaking region) in the PIR query. Let $PIR = \{PIREncode, PIRProcess, PIRDecode\}$ be some PIR protocol where $PIREncode$, $PIRProcess$, and $PIRDecode$ are the query encoding, response processing, and response decoding protocols, respectively.

Query Generation (by PIR client)

Let n be the total number of items (or POIs) in the PIR database or databases (in the case of a PIR protocol with replicated databases), σ be the number of VHC grid cells in the map where each grid cell has n/σ items. Let i be the index of the database block that the user is after, and $\rho \in [0, 1]$ be the level of query privacy preset by the user. In models where users pay for computational resources used by PIR servers, this privacy level must have been negotiated by the user and LBS provider. The user selects this level by using an application interface on the smartphone. The level is specified in terms of cities/towns (city level), state/provinces (provincial level), and so on.

i. Compute $w = \lceil \rho n \rceil$ and set $R = \{r_1, r_2, r_3, \ldots, r_w\}$ to be the set of indices for the items corresponding to the cloaking region.

ii. Compute $(q, \tau) = PIREncode_R(i)$ as the PIR query encoding for i, using only the item indices in R (i.e., not all the indices in all geographical grid locations in the entire map are required). Here, q is the query to be sent to the database server, and τ is some state information that will be used to decode the server's response.

iii. Send $\{q, R\}$ to the database (or PIR server). Instead of sending R, it may be more efficient to send only the top left and bottom right coordinates of the bounding rectangle that covers the cloaking region, or the range of identifiers (or numbers) for the VHC grid cells that are within the cloaking region, or for the geographic cells that contain them. In any of these cases, the PIR server can use the provided information to determine R.

Response Encoding (by PIR server)

i. Retrieve a database portion $D = \{d_1, d_2, d_3, \ldots, d_w\}$, where each database item in D correponds to an index in R. Each item is a POI data structure with attributes longitude, latitude, name, address, phone, category, website address, and so on.

ii. Execute $PIRProcess_D(q)$ to obtain response r, which should be the block of POIs in the user's VHC grid cell.

iii. Return r back to the client.

Response Decoding (by PIR client)

i Execute $PIRDecode_R(\tau, r)$ to obtain a database response to the query. The response should be the set of POIs that the query requested.

ii The client can locally compute the nearest neighbour using the set of POIs that was returned.