

Revisiting the Computational Practicality of Private Information Retrieval^{*}

Femi Olumofin and Ian Goldberg

Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada N2L 3G1
{fgolumof,iang}@cs.uwaterloo.ca

Abstract. The retrieval of information from a remote database server typically demands providing the server with some search terms to assist with the retrieval task. However, keeping the search terms private without undermining the server’s ability to retrieve the information is desirable for many privacy-preserving systems. Private information retrieval (PIR) provides a cryptographic means for retrieving data from a database without the database or database administrator learning any information about which particular item was retrieved.

A study of the computational practicality of PIR by Sion and Carbunar in 2007 concluded that no existing construction is as efficient as *the trivial PIR scheme* — the server transferring its entire database to the client. While often cited as evidence that PIR is impractical, that paper did not examine multi-server information-theoretic PIR schemes, which are orders of magnitude more computationally efficient; further, a single-server lattice-based scheme by Aguilar-Melchor and Gaborit has recently been introduced, which is also much more computationally efficient than the schemes examined by Sion and Carbunar. In this paper, we report on a performance analysis of the above single-server lattice-based PIR scheme 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.

1 Introduction

The retrieval of information from a remote database server typically demands providing the server with clues in the form of data indices, search keywords, or structured queries to assist with the retrieval task. However, keeping retrieval clues private without undermining the server’s ability to retrieve the desired

^{*} This is an extended version of our Financial Cryptography 2011 paper [36].

information is a requirement that is common for user-centric privacy-preserving systems. Private information retrieval (PIR) provides a means of retrieval that guarantees access privacy, by preventing the database administrator from being able to learn any information about which particular item was retrieved. In other words, PIR hides the clues in the user’s query, without inhibiting retrieval functions.

Today’s most developed and deployed privacy-preserving techniques, such as onion routers and mix networks, are limited to anonymizing the identity of users. PIR, on the other hand, by protecting the *contents* of queries, can protect important application domains like patent databases, pharmaceutical databases, online censuses, real-time stock quotes, location-based services, online behavioral analysis for ad networks, and Internet domain registration [4, 20, 26, 38].

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. [13, 14] 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 [13, 14], 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 transferring $\frac{1}{3}(1+o(1)) \cdot \log_2^2 n \cdot \log_2 \log_2(2n)$ 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 [13, 14], while some others [4, 8, 19, 22] have addressed such problems as using amortization and preprocessing to reduce server-side computational overheads and improving query robustness, amongst others.

Chor and Gilboa [11] 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 [28] 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. These include schemes based on group-homomorphic encryption [43], length-flexible additive homomorphic public-key cryptosystems [29], Φ -Hiding Assumption (Φ HA) trapdoor predicates (the hardness of deciding if a small prime is a factor of the totient of a composite integer of unknown factorization) [10], and more recently, a lattice-based scheme by Aguilar-Melchor and Gaborit [2].

In 2007, Sion and Carbunar [40] 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 [31] and Nielsen’s Law [34] 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 [40] 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 [2, 45] costs much less than one modular multiplication per database bit. Besides, the projections from [40] 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 paper, we revisit the computational practicality of PIR in general by extending and clarifying the results in [40]. First, we provide a detailed performance analysis of a recent single-server PIR scheme by Aguilar-Melchor and Gaborit [1–3], 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 [47]. They proposed and implemented the protocols, but their analysis was limited to server-side computations by the PIR server [1] 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 [40], 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 [2] 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. [13] and by Goldberg [22], which is novel to this paper. 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.

1.1 Preliminaries

We begin by outlining a few building blocks, some of which are based on [40]. 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 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 scheme in [2] 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: average home-user last-mile connection [24], Ethernet LAN, and commercial high-end inter-site connections [25, 42, 44]. Table 1 shows various network connection speeds (Mbps) since 1995, when PIR was introduced. The values up until 2006 are reused from [40], while we provided the subsequent values based on the capacity of today’s network bandwidths.¹

Modular multiplication. The work in [40] 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 [28]. Such CPUs have long been retired by Intel and are no longer representative of today’s multi-core CPUs. In addition, the Dhrystone benchmark, which found widespread usage at the time it was introduced in

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 1. Bandwidth estimates (in Mbps) for 1995 to 2010. We adapted values up to 2007 from [40] and those after 2007 are based on the Internet speed data for Canada and US from [37].

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

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 [46].

Instead, we measure the time directly. Using the key size schedule from RSA Labs [27], NIST [32], and NESSIE [33], the current recommended key size for the security of the Kushilevitz and Ostrovsky scheme is 1536 bits. 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 $t_{mul} = 3.08 \pm 0.08 \mu s$.

Projections. Moore’s Law [31] has an annual growth rate of 60%, which surpasses the 50% growth rate of Nielsen’s Law [34]. 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.

2 Related Work

A PIR scheme protects the access privacy of users to databases by providing a means to query the database without revealing any information about the item that was retrieved. Every PIR scheme consists of three basic algorithms: query generation, response encoding, and response decoding. Given an 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 , $j \in \{1, \dots, r\}$ with the query and returns an encoded response to the user. Finally, the user decodes the response using the response decoding algorithm. In the multi-server information-theoretic setting, we assume two or more non-colluding database replicas and the user must concurrently interact with all replicas in a manner similar to the above. Note that this describes the basic “block-retrieval” PIR scheme; other work has extended PIR to allow more expressive queries, including by keywords [12] and by SQL [35]. As block-retrieval PIR underlies the latter schemes as well, our analysis extends to those improved schemes.

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 [16]. In a trivial PIR scheme, the database sends all r blocks to the user, who can then retrieve the block of interest at index i (i.e., X_i). Hence, the trivial PIR scheme is private in the information-theoretic sense, but carries a very poor communication complexity, particularly for large databases used in the real world.

Standard PIR schemes preserve access privacy for user queries, but it is possible for a query to retrieve more data items than the user actually needs. An extension, known as *Symmetric PIR* [30] constrains the number of data items that may be retrieved per query, thereby preserving database privacy as well. We note that such a restriction is not possible with the the trivial PIR scheme.

The literature has mainly focused on improving the communication complexity of PIR schemes because communication between the user and the server(s) is considered to be the most expensive resource [7]. Despite achieving this goal, other barriers continue to limit realistic deployment of PIR schemes; the most limiting of these barriers is the computational requirement of PIR schemes. The performance measure of a scheme in terms of its computational complexity has only received attention much more recently. The first of these is the work by Beimel et al. [6, 7] which shows that standard PIR schemes cannot avoid a computation cost that is *linear* in the database size because each query for block X_i must necessarily process all database blocks X_j , $j \in \{1, \dots, r\}$. 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. [4, 5] and Smith et al. [41] similarly explore preprocessing for reducing server-side computation. However, the specialized hardware requirement at the server makes this solution less desirable.

In 2006, panelists from SECURECOMM [16] 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 [40] compares the bandwidth cost of trivial PIR to the computation and bandwidth cost of a single-server computational PIR scheme [28], which they considered to be the most efficient at that time. The motivation of [40] 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 [2, 3, 45] to argue otherwise by introducing more efficient computational PIR schemes that may be practical to deploy. Our work extends the work from [40] in important ways. First, their analysis was based on a number-theoretic computational PIR scheme [28], whereas we considered different varieties of computational PIR schemes: a number-theoretic scheme [28] and a

lattice-based linear algebra scheme [2]. 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 the most mitigating factor to the practicality of PIR schemes. Secondly, we extend the analysis of practicality to multi-server PIR schemes which has never been considered by any previous measurement study. 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 [35]. Even in domains where the database is not distributed, deployment is possible using servers containing random data [19], which eliminates the need for an organization to replicate its data to foreign servers.

Williams and Sion [48] describe 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 client storage, on the order of $\Theta(\sqrt{n})$, is available.

Aguilar-Melchor and Gaborit [2, 1] 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 [47]. Aguilar-Melchor et al. [1] 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. That work makes two main contributions. First, it shows that its scheme exhibits one order of magnitude speedup by using 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 for most realistic bandwidth situations; this attempts to dispel the conclusions by Sion and Carbunar [40] with respect to the practicality of single-server PIR schemes. However, the evaluation from Aguilar-Melchor et al. [1] consider 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 query generation and response decoding costs, as well as the query and response transfer cost. 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 [18, 29] 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 [18, 29], this scheme gives a much better server-side processing rate both on CPUs and GPUs.

Trostle and Parrish [45] 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 [18, 29], it is not quite as efficient as the scheme of Aguilar-Melchor and Gaborit [2, 1].

In the context of keyword search using PIR, Yoshida et al. [49] considered the practicality of a scheme proposed by Boneh et al. [9]. 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 Efficient Single-server PIR (LPIR-A)

We experimentally evaluated an implementation of the single-server PIR scheme by Aguilar-Melchor et al. [1]. 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.

3.1 Experiment

We obtained the source code [23] for this scheme, removed interactivity, changed the default parameters to one that guarantees security in a practical setting (complexity of over 2^{100} operations) [2], 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 [23] prevented us from testing larger databases for the selected security parameters. We did not fix the bug 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.

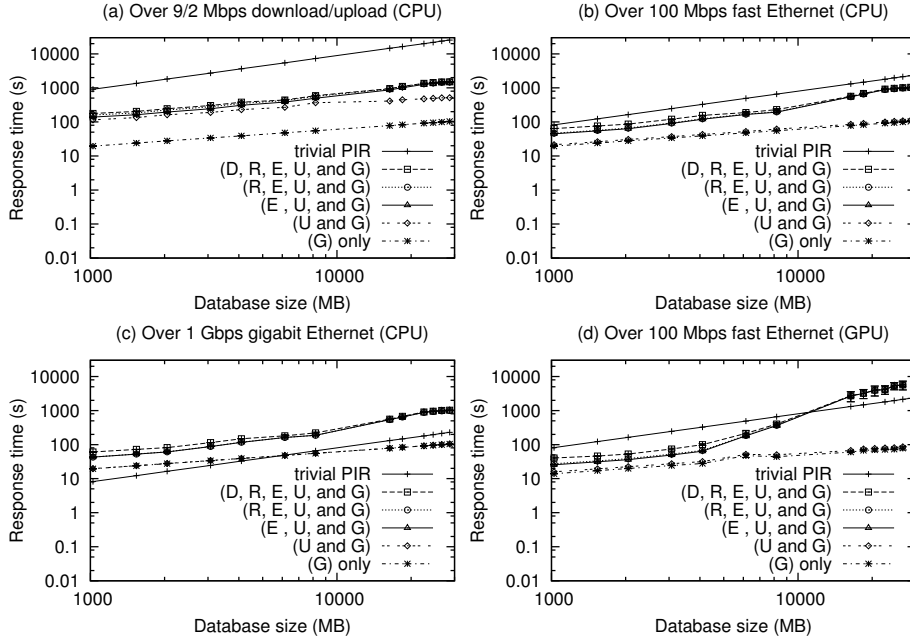


Fig. 1. Logarithmic scale plots for query generation (G), query upload(U), response encoding (E), response download (R), and response decoding (D) times for the Aguilar-Melchor et al. single-server PIR scheme [1] and the trivial PIR scheme in different bandwidth scenarios.

We ran the experiment on the server hardware described in Section 1.1. 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.

3.2 Result

Figure 1 shows the log-log plots of our results with breakdowns of the time for query generation and upload, response encoding and download, response decoding, as well as the trivial download time for the different sizes of databases we tested. Plots (a), (b), (c), and (d) respectively reflect bandwidth values typical of an Internet connection in the US and Canada, a 100 Mbps fast Ethernet, a 1 Gbps gigabit Ethernet, and a 100 Mbps fast Ethernet on 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 [37]. 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. For small databases, 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 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 [37], 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 [34]. We contrast these users from “low-end” users [34] that the above bandwidth averages from the Internet speed data [37] 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 in [1], but we used much larger databases.

4 Multi-server PIR

In this section, we provide detailed performance analyses of two multi-server information-theoretic PIR schemes, from Chor et al. [13] and from Goldberg [22]. We begin with an overview of these schemes and later show how they compare with the single-server scheme by Aguilar-Melchor et al. [1], and the trivial PIR scheme. The reason for choosing [13] is its simplicity, being the first PIR protocol invented. The reason for choosing [22] is its comprehensiveness and source code availability which allows for easy experimental analysis. The implementa-

tion of [22], known as Percy++ [21], is an open-source project on SourceForge. We also reproduce and reuse the results by Sion et al. [40] for the purpose of comparison with the two multi-server PIR schemes. Their result features the single-server PIR scheme by Kushilevitz and Ostrovsky [28], 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 [35], 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 [15] and Tor [17], also make the assumption that not all of the servers involved are colluding against the user.

4.1 First Scheme (MPIR-C)

We first describe the simple $O(\sqrt{n})$ protocol by Chor et al. The database D is treated as an $r \times b$ matrix of bits, where the k^{th} row of D is the k^{th} block of the database. Each of ℓ servers stores a copy of D . The client, interested in block i of the database, picks ℓ random bitstrings ρ_1, \dots, ρ_ℓ , each of length r , such that $\rho_1 \oplus \dots \oplus \rho_\ell = e_i$, 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 . 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. The client computes $R_1 \oplus \dots \oplus R_\ell = (\rho_1 \oplus \dots \oplus \rho_\ell) \cdot D = e_i \cdot D$, which is the i^{th} block of the database.

Sion and Carbunar [40] used a closed-form expression for the computation and communication cost of the PIR scheme in [28]. 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 closed-form 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.

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 the cost of accessing the contents of the database from memory or disk, 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

$$T_{M\text{PIR}-C} < (2(\ell - 1)\sqrt{n}/m + n/m) \cdot t_{\oplus} + 2\ell\sqrt{n} \cdot t_t + 2n/m \cdot t_{ac} + n \cdot t_{ov}$$

The most dominant term is $n \cdot (\frac{1}{m}t_{\oplus} + \frac{2}{m}t_{ac} + t_{ov})$, which will suffice for the entire expression when the value of n is large. Nevertheless, it is better to use the entire equation to determine the upper bound on execution time for more objective comparison purposes.

The work in [40] 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 1.1.

4.2 Second Scheme (MPIR-G)

We similarly derive an upper bound expression for the execution time for Goldberg’s multi-server PIR scheme, which 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 [39] 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 $r \times b$ matrix of w -bit words (i.e., elements of $GF(2^w)$), where again r is the number of blocks and b 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, in this paper we will only consider honest servers, which respond correctly. For head-to-head comparison with the Chor et al. protocol, we set the privacy level t (the number of servers which can collude without revealing the client’s query) to $\ell - 1$. As before, we choose $r = b$, but now $r = b = \sqrt{n/w}$. We also choose $w = 8$ to simplify the cost of computations; in $GF(2^8)$, additions are XOR operations on bytes and multiplications are lookup operations into a 64 KB table. These are the choices made by the open-source implementation of this protocol [21].

A client encodes a query for database block i by first uniformly choosing ℓ random distinct non-zero indices $\alpha_1, \dots, \alpha_{\ell}$ 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_k are random elements of $GF(2^8)$, while those for the constant terms should be 1 if $i = k$ 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, $\rho_j = [f_1(\alpha_j), \dots, f_r(\alpha_j)]$. (Note that each $f_k(\alpha_j)$ is an element of $GF(2^8)$ — a single byte.) 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 b elements of vector R_j 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 $GF(2^8)$. Using the protocol description in [22] and the source code [21], 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

$$T_{M\text{PIR-G}} < \left((\sqrt{n/8} + 3)\ell^2 - (\sqrt{n/8} - 3)\ell + n/8 \right) (t_{\oplus} + 3t_{ac}) + 2\ell\sqrt{8n} \cdot t_t + n \cdot t_{ov}$$

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

4.3 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++) [21] 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 server. 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 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 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

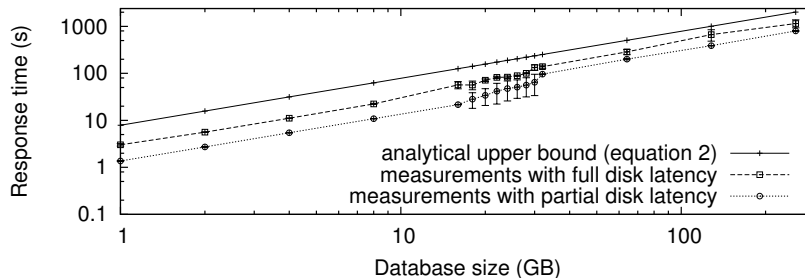


Fig. 2. Analytical and experimental measurements of the response time of Goldberg’s multi-server PIR scheme [22] (computations only). The upper line is derived from equation (2), but excluding time for communications. The middle line is the time for the first query, which includes startup overhead and 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.

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. [13] MPIR-C scheme using an implementation we developed. The implementation differed from [21] by doing XORs in 64-bit words, instead of by 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.

5 Comparing Multi-server, Single-server and the Trivial PIR Schemes

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 [28] 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

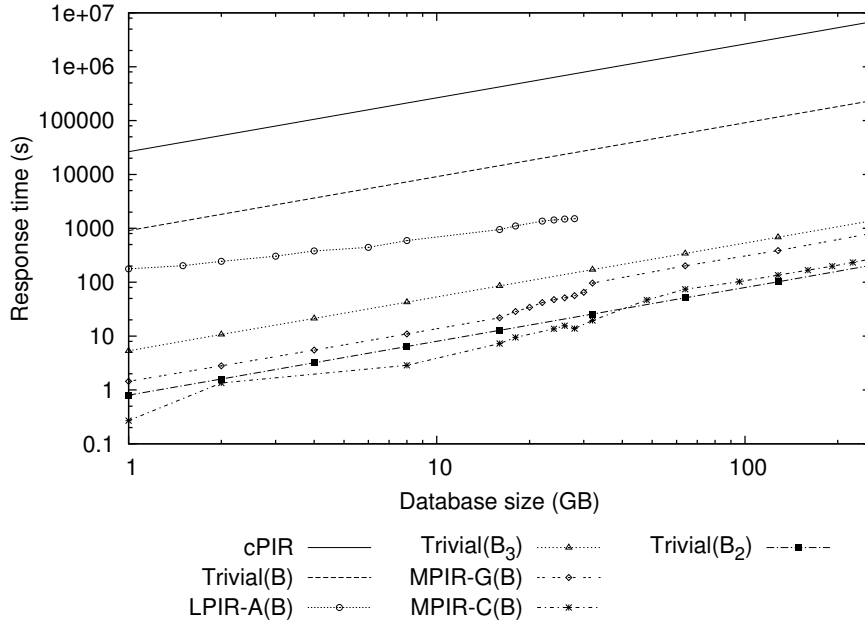


Fig. 3. Comparing the response times of PIR schemes by Kushilevitz and Ostrovsky (cPIR) [28], Aguilar-Melchor [1] (LPIR-A), Chor et al. [13] (MPIR-C), and Goldberg [22] (MPIR-G), as well as the trivial PIR scheme over three current network bandwidths data in Table 1, using different database sizes. The bandwidth used for the non-trivial PIR schemes is B .

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 1).

Figure 3 shows the log-log plot of the response times for the multi-server and lattice-based PIR schemes against the earlier results from [40], which include the trivial scheme and the Kushilevitz and Ostrovsky scheme [28]. As in [40], we give maximal benefit to the scheme in [28] by ignoring all costs except those of modular multiplication for that scheme, using the value for t_{mul} given in Section 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 servers for the multi-server schemes is $\ell = 2$.

We can see from the plot that, as reported in [40], 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. 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. We include one plot for a database of 16 GB, which fits in RAM, and one for 28 GB, which does not. 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.

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.

5.1 Lessons Learned, Recommendations and Justification

This section reports some of the lessons learned from our analysis and experimentation, gives some recommendations on how to mitigate practical issues about deploying PIR schemes, and provide other justification for deploying PIR-based systems.

- A closed-form analytic expression of the computation and communication costs of PIR schemes and any algorithm for that matter cannot be used to predict exact execution costs. Such expressions only give upper-bound execution cost on x86 CPUs without any support for hardware-level parallelization. However, all modern x86 CPUs support some form of parallelization. Hence, execution costs should be obtained by experimentally taking measurements on the implementation of the algorithm on benchmark hardware.
- The category of users referred to by Nielsen’s Law of Internet bandwidth is high-end. These users are more technologically savvy and could afford higher Internet bandwidth compared to typical user; referred to as low-end users. Low-end users are broader representation of Internet users. The measured

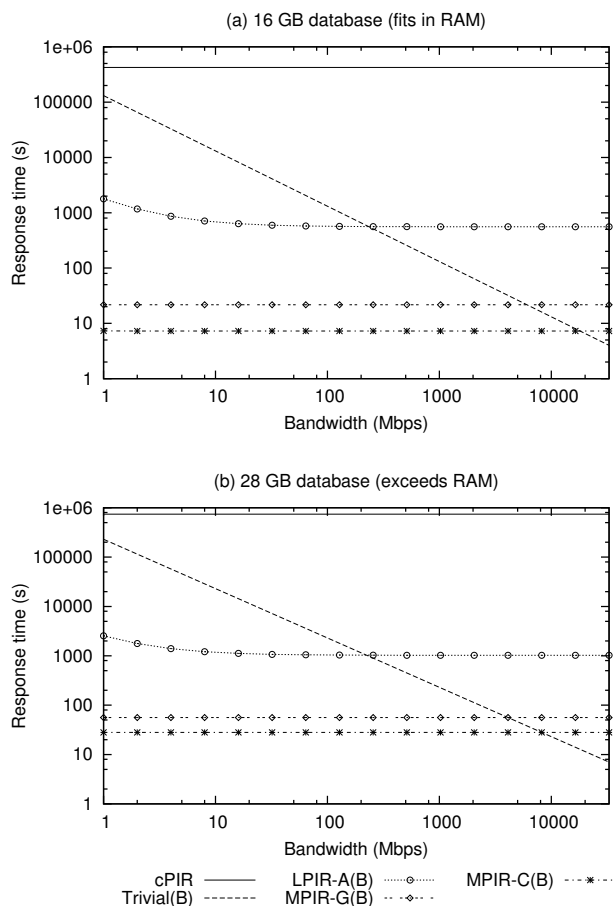


Fig. 4. Plots of response time vs. bandwidth for the PIR schemes by Kushilevitz and Ostrovsky (cPIR) [28], Aguilar-Melchor [1] (LPIR-A), Chor et al. [13] (MPIR-C) and Goldberg [22] (MPIR-G), as well as the trivial PIR scheme for database sizes that fit in RAM (16 GB) and that exceed RAM (28 GB).

bandwidth average for these users is a more appropriate bandwidth to use for any study examining the practicality of some new technologies. One good source for Internet bandwidth measurement data is [37].

- One of the main challenges in deploying multi-server PIR schemes is the assumption that an organization deploying multi-server PIR must outsource its database to one or more servers managed by some independent holders who should not collude to break users' query privacy. However, following the random server model [19] for deploying existing multi-server PIR schemes does not require an organization to replicate its data to holders who might

otherwise not be trusted. In the random server model, a PIR client would query some servers holding copies of a random string as well as the database server and compute the query result from the responses from these servers. Furthermore, some PIR schemes [22] provide constructions that resist a coalition of servers, guaranteeing either computational or information-theoretic privacy protection.

6 Conclusions

We reexamined the computational practicality of PIR following the earlier work by Sion and Carbunar [40]. Some interpret [40] 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 in [28] 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 [2] 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 considering cost-effective options 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.

Acknowledgments. We would like to thank Gregory Zaverucha and the anonymous reviewers for their helpful comments for improving this paper. We also gratefully acknowledge NSERC and MITACS for funding this research.

References

1. C. Aguilar Melchor, B. Crespín, P. Gaborit, V. Jolivet, and P. Rousseau. High-Speed Private Information Retrieval Computation on GPU. In *SECURWARE’08*, pages 263–272, 2008.
2. C. Aguilar-Melchor and P. Gaborit. A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol. In *WEWORC 2007*, July 2007.
3. C. Aguilar Melchor and P. Gaborit. A Fast Private Information Retrieval Protocol. In *ISIT’08*, pages 1848–1852, 2008.
4. D. Asonov. *Querying Databases Privately: A New Approach To Private Information Retrieval*. SpringerVerlag, 2004.

5. D. Asonov and J.-C. Freytag. Almost Optimal Private Information Retrieval. In *PETS'02*, pages 209–223, 2003.
6. A. Beimel, Y. Ishai, and T. Malkin. Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. In *CRYPTO'00*, pages 55–73, 2000.
7. A. Beimel, Y. Ishai, and T. Malkin. Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. *J. Cryptol.*, 17(2):125–151, 2004.
8. A. Beimel and Y. Stahl. Robust Information-Theoretic Private Information Retrieval. *J. Cryptol.*, 20(3):295–321, 2007.
9. D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith, III. Public key encryption that allows PIR queries. In *CRYPTO'07*, pages 50–67, 2007.
10. C. Cachin, S. Micali, and M. Stadler. Computationally Private Information Retrieval with Polylogarithmic Communication. In *EUROCRYPT'99*, pages 402–414, 1999.
11. B. Chor and N. Gilboa. Computationally Private Information Retrieval (extended abstract). In *STOC'97*, pages 304–313, 1997.
12. B. Chor, N. Gilboa, and M. Naor. Private Information Retrieval by Keywords. Technical Report TR CS0917, Department of Computer Science, Technion, Israel, 1997.
13. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. In *FOCS'95: Proceedings of the 36th Annual Symposium on the Foundations of Computer Science*, pages 41–50, Oct 1995.
14. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private Information Retrieval. *J. ACM*, 45(6):965–981, 1998.
15. G. Danezis, R. Dingleline, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *IEEE S&P*, pages 2–15, May 2003.
16. G. Di Crescenzo et al. Towards Practical Private Information Retrieval. Achieving Practical Private Information Retrieval Panel, SecureComm 2006, Aug. 2006. <http://www.cs.sunysb.edu/~sion/research/PIR.Panel.Securecomm.2006/giovanni.pdf>.
17. R. Dingleline, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
18. C. Gentry and Z. Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In *ICALP'05*, pages 803–815, 2005.
19. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting Data Privacy in Private Information Retrieval Schemes. In *STOC'98*, pages 151–160, 1998.
20. G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private Queries in Location Based Services: Anonymizers are not Necessary. In *SIGMOD'08*, pages 121–132, 2008.
21. I. Goldberg. Percy++ project on SourceForge. <http://percy.sourceforge.net/>.
22. I. Goldberg. Improving the Robustness of Private Information Retrieval. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 131–148, 2007.
23. GPGPU Team. High-speed PIR computation on GPU on Assembla. http://www.assembla.com/spaces/pir_gpgpu/.
24. InformIT. Introduction to DSL. <http://www.informit.com/articles/article.aspx?p=31699&seqNum=3>.
25. Intel. Intel and Ethernet. http://www.intel.com/standards/case/case_ethernet.htm.
26. A. Juels. Targeted advertising ... and privacy too. In *CT-RSA 2001*, pages 408–424, 2001.
27. B. Kaliski. TWIRL and RSA Key Size, 2003. <http://www.rsa.com/rsalabs/node.asp?id=2004>.

28. E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *FOCS'97*, page 364, 1997.
29. H. Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In *ISC'05*, pages 314–328, 2005.
30. S. K. Mishra and P. Sarkar. Symmetrically Private Information Retrieval. In *INDOCRYPT'00*, pages 225–236, 2000.
31. G. E. Moore. Cramming More Components Onto Integrated Circuits. *Electronics Magazine*, 38(8), 1965.
32. National Institute of Standards and Technology. Key Management Guideline, 2007. <http://csrc.nist.gov/groups/ST/toolkit/index.html>.
33. New European Schemes for Signatures, Integrity, and Encryption. Final Report IST-1999-12324, 2004. <https://www.cosic.esat.kuleuven.be/nessie/>.
34. J. Nielsen. Nielsen's Law of Internet Bandwidth, April 1988. <http://www.useit.com/alertbox/980405.html>.
35. F. Olumofin and I. Goldberg. Privacy-preserving Queries over Relational Databases. In *PETS'10*, Berlin, 2010.
36. F. Olumofin and I. Goldberg. Revisiting the Computational Practicality of Private Information Retrieval. In *Financial Cryptography and Data Security '11*, St. Lucia, 2011.
37. Ookla. Canada and US Source Data. <http://www.netindex.com/source-data/>.
38. F. 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.
39. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.
40. R. Sion and B. Carbunar. On the Computational Practicality of Private Information Retrieval. In *NDSS'07*, 2007.
41. S. W. Smith and D. Safford. Practical Server Privacy with Secure Coprocessors. *IBM Syst. J.*, 40(3):683–695, 2001.
42. C. E. Spurgeon. *Ethernet: The Definitive Guide*. O'Reilly & Associates, Inc., USA, 2000.
43. J. P. Stern. A New Efficient All-Or-Nothing Disclosure of Secrets Protocol. In *ASIACRYPT'98*, pages 357–371, 1998.
44. A. Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 2002.
45. J. Trostle and A. Parrish. Efficient Computationally Private Information Retrieval From Anonymity or Trapdoor Groups. ePrint 2007/392, 2007.
46. A. R. Weiss. Dhrystone Benchmark: History, Analysis, Scores and Recommendations. *EEMBC White Paper*, 2002.
47. C. Wieschebrink. Two NP-complete Problems in Coding Theory with an Application in Code Based Cryptography. In *ISIT'06*, pages 1733–1737, July 2006.
48. P. Williams and R. Sion. Usable PIR. In *NDSS'08*, 2008.
49. R. Yoshida, Y. Cui, R. Shigetomi, and H. Imai. The Practicality of the Keyword Search Using PIR. In *ISITA'08*, pages 1–6, December 2008.