On the Computational Practicality of Private Information Retrieval

Radu Sion [*]
Network Security and Applied Cryptography Lab
Computer Sciences, Stony Brook University
sion@cs.stonybrook.edu

Bogdan Carbunar
Pervasive Platforms and Architectures
Motorola Labs
carbunar@motorola.com

Abstract

We explore the limits of single-server computational private information retrieval (PIR) for the purpose of preserving client access patterns leakage. *We show that deployment of non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude less time-efficient than trivially transferring the entire database. We stress that these results are beyond existing knowledge of mere "impracticality" under unfavorable assumptions. They rather reflect an inherent limitation with respect to modern hardware, likely the result of a communication-cost centric protocol design. We argue that this is likely to hold on non-specialized traditional hardware in the foreseeable future. We validate our reasoning in an experimental setup on modern off-the-shelf hardware. Ultimately, we hope our results will stimulate practical designs.*

1 Introduction

Private Information Retrieval, (PIR) has been proposed as a primitive for accessing outsourced data over a network, while preventing its storer to learn anything about client access patterns [27].

In initial results, Chor et al.[27] proved that in an information theoretic setting in which queries do not reveal any information at all about the accessed data items, any solution requires $\Omega(n)$ bits of communication. To avoid this overhead, if multiple non-communicating databases can hold replicated copies of the data, PIR schemes with only sub-linear communication overheads are shown to exist [27]. We discuss related single-server PIR schemes in Section 4. The related notion of *symmetric* PIR (SPIR) [36, 56] handles the scenario in which privacy of server data is of concern and a client is allowed to retrieve only a limited number of data bits. We discuss SPIR in Section 5.1.

[*]Sion is supported in part through the NSF CyberTrust Award CNS-0627554 and by the Stony Brook Office of the Vice President for Research.

Here we discuss single-server computational PIR *for the purpose of preserving client access patterns leakage*. We show that deployment of non-trivial single server private information retrieval protocols on real hardware of the recent past would have been orders of magnitude more time-consuming than trivially transferring the entire database. The deployment of computational PIR would in fact *increase* overall execution time, as well as the probability of *forward* leakage, when the deployed present trapdoors become eventually vulnerable – e.g., today's queries will be revealed once factoring of today's values will become possible in the future.

We stress that this is beyond existing knowledge of mere "impracticality" under unfavorable assumptions. On real hardware, *no* existing non-trivial single server PIR protocol could have possibly had outperformed the trivial client-to-server transfer of records in the past, and is likely not to do so in the future either. This is due to the fact that on any known past general-purpose Von Neumann hardware, it is simply more expensive to PIR-process one bit of information than to transfer it over a network.

In particular, this impacts the type of complexity reasoning as found in [28] (section 2.4, page 971). The complexities discussed there do not consider the *significant* computation times associated with individual operations. While the claimed "linear in n" behavior of server-side computation seems satisfactory, unfortunately, the constants prove to be orders of magnitude higher than in the case of trivial (linear communication) transfer of data, in effect invalidating the case for PIR as a whole.

We are no wizards. The future prediction section of our results is at most well-documented guess-work. Indeed, in the event of an un-expected surge in processing speed per deployed processor gate (if not simultaneously reflected in network performance increases) it is possible that current single-server PIR protocols could become useful. But this is unlikely to happen until Moore's Law impact in computing *performance* will significantly out-perform Nielsen's Law of network *bandwidth*. Moreover, such a surge will not immediately make PIR usable as it will also likely increase the required minimum key sizes due to faster factoring available to adversaries.

Scope. We consider single-server computational PIR only. While we believe also the practicality of the *multi-server* case is hampered by hard-to-realize non-collusion assumptions, we will not explore multi-server PIR here. We note that more computationally efficient, sub-linear communication protocols for multi-server PIR exist, including clever pre-processing schemes [20] and the use of auxiliary servers [35] to "hide" the computation costs away from the main database server. We discuss how our results apply to SPIR scenarios, in Section 5.1.

Also, it is not in our intention to survey the inner workings (beyond complexity considerations) of various PIR mechanisms or of associated but unrelated research. We invite the reader to explore a multitude of existing sources, including the excellent, almost complete survey by William Gasarch [31, 32]. We will however briefly discuss general lower bounds on server-side per-data-bit computation as well as the associated communication complexity. For illustration purposes we will exemplify our numbers with one single-server PIR instance, namely [46], based on the quadratic residuosity assumption.

Ultimately, we hope this work will stimulate the design of practical single server PIR protocols.

2 Building Blocks

We start by outlining a few building block elements to our argument: arithmetic capabilities of modern commodity hardware, the state of the art in modular arithmetic (in particular multiplications), as well as an instance of a well-known single-server computational PIR mechanism.

2.1 Hardware

Given specification information availability as well as a large market share of small business and server markets, we will illustrate our results for top of the line server CPU x86 architecture hardware. We argue that the results hold certainly for other architectures in the past and are likely to do so also for traditional commodity Von Neumann architectures in the future [37]. Moreover, we are not concerned with expensive specialized fast modular arithmetic hardware. While commercially available accelerators do not surpass our considered baseline [1], arguably, more expensive, specialized high-throughput hardware exists or will become available. This, however, does not contradict the thesis of this paper. Indeed, the results below show that investing a few orders of magnitude more cash resources at the server side could yield just enough computation speed to keep up with trivial database transfer. Nevertheless, requiring such large amounts of resources constitutes an impractical deployment instance in itself. We consider instead realistic scenarios, using top of the line hardware likely available to commercial and governmental institutions.

[1]For example, the IBM 4764 PCI-X Cryptographic Coprocessor (PCIXCC) [14] can generate around roughly 900 RSA signatures per second. A 3.6GHz P4 yields a comparable throughput of 400 using the GNU Multiple Precision Arithmetic Library 4.2.1 [12]. Investing an order of magnitude more for such hardware for the sole purpose of accelerating modular arithmetic becomes questionable. We believe the reason for this anomaly is simply the lack of a market, as x86 performance is more than adequate for current applications and can be cheaply mass-produced.

Pentium 4. For illustration and experimental consistency purposes we will focus on the latest top of the line Intel [3] Pentium 4 (P4) [43] rated around 10500-11500 (DhryStone) MIPS in various instances and core frequencies [38, 61]. We mention that the results of this paper hold regardless of minor differences in architectural components and functionality reasoning [40], including AMD [1] and Motorola [5] chips. In particular they are true even if the arithmetic abilities of the processing plant were to be one order of magnitude faster. Moreover, choosing the x86 as the architecture of choice is justified by the recent shift toward x86 hardware even in traditional competitors of Intel/Microsoft [16].

For brevity we will discuss only relevant functionality and performance elements of the P4, namely its arithmetic abilities [17, 18]:

"[The P4 contains three *arithmetic and logic* units (ALU)], one complex, slow ALU and two simple, fast ALUs. [...] The complex ALU handles complex integer instructions like multiply, divide, and some special-purpose register instructions. Instructions sent to this ALU generally take *four cycles* to complete. [...] The P4's execution core exhibits one major peculiarity that sets it apart from any other architecture: two of its integer execution units run at twice the core clock speed. This allows each double-speed unit effectively to act as two regular-speed units, because each unit can take in and spit out two instructions per clock cycle (one on the clock's rising edge and one on its falling edge). [...] That's a lot of integer horsepower, and indeed the P4 does quite well in integer benchmarks, especially at higher clock speeds.]"

For simplification reasons we will favor the case for PIR and assume that the deployed CPU will be able to perform *at least 1 digit multiplication per cycle*. Additionally, we note that we will consider this also for older architectures (certainly requiring more cycles per multiplication), again favoring PIR.

Parallelism. It is also important to note that the addition of multiple units of computation (whether as multi-core architectures [41] or as traditional multi-processor systems) can only speed up computation so much. The minimal degree of parallelism that would make PIR practical is explored in

Section 6. Ultimately this reduces to balancing a trade-off between financial and performance concerns.

MIPS. For the prediction section of our results, we will use MIPS (Millions of Instructions Per Second) figures as a baseline to provide an approximate scale for future performance. While commonly used as a metric of processor speed, MIPS constitute at its best just a relative measure of performance for general purpose applications [68]. Nevertheless, the less I/O intensive nature of modular arithmetic makes MIPS a better predictor to use in this case. Additionally, this is reasonable because the claims below hold within one order of magnitude or more (i.e., even if the CPU were one order of magnitude faster). Moreover, we will favor PIR by always using optimistic speed estimates (assuming the fastest CPU). Let M denote the MIPS figure for the currently considered CPU.

2.2 Fast Modular Arithmetic Algorithms

In this section we briefly survey current fast modular multiplication algorithms.

Beyond Montgomery reduction [52, 54] which results in a cost of $2m^2 + 2m$ digit operations (m is the number of digits in the operands) a multitude of results have aimed at reducing the constants. For example, the method in [65] yields the following execution times:

$$t_{mul} \approx (m^2+7m)t_d+(4m^2+20m)t_a+(4m^2+2m)t_{mem} \quad (1)$$

where we denoted by t_d, t_a and t_{mem} the digit multiplication, digit addition and memory access times.

For illustration purposes we will make a few simplifying assumptions, favorable to the deployment of computational PIR. We will ignore additions and memory accesses as well as insignificant factors in (1). We will approximate the number of digits in the operands by $m \approx \frac{|N|}{d}$ where $|N|$ is the bit-size of N and d is the bit-size of a digit. We define:

$$t_{mul}(|N|) \approx (\frac{|N|}{d})^2 \times t_d$$

Normally [48, 50] we have $d = log_2(10) \approx 3.3$. The decision for the value of d should be made based on the computing platform and the programming language used to implement modular reduction [21]. To account for pipelining and inter-ALU optimizations on the considered Intel platforms, in the evaluation experiments we will make the empirical, PIR-favorable approximation of $d \approx 5$ (operating in base 32). This decision is favoring PIR because in effect it is reducing the number of digits in the main operands [2].

[2]Thus also reducing the claimed complexity of server-side operations, if 1 digit operation is performed per second as discussed in Section 2.1.

From Section 2.1 we have $t_d \approx \frac{1}{M}$ and thus:

$$t_{mul}(|N|) \approx \frac{|N|^2}{M \times d^2} \quad (2)$$

We note that in our experimental evaluation in Section 3.2 this result validates well.

2.3 Quadratic Residuosity PIR

It is known that single-server PIR requires a full transfer of the database [27, 28] for computationally unbounded adversaries (servers). For bounded adversaries, *computational* PIR (cPIR) mechanisms have been proposed.

For illustration purposes, we will consider here one such protocol [46]. We note that our results hold immediately for other mechanisms such as [24, 26, 49]. This is due to the following two reasons. First, even though these newer cPIR solutions have a lower communication complexity than [46], our analysis ignores the communication costs of cPIR. Second, an $O(n)$ lower bound on server-side computation complexity is trivial to establish – and for privacy, the per-bit operation(s) for current protocols rely on expensive modular arithmetic leveraged in the instantiation of some trapdoor. In fact, in Section 4 we show that the computation costs of [24, 26, 49] exceed those of [46].

The hardness problem of choice is the quadratic residuosity assumption and its equivalent, factoring. We investigate PIR computation times and compare against the alternative of transferring the entire database to the client.

In the following we briefly discuss the cPIR mechanisms in [46]. The n bits of the database are organized logically at the server as a bi-dimensional matrix M of size $\sqrt{n} \times \sqrt{n}$. To retrieve bit $M(x, y)$ with computational privacy, the client:

- randomly chooses two prime numbers p and q of similar bit length, computes their product, $N = pq$ and sends it to the server.
- generates $\sqrt{n}$ numbers $s_1, s_2, \ldots, s_{\sqrt{n}}$, such that s_x is a quadratic non-residue (QNR) and the rest are quadratic residues (QR) in $\mathbb{Z}_N^*$.
- sends $s_1, s_2, \ldots, s_{\sqrt{n}}$ to the server.

For each "column" $j \in (1, \sqrt{n})$ in the $\sqrt{n} \times \sqrt{n}$ matrix, the server:

- computes the product $r_j = \prod_{0<i<\sqrt{n}} q_{ij}$ where $q_{ij} = s_i^2$ if $M(i, j) = 1$ and $q_{ij} = s_i$ otherwise [3].

[3]In fact this apparently works also with less work by making $q_{ij} = s_i$ if $M(i, j) = 1$ and $q_{ij} = 1$ otherwise. In the remainder of the paper we will favor PIR by only assuming 1 multiplication per bit is required

- sends $r_1, \ldots, r_{\sqrt{n}}$ to the client

The client then simply checks if r_y is a QR in $\mathbb{Z}_N^*$ which implies $M(x,y) = 1$, else $M(x,y) = 0$.

The last step can be done also recursively, further reducing communication. This however, only leads to increases in computation times, and would make the case for deploying PIR vs. transferring the data even more difficult. We will thus rather just look at the lesser PIR cost of the simple protocol above.

Let M be the CPU processing speed measured in MIPS and B the available (bps) network bandwidth between client and server. We will denote $t_t = \frac{1}{B}$ the time required to transmit *one bit* between the server and the client, and $t_{qrv}(b)$ the time required to verify the quadratic residuosity of one b-bit number.

The above computational PIR protocol will incur the following cost (including communication and execution time):

$$T_{pir} = nt_{mul}(|N|) + 2\sqrt{n}(|N|)t_t + \sqrt{n}t_{qrv}(|N|) \quad (3)$$

We will simplify further, by ignoring any other costs besides the $nt_{mul}(|N|)$ factor:

$$T_{pir} \approx n \times t_{mul}(|N|)$$

We emphasize that this is favorable to PIR, by ignoring the quadratic residuosity verification of $\sqrt{n}\,|N|$-bit numbers (as well as the PIR communication costs), which can be significant. We will compare against the time required to transfer the entire database:

$$T_{trans} = n \times t_t$$

Let us then consider ΔT, the difference in execution time between a protocols involving PIR vs. simple complete database transfer to the client:

$$\Delta T = T_{pir} - T_{trans} \leq n \times (t_{mul}(|N|) - t_t)$$

It can be seen that if the server-side 1-bit processing time exceeds a 1-bit transmission time, the single-server computational PIR protocol will take longer ($\Delta T > 0$) than simply transferring the database over to the client.

Next, to evaluate the above in a PIR-favorable scenario, we will assume that very fast modular arithmetic algorithms are put in place, as discussed above in Section 2.2. Using equation (2) in Section 2.2, we write:

$$((\frac{|N|^2}{M \times d^2} - \frac{1}{B}) > 0) \Rightarrow (\Delta T > 0) \quad (4)$$

Equation (4) represents the boundary condition that determines whether single-server computational PIR is slower than downloading the entire database to the client. In Section 3 we evaluate this condition and show that its left side holds true on traditional hardware.

2.4 Key Sizes

By operating under an assumption of a *computationally* bounded adversary, it is important to assess associated bounds and relate them to the deployed privacy-enabling trapdoor. Because the single-server computational PIR setting of choice relies on the quadratic residuosity assumption we will consider here the (equivalent) assumed hardness of factoring as a metric for achieved privacy.

RSA Labs [8] has started evaluating and recommending key sizes for RSA since 1995 [11] when 768 bit sizes were deemed appropriate for most application. Both the RSA [9] and the National Institute of Standards and Technology (NIST) key schedules [58] (last updated in August 2005) propose 1024 bits minimum until 2010. Secrets that are to live beyond 2010, but not after 2030, are to be protected by minimum 2048 bit RSA keys. Beyond 2030, a minimum key size of 3072 bit and above is recommended [9] (we omit the symmetric key sizes and discussion for brevity). See also the RSA Factoring Challenge effort [7]. Additionally, in 2004, the NESSIE (New European Schemes for Signatures Integrity and Encryption) Project [6] recommended a minimum of 1536 bits for RSA signatures.

$target$	1995	$2000-2010$	$2011-2030$	$2030-$
$bits$	768	1024-1536	2048	3072

Figure 1. RSA key size schedule.

These recommendations are important to consider also in the light of new specialized factoring hardware such as the Weizmann Institute Relation Locator [10] that claims factoring times of no more than 1 year for 1024 bit sizes, at a cost of a few dozen million US dollars. In the following we will use these recommendations (Figure 1) to establish the values of $|N|$ for different points in time.

3 Timeline

To evaluate the behavior of boundary condition (4) we will basically analyze its left side. Specifically, we will compare the time required to perform a modular $|N|$-bit multiplication, t_{mul} with the time taken to transfer one bit of information, t_t.

As a lower bound baseline we consider Intel CPUs [25, 42], and a variety of network setups, including average home-user last-mile connection bandwidths [4], Ethernet as well as commercial high-end inter-site connections [2, 70, 72]. These are arguably settings in which PIR would be very likely of deployment use.

3.1 Past: 1995-2005

We start by discussing the evolution of t_{mul} and t_t between 1995 and 2005[4].

year	M	B	B_2	B_3
1995	200	0.028	10	0.256
1997	300	0.056	100	0.768
1998	400			1.000
1999	744	0.768	1000	10
2000	1500			
2001	2500	1.000		100
2005	15000	4.000	10000	1000
2006	25000	6.000	10000	1500

Figure 2. Estimated average values for x86 CPU MIPS, end-user home commodity Internet (B), Ethernet LAN (B_2) and commercial high-end inter-site (B_3) bandwidth (Mbps), between 1995 and 2006.

Figure 2 shows averages of MIPS (M) [25, 42] and bandwidth (B) values for commodity hardware and various types of networks [2, 30, 70, 72], between 1995 and 2006.

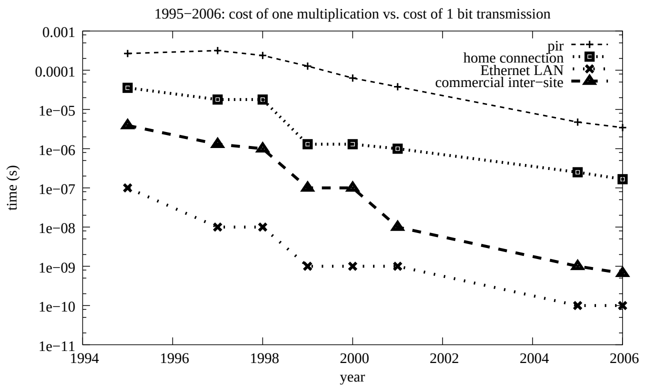

Figure 3. Comparison between the time required to perform PIR and the time taken to transfer the database, between 1995 and 2005. PIR is orders of magnitude slower. (logarithmic)

In Figure 3 (logarithmic scale) we plot t_{mul} and t_t for hardware between 1995 and 2005 (see also Figure 2). It can be seen that the time required for performing one $|N|$-bit multiplication has consistently been *one order of magnitude* larger than the time of transferring one bit on a low-end connection and *two orders of magnitude* larger than the transfer on a high-end connection. Note that the increase of $|N|$ from 768 to 1024 bits, reflects also in the increase of PIR cost between 1995 and 1997.

[4]Single-server computational PIR was introduced in 1995.

3.2 Present: 2006

We now consider current hardware and start by validating equation (2). This is important so as to ensure better prediction ability for Section 3.3. For this purpose, we benchmarked 1024 bit operations on a Intel(R) Pentium(R) 4 CPU running at 3.60GHz with 1GByte RAM, using the GNU Multiple Precision Arithmetic Library [5] 4.2.1 [12]. For modular multiplication GMP uses Montgomery's REDC [54] method [12]. Running in semi-controlled light-load multi-user mode on a Linux box, we obtained a throughput of around 0.273 million 1024-bit modular multiplications per second.

For this processor, rated at around 11000 MIPS (see Section 2.1), this value is predicted with surprising accuracy by equation (2) – considering all the simplifying assumptions made in deriving it. The value predicted is around 0.275 million. The higher actual throughput is likely due to pipelining, predictive branching and multi-ALU operations.

For cross-validation purposes, we further repeated the same experiments on other platforms, including a Pentium(R) M CPU running at 1.80GHz, rated at 6500-7500 (Dhrystone) MIPS [53, 62] (despite its lower clock-rate). We achieved similar good prediction: 0.197 million 1024-bit modular multiplications were benchmarked, 0.189 million were predicted by equation (2).

In the Pentium(R) 4 CPU setup above, PIR processing one single bit would require around 3700 nanoseconds. Considering even a home-user cable-modem bandwidth of only 10Mbps [6] transferring one bit would require roughly 100 nanoseconds. Even if one would consider significant transfer protocol overheads this would still be over 30 times faster than PIR processing.

Drastically Limited Bandwidth. For extremely limited bandwidth however, it seems like PIR could be more efficient. In this particular scenario, PIR seems to become usable for bandwidths of 300Kbps or less. This is not the case immediately however. The PIR-favorable simplifications and previously ignored factors now become significant. For example, on a slow connection, the $2\sqrt{n}(|N|)$ PIR associated network traffic in equation (3) cannot be ignored anymore. This will (i) further reduce the bandwidth threshold below which PIR is usable and (ii) make this threshold dependent on n (thus impractical in deployment).

[5]Which proved faster than the OpenSSL 0.9.7j Library [63], thus was chosen instead.

[6]At the time of this writing, one of the author's home connection offers 15Mbps down-stream and 5Mbps up-stream throughputs for \$29.95/month.

We illustrate this borderline behavior in Figure 4. The dependency of n becomes apparent when considering 10 MBytes vs. 1 MBytes databases. It can be seen that for 10 MBytes databases, PIR become profitable if the bandwidth is below 210 Kbps, whereas in the case of a 1 MByte database, this threshold goes down to 70Kbps. It is also important to note that for $\sqrt{n} < |N|$ this PIR protocol would require more communication than a trivial database transfer (thus it cannot be used for small databases, e.g. $<$ 1MBit).

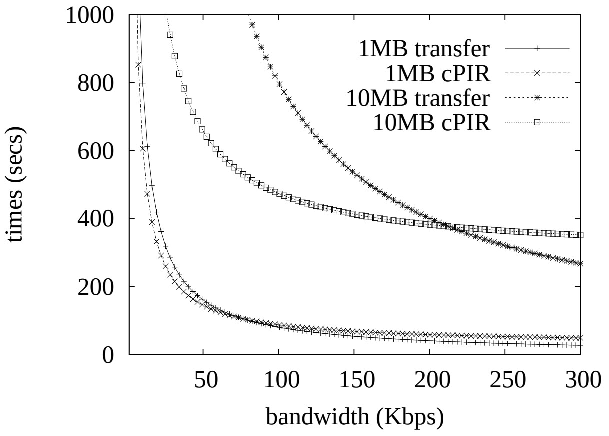

Figure 4. Low Bandwidth ($t_{mul} < t_t$, condition (4) does not hold): behavior of execution times for cPIR for the Pentium(R) 4 CPU setting vs. database transfer times. If its (previously ignored) communication overheads are considered, the bandwidth thresholds below which cPIR becomes usable (faster than trivial transfer) further decrease.

3.3 Future: 2006-2035 and beyond

We have shown that single-server computational PIR has not been yet an efficient alternative to a straightforward database transfer for the purpose of privacy of client access patterns. Now we explore if this is likely to change in the next 30 years. We model advances in processing power by Moore's Law [55]. We model future bandwidth figure by Nielsen's Law of Internet Bandwidth [57].

In its current form, believed to hold for at least two more decades [25, 42], Moore's Law states that the number of transistors on integrated circuits doubles every 18 months (increases 100 times every ten years). Intel notes the law holds very well, roughly doubling the processor MIPS figures every two years [42]. Nielsen's Law states that high-end network bandwidth grows at least by 50% per year (increases 57 times every ten years). It has been so far (over)validated [2, 70, 72].

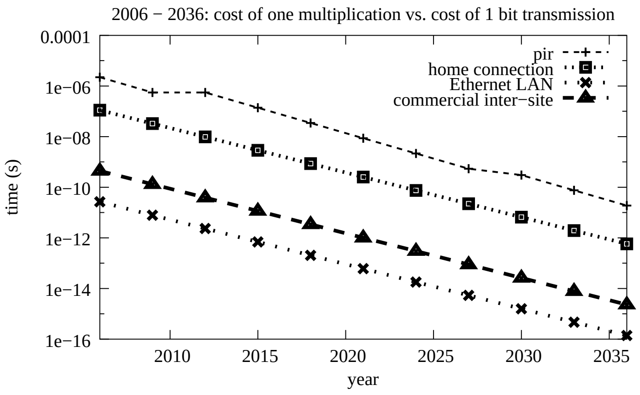

Figure 5. Prediction of future PIR execution vs. database transfer times. PIR is orders of magnitude slower. (logarithmic)

It is apparent that processing power increases faster than the available bandwidth. The increase in processing power however implies not only decreases in t_{mul} but also mandates larger $|N|$ values as factoring speeds will increase too. It is important to evaluate which of these trends (if any) will dominate. Figure 5 (logarithmic scale) plots the predicted evolution for t_{mul} and t_t between 2005 and 2035 by also considering the recommended key schedules discussed in Section 2.4. It can be seen that t_{mul} can be predicted to continue to be at least one order of magnitude larger than t_t (even for slow home connections !), effectively neutralizing the impact of the accelerated increase in processing power vs. bandwidth. In other words PIR execution times are likely to continue to be orders of magnitude higher than trivial database transfers.

4 Other Protocols

We argued above that deploying any single-server computational PIR protocol is necessarily less efficient than a simple transfer of the entire database, in real scenarios. We have done so by illustrating our results with the aid of a particular well-known computational PIR method.

In this section we aim to understand if other existing single-server PIR variants can surpass these results and in fact be efficient enough to become usable. We previously (Section 2.3) argued that this is unlikely due to the $O(n)$ lower bound on server-side processing mandating expensive trapdoor operations per bit, for privacy. We now elaborate on existing single-server PIR mechanisms.

Cachin et al.[24] propose the use of the ϕ-Hiding Assumption to perform PIR with poly-logarithmic communication complexity. We note the protocol requires the server to perform n exponentiations modulo m. As the security of the protocol relies on the assumed hardness of factoring m, this immediately implies that the bit size of m has to be at least as recommended in Section 2.4. This would render the method significantly more expensive than the quadratic residuosity PIR version discussed above, as modular exponentiation is costlier.

Lipmaa [49] extends the work of Stern [71] to provide a PIR scheme with $O(log^2(n))$ communication complexity. However, the server performs more than n exponentiations modulo k^{s+j}, $j = 1..\alpha$, where k is a public key and s and α are fixed. Since the bit size of the modulus is $|k|s$, we expect these exponentiations to be certainly more expensive than the modular multiplications analyzed in our work.

Chang [26] introduces a single database computational PIR scheme for which the server side communication complexity is $O(log(n))$. This scheme relies on Paillier [59]. The computation complexity significantly exceeds the complexity of our considered algorithm above [46] as the server is required to operate in $\mathtt{Z}^*_{\mathtt{n}^2}$. Thus the results hold [7].

Kushilevitz and Ostrovsky [47] generalize a set of previous results and show that PIR can be performed with communication strictly less than n when using one-way trapdoor permutations (TDPs). They propose a protocol that requires the client to send $O(k^2)$ bits and the server to reply with $n - n/2k$ bits, where k is a security parameter of the TDPs. In addition, the server has to perform n/k TDPs on strings of k bits and n bitwise XORs.

Thus the protocol decreases communication by $n/2k$ bits at the expense of n/k TDPs. For consistency, it is natural to require the PIR method to offer at least the security attained under the hardness of factoring trapdoor as considered above. Unless a cheaper instance of such type of trapdoor can be materialized, for the time being it is reasonable to assume the trapdoor is at least as computationally intensive as modular squaring [8]. In fact our argument requires less, namely that the trapdoor is at least as expensive as *one half* the cost of a modular multiplication.

This is true because the protocol effectively reduces the argument of PIR's practicality to a comparison between the cost of the trapdoor and *half* of the cost of transferring a bit. This in fact strengthens the argument against PIR that was made above, which relied on comparing the cost of transferring a *full* bit with the cost of the trapdoor. Thus, the above results hold.

Mann [51] extends the work of [46] with a trapdoor predicate with homomorphic properties; the server is required to perform n modular multiplications.

[7] In fact we have designed and experimentally evaluated a customized, more efficient (than [26]) use of Paillier for PIR and validated the fact that [46] is much faster.

[8] It is known that TDPs imply the existence of public key encryption mechanisms. Known TDPs are RSA, Rabin [64] and Paillier. RSA requires one modular exponentiation and Rabin relies on modular squaring. Paillier [60] requires two exponentiations and a multiplication in $\mathtt{Z}^*_{\mathtt{n}^2}$.

5 Limitations of Our Argument

We now briefly describe several existing PIR extensions for which our analysis seems to not directly apply.

5.1 Symmetric Private Information Retrieval

In the analysis above, we specifically did not address the symmetric PIR case. We do so here.

In the multi-server space, Gertner et al. [36] propose a transformation from any PIR scheme to a SPIR scheme at the expense of an increased number of non-cooperating servers. Naor and Pinkas [56] have shown a PIR-to-SPIR transformation for any PIR scheme for the single server case. The transformation uses their 1-out-of-n Oblivious Transfer (OT) protocol. The idea of the transformation is for the server to generate $l = \log n$ key pairs, $(k_1^0, k_1^1), .., (k_l^0, k_l^0)$ and mask its i-th data item, $i = 1..n$, with a subset of the keys (one from each pair), corresponding to the binary representation of i. That is, if the server's values are $x_1, .., x_n$ and the binary representation of i is $i_1..i_l$, then the server masks value x_i using keys $k_1^{i_1}, .., k_l^{i_l}$. Let x_i be the value the client wants to retrieve and let y_i denote the masked x_i value. Using any existing 1-out-of-2 OT protocol, the client retrieves only one key out of each pair, the one corresponding to the binary representation of the index of interest ($k_1^{i_1}, .., k_l^{i_l}$).

The PIR-to-SPIR transformation uses any PIR scheme to transfer to the client only the masked item of interest. The transformation adds to the communication complexity of the PIR scheme only $\log n$ invocations of the 1-out-of-2 OT protocol. The per query computation complexity increases with $n \log n$ pseudo-random function evaluations.

However, as argued in this paper, trivial PIR (transferring the entire database at the client) is more efficient than performing computation intensive PIR with reduced communication costs. Consequently, it seems that having the client simply transfer all n masked items, $y_1, .., y_n$ would be more efficient in this case. The symmetric assurances still hold, since the client can retrieve only one key out of each key pair, it can unmask only one of the n items. In conclusion, symmetric PIR seems to be achievable without the aid of non-trivial computational PIR schemes.

5.2 Computation-Amortized PIR

Ishai et al. [44] proposed the use of batch codes to amortize the server-side computation complexity of PIR over k queries performed by a single client. The solution allows the simultaneous retrieval of k data items using only $n^{1+o(1)}$ server-side computation. Thus, its applicability is constrained to cases where clients can wait for the collection of k queries. In [45] the same authors have also investigated multi-client computation and communication amortization techniques. They propose the use of two-way anonymous communication channels and amortization over multiple clients in order to provide a PIR solution close to optimal in terms of both communication and computation. Its security relies on the hardness of interpolating noisy low-degree curves in a low dimensional space.

Specifically, the server stores the database entries as coefficients of a c-variate polynomial q of degree $d = O(n^{1/c})$, where n is the database size. For each item x_i in the database, there exists a point $z_i \in F^c$, such that $q(z_i) = x_i$ (where F is a field). For a query for item x_i a client randomly generates c polynomials of degree k, $p_1, .., p_c$ such that $z_i = (p_1(0), .., p_c(0))$. It then picks $kd + 1$ points from F and generates for each such point a sub-query consisting of the evaluation of its c polynomials on the point. It then sends the resulting c points to the server, through an anonymizer. Upon receiving a sub-query, the server evaluates q on the c points of the sub-query, and anonymously sends back the result. Thus, the server is prevented from correlating sub-queries and clients. Using the answers to $kd+1$ sub-queries, the client can reconstruct x_i.

To provide computational privacy, the above mechanism requires each client to add “noise” to its query. That is, a client needs to mix its previous $kd + 1$ sub-queries with a set of random points. The total amount of noise sent by all uncorrupted (not cooperating with the server) clients must be $\omega(kn^{1+1/c})$. Each sub-query, including each noise point, is sent separately through the anonymizer.

While the maximum number of clients concurrently accessing the database, C, can be on the order of thousands, the size of average databases can be on the order of millions of items. Hence, the number of noise containing sub-queries per client query, $\omega(kn^{1+1/c}/C)$ is likely to be quite large. For instance, for a database of one million items, concurrently accessed by 10000 clients and for values of $k = 5$ and $c = 20$, the number of noise sub-queries per query needs to be larger than 1000. This, together with the relatively high latency of anonymizers[9] (on the order of hundreds of milliseconds), can lead to high response times.

Experiments in Tor [29] (similarly in [23]) show anonymizer-induced latencies to be anywhere in the 0.1-5s range. Even in the presence of optimal query pipe-lining, and, under the favorable (yet unrealistic) assumption that all clients “arrive” simultaneously at the anonymizer, this can result in significant overheads, often rendering trivial multi-client transfer of the database more efficient. Moreover, in [29] the authors note the fact that, with increasing network loads (e.g., many clients – as required by the mechanisms above), “the chance of building a slow circuit is increasing”, thus leading to increased latencies in average expectation.

[9]Low latency anonymizers are open to timing attacks [22].

6 Conclusions

We explored single-server PIR for client access privacy. We showed that single-server PIR protocols, running on modern high-end non-specialized hardware and networks, are mostly orders of magnitude slower than the trivial transfer of the entire database to the client. We illustrated this for past hardware and experimentally validated our claim on current hardware. We predicted the results to hold also in the future based on considerations of future network and computation devices.

We explored settings in which existing single-server PIR protocols may become usable. In particular, this is the case for scenarios involving highly limited bandwidth (tens of KBps or less) networks. Moreover, such protocols can be leveraged if large (10-1000 CPUs) amounts of server processing units are available to overcome the orders of magnitude difference between per-bit privacy processing and bit network transfer. This is likely impractical from a dollar-cost point of view. Purchasing hundreds of CPUs to achieve the same privacy level as offered by transferring the entire data over a cheap network link is hardly sound. Informally, single-server PIR can only become usable if communication can be traded for orders of magnitude more computation. We argue this is an unrealistic proposition given likely mainstream or even specialized application scenarios.

Recommendations. We hope this work will stimulate work on practical designs [13]. We believe it is important to explore protocols for single-server PIR in the presence of server-side trusted hardware [15, 69]. This should allow the delegation of client-logic in closer proximity to the data and might yield significant benefits. The dominant component of current solutions [19, 39, 73] is the periodic reshuffling of the database, performed by the secure CPU. The period is determined by the size of the secure CPU’s tamper proof cache. For reshuffling, the operations performed by the secure CPU are encryptions, decryptions and communication with the host. Asonov [19] proposed a solution that requires $O(n\sqrt{n})$ operations for a reshuffle. Iliev and Smith [39] use Benes networks to decrease this overhead to $O(n \log n)$ operations. Wang et al.[73] further reduce this overhead to $O(n)$ operations.

We argue that “run client p̈roxyïnside secure CPU” approaches [19, 39, 73] are likely to fail as typically such hardware is orders of magnitude slower than main CPUs (e.g., due to heat dissipation concerns). The main CPU will remain starkly under-utilized and the entire cost-proposition of having fast (unsecured) main CPUs and an expensive and slow secured CPU will be defeated. We believe efficient protocols are likely to access the secure hardware just sparsely, in critical portions, not synchronized with the main data flow.

Additionally, novel PIR protocols with lower server-side per-bit computation requirements should be designed. Main directions of research here include (i) the design of block-level protocols that amortize expensive modular operations over larger block sizes, and (ii) protocols based on novel hard problems that allow cheaper server-side arithmetic.

In promising ongoing work [33, 34] a method replacing modular multiplication with modular addition (based on novel security results [66, 67]), has been proposed.

7 Acknowledgments

We would like to thank Dan Boneh, Giovanni Di Crescenzo, Bill Gasarch, Aggelos Kiayias, Rafail Ostrovsky, Gene Tsudik, Rebecca Wright, Moti Yung, as well as our anonymous reviewers for their great feedback throughout the process that led to this paper.

References

[1] Advanced Micro Devices. Online at `http://www.amd.com`.

[2] Intel and Ethernet. Online at `http://www.intel.com/standards/case/case_ethernet.htm`.

[3] Intel Corporation. Online at `http://www.intel.com`.

[4] Introduction to DSL. Online at `http://www.informit.com/articles/article.asp?p=31699&seqNum=3&rl=1`.

[5] Motorola. Online at `http://www.motorola.com`.

[6] New European Schemes for Signatures Integrity and Encryption. Online at `http://www.cryptonessie.org/`.

[7] RSA Factoring Challenge. Online at `www.rsasecurity.com/rsalabs/challenges/factoring/`.

[8] RSA Labs. Online at `http://www.rsasecurity.com/rsalabs`.

[9] TWIRL and RSA Key Size. Online at `http://www.rsasecurity.com/rsalabs/node.asp?id=2004`.

[10] TWIRL: The Weizmann Institute Relation Locator. Online at `http://www.wisdom.weizmann.ac.il/~tromer/twirl/`.

[11] RSA CryptoBytes, Summer 1995. Online at `ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto1n2.pdf`, 1995.

[12] GMP: GNU Multiple Precision Arithmetic Library. Online at `http://www.swox.com/gmp/`, 2005.

[13] Achieving Practical Private Information Retrieval (Panel). Online at `https://www.cs.stonybrook.edu/~sion/research/PIR.Panel.Securecomm.2006/`, 2006.

[14] IBM 4764 PCI-X Cryptographic Coprocessor (PCIXCC). Online at `http://www-03.ibm.com/security/cryptocards/pcixcc/overview.shtml`, 2006.

[15] IBM Cryptographic Hardware. Online at `http://www-03.ibm.com/security/products/`, 2006.

[16] Apple. Apple to Use Intel Microprocessors Beginning in 2006. Online at `http://www.apple.com/pr/library/2005/jun/06intel.html`, June 2005.

[17] Arstechnica.com. The Pentium 4 and the G4e: an architectural comparison – Part II: the execution core. Online at `http://arstechnica.com/articles/paedia/cpu/p4andg4e2.ars/2`, Nov. 2001.

[18] Arstechnica.com. The Pentium: An Architectural History of the World's Most Famous Desktop Processor (Part II). Online at `http://arstechnica.com/articles/paedia/cpu/pentium-2.ars/3`, July 2004.

[19] D. Asonov. *Querying Databases Privately: A New Approach to Private Information Retrieval*. Springer Verlag, 2004.

[20] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. *Lecture Notes in Computer Science*, 1880:55–??, 2000.

[21] A. Bosselaers, R. Govaerts, and J. Vandewalle. Comparison of three modular reduction functions. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 175–186, 1994.

[22] Brian N. Levine and Michael K. Reiter and Chenxi Wang and Matthew Wright. Timing attacks in low-latency mix-based systems. In *Proceedings of Financial Cryptography*, 2004.

[23] M. Burnside and A. Keromytis. Low latency anonymity with mix rings. In *Proceedings of the 9th International Information Security Conference (ISC)*, 2006.

[24] C. Cachin, S. Micali, and M. Stadler. Private Information Retrieval with Polylogarithmic Communication. In *Proceedings of Eurocrypt*, pages 402–414. Springer-Verlag, 1999.

[25] C. D. Carothers. Evolution of Intel microprocessors: 1971 to 2007. Online at `http://www.cs.rpi.edu/~chrisc/COURSES/CSCI-4250/SPRING-2004/slides/cpu.pdf`, 2004.

[26] Y. Chang. Single-Database Private Information Retrieval with Logarithmic Communication. In *Proceedings of the 9th Australasian Conference on Information Security and Privacy ACISP*. Springer-Verlag, 2004.

[27] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.

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

[29] R. DingleDine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[30] Ethernet Alliance. A historical perspective of ethernet. `http://www.ethernetalliance.org/technology/presentations`.

[31] W. Gasarch. A WebPage on Private Information Retrieval. Online at `http://www.cs.umd.edu/~gasarch/pir/pir.html`.

[32] W. Gasarch. A survey on private information retrieval, 2004.

[33] W. Gasarch and R. Sion. Personal Communication, 2006.

[34] W. Gasarch and A. Yerukhimovich. Computational Inexpensive PIR (unpublished manuscript), 2006.

[35] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval or how to achieve information theoretic PIR avoiding database replication. *Lecture Notes in Computer Science*, 1518:200–??, 1998.

[36] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 151–160, New York, NY, USA, 1998. ACM Press.

[37] J. L. Hennessy and D. Goldberg. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 1996.

[38] Hothardware.com. Pentium 4 670 3.8GHz Performance Profile. Online at `http://www.hothardware.com/viewarticle.aspx?articleid=686`.

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

[40] Intel. *Intel Itanium 2 Processor Reference Manual For Software Development and Optimization*. Intel Corporation, 2004.

[41] Intel. Intel Multi-Core Processing. Online at `http://www.intel.com/cd/ids/developer/asmo-na/eng/strategy/multicore/index.htm`, May 2006.

[42] Intel Circuit Research Labs. Gigascale Integration-Challenges and Opportunities. Online at `http://www.intel.com/cd/ids/developer/asmo-na/eng/182440.htm`.

[43] Intel Corporation. Intel Pentium 4 Processor. Online at `http://www.intel.com/products/processor/pentium4`.

[44] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of com puting*, pages 262–271, 2004.

[45] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from anonymity. In *Proceedings of 47st Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, 2006.

[46] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings of FOCS*. IEEE Computer Society, 1997.

[47] E. Kushilevitz and R. Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In *Proceedings of EUROCRYPT*, 2000.

[48] J.-Y. Leu and A.-Y. Wu. A scalable low-complexity digit-serial VLSI architecture for RSA cryptosystem. In *Proceedings of the IEEE Workshop Signal Processing Systems SIPS*, 1999.

[49] H. Lipmaa. An oblivious transfer protocol with log-squared communication. Cryptology ePrint Archive, 2004.

[50] Mads Oesterby Olesen and Henrik Sandmann and Christopher Mosses. Implementing Fast Modular Arithmetic (Course). Online at `http://www.daimi.au.dk/~cmosses/crypt/`, 2002.

[51] E. Mann. Private access to distributed information. Master's thesis, Technion - Israel Institute of Technology, 1998.

[52] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[53] MobilityGuru.com. The First Pentium-M Notebook Put To The Test. Online at `http://www.mobilityguru.com/2003/02/05/the_first_pentium/page10.html`.

[54] P. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[55] G. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, April 1965.

[56] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254, New York, NY, USA, 1999. ACM Press.

[57] J. Nielsen. Nielsen's Law of Internet Bandwidth. Online at `http://www.useit.com/alertbox/980405.html`, Apr. 1998.

[58] N. I. of Standards and T. (NIST). The key management guideline. Online at `http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html`, Aug. 2005.

[59] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of EuroCrypt*, 1999.

[60] P. Paillier. A trapdoor permutation equivalent to factoring. In *PKC '99: Proceedings of the Second International Workshop on Practice and Theory in Public Key Cryptography*, pages 219–222, London, UK, 1999. Springer-Verlag.

[61] PCStats.com. Intel Pentium 4 3.2GHz Extreme Edition Processor Review. Online at `http://www.pcstats.com/articleview.cfm?articleid=808`.

[62] PCStats.com. Shuttle XPC SD11G5 Small Formfactor PC Review. Online at `http://www.pcstats.com/ArtRSS.cfm?articleid=1905`.

[63] T. O. Project. OpenSSL: The open source toolkit for SSL/TLS. `www.openssl.org`, April 2003.

[64] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Cambridge, MA, USA, 1979.

[65] Rainer Bluemel and Ralf Laue and Sorin A. Huss. A highly efficient modular Multiplication Algorithm for Finite Field Arithmetic in GF(P). In *Proceedings of ECRYPT Workshop, Cryptographic Advances in Secure Hardware*, 2005.

[66] O. Regev. New lattice based cryptographic constructions. *Journal of the ACM*, 51(6):899–942, 2004.

[67] O. Regev. Lattice-based cryptography. In *Proceedings of Crypto*, 2006.

[68] R. D. Silverman. Exposing the Mythical MIPS Year. *Computer*, 32(8):22–26, 1999.

[69] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Systems Journal*, 40(3):683–695, 2001.

[70] C. E. Spurgeon. *Ethernet: The Definitive Guide*. O'Reilly and Associates, 2000.

[71] J. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *Proceedings of Asia Crypt*, pages 357–371, 1998.

[72] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 2002.

[73] S. Wang, X. Ding, R. Deng, and F. Bao. Private information retrieval using trusted hardware. In *11th European Symposium On Research In Computer Security (ESORICS)*, 2006.