High-Speed Single-Database PIR Implementation

Carlos Aguilar Melchor

XLIM-DMI,
Université de Limoges,
123, av. Albert Thomas
87060 Limoges Cedex, France
carlos.aguilar@xlim.fr

Abstract. In this HotPETs session we would like to present an implementation of a single-database Private Information Retrieval (PIR) scheme that can process a database at 2 Gbits/s using a commodity Graphics Processing Unit (GPU).
This session will have three goals :

- Dispel the idea that single-database PIR schemes are unusable because too expensive from a computational point of view
- Provide a tool to do fast single-database PIR for higher-level applications and tests
- Highlight that "Lattices + GPUs = Huge speedup" compared to number-theory schemes

In order to do this we will first give a quick introduction to single-database PIR schemes and highlight the computational issues. Then after a one slide presentation of how GPUs can be used to do general purpose computations, we will present in a very schematic way the scheme implemented and why it is well adapted to GPUs. Finally, we will present a performance comparison over different database sizes with mean and variance values.
One or two demos are possible if the organizers agree with them.

IMPORTANT NOTE: Our implementation can be donwloaded from
http://www.assembla.com/spaces/pir_gpgpu2008

Project members : Carlos Aguilar Melchor, Benoit Crespin, Philippe Gaborit, Vincent Jolivet and Pierre Rousseau (lead programmer)

High-Speed Single-Database PIR Implementation

IMPORTANT NOTE: Our implementation can be donwloaded from
http://www.assembla.com/spaces/pir_gpgpu2008

Abstract. In this HotPETs session we would like to present an implementation of a single-database Private Information Retrieval (PIR) scheme that can process a database at 2 Gbits/s using a commodity Graphics Processing Unit (GPU).
This session will have three goals :

- Dispel the idea that single-database PIR schemes are unusable because too expensive from a computational point of view
- Provide a tool to do fast single-database PIR for higher-level applications and tests
- Highlight that "Lattices + GPUs = Huge speedup" compared to number-theory schemes

In order to do this we will first give a quick introduction to single-database PIR schemes and highlight the computational issues. Then after a one slide presentation of how GPUs can be used to do general purpose computations, we will present in a very schematic way the scheme implemented and why it is well adapted to GPUs. Finally, we will present a performance comparison over different database sizes with mean and variance values.
One or two demos are possible if the organizers agree with them.

1 Introduction

1.1 PIR schemes

Usually, to retrieve an element from a database, a user will send a request pointing out which element he wants to obtain, and the database will send back the requested element. Which element a user is interested in may be an information he would like to keep secret, even from the database administrators.

A Private Information Retrieval scheme is a protocol in which a user retrieves a record out of n from a replicated database, while hiding from the database which record has been retrieved, as long as the different replicas do not collude (see [1]).

In this session we just consider a specially interesting sub-field of research, called single-database Private Information Retrieval, which deals with the schemes that allow a user to retrieve privately an element of a non-replicated database. In these schemes [2–7], user privacy is related to the intractability of a mathematical problem, instead of being based on the assumption that different replicas exist and do not collude against their users. To lighten our text, PIR will implicitly represent single-database Private Information Retrieval. When dealing with schemes that need the database to be replicated we will explicitly say replicated-database PIR.

In a PIR protocol, a user wanting to retrieve an element of index i from a database, uses a PIR query generation algorithm with input i and sends the resulting query to the database (see Figure 1). The database combines its elements to the query using a reply generation algorithm and obtains a result which is sent back to the user. Finally, the user decodes the answer through a reply decoding algorithm. The protocol is said to be correct if the decoding results is the i-th element of the database, and private if the database is unable to learn anything about i from the query or the reply generated.

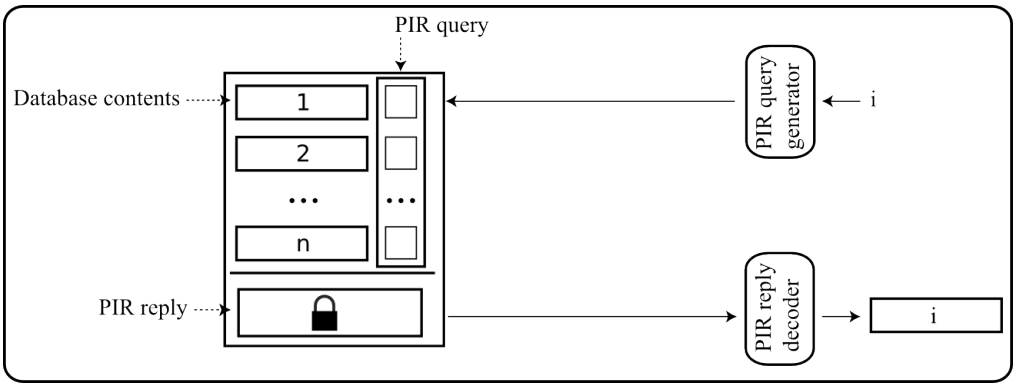

Fig. 1. PIR retrievals.

1.2 Computational cost

Single-database PIR schemes have generated an enormous amount of research in the privacy protection field during the last two decades. However, many scientists believe, that these are theoretical tools unusable in almost any situation. It is true that these schemes usually require the database to use an enormous amount of computational power, but considering the huge amount of applications these protocols have, it is important to develop practical protocols that provide acceptable performances for as many applications as possible.

A major issue with single-database PIR schemes is that they are computationally expensive. Indeed, in order to answer a query, the database must process all of its entries. If in a given protocol it does not process some entries, the database will learn that the user is not interested in them. This would reveal to the database partial information on which entry the user is interested in, and therefore it is not as private as downloading the whole database and retrieving locally the desired entry.

The computational cost for a server replying to a PIR query is therefore linear on the database size. Moreover, **number-theoretic schemes have a very expensive cost per bit in the database: a multiplication over a large modulus**, which usually is 1024 or 2048 bits long. This limits both the database size and the throughput shared by the users, limiting as well their usage for many databases as for other applications such as low-latency unobservable communications [8] or private keyword search [9].

Recent proposals [10, 11] introduce noise-based protocols in which the cost per bit in the database is a vector addition which, depending on the parameters, can be much cheaper than the modular multiplications used in number theory schemes.

2 Our implementation

We present a proof-of-concept implementation of a single-database PIR scheme we proposed in 2007 [11]. A security proof for this scheme will be presented this year at the 2008 IEEE International Symposium of Information Theory.

This implementation can run in a CPU or a GPU using CUDA, nVidia's library for General Purpose computing on Graphics Processing Units (GPGPU). The performance results highlight that

lattice-based PIR schemes allow to process database contents several orders of magnitude faster than previous implementations specially if using nowadays' highly parallelized GPUs.

2.1 Basic description of the scheme

In this section we give an overview of the PIR scheme. Query generation and information extraction from the database reply are not described for two reasons. First, the bottleneck in PIR schemes is reply generation, not query generation or reply extraction. Second, all our implementation efforts and the GPGPU computation are on the reply generation phase. Query generation and reply extraction have been implemented straightforwardly from the scheme description. The formal description of the three phases is available in [11, 12].

We describe a database as a set of n elements. Each element a_i (for $i \in \{1, \cdots, n\}$) is split in ℓ_0-bit sub-elements and represented as a matrix A_i

$$A_i = \begin{bmatrix} a_{i,1,1} & \cdots & a_{i,1,N} \\ \vdots & \vdots & \vdots \\ a_{i,L,1} & \cdots & a_{i,L,N} \end{bmatrix}$$

N and ℓ_0 being two security parameters (which in our implementation we set respectively to 48 and 16). Noting S_{max} the size of the largest element in the database, parameter L is set to $L := \lceil S_{max}/(N \times \ell_0) \rceil$. If a database element is smaller than $L \times N \times \ell_0$ the end of the matrix filled up with a standard padding technique.

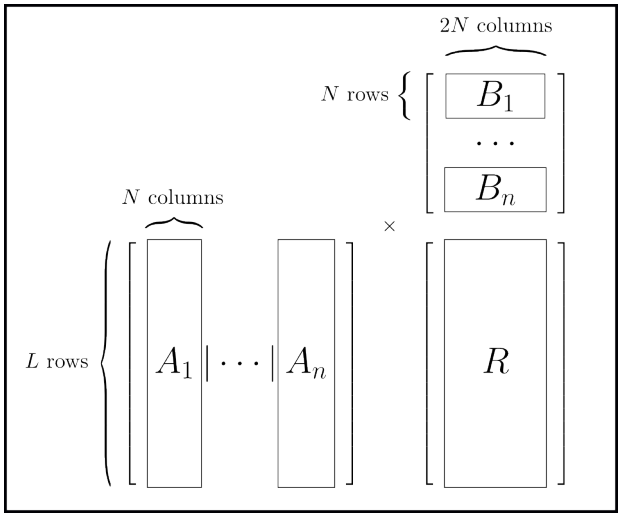

Fig. 2. Scheme overview.

A user wanting to retrieve an element generates a query formed of n matrices $(B_1, \cdots, B_n)$, one for each database element. Each matrix is of dimension $N \times 2N$ with scalars in $\mathbb{Z}/p\mathbb{Z}$, p being

a $3\ell_0$-bit prime. If the user wants to retrieve element a_{i_0}, he generates a query such that the i_0-th matrix has a special property that is invisible to the server. This property ensures that the user will be able to extract from the server reply the desired element.

The user sends the query to the server hosting the database. To generate the reply, the server multiplies the column concatenation of the database element matrices and the row concatenation of the query matrices, as shown in Figure 2.

The resulting matrix, which we note R, is a $L \times 2N$ matrix with $3\ell_0$-bit scalars (in $\mathbb{Z}/p\mathbb{Z}$) and thus, is six times larger than a database element matrix. Query size is $n \times N \times 2N \times 3\ell_0$, which for the given parameters results in $28 \times n$ Kbytes. Using the recursive construction that Kushilevitz and Ostrovsky proposed in the seminal paper on single-database PIR [2], it is possible to lower the query size to $28 \times d \times n^{1/d}$ Kbytes (for d levels of recursion). However recursion must be used carefully as it results in a larger reply expansion factor, which becomes 6^d. Recursion usage does not change much PIR computational performance and thus we have not included it in the current test implementation but should come in its next version.

2.2 Reply generation using GPGPU

The concept of general-purpose computation using graphics processing units (or GPGPU) has arosen in recent years. GPGPU applications make use of graphic processing units (GPU) as massively parallel processors, turned away from their original purpose but nonetheless highly efficient in numerous application domains. Combining both speed and bandwidth due to their parallel architecture (and accessible cost), GPU are an attractive choice for complex computations compared to traditional CPU since they exhibit significant performance overheads, and are supported by constantly improving high-level programming language provided by both GPU vendors and the academic community. GPU have now evolved from highly-specialized graphics processors into powerful, flexible programmable units, and become increasingly popular for a wide range of applications [13, 14].

All the existing PIR schemes are highly parallelizable. However, in the classic schemes, the basic operation for reply generation (per bit on the database) is a multiplication over a 1024 or 2048 bit modulus. The stream processors that form a GPU are not adapted to do directly such operations and trying to do a straightforward parallelization would result in very poor performance. The correct approach is to use the whole set of stream processors to split an exponentiation among them. However, even if this approach has been proven to run 100 times faster than CPU computation for a 190 bit modulus (see [15]), it only runs 2-3 times faster for 1024 bits modulus (see [16]).

The situation is very different for the protocol we have decided to implement. Indeed, the basic operation (per group of ℓ_0 bits in the database) is a multiplication of an $\ell_0 = 16$-bit scalar by a $3 \times \ell_0 = 48$-bit scalar, which can be efficiently encoded through SIMD (Simple Instruction Multiple Data) instructions, on which GPUs are specialized. Each of these operations can be executed inside a single stream processor and the large number of these processors in modern GPUs results in a significative performance improvement.

2.3 Package description

Our implementation (on the client-side) uses NTL [17], the Number Theory Library of Victor Shoup, and is distributed with it.[1]It is possible to compile the server and client with or without CUDA in

[1] The `./install` script provided with the package proposes the user to build and install it if needed.

order to test GPU and CPU or CPU-only performance. All the instructions for installation are given with the package.

We have implemented a basic server that is compiled in the `server` directory during the installation process. The command `PIRServer configfile [port]` runs the server on port `port`. The file `configfile` must contain a set of files (for example a few songs) that the server will use to form the database element matrices $A_1, \cdots, A_n$. Once these matrices are set up, the server waits for a client connexion.

The client is located in the `client` directory and is run with `PIRClient [port]`. When run, the client first asks for the server IP (the default being `127.0.0.1` for an easy test). Then, it connects to the server, and receives a list of the filenames available. The user chooses an index for a file and the client generates a query for this element using NTL. After the query is generated (following the query generation algorithm described in [11]), it is sent to the server.

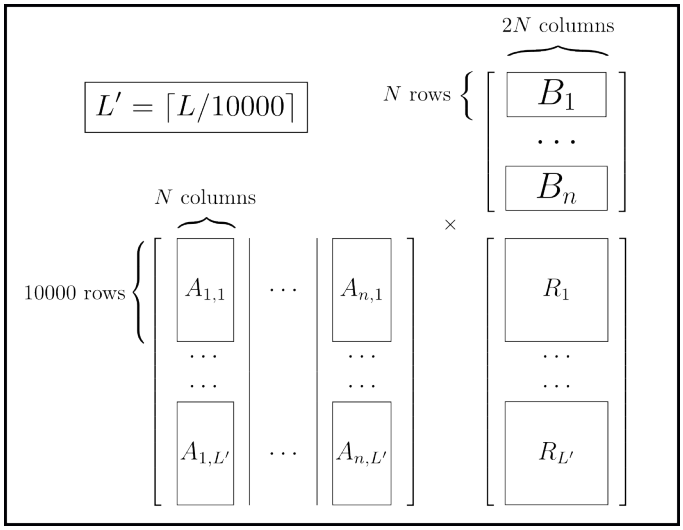

Fig. 3. Block matrix multiplication.

Upon reception of the query, the server begins the reply computation. In the current implementation the server only handles one client at a time. In order to optimize cache usage, we have chosen to use a block matrix multiplication algorithm. The database elements are split in matrix blocks of 10000 rows (see Figure 3) and the server asks iteratively the GPU to compute $A_{i,1} \times B_i$ and sum the results progressively. After n iterations the server obtains R_1 and sends it to the user while restarting the process for each of the following block lines until sending $R_{L'}$.

3 Computational performance

From a computational point of view, reply generation is the limiting factor for single-database PIR. We will therefore not analyze the computational cost for a user to generate a query and to decode a PIR reply.

In this section, we present the performance results between the GPU and the CPU approach for Aguilar and Gaborit's scheme. We also provide a comparison with two number-theory based schemes. Lipmaa's scheme [6] is the outcome of a set of protocols [2, 3, 5] based on homomorphic encryption. Similarly, Gentry and Ramzan's scheme [7] is the outcome of the second known approach to obtain number-theory PIR schemes. Note that other linear algebra based schemes than Aguilar and Gaborit's exist. However, their security has either been broken [18] or require parameters resulting in unrealistic communication costs [10], and thus they have not been included in our performance comparison.

Currently two PIR implementations are known to the authors. The first is a single-database PIR implementation [19] that it is not at all focused on performance optimization and thus, no comparison is done with it. The second is a replicated-database PIR implementation presented at IEEE Security & Privacy 2007 [20]. Computational issues in replicated-database PIR schemes are very different from the ones in single-database schemes. Indeed, even the seminal replicated-database PIR schemes presented Chor et al. in [1] were optimal from a computational cost point of view (one bit operation per bit on the database). Computational complexity derives from other issues like trying to add robustness properties to the distributed protocol, which has nothing to do with our work. We therefore not compare our single-database PIR results with the implementation proposed in [20] for robust replicated-database PIR schemes.

3.1 Experimental setup

We have not implemented Lipmaa's and Gentry and Ramzan's schemes. On the other hand, a pretty good performance estimation can be done. Indeed, in these schemes, the server's computational cost is just a large set of standard repetitive operations (a 1024 or 2048 bit modular multiplication per bit in the database). It is therefore very easy to make an analytical approximation of this cost. Moreover, standard utilities, such as the openssl library, provide implementations of these operations optimized for many years by the research community. These utilities allow to transform out analytical results into practical figures with pretty good confidence that the result will be very close to the one of a real implementation. We have set all the parameters in these protocols to optimize computational performance. The results in our figure may decrease if these parameters vary.

We have run the tests over six different systems: three systems for the CPU tests and three for the GPU tests. The three processors used for CPU tests correspond to:

- System 1: an Athlon™ 5000+ (2GHz, 1Mbyte of cache)
- System 2: an Intel® dual-core Xeon™ 5160 (3GHz, 4Mbytes of cache)
- System 3: an Intel® quad-core Xeon™ 5345 (2.33GHz, 4Mbytes of cache).

All the GPU tests have been done on a machine with an Athlon™ 5000+, using the following graphics cards:

- System 1: a ASUS® GeForce™ 8600 GTS (32 stream processors 540MHz)

– System 2: an MSI® GeForce™ 8800 Ultra (128 stream processors 660MHz)
– System 3: two SLI enabled GeForce™ 8800 GTX graphics cards

The CPUs and GPUs used in the corresponding systems have similar prices (namely 150$, 500$ and 1100$).

Each test has been run twenty times. We present only the mean value, the deviance being negligible. Of course, this is the expected behaviour defined by the central limit theorem when running millions of times a repetitive task (in this case executing a kernel in a stream processor). Each file has been set to exactly 3 Mbytes, and the number of files to twelve.

Processing throughput is almost invariant on the number of files or on their size, as long as the whole database can be cached into RAM. If not, there is a large performance drop, which comes from the fact that we have not pipelined disk reads and computation. Indeed, the server computes the whole answer for the data in the RAM and when finished stops computing and caches again another block of data to the RAM before restarting computation. Pipelining and many other improvements as recursion have not been implemented as this software is just meant to be a proof-of concept and a very basic test tool for the moment.

3.2 Results

Figure 4 presents the throughput at which the database is processed when the server generates the reply.

Scheme	Processing throughput		
	System 1	System 2	System 3
Lipmaa[1]	160 Kbits/s	280 Kbits/s	490 Kbits/s
Gentry and Ramzan[1]	490 Kbits/s	890 Kbits/s	1500 Kbit/s
Aguilar and Gaborit (CPU)[2]	33 Mbits/s	130 Mbits/s	230 Mbits/s
Aguilar and Gaborit (GPU)[2]	270 Mbits/s	1.2 Gbits/s	2 Gbits/s

[1] Estimation using `openssl speed rsa` to evaluate analytical complexity (see Section 3.1).
[2] Experimental results using our implementation over 12 files of 3 Mbytes each.

Fig. 4. Computation performance comparison.

With any of the presented PIR schemes, when processing an n-element database, only one n-th of the computation processes the element the user wants to get. Thus, in order to obtain the throughput at which a user gets the element (after decoding the reply data), the results in Figure 4 must be divided by n. For example if the user retrieves a song from a 1000 files databases, the reply decoding process will output the file at 270Kbits/s for the cheapest GPU tested.

There is a large leap between the performance of the linear algebra scheme and number theory schemes, namely between a factor 100 using the CPU implementation and a factor 1000 with the

GPU implementation. Of course this result has an important impact in PIR usability. Indeed a server will be able to use such a PIR scheme over larger databases, handling more users, and providing more throughput.

This performance leap is critical. Indeed, in [21], Sion and Carbunar show that in a large variety of situations number theory schemes are much slower than the trivial solution to the PIR problem (i.e. downloading the whole database). The speedup of the trivial solution is a factor between 10 and 1000 (depending on whether the communications are done through a home connection, or a high-speed LAN). The performance improvement that is brought by linear algebra schemes inverses this factor. Thus, using a linear algebra scheme is the only way to be faster (and much more communication efficient) than the trivial solution in the situations described by Sion and Carbunar.

4 Conclusion

We have presented a proof-of-concept implementation of a linear algebra PIR scheme which can process over one Gbit/s of data with commodity graphics cards. This implementation allows also to process 230 Mbits/s of data with a high-end CPU.

Using the `openssl` library to test the performance attainable with existing schemes we have proved that the linear algebra scheme is between 100 (for the CPU implementation) and 1000 (for the GPU implementation) times faster than number theory schemes. We have noted the importance of this improvement, specially considering the previous work on PIR usability by Sion and Carbunar.

As Aguilar and Gaborit note in [11], their scheme result in larger communication costs, but with current bandwidths and through the usage of recursion it remains usable in many situations. On the other side, their security assumptions are pretty new and thus, the security of the system may be easier to break than for other schemes. In any case, the results we provide are true from a general point of view for linear algebra schemes. Indeed, these schemes can be really fast, specially if implemented over a GPU, and other schemes based on linear algebra should be carefully considered (as for example the proposal of Gasarch and Yerukhimovich [10]).

As a consequence of the good performance results we would like to polish and complete our implementation to transform it in a robust usable library for other users. We would also like to use other hardware improvements that GPUs can provide but that we have been unable to test for the moment. Finally we would like to implement an interface to common database languages such a MySQL, etc.

We hope that the provided results will motivate further research in linear algebra PIR schemes, and more generally in the privacy domain.

References

1. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. In: 46th IEEE Symposium on Foundations of Computer Science (FOCS'95), Pittsburgh, PA, USA, IEEE Computer Society Press (1995) 41–50
2. Kushilevitz, E., Ostrovsky, R.: Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval (extended abstract). In: FOCS: IEEE Symposium on Foundations of Computer Science (FOCS). (1997) 364–373

3. Stern, J.P.: A New Efficient All-Or-Nothing Disclosure of Secrets Protocol. In: 13th Annual International Conference on the Theory and Application of Cryptology & Information Security (ASIACRYPT'98), Beijing, China. Volume 1514 of Lecture Notes in Computer Science., Springer (1998) 357–371
4. Cachin, C., Micali, S., Stadler, M.: Computationally Private Information Retrieval with Polylogarithmic Communication. In: 18th Annual Eurocrypt Conference (EUROCRYPT'99), Prague, Czech Republic. Volume 1592 of Lecture Notes in Computer Science., Springer (1999) 402–414
5. Chang, Y.C.: Single Database Private Information Retrieval with Logarithmic Communication. In: Information Security and Privacy: 9th Australasian Conference (ACISP'04), Sydney, Australia. Volume 3108 of Lecture Notes in Computer Science., Springer (2004) 50–61
6. Lipmaa, H.: An Oblivious Transfer Protocol with Log-Squared Communication. In: 8th Information Security Conference (ISC'05), Singapore. Volume 3650 of Lecture Notes in Computer Science., Springer (2005) 314–328
7. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: 32nd Annual International Colloquium on Automata, Languages and Programming (ICALP'05), Lisbon, Portugal. Volume 3580 of Lecture Notes in Computer Science., Springer (2005) 803–815
8. Aguilar Melchor, C., Deswarte, Y., Iguchi-Cartigny, J.: Closed-Circuit Unobservable Voice Over IP. In: 23rd Annual Computer Security Applications Conference (ACSAC'07), Miami, FL, USA, IEEE Computer Society Press (2007)
9. Ostrovsky, R., E. Skeith III, W.: Private Searching on Streaming Data. In: Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 223–240
10. Gasarch, W., Yerukhimovich, A.: Computational inexpensive PIR (2006) Draft available online at http://www.cs.umd.edu/ arkady/pir/pirComp.pdf.
11. Aguilar Melchor, C., Gaborit, P.: A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol. In: Western European Workshop on Research in Cryptology (WEWoRC'2007), Bochum, Germany. Book of Abstracts. (2007) 50–54, Extended version available on IACR eprints `http://eprint.iacr.org/2007/446`
12. Aguilar Melchor, C., Gaborit, P.: A Fast Private Information Retrieval Protocol. In: The 2008 IEEE International Symposium on Information Theory (ISIT'08), Toronto, Ontario, Canada, IEEE Computer Society Press (2008) To appear
13. Houston, M., Govindaraju, N., eds.: GPGPU: general-purpose computation on graphics hardware. ACM Press (2007)
14. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Computer Graphics Forum **26**(1) (2007) 80–113
15. Fleissner, S.: GPU-accelerated montgomery exponentiation. In: 7th International Conference on Computational Science (ICCS'07), Beijing, China. Volume 4487 of Lecture Notes in Computer Science., Springer (2007) 213–220
16. Moss, A., Page, D., Smart, N.: Toward acceleration of rsa using 3d graphics hardware. In: Cryptography and Coding. Volume 4887 of Lecute Notes in Computer Science., Springer (2007) 369–388
17. Shoup, V.: NTL: A library for doing Number Theory. http://www.shoup.net/ntl/ (2007)
18. Kiayias, A., Yung, M.: Secure Games with Polynomial Expressions. In: ICALP: Annual International Colloquium on Automata, Languages and Programming. (2001)
19. Saint-Jean, F.: A Java Implementation of a Single-Database Computationally Symmetric Private Information Retrieval (cSPIR) protocol. Technical Report 1333, Yale University (2006)
20. Ian Goldberg: Improving the Robustness of Private Information Retrieval. In: The 2007 IEEE Symposium on Security and Privacy (S&P'07), Oackland, CA, USA, IEEE Computer Society Press (2007) 131–148
21. Sion, R., Carbunar, B.: On the Computational Practicality of Private Information Retrieval. In: 14th ISOC Network and Distributed Systems Security Symposium (NDSS'07), San Diego, CA, USA. (2007)