Almost Optimal Private Information Retrieval

Dmitri Asonov* and Johann-Christoph Freytag

Humboldt-Universität zu Berlin,
10099 Berlin, Germany
{asonov, freytag}@dbis.informatik.hu-berlin.de

Abstract. A private information retrieval (PIR) protocol allows a user to retrieve one of N records from a database while hiding the identity of the record from the database server.
With the initially proposed PIR protocols to process a query, the server has to process the entire database, resulting in an unacceptable response time for large databases. Later solutions make use of some preprocessing and off-line communication, such that only $O(1)$ on-line computation and communication are performed to execute a query.
The major drawback of these solutions is off-line communication, comparable to the size of the entire database.
Using a secure coprocessor we construct a PIR scheme with $O(1)$ on-line computation and communication, periodical off-line preprocessing, and zero off-line communication.
The protocol is almost optimal. The only parameter left to improve is the server's off-line preprocessing complexity - the least important one.

Keywords: Efficient realization of privacy services.

1 Introduction

A private information retrieval (PIR) protocol allows a user to retrieve one of N records from a database while hiding the identity of the record from a database server. That is, with a PIR protocol, the client can perform his query "return me the i-th record" in such a way that no one (including the server) receives any information about i.

Many practical e-commerce applications could benefit from using PIR to address user privacy [Aso01]. An obvious, common application is trading digital goods. Using PIR, a client may retrieve a selected subject (a digital article, an e-book, or a music file etc.) privately. "Privately" means that a digital good is retrieved such that no one except the client observes the identity of the good. At the same time the retrieval is controlled by the server, so billing can be performed as well.

Naturally, the quality of every PIR protocol is measured by the two following parameters: the complexity of the computation to perform one query, and the

* This research was supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316.)

complexity of the communication done between the client and the server to execute one query.

1.1 Motivation

Initially developed PIR protocols lack scalability dramatically, making it impossible to use them in the real world. In order to process a PIR query to retrieve a single record, the server must perform complex computations with each record of the entire database.

Several attempts were made to address this problem. Two papers present the state of the art [BDF00,SJ00]. PIR protocols by Bao et. al. and Schnorr et. al., being developed independently, present very similar ideas to address the problem, although they introduce a new one. Namely, off-line communication, comparable to the size of the entire database, must be performed between the client and the server before these protocols start. (Sect. 2.3 discusses these protocols in details.)

1.2 Our Results

We present a protocol that addresses the problem of constructing a PIR protocol with O(1) answer time and on-line communication, and no off-line communication. For our protocol, $O(1)$ records have to be processed on-line in order to answer a query. But, in contrast to [BDF00,SJ00], the protocol eliminates the off-line communication completely. Our protocol is almost optimal in the sense that the only parameter left to be optimized is the server's preprocessing complexity - the least critical one.

1.3 Preliminaries and Assumptions

In the following, N denotes the number of records in the database. The only type of query considered is "return me the i-th record", $1 \leqslant i \leqslant N$.

As in [SS00,SS01,BDF00,SJ00], we omit the precise mathematical definition of privacy while presenting the protocol. The definition "no information about queries is revealed" is enough. However, we introduce a formal definition of privacy later in this paper in order to formally prove that the protocol fulfills the privacy property.

We say that a PIR protocol has $O(A)$ communication complexity and $O(B)$ computation complexity if only $O(A)$ records must be communicated between the server and client, and only $O(B)$ records must be processed by the server (in oder to answer one query). For example, we say that computation complexity is $O(1)$ if the number of records, that has to be processed by the server to answer a query, is independent from N.

Note that our measure of the communication complexity does not include the size of the query itself. This is because the query size for most PIR protocols is much less then the size of an average record in the applications considered. For example, let us take a theoretical limit for a query size: lgN bits. It is practical to say that lgN bits are much less than the record size (5Mb in case of an mp3 file).

1.4 Structure of the Paper

Having analyzed the related work in the next section, we present our basic protocol in Sect. 3. Further details on the protocol and discussion are presented in Sect. 4. We finish the paper with ideas about future work.

Furthermore, Appendix A formalizes the protocol; we introduce a formal definition of privacy based on information theory in Appendix B.1. Based on the formal description of the protocol and the definition of privacy we prove that the protocol is private in Appendix B.2. Finally, a short introduction to information theory is given in Appendix C.

2 Related Work

The PIR problem was first formulated by Chor et al. [CGKS95]. From the very beginning two fundamental limitations became clear:

1. PIR is impossible, unless we consider sending the entire database to the client as a solution. That is, the communication complexity of any PIR protocol to perform one query is proven to be $\Omega(N)$. [1]
2. In order for any PIR protocol to answer one query, the entire database must be read. This conclusion is based on the following simple observation: Independently from how a PIR protocol works, if the server does not read some of the database records while answering a query, then the (malicious) server may observe the records that the client did not request. This is a privacy violation by definition.

While the first limitation affects the first parameter of a PIR protocol - the communication complexity, the second limitation affects the second parameter of a PIR protocol - the computation complexity (or, response time) of the server.

The following three sections show the efforts made to overcome these limitations. After each description we summarize the pros and cons of each protocol. Finally, jumping ahead over the description of our protocol (for the convenience of the presentation), we give a comparison of the state of the art with our protocol in Sect. 2.4.

2.1 Computational PIR

Although PIR with communication complexity less than $\Omega(N)$ is impossible theoretically, it is found to be possible if computational cryptography is used [KO97,CMS99,KY01].

[1] There is also a modification of the problem setting (called multi-server PIR), where several servers hold copies of the database. A communication complexity better than $\Omega(N)$ may be achieved under the assumption that the servers do not communicate to each other [CGKS95,CG97,Amb97,BI01]. The idea is to send different queries to different servers, so that i is not derivable from any single of them. But having the all answers gathered, the client can derive the i-th record. In this paper we do not consider the schema based on several servers non-communicating to each other.

The underlying idea is to rely on some intractability assumptions (the hardness of deciding quadratic residuosity, in case of [KO97]). Then, a protocol works as follows. The client encrypts a query "return me the i-th record" in such a way, that the server still can process it using special algorithms and the entire database as an input. However, under an intractability assumption, the server recognizes neither the clear-text query nor the result. The result can be decrypted by the client only.

Pros and Cons. Computational PIR protocols break through the first limitation; [KO97] provides polynomial communication complexity ($O(N^c)$, for any given $c < 1$), improved by polylogarighmic communication complexity in [CMS99,KY01].

Still, the second limitation works for such protocols: the server has to process each record of the entire database to answer one query. Although these protocols are beautiful research jobs from the viewpoint of mathematics, $O(N)$ computation complexity makes them practically infeasible even for small databases [BDF00].

2.2 Hardware-based PIR

Smith et al. [SS00,SS01] make use of a tamper-proof device to implement the following PIR protocol.

The idea is to use a secure coprocessor (a tamper-proof device) as a black box, where the selection of the requested record takes place. Although hosted at the server side, the secure coprocessor (SC) is designed so that it prevents anybody from accessing its memory from outside [SPW98].

The basic protocol runs as shown in Figure 1. The client encrypts the query "return me the i-th record" with a public key of the SC, and sends it to the server. The SC receives the encrypted query, decrypts it, reads through the entire database, but leaves in memory the requested record only. The protocol is finished after the SC encrypts the record and sends it to the client.

To provide integrity, the SC keeps all records of the database encrypted. We discuss this in details in Sect. 4.3.

Pros and Cons. This PIR protocol improves the computation complexity. In comparison to computational PIR protocols, ordinary decryption and encryption have to be made with each of the N records to process a query.

The main disadvantage of this PIR is the same as that of the computational PIR protocols: the second limitation, e.i. , $O(N)$ computation complexity.

2.3 PIR with Preprocessing and Off-line Communication

Although it does not seem feasible to break through the second limitation - $O(N)$ computation, one could try to move off-line as much work as possible using preprocessing. Such that, when a query is submitted, it would cost only $O(1)$ computation to answer it on-line.[2]

[2] As already explained above, we do not consider here approaches oriented for a setting with several servers non-communicating to each other [BIM00,CIO98,GGM98].

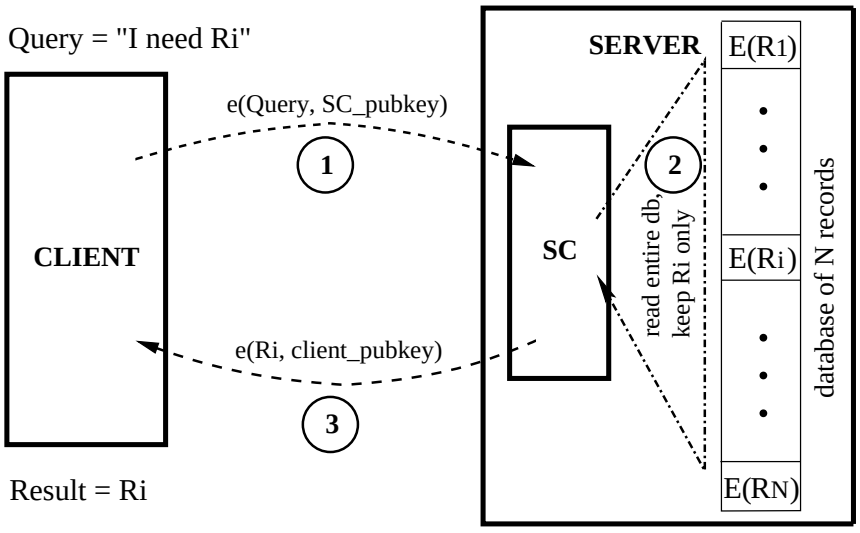

Fig. 1. An example of a PIR protocol with SC.

With this idea in mind, [BDF00,SJ00] present independently very similar PIR protocols. Both utilize a homomorphic encryption, which is used by the server to encrypt off-line every record of the database. All these encrypted records are sent (off-line) to the client. This communication has to be done only once between the client and the server before the PIR protocol starts, independently from how many PIR queries will be processed on-line.

If the client wants to buy a record, he selects the appropriate (locally stored) encrypted record and re-encrypts it. Then, the client sends it to the server and asks to remove the server's encryption. The server is able to do it because of the homomorphic property of the encryption. The server removes its encryption, but cannot identify the record because of the client's encryption. He sends it back to the client. The client removes his encryption. The protocol is done. Figure 2 demonstrates every step of the protocol.

Pros and Cons. The protocols with preprocessing and off-line communication overcome the second limitation: Only $O(1)$ computation is required on-line to answer one query, i.e., these protocols ensure a practical response time.

However, the protocols suffer from another drawback: This is off-line communication comparable to the size of the entire database, that makes their practical applicability questionable. (Imagine a client decides to buy a single digital book or a music file at some digital store. He will probably react negatively after being asked to download the entire encrypted content of the digital store in oder to proceed. Another problem is keeping the client's database copy updated.)

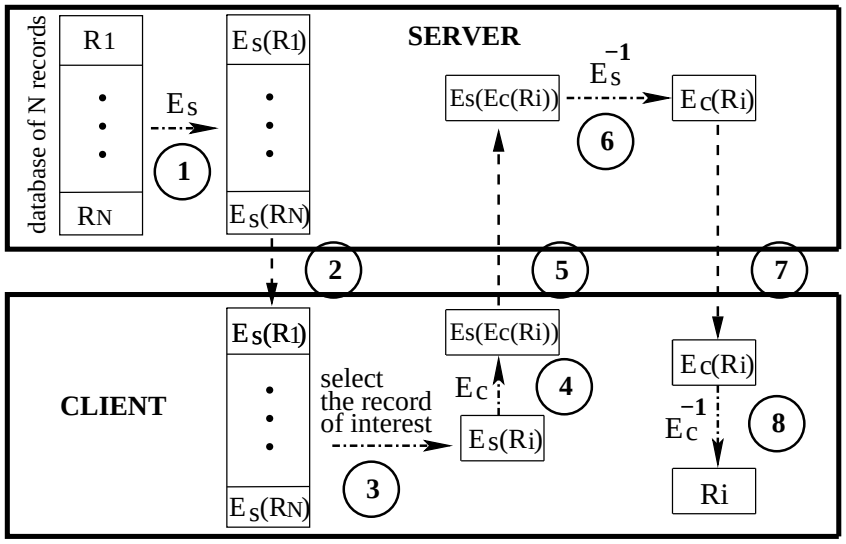

Fig. 2. An example of a PIR protocol with preprocessing and off-line communication. Steps 1 and 2 are made off-line once, and the other steps are performed on-line for every query submission.

2.4 State of The Art and Our Results

In Table 1 we summarize the existing PIR protocols and compare them with the proposed PIR protocol.

Table 1. Comparative analysis of the proposed protocol.

Parameter	**PIR Protocol**			
	Computational [CMS99,KY01]	With SC [SS00,SS01]	With Preprocessing [BDF00,SJ00]	The Proposed (With SC)
On-line communication	assimpt. optimal	optimal	optimal	optimal
Computation	O(N)	O(N)	O(1)	O(1)
Off-line comm.	no	no	O(N)	no
Preprocessing	no	yes	yes	yes

In summary, protocols with preprocessing [BDF00,SJ00] are the most effective in terms of on-line computation and on-line communication complexity. Our PIR protocol retains these parameters, but it does not require off-line communication in comparison to [BDF00,SJ00].

3 The Basic Protocol

We start with the same basic model as described in Sect. 2.2. But in addition, before starting the PIR protocol, the SC shuffles the records off-line. That is, the SC computes a random permutation of the records, and stores this permutation in an encrypted form. Now, the server has no evidence of which record is which.

After the client sends his query "return me the i-th record", the SC does not need to read the entire database anymore. Instead, the SC accesses the desired encrypted record directly. Then the encrypted record is decrypted inside the SC, encrypted with the user public key, and sent to the user. To answer this query, $O(1)$ computation and communication is made on-line by the server.

To answer a second query, the SC reads the previously accessed record first, then the desired record. If the previously accessed record is not read by the SC, then the privacy of the second query could obviously be broken. [3] In case the second query requests the same record as the first query, the SC chooses some random record to be read.

So, to answer the k-th query, the SC has to read the $k-1$ previously read records first. Then the SC reads the desired record. Evidently, the SC has to keep track of the accessed records.

It is up to the server to decide at which $m = max(k)$ $(1 \leqslant m \leqslant N)$ to stop and to switch to another preprocessed (shuffled) copy of the database, so that k would equal one again. Since m is a constant independent of N, we can say that the server has to perform $O(1)$ on-line computations (and read operations) to answer each query.

Now that the basic idea has been introduced, we go into details of our protocol in the next section.

4 The Details

We describe the shuffling algorithm in Sect. 4.1. A problem related to the indexing of the encrypted database is discussed in Sect. 4.2. A hypothetical attack is considered in Sect. 4.3. We demonstrate a trade-off between off-line and on-line computation in our protocol and discuss how to choose the optimal trade-off in Sect. 4.4 and 4.5 respectively. Finally, we consider shortly the cases with multiple queries and multiple secure coprocessors.

4.1 Shuffling Algorithm

The purpose of a shuffling algorithm is to get a random permutation of records ([Knu81], Sect. 3.4.2). However, the specificity of our shuffling algorithm is in

[3] Assume that the server issued the first query itself. Then he observed which record was read by the SC, and he got back the original record as an answer to his query. So he knows the identity of the one encrypted record. Now, the SC reads another encrypted record to answer some client's query. The server can observe that the client is interested in the record different to the record that the server requested before. This is a privacy violation.

that it should be done by the SC in a way, that does not reveal the resulting permutation of the encrypted records to anyone. In our context we focus on building this specific algorithm, omitting how the permutation vector itself is obtained inside the SC.

The SC works as follows. The SC invokes (off-line) N times a PIR protocol similar to the one described in Sect. 2.2. With each of these invocation it privately reads one record (which is chosen accordingly with the permutation vector). It then encrypts and writes this record to a new database. As a result of this shuffling algorithm, the SC generates a database of encrypted and randomly permutated records.

Having stored an encrypted index to address these records[4], the SC can now access any encrypted record directly, while not revealing the identity of the accessed record.

The shuffling algorithm could be run off-line any given number of times beforehand to produce several shuffled databases. The only limitation for a preprocessing algorithm is the size of additional storage available to the server. Therefore in Sect. 4.4 we define an off-line load parameter as an average amount of additional storage (per query) used during preprocessing.

4.2 Indexing

The SC has to maintain some sort of index, in order to know which record is which. In case the index is stored outside the SC, it should be encrypted. Also, the SC has to read the entire index (in addition to $O(1)$ records) in order to answer one query. However, since the records are large, reading the index would take much less time then reading one record.

Example 1 (Comparing the sizes of the index and a record). We use an example similar to one given in [SS01]. Assume that the size of a record is $\mathcal{S}_{record} = 5Mb$, there are $N = 10000$ records, and $i = 10$ bytes, which is enough to address a record. Then, the size of an index I may be estimated as

$$I = N * i = 100\text{Kbytes};\ I \ll \mathcal{S}_{record}. \tag{1}$$

□

So reading the index does not influence the on-line workload much in comparison to reading a record on-line. Therefore, to estimate on-line computation work, we only consider a number of records that must be read to answer a query.

The difference between times to read an index and to read a record even grows, if the entire index fits in the SC's internal memory which is a quite reasonable assumption for commercially available secure coprocessors [DLP+01].

[4] The stored (in an encrypted form) permutation vector may serve as a simple index. We also discuss indexing in Sect. 4.2.

4.3 Active Attacks

In [SS00] an attack is considered, where the malicious server destroys or modifies an arbitrary record before the PIR protocol starts. If the client complains after the PIR query is performed, the malicious server concludes that the client was interested in the modified record, thus breaking the privacy of the client. The solution proposed is to check the granularity of every record in the database (while reading the entire database through) for every query. If a record with the broken granularity appears, the SC aborts PIR protocol, independently from whether the forged record is requested by the client or not [SS00]. In order to provide the granularity control, each record is stored in an encrypted form.

The malicious server might try the same attack within our PIR protocol. In this case, the SC does not have to check the integrity of each record in the database to process one query. It is enough to check the granularity of the requested encrypted record only.

4.4 Trade-Off Between Preprocessing and On-Line Computation

In our protocol, it is possible to balance the workload between the on-line and off-line phases. Decreasing the amount of on-line work increases the off-line work and vice versa. Let m ($1 \leqslant m \leqslant N$) be a maximal number of records allowed to be read on-line to answer a single query, as explained in Sect. 3. Obviously, m is a trade-off parameter. Reducing m will decrease the on-line computation (and, consequently the response time of the server), but will increase the amount of the off-line preprocessing.

Let $\mathbf{r}_{on-line}$ be the average number of encrypted records that the SC reads on-line to answer a query. This parameter characterizes the average response time of the server. Let $\mathbf{w}_{off-line}$ be the average number of encrypted records that the SC writes off-line (during the preprocessing stage) in order to be prepared to answer one query. This parameter characterizes the average amount of additional storage used by the SC for answering one query. Our equations below show both parameters expressed using the trade-off parameter.

$$\mathbf{r}_{on-line} = \frac{1+2+3+...+m}{m} = \frac{m*(m+1)}{2*m} = \frac{m+1}{2} \tag{2}$$

$$\mathbf{w}_{off-line} = \frac{N}{m} \tag{3}$$

The dependencies between the trade-of parameter m, the on-line work $\mathbf{r}_{on-line}$, and the preprocessing parameter $\mathbf{w}_{off-line}$ are shown in Figure 3 (for $N = 10000$). From equations 2 and 3 we derive the dependence between the on-line ($\mathbf{r}_{on-line}$) and off-line ($\mathbf{w}_{off-line}$) parameters of the protocol.

$$\mathbf{r}_{on-line} = \frac{N}{2*\mathbf{w}_{off-line}} + 1, \quad \mathbf{r}_{on-line} = \Theta\left(\frac{1}{\mathbf{w}_{off-line}}\right) \tag{4}$$

The last equation exhibits that each reduction of the response time by an order leads to a blow up in preprocessing work by an order.

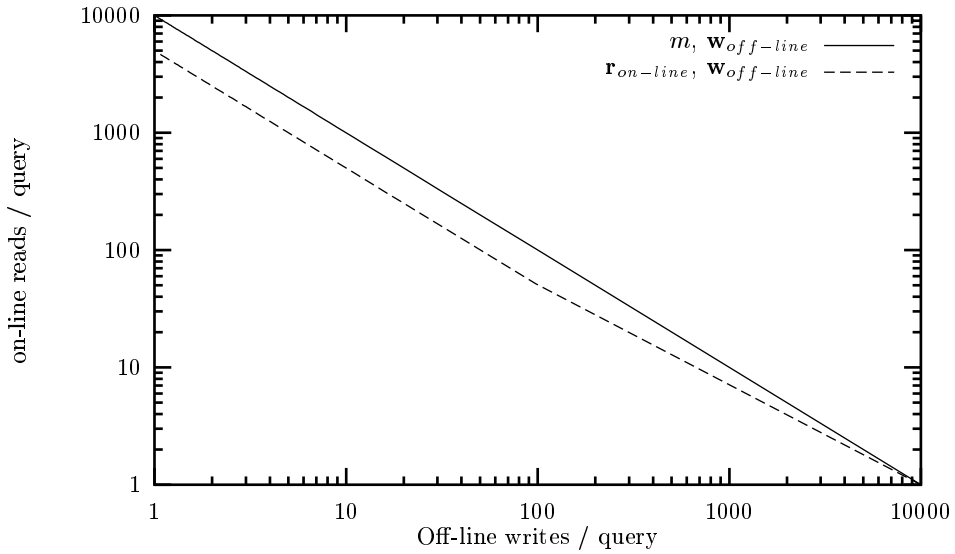

Fig. 3. The dependence between on-line performance (max and average number of records to read on-line per query) and preprocessing load (number of off-line write operations per query).

4.5 Choosing the Optimal Trade-Off

Using the required response time of the server one could determine the trade-off parameter m. That is, if the maximal allowed response time is fixed, choosing the trade-off parameter is a straightforward task.

Another strategy for choosing the trade-off parameter might be minimizing the overall work $S(m)$, defined as the sum of the normalized on-line and off-line work parameters.

We show in Figure 4 that the overall work $S(m)$ does not remain constant while varying trade-off parameter. To determine the optimal trade-off parameter we must find the minimum of the following function:

$$S(m) = \mathbf{r}_{on-line} * k_{norm} + \mathbf{w}_{off-line} \tag{5}$$

where k_{norm} is the normalization coefficient used to normalize the two parameters.

We resolve the optimal trade-off by finding the roots of the derivative of $S(m)$:

$$S'(m) = \left(\frac{(m+1) * k_{norm}}{2} + \frac{N}{m}\right)' = \frac{k_{norm}}{2} - \frac{N}{m^2} \tag{6}$$

$$\frac{k_{norm}}{2} - \frac{N}{m_{opt}^2} = 0, \qquad m_{opt} = \left\lceil \sqrt{\frac{2 * N}{k_{norm}}} \right\rceil \tag{7}$$

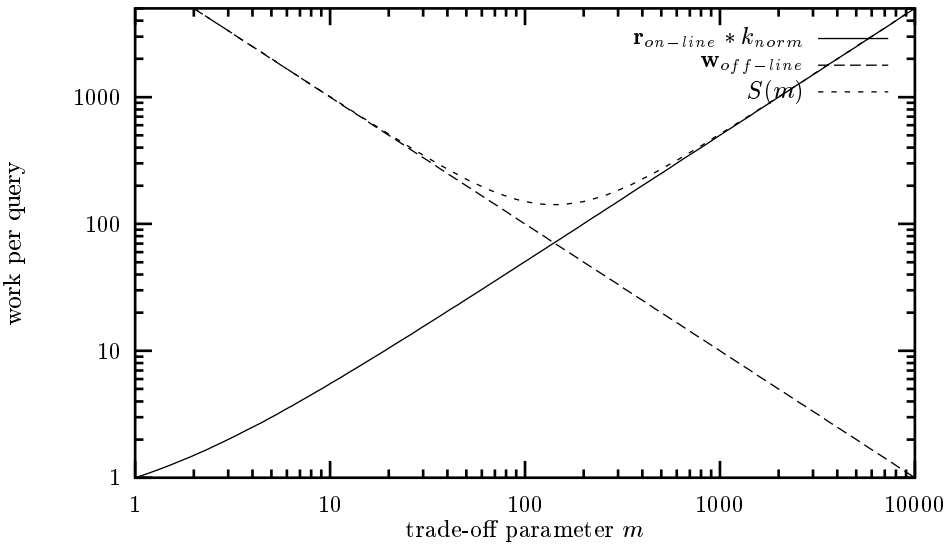

Fig. 4. The overall work done per query (calculated as a sum of normalized on-line and off-line parameters) is not constant for different values of the trade-off parameter.

For example, if $k_{norm} = 1$ (reading one record on-line is considered equal to writing and storing one record off-line) and $N = 10000$, then the optimal trade-off parameter is $m_{opt} = [\sqrt{2 * N}] = 141$.

4.6 Multiple Queries and Multiple Coprocessors

Multi-query optimization may be advantageous for our protocol. When several queries arrive, the SC may read previously accessed records only once, thus eliminating the need to perform this operation for every query.

Shifting from a single SC to multiple SCs is not a trivial task for the PIR scheme in [SS00]. For our scheme, distributing the work between several SCs is obvious. For example, due to a small on-line workload, one SC might be dedicated to answering queries; and the rest secure coprocessors can do the preprocessing work, i.e. preparing several shuffled copies of the database. Such a simple parallelization is possible because on-line and preprocessing algorithms are practically independent.

5 Future Work

We do not discuss key management issues that might arise in our scheme. Thus prototyping the protocol is an interesting future task.

Another open question is whether the preprocessing complexity of our protocol is optimal or not.

6 Conclusion

Private Information Retrieval (PIR) can solve privacy issues in many practical e-commerce applications by enabling the user to retrieve a record of his choice from the database in a way, that no one, not even the database server, observes the identity of the record.

The existing PIR protocols either incur intolerable query response time (linear in the size of the database) or introduce off-line communication of the size of the entire database between the user and the server. Thus the applicability of both types of protocols is questionable from a practical point of view.

We presented a new PIR protocol with preprocessing that has $O(1)$ response time, optimal on-line communication complexity, and does not require off-line communication. This property is due to a new preprocessing algorithm based on shuffling and due to the usage of a secure coprocessor.

We showed the trade-off between the on-line and preprocessing workloads for the protocol. The protocol is scalable for multiple queries and multiple secure coprocessors.

References

[Amb97] A. Ambainis. Upper bound on the communication complexity of private information retrieval. In *Proceedings of 24th ICALP*, 1997.

[Aso01] D. Asonov. Private information retrieval - an overview and current trends. In *Proceedings of the ECDPvA Workshop, Informatik 2001, Vienna, Austria*, September 2001.

[BDF00] F. Bao, R. H. Deng, and P. Feng. An efficient and practical scheme for privacy protection in the e-commerce of digital goods. In *Proceedings of the 3rd International Conference on Information Security and Cryptology*, December 2000.

[BI01] A. Beimel and Y. Ishai. Information-theoretic private information retrieval: A unified construction. ECCC Report TR01-015, February 2001.

[BIM00] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *Proceedings of CRYPTO'00*, 2000.

[CG97] B. Chor and N. Gilboa. Computationally private information retrieval. In *Proceedings of 29th STOC*, 1997.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of 36th FOCS*, 1995.

[CIO98] G. D. Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for database private information retrieval. In *Proceedings of 17th PODC*, 1998.

[CMS99] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Proceedings of EUROCRYPT'99*, 1999.

[DLP+01] J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the ibm 4758 secure coprocessor. *IEEE Computer*, 34(10):57–66, October 2001.

[GGM98] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In *Proceedings of 2nd RANDOM*, 1998.

[Jay94] E. T. Jaynes. *Probability theory: the logic of science*. `http://omega.math.albany.edu:8008/JaynesBook.html`, 1994.

[Knu81] D. E. Knuth. *The art of computer programming*, volume 2. Addison-Wesley, second edition, Jan 1981.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: Single-database computationally private information retrieval. In *Proceedings of 38th FOCS*, 1997.

[KY01] A. Kiayias and M. Yung. Secure games with polynomial expressions. In *Proceedings of 28th ICALP*, 2001.

[Mac00] D. J. MacKay. *Textbook on Information Theory*. `http://wol.ra.phy.cam.ac.uk/mackay/Book.html`, 2000.

[Sch96] B. Schneier. *Applied Cryptography*. Wiley, New York, 2nd edition, 1996.

[Sha48] Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27, 1948.

[SJ00] C. P. Schnorr and M. Jakobsson. Security of signed elgamal encryption. In *Proceedings of ASIACRYPT'00, LNCS 1976*, December 2000.

[SPW98] S. W. Smith, E. R. Palmer, and S. H. Weingart. Using a high-performance, programmable secure coprocessor. In *Proceedings of the 2nd International Conference on Financial Cryptography, Springer-Verlag LNCS*, February 1998.

[SS00] S. W. Smith and D. Safford. Practical private information retrieval with secure coprocessors. Technical report, IBM Research Division, T.J. Watson Research Center, July 2000.

[SS01] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Systems Journal*, 40(3), September 2001.

A The Formal Protocol

This part of the paper formally presents the concepts and algorithms discussed in this paper. Namely, we start with the proposed PIR protocol, which refers in turn to the database shuffling algorithm (formalized in Appendix A.2) and to the protocol for answering k-th query (formalized in Appendix A.3).

A.1 Almost Optimal PIR Protocol

We give a simple version of the protocol, with one shuffled database. The re-shuffling takes place when a fixed number of queries have been answered. The protocol can easily be generalized by preparing several shuffled copies of the database, and by producing shuffled copies in parallel with the process of answering queries.

The preprocessing phase of the protocol is handled by the SC and consists of periodically producing a number of shuffled databases (with appropriate indexes) using Algorithm 1.

The on-line phase of the protocol works as follows.

(1) The SC initializes a query counter $k = 1$, loads the index V' of a shuffled database into the internal memory, and initializes the track of accessed records $T = \{\varnothing\}$.
(2) The client (i.e., the user) comes up with a query $Q =$ "return me the i-th record"
(3) The client and the SC generate and exchange symmetric keys Key_c and Key_{sc} using a public key infrastructure.
(4) The client sends the encrypted query $E(Q, Key_{sc})$ to the server.
(5) The SC receives and decrypts the query.
(6) The SC runs Algorithm 2 to get the answer $A = R_i$.[5]
(7) The SC sends the encrypted answer $E(A, Key_c)$ to the client.
(8) The client decrypts the answer to his query.
(9) The SC increments k by one.
If $k > m$, the SC switches to a new shuffled database, reloads the corresponding index, re-initializes the query counter $k = 1$ and the track of accessed records $T = \{\varnothing\}$.
(10) Step 2 may be repeated.

A.2 Database Shuffling Algorithm

To provide preprocessing for the PIR protocol the database shuffling algorithm is executed inside the secure coprocessor. The only operations observable from outside the SC are *read* and *write* operations, which are used to manage the external storage. The complexity of this algorithm is $O(N^2)$. A basic realization of the database shuffling algorithm is presented as Algorithm 1.

A.3 An Algorithm for Processing the k-th Query

This algorithm (Algorithm 2) is executed inside the secure coprocessor, and is used as a part of the on-line phase of the PIR protocol. The only operations observable from outside the SC are *read* operations to access the shuffled database. As discussed above, the complexity of this algorithm is $O(1)$.

B Privacy Definition and Proof of Protocol

Based on the formalization of the protocol in Appendix A, we give a formal proof for the privacy property of the proposed protocol. However, we first discuss how we define privacy formally. The formal definition of privacy is based on Shannon's information theory sketched in Appendix C.

[5] Algorithm 2 uses i, V', and T to privately retrieve the requested record into the SC; it also updates T appropriately.

Input: DB: a database of N records
Output: $DB_{shuffled}$: a shuffled copy of DB, each record is encrypted; $INDEX_{shuffled}$: an encrypted index of $DB_{shuffled}$

1: $V = [1, ..., N]$ {Index of the database DB}
2: $V' = shuffle(V)$ {Prepare index for the shuffled database $DB_{shuffled}$}
3: **for** $g = 1$ to N **do**
4: **for** $h = 1$ to N **do**
5: $read(Temp \Leftarrow DB[h])$ {Read the h-th record into the SC}
6: **if** $h = V'[g]$ **then**
7: $Record = Temp$ {Save the $V'[g]$-th record of the database internally}
8: **end if**
9: **end for**
10: $write(DB_{shuffled}[g] \Leftarrow Record)$ {Produce the g-th record of $DB_{shuffled}$}
11: **end for**
12: $V'_{encrypted} = encrypt(V')$ {Encrypt the index with some key of the SC}
13: $write(INDEX_{shuffled} \Leftarrow V'_{encrypted})$ {Store the encrypted index of $DB_{shuffled}$}

Algorithm 1: The basic database shuffling algorithm

B.1 Privacy Definition

To perform the formal proof for privacy, we need a formal definition for privacy. We must formally capture the notion "no information about user queries is revealed" in mathematical terms. Definitions based on previous work, such as "communication between the server and client must be indistinguishable", are difficult to apply in our case. In our protocol, not only communication between the server and client is observable, but preprocessing work of the SC is observable too. We need a precise and universal definition.

We exploit Shannon's information theory to use its definition of information measure [Sha48]. There are several reasons for choosing this definition of information measure. It is formal and universal: Many scientific societies (in computer science, physics, economics etc.) accepted it as a classical measure of information ([Jay94], Chapter 11). It is well developed: We can avoid a long list of preliminary theorems by referencing to previous work on information theory [Mac00,Sch96,Jay94].

First, we give a sketch of how the amount of information is measured using information theory in Appendix C. Informally, the information known about a variable i is defined as a measure of the predictability of this variable [Jay94]. The measure of predictability is defined using the measure of unpredictability (entropy) - the central notion in the Shannon's theory.

Second, based on the introduction to information theory, we define privacy as an absence of information about a set of queries $Q_1, ..., Q_k$. Due to the information theory, there is no information revealed about the set of variables $Q_1, ..., Qk$ if and only if the joint entropy $H(Q_1, ..., Q_k)$ of these variables reaches its maximum.

Input: $DB_{shuffled}, V'$: a shuffled copy of DB (each record is encrypted) and its index;
k: the sequence number of the query being processed using $DB_{shuffled}$;
i: the number of the DB record requested
Output: $Answer$: record R_i of DB privately retrieved into the SC

```
 1: g = 1
 2: GotAnswer = No          {An indicator of the presence of Answer inside the SC}
 3: while g < k do
 4:    read(Temp ⇐ DB_shuffled[T[g]]) {Read previously accessed records one by one}
 5:    if V'[T[g]] = i then
 6:       Answer = Temp          {One of the accessed records is the answer, save it}
 7:       GotAnswer = Yes
 8:    end if
 9:    g = g + 1
10: end while
11: if GotAnswer = No then
12:    obtain i' : V'[i'] = i     {Get the position of the i-th DB record in DB_shuffled}
13:    read(Answer ⇐ DB_shuffled[i'])          {Access the required record directly}
14:    T[k] = i'                  {The track list is updated with the k-th item}
15: else
16:    UnRead = {1, ...N} \ {T[1], ..., T[k − 1]}
17:    h = select_random_from(UnRead) {Select randomly one of the unread records}
18:    read(Temp ⇐ DB_shuffled[h])          {Read the selected record into the SC}
19:    T[k] = h                   {The track list is updated with the k-th item}
20: end if
21: return Answer
```

Algorithm 2: An algorithm for processing k-th query

Definition 1 (Privacy). *No information about a set of variables is revealed iff the joint entropy of these variables is maximal.*

We consider the following example to demonstrate the definition.

Example 2 (Calculating the Joint Entropy). We consider two queries Q_1, Q_2. Each of these queries is presented as a variable equal to a number from 1 to N, meaning the record number to be retrieved. We consider two cases. First case: the observer has no information about set of variables Q_1, Q_2. Second case: the observer has no information about variables but the fact that $Q_1 = Q_2$.

Intuitively, half of the information about the set of two variables is revealed in the second case. After calculating the joint entropies for both cases, we check if the joint entropies correlate in the same way.

To calculate joint entropies we need the individual entropies $H(Q_1)$ and $H(Q_2)$. The individual entropies $H(Q_1)$ and $H(Q_2)$ are calculated with (9) using the correspondent conditional probabilities. The conditional probabilities are equal and are the same for the both cases:

$$P(Q_1 = 1) = P(Q_1 = 2) = ... = P(Q_1 = N) = \frac{1}{N},$$

$$P(Q_2 = 1) = P(Q_2 = 2) = ... = P(Q_2 = N) = \frac{1}{N}.$$

In both cases the individual entropies $H(Q_1)$ and $H(Q_2)$ are maximal (9,11) :

$$H(Q_1) = \sum_{1 \le j \le N} P(Q_1 = j) * log \frac{1}{P(Q_1 = j)} = H(Q_2) = logN$$

In the first case, the joint entropy is calculated as the sum of the individual entropies (13) :

$$H(Q_1, Q_2) = H(Q_1) + H(Q_2) = logN + logN = 2 * logN$$

In the second case, the joint entropy is calculated due to (12) :

$$H(Q_1, Q_2 \equiv Q_1) = \sum_{1 \le j \le N, 1 \le k \le N} P(Q_1 = j, Q_2 = k) * log \frac{1}{P(Q_1 = j, Q_2 = k)}$$
$$= \sum_{1 \le j \le N} P(Q_1 = j) * log \frac{1}{P(Q_1 = j)} = logN$$

The last two equations demonstrate the correspondence between the notion of entropy and our intuition in that the entropy of two unknown independent variables is twice as large as the entropy of two unknown equal variables. □

B.2 Proving the Privacy Property

Based on the definition of privacy (Appendix B.1, Definition 1) and on the formal description of the protocol (Appendix A), we formally prove that our protocol has privacy property.

Theorem 1 (The proposed protocol is private). *Let S be a set of queries $Q_1, ...Q_k$ (for any k) executed so far using the protocol proposed in Appendix A. Then, no information (due to Definition 1) is revealed about this set.*

Proof. Given the proposed protocol, we must prove that "no information is revealed about the set of queries executed". That is, due to the definition of privacy (Appendix B.1), we have to prove that, for an observer, the joint entropy of the set of queries is maximal:

$$H(Q_1, ..., Q_k) = k * logN \tag{8}$$

To prove (8), it is sufficient (14) to prove that:

1. The queries are independent for an observer (the first claim):

$$P(Q_1, ..., Q_k) = P(Q_1) * P(Q_2) * ... * P(Q_k)$$

2. The entropy of each query is maximal (the second claim):

$$H(Q_1) = H(Q_2) = ... = H(Q_k) = logN$$

We prove both claims by induction. First, we consider the number of queries $k = 1$. Second, we also consider the case of $k = 2$. Third, we assume that the claims are true for $k = K$ and prove the same for $k = K + 1$.

For $k = 1$, only one query Q_1 is processed after the database was shuffled with Algorithm 1. Due to Algorithm 2, the SC reads directly the required encrypted record to answer the query. The first claim is obviously true because the set of answered queries contains only one query. Since the records were randomly permutated with the shuffling algorithm, reading the encrypted record reveals no correspondence to the original record. This proves the second claim to be true:

$$P(Q_1 = 1) = ... = P(Q_1 = N) = \frac{1}{N}, \quad H(Q_1) = H_{max}(Q_1) = logN$$

Consider the case $k = 2$. Due to the protocol (Appendix B.1), the server answers the second query after reading the previously accessed record and one of the unread records from the shuffled database. Since the server reads one of the unread records independently from whether $Q_1 = Q_2$ or not (commands 18 and 13 of Algorithm 2 respectively), Q_1 and Q_2 are independent variables for the observer.[6] This proves the first claim to be true.

Because the database is shuffled, Q_2 may be any number from 1 to N with equal probabilities. This proves the second claim to be true.

$$P(Q_1, Q_2) = P(Q_1) * P(Q_2); \qquad H(Q_1) = H(Q_2) = logN$$

We assume that the claims are true for $k = K$, i.e.,

$$P(Q_1, ..., Q_K) = P(Q_1) * ... * P(Q_K); \qquad H(Q_1) = ... = H(Q_K) = logN$$

We consider the execution of the $k = (K + 1)$-th query with Algorithm 2. Since the SC reads all K previously read records plus one, there is no relationship between the new query and the previous ones. Taking into account the last equation, we have:

$$P(Q_1, ..., Q_K, Q_{K+1}) = P(Q_1) * ... * P(Q_K) * P(Q_{K+1})$$

Similarly, since the SC accesses a shuffled database, Q_{K+1} could be of any value with equal probability for the observer.

$$H(Q_1) = ... = H(Q_K) = H(Q_{K+1}) = logN$$

The proof by induction is complete. □

[6] That is, if the observer knows one query it does not provide him any information about another one.

C An Introduction to Information Theory

During the preparation of this short survey several sources were used, including [Sha48,Jay94,Mac00,Sch96].

Let X be a random variable, and let A_X be the set of values this variable may take. Let the number of elements of the set A_X be N. Finally, by x_j we denote the j-th element of A_X, $x_j \in A_X$. A random variable X is presented as a vector of probabilities $X =< P(x_1), ..., P(x_N) >$. Similarly, $Y =< P(y_1), ..., P(y_N) >$.

The amount of information known about the variable X is measured by the entropy of this variable $H(X)$. Informally, the entropy is a measure of the "uncertainty" of X. If the entropy is zero, one knows the exact value of the variable – as shown in Equation 10 below. If the entropy is the maximal for this variable, one knows nothing about this variable except its size – as shown in Equation 11 below.

The entropy is defined as a function H with the following properties:

1. H should be continuous in the $P(x_j)$. Otherwise an arbitrary small change in the probability distribution would still lead to the same big change in the amount of uncertainty.
2. It is required, that this function should correspond qualitatively to common sense in that when there are many possibilities, we are more uncertain than when there are few. This condition takes the form that in case the $P(x_i)$ are all equal, the quantity
$$h(N) = H\left(\frac{1}{N}, ..., \frac{1}{N}\right)$$
is a monotonically increasing function of N.
3. Informally, the measure H should be consistent, i.e., if there is more than one way of working out its value, we must get the same answer for every possible way. Formally, if a choice is broken down into two successive choices, the original value of H should be the weighted sum of the values of H for individual choices. The meaning of this is illustrated in Figure 5. For this

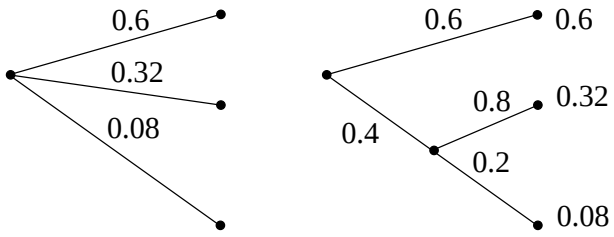

Fig. 5. Decomposition of a choice from three possibilities.

special case, we require that

$$H(0.6, 0.32, 0.08) = H(0.6, 0.4) + 0.4 * H(0.8, 0.2).$$

The coefficient 0.4 is the weighting factor.

Shannon's Theorem proves that the only function satisfying the given properties is the following one (a multiplicative constant is usually omitted):

$$H(X) = \sum_{x \in A_X} P(x) * log \frac{1}{P(x)} \tag{9}$$

With the definition of entropy, one can prove the further properties:

$$H(X) \geqslant 0 \quad \text{with equality iff} \quad P(x_j) = 1 \quad \text{for one } j \in 1, ..., N \tag{10}$$

$$H(X) \leqslant \log N \quad \text{with equality iff} \quad P(x_j) = 1/N \text{ for all } j \in 1, ..., N \tag{11}$$

The joint entropy, i.e., the entropy of a set of variables is calculated by:

$$H(X,Y) = \sum_{xy \in A_X A_Y} P(x,y) * log \frac{1}{P(x,y)} \tag{12}$$

In case of variable independence, the joint entropy is calculated as a sum of entropies of the variables:

$$H(X,Y) = H(X) + H(Y) \quad \text{if} \quad P(x,y) = P(x) * P(y) \tag{13}$$

The joint entropy of a set of variables has the same meaning as the entropy of a variable. That is, no information about a set of variables is revealed iff the joint entropy is maximal.

It can be shown, that the maximal joint entropy may be reached only if the variables are independent and the individual entropies of the variables are maximal. So, no information is revealed about a set of variables, if these variables are independent[7] and no information about each of these variables is revealed. Formally:

The joint entropy H(X,Y) is maximal iff

$$H(X) = logN, \quad H(Y) = logN, \text{ and } P(x,y) = P(x) * P(y) \tag{14}$$

The maximal entropy is then due to (13) and (14) :

$$H(X,Y) = H_{max}(X) + H_{max}(Y) = logN + logN = 2 * logN$$

In summary, we say that no information is revealed if the corresponding entropy is maximal. In particular, no information is revealed about a set of variables if the joint entropy of these variables is maximal. Only for independent variables the joint entropy may reach its maximum and can be calculated by the sum of the entropies of the variables.

[7] This basically means, that if we know one variable it gives us no information about another variable.