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 offline communication, such that only $O(1)$ online computation and communication are performed to execute a query. The major drawback of these solutions is offline communication, comparable to the size of the entire database.
Using a secure coprocessor we construct a PIR scheme that eliminates both drawbacks. Our protocol requires $O(1)$ online computation and communication, periodical preprocessing, and zero offline communication. The protocol is almost optimal. The only parameter left to improve is the server's 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 user can perform the query "return me the i-th record" in such a way that no one, not even 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 user 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. Billing can be performed as well, because the retrieval is controlled by the server.

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

* Published at *Privacy Enchancing Technologies 2002.* ©Springer-Verlag
** 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, employ very similar ideas to address the problem, although they introduce a new one. Namely, offline 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) complexity of query response time and communication, and no offline communication. For our protocol, $O(1)$ records have to be processed online in order to answer a query. But, in contrast to [BDF00,SJ00], the protocol eliminates the offline 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.

By the client we denote the computer the user forms his queries with. We assume that the user trusts the client; i.e., the sensitive information stored or processed at the client is assumed to be hidden from every one except the user.

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 are presented in Sect. 4. We finish the paper with the discussion and future work ideas.

Furthermore, Appendix A formalizes the protocol. Due to space limitation, a formal definition of privacy based on information theory and the proof that the protocol is private are not included in the final version of the paper. Both the privacy definition and the protocol proof can be found in [AF01].

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 server computation complexity (or, query response time).

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, before the description of our protocol, 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 collude against the user [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, improved by polylogarighmic communication complexity in [CMS99,KY01].

Still, the second limitation holds 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 Fig. 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. 4.1.

Pros and Cons. This PIR protocol improves the computation complexity. In comparison to computational PIR protocols, ordinary decryption has to be performed for 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 Offline Communication

Although it does not seem feasible to break through the second limitation - $O(N)$ computation, one could try to preprocess as much work as possible. Such that,

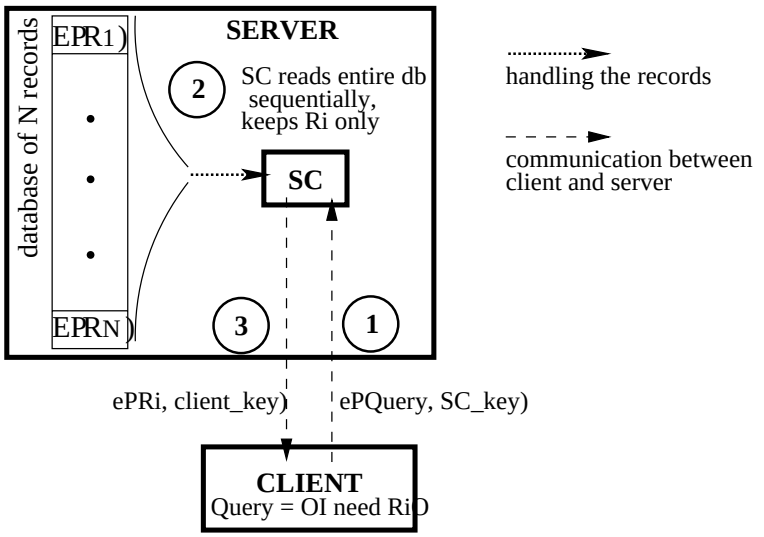

Fig. 1. An example of a PIR protocol with SC.

when a query is submitted, it would require only $O(1)$ computation to answer it online.[2]

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 offline every record of the database. All these encrypted records are sent 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 online.

If the user 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 users's encryption. The server sends it back to the client. The user removes his encryption; the protocol is done (Fig. 2).

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

However, the protocols suffer from another drawback: This is offline communication comparable to the size of the entire database, that makes their practical applicability questionable. (Imagine a user decides to buy a single digital book or a music file at some digital store. He will probably react negatively after being

[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].

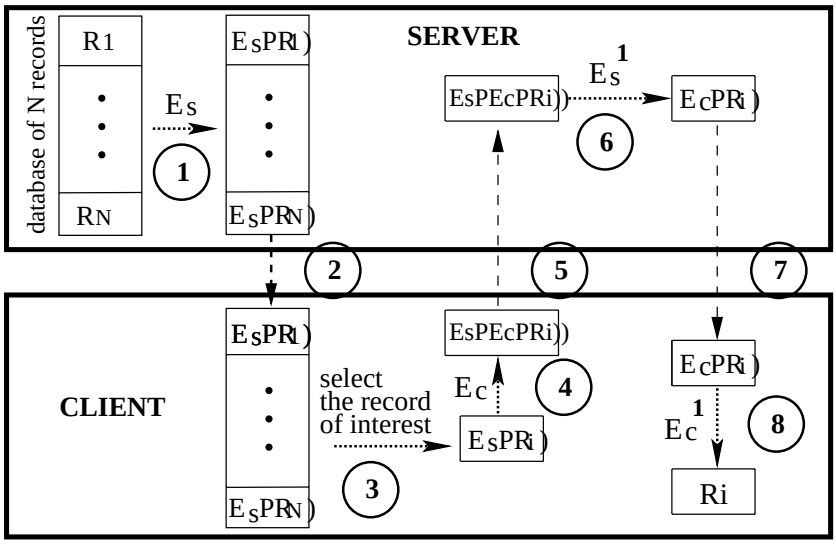

Fig. 2. An example of a PIR protocol with preprocessing and offline communication. Steps 1 and 2 are made offline once, and the other steps are performed online for every query submission.

asked to download the entire encrypted content of the digital store in oder to proceed. Another problem is keeping the user's database copy updated.)

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.

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

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)
Online communication	assimpt. optimal	optimal	optimal	optimal
Computation	O(N)	O(N)	O(1)	O(1)
Offline comm.	no	no	O(N)	no
Preprocessing	no	yes	yes	yes

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 offline. 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 SC receives the first query "return the i-th record", the SC does not need to read the entire database. Instead, the SC accesses the desired encrypted record directly. Then the encrypted record is decrypted inside the SC, encrypted with the user's key, and sent to the user (Fig. 3).

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 reads a random (previously unread) record. Otherwise some information about the queries is revealed, namely, it would be observable that the queries are identical.

So, to answer the k-th query, the SC has to read the $k-1$ previously read records first. Then the SC reads one of the unread records. 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 shuffled copy of the database, so that k would be equal one again. Since m is a constant independent of N, we can say that the server has to process $O(1)$ records online to answer each query.

Now that the basic idea has been introduced, we go into details of our protocol in the next section. The formalization of the protocol is presented in Appendix A.

4 The Details

A hypothetical attack is considered in Sect. 4.1. We demonstrate a trade-off between offline and online computation in our protocol and discuss how to choose the optimal trade-off in Sect. 4.2 and 4.3 respectively. Finally, we consider shortly the cases with multiple queries and multiple secure coprocessors.

4.1 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 user complains after the PIR query is performed, the malicious server concludes that the user was interested in the modified record, thus breaking the privacy of the user. The solution proposed is to check the granularity of every record in the database

[3] Assume that the server issued the first query itself. Then the server observed which record was read by the SC. So the identity of the one encrypted record is known to the server. Now, the SC reads another encrypted record to answer a users'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.

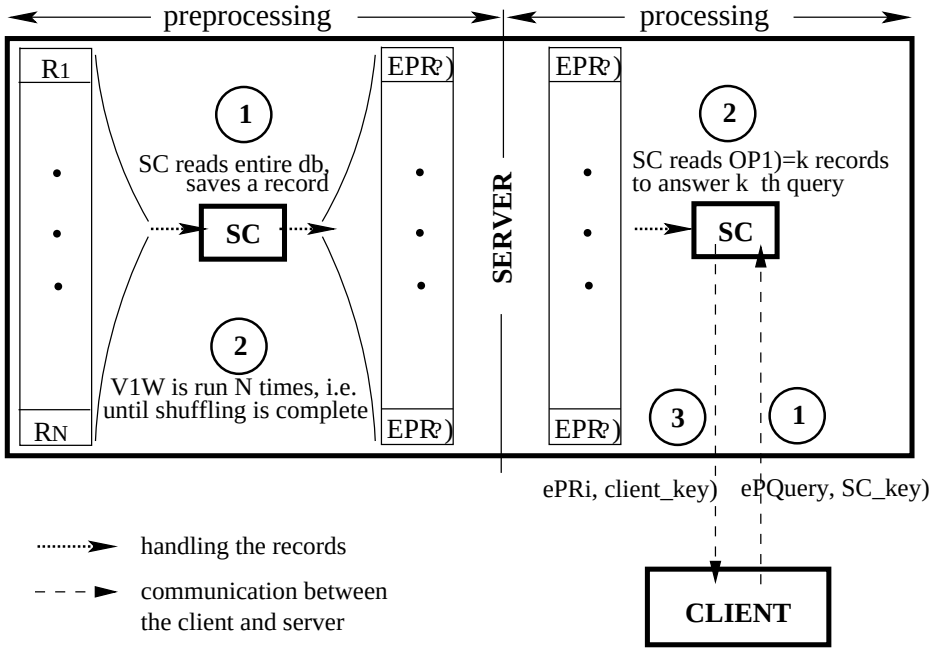

Fig. 3. I/O flows in the proposed PIR protocol.

(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.

4.2 Trade-Off Between Preprocessing and Online Computation

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

Let $\mathbf{r}_{online}$ be the average number of encrypted records that the SC reads online to answer a query. This parameter characterizes the average response time of the server. Let $\mathbf{w}_{offline}$ be the average number of encrypted records that the SC

writes 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.

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

$$\mathbf{w}_{offline} = \frac{N}{m} \tag{2}$$

The dependencies between the trade-of parameter m, the online workload $\mathbf{r}_{online}$, and the preprocessing workload $\mathbf{w}_{offline}$ are shown in Fig. 4 (for $N = 10000$). From equations 1 and 2 we derive the dependence between the online ($\mathbf{r}_{online}$)

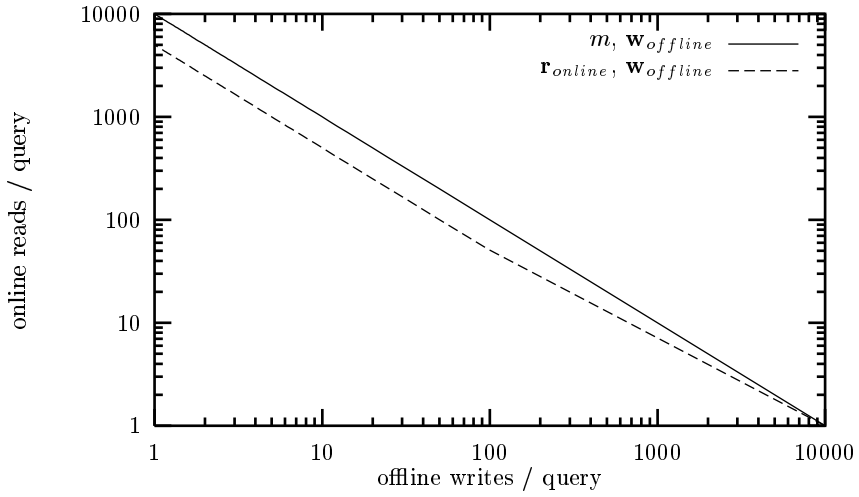

Fig. 4. The dependence between online performance (max and average number of records to read online per query) and preprocessing workload (number of offline write operations per query).

and offline ($\mathbf{w}_{offline}$) parameters of the protocol.

$$\mathbf{r}_{online} = \frac{N}{2*\mathbf{w}_{offline}} + 1, \quad \mathbf{r}_{online} = \Theta\left(\frac{1}{\mathbf{w}_{offline}}\right) \tag{3}$$

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.

4.3 Choosing the Optimal Trade-Off

If the maximal allowed query response time is fixed, choosing the trade-off parameter m 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 online and preprocessing workload parameters.

We show in Fig. 5 that the overall work $S(m)$ does not remain constant while varying trade-off parameter. To determine the optimal trade-off parameter we

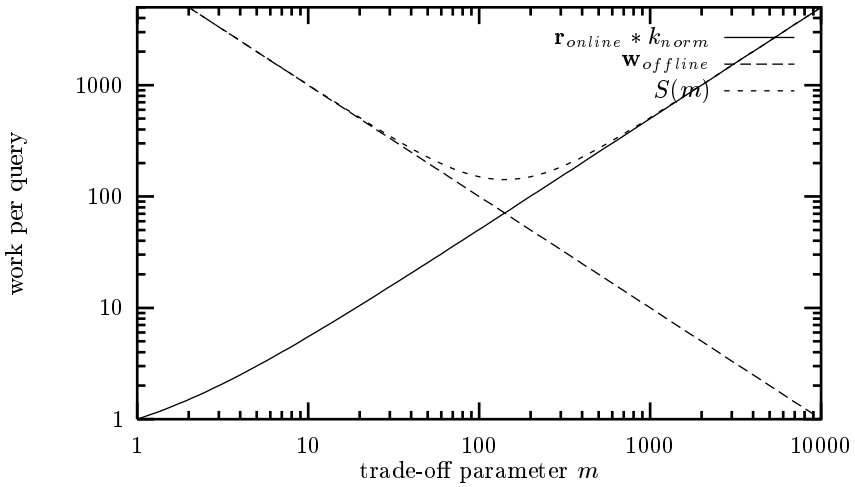

Fig. 5. The overall work done per query (calculated as a sum of normalized online and offline parameters) is not constant for different values of the trade-off parameter.

must find the minimum of the following function:

$$S(m) = \mathbf{r}_{online} + \varphi_{norm} * \mathbf{w}_{offline} \tag{4}$$

where φ_{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 $S(m)$:

$$S'(m) = \left(\frac{m+1}{2} + \varphi_{norm} * \frac{N}{m}\right)' = \frac{1}{2} - \frac{\varphi_{norm} * N}{m^2} \tag{5}$$

$$\frac{1}{2} - \frac{\varphi_{norm} * N}{m_{opt}^2} = 0, \qquad m_{opt} = \left[\sqrt{2 * \varphi_{norm} * N}\right] \tag{6}$$

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

4.4 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 online 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 since preprocessing can be done independently from query processing.

5 Discussion

In this section we consider some questions that arise while reading the paper.

5.1 Trust in SC

One might assume, that users must trust the SC. In the following, we clarify that the SC must not be trusted by users. Instead, the trust in SC is reduced to the trust in statement 3 below, given statements $1, 2$ can be checked independently from user's believe.[4]

1. The SC is a product of company A.
2. The SC is unbreakable, if it is designed in accordance to the product design documentation of company A.
3. The SC is designed in accordance to the product design documentation.

In summary, the only possible concern about the SC is that it might not (theoretically speaking) be designed in accordance to its specification, thus intentionally leaving some hole on the physical or API level. In addition, in order for the attack to succeed, the company that produces SC must cooperate with the company that trades digital goods.

Furthermore, this concern may be (hypothetically) backed up by the company A itself, by providing the trust in statement 3 with monetary incentives or a prize for everybody who proves that a SC misbehaves or can be broken.

5.2 Private vs. Anonymous Retrieval of Information

There is a principal difference between PIR approaches and anonymous techniques, such as those based on Chaum's MIX. PIR hides the content of the queries and does not depend on third parties. Anonymous techniques do not

[4] Statements $1, 2$ must not be really trusted by users, because (i) the first statement can be checked via PKI, (ii) the second statement follows from conclusions of several tests of independent research labs. Both issues are sketched in an overview [DLP+01].

hide the content of the queries from the server and do depend on third parties [5]. The consequences are that the administration complexity is higher and the privacy is not perfect because it does not withstand "all-against-one" attack.

5.3 Preprocessing Complexity

The preprocessing complexity of our protocol is high (i.e. quadratic in N), even though it can be made offline as many times as secondary storage is available. We currently work on decreasing the preprocessing complexity [AF02].

6 Conclusion and Future Work

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 offline 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 $O(1)$ query response time, optimal online communication complexity, and does not require offline communication. This property is due to a novel preprocessing algorithm based on shuffling and due to the usage of a secure coprocessor. We showed the trade-off between the online and preprocessing workloads for the protocol. The protocol is scalable for multiple queries and multiple secure coprocessors.

An open question is whether the preprocessing complexity of our protocol is optimal or not. Minor additions should be made to the algorithms to withstand timing attacks.

Acknowledgements: We are especially thankful to Roger Dingledine, Andreas Pfitzmann, Adam Shostack, Sean Smith and Paul Syverson for the fruitful comments on the talk. Thanks also go to Bram Cohen, George Danezis and Stephan J. Engberg for the interesting thoughts on the subject.

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

References

[AF01] D. Asonov and J.-C. Freytag. Almost optimal private information retrieval. Tech. Report HUB-IB-156, Humboldt University Berlin, November 2001.

[5] That is, anonymous approaches require "other people (other entities) doing other things", in opposite to PIR that does not require anything but one server.

[AF02] D. Asonov and J.-C. Freytag. Private information retrieval, optimal for users and secure coprocessors. Tech. Report HUB-IB-159, Humboldt University Berlin, May 2001.

[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 *Proc. of the 3rd Intl. 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. Doorn, S. Smith, and S. Weingart. Building the IBM 4758 Secure Coprocessor. In *IEEE Computer*, 43(10), 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.

[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*, 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 reshuffling 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 online 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 $A = R_i$.[6]
(7) The SC sends the encrypted answer $E(A, Key_c)$ to the client.
(8) The client decrypts the answer to the 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 SC. The only operations observable from outside the SC

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

are *read* and *write* operations, which are used to manage the external storage. The complexity of this algorithm is $O(N^2)$. The function *shuffle* that provides a shuffled index is constructed in accordance with [Knu81], Sect. 3.4.2. A basic realization of the database shuffling algorithm is presented as Algorithm 1.

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: $write(DB_{shuffled}[g] \Leftarrow encrypt(Record)$ {Produce 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

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 online 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 $O(1)$.

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

Algorithm 2: An algorithm for processing k-th query