Private Information Retrieval, Optimal for Users and Secure Coprocessors

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. A PIR protocol is optimal for users, if the query response time is independent from N, and the communication between the user and server is of an order of one record. Recently such a protocol was proposed [AF01,AF02]. However, an ultimate condition for that protocol is using a secure coprocessor (SC) that processes the database records.
We focus on the problem of minimizing the number of records handled by the SC, both for preprocessing and for query processing. In this paper, we present a set of preprocessing algorithms with $O(N^2/p)$ I/O's for a SC (for any p) and $O(pN)$ I/O's for an untrusted computer (UC), compared to $O(N^2)$ I/O's for a SC in [AF01,AF02]. For $p = \sqrt{N}$, the algorithm is of complexity $O(\sqrt{N}N)$ for a SC and UC. Also, query processing time may be cut from $O(1)$ to 2 by preprocessing online.

Keywords: Information hiding aspects of privacy; efficient realization of privacy services.

1 Introduction

In many e-commerce scenarios the user retrieves data from the server and pays for it. Sometimes it is vital for the user to hide the content of his queries from every one else. Encrypting the communication channel between the client and server does not help, since the server would have access to the user queries. Solutions that help are called Private Information Retrieval (PIR) protocols.

Formally, a PIR protocol allows a user to retrieve one of N digital records of his choice from the server, while revealing nothing about the record identity, not even to the server that processes the query.

Trading digital goods online is a straightforward application for PIR. With PIR the user privacy concerns would be satisfied: The server cannot misuse the information contained in the user query, because the server simply does not have information about content of user queries.

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

1.1 Motivation

Initial PIR protocols process all N records to answer a user query "return the i-th record" [KO97,CMS99,SS00,SS01], resulting in impractical query response time. Later protocols fixed this problem by introducing preprocessing on the server side [BDF00,SJ00]. However, the communication proportional to the size of the database between the client and server is needed to implement these latter protocols.

A recently proposed protocol achieves constant query response time and optimal communication simultaneously [AF01,AF02], thus being an optimal protocol from the user point of view. This protocol employs a tamper-proof device, namely a secure coprocessor (SC), that processes database records (Section 2.1 gives a sketch of the protocol). This approach raises the following two concerns:

– The PIR protocol uses special hardware (a SC) at the server site, so the implementation of the protocol is impossible with general-purpose hardware only.
– The PIR protocol uses special hardware intensively. Namely, the tamper-proof device must process k records to answer the k-th query online. Moreover, by starting with $k = 1$, the tamper-proof device must preprocess $O(N^2)$ records.

In this paper, we tackle the second concern and achieve results outlined in the next subsection.

1.2 Our Results

We improve the complexity of both preprocessing and processing as following.

In Section 3, we construct a set of preprocessing protocols with $O(N^2/p + pN)$ complexity, where p is a parameter identifying a member of the set. We show, that the overall number of I/O's is minimal for $p \approxeq \sqrt{N}$ (Section 3.2). Consequently, the optimal protocol in the set has a complexity of $O(N\sqrt{N})$, in contrast to $O(N^2)$ in [AF02]. For practical scenarios, this result cuts weeks of preprocessing time down to a few hours, as discussed in Section 2.2.

In Section 4, we assume there is a free time slot after the $(k-1)$th query has been processed. We show how to reduce the query response time for k-th query from k down to 2 (a constant) if k^2 time is available online for running an appropriate algorithm. Evidently, runtime of this algorithm grows with the number of queries answered so far.

In Section 5 we compare our approaches with the protocol in [AF01,AF02]. To do this using one scale, we propose a normalized measure of the complexity of a protocol based on the processing and preprocessing complexities.

1.3 Preliminaries and Assumptions

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

We assume that the access to secondary storage takes several orders of magnitude more time than an operation on data in main memory. Thus, we measure the preprocessing and query response time of a PIR protocol by the number of records accessed on secondary storage, i.e. by the number of I/O's of blocks of size L.

For simplicity of presentation, we assume that $O(L)$ bits of memory of a SC are available. Our protocols can be slightly modified in order to support the general case.

2 Related Work

Appendix A sketches the PIR protocols from [SS00,SS01,BDF00,SJ00], that suffer from $O(N)$ query response time or $O(N)$ communication required to initiate the protocol. Additionally, it briefly reviews the properties of a SC briefly in Section A.1.

Section 2.1 summarizes the protocol from [AF01,AF02], that both has $O(1)$ query response time and communication complexity.

Section 2.2 reviews the related work and compares those results with the proposed protocol.

2.1 PIR Optimal for Users but Costly for SC

The protocols described in [SS01,BDF00,SJ00] (summarized in Section 2.2, Table 1) raise the question, if it is possible to design a PIR protocol with $O(1)$ query response time and $O(1)$ communication. Such protocol would be optimal from the user's point of view. A solution is described in [AF02] by suggesting the following protocol.

Using Algorithm 1, the SC shuffles the records of a database in such a way, that no one observes the new order (Fig. 1). In Algorithm 1, the SC reads the entire database N times record by record. Each time the SC leaves a record in its secure memory and writes it to the (shuffled) database in an encrypted form. When the shuffled database is complete, the SC has read N^2 database records, i.e. N^2 (sequential) blocks of size L.

After this preprocessing is complete, the SC answers the user queries in $O(1)$ time. The SC must not read the entire database to answer one query. Instead, as for the first query, the SC reads only one record to answer a query. To answer a k-th query, the SC must read k records: These are $k-1$ previously read records plus one of the unread records in the shuffled database.

After some threshold number m of records to read to answer a query is reached, the SC switches to another shuffled database to keep the response time from breaking through the threshold.

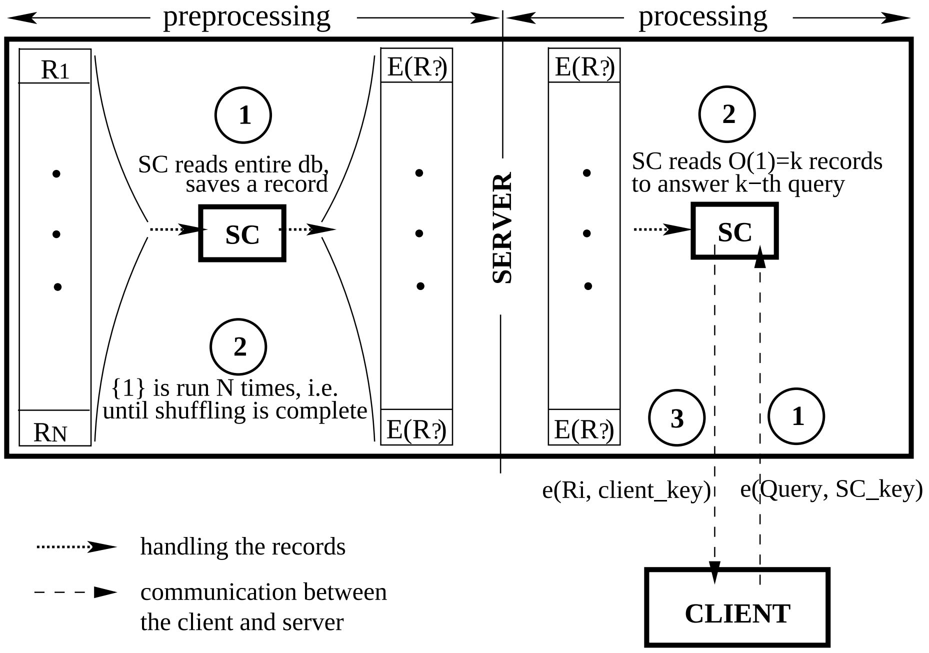
preprocessing
processing
R1
RN
E(R?)
SC reads entire db, saves a record
{1} is run N times, i.e. until shuffling is complete
SC
SERVER
SC reads O(1)=k records to answer k−th query
e(Ri, client_key)
e(Query, SC_key)
CLIENT
handling the records
communication between the client and server

Fig. 1. I/O flows in the PIR in [AF01].

2.2 State of the Art and Our Results

We call a protocol optimal for users if both the communication and query response time are optimal [1]. In the related work, summarized in Table 1, there is only one protocol optimal for users – [AF02].

However, this protocol requires preprocessing of high complexity which must be repeated periodically. Assuming that accessing one database record takes $0.01sec$ for a SC and $N = 10000$, these are $N^2 * (0.01sec/record) = 10000 * 10000 * 0.01sec \approx 2$ weeks to prepare one shuffled database. [AF02] recommends to use one shuffled database for answering no more than about 140 queries.

One of the preprocessing protocols proposed in this paper exhibits a complexity of $O(N^{1.5})$, meaning the time of preprocessing of 2 weeks is cut down to approximately 3 hours. Note, that the query response time of our protocol (identified to be $\leq k$ in Table 1) can take values up to 2. The exact value depends on the amount of time between the processing of a query and the arrival of the next query, as explained in details in Section 4.

[1] We say, that the communication complexity is optimal if communication between the client and server is of an order of one record per query "return the i-th record". Query response time is optimal if it is independent form the size of the database, i.e., if it is independent from N

```
Input: $DB$: a database of $N$ records
Output: $DB_{shuffl}$: a shuffled copy of $DB$, each record is encrypted; $INDEX_{shuffl}$:
    an encrypted index of $DB_{shuffl}$
 1: $V = [1, ..., N]$                                   {Index of the database $DB$}
 2: $V' = shuffle(V)$                {Prepare index for the shuffled database $DB_{shuffl}$}
 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:   $write(DB_{shuffl}[g] \Leftarrow encrypt(Record))$      {Produce $g$-th record of $DB_{shuffl}$}
11: end for
12: $V'_{encrypted} = encrypt(V')$              {Encrypt the index with some key of the SC}
13: $write\_from\_SC(INDEX_{shuffl} \Leftarrow V'_{encrypt})$     {The encrypted index of $DB_{shuffl}$}
```

Algorithm 1: The basic database shuffling algorithm

3 Improving the Preprocessing Complexity

Section 3.1 presents a preprocessing algorithm with $O(N^2/p)$ complexity for a SC (for any $p \leq L$) and $O(pN)$ complexity for an untrusted computer (UC).

Section 3.2 shows the optimal complexity for a SC together with an UC to be $O(\sqrt{N}N)$ (for $p = \sqrt{N}$), in contrast to $O(N^2)$ in [AF01].

3.1 The Preprocessing Algorithm with Reduced I/O's Complexity

Algorithm 2 formalizes the proposed preprocessing algorithm referring to three sub-algorithms that are explained in the following.

Algorithm 3 splits each record of the database $DB[N]$ in p equal parts. For example, the database $DB_1[N]$ would consist of the first parts of the records of the original database; each record of the database $DB_1[N]$ would be of size L/p. Obviously, there would be p such databases $(DB_1[N], ..., DB_p[N])$. No SC's is needed to perform this algorithm, because it can be performed by an untrusted computer (UC). Fig. 2(b) shows the output of this algorithm for $p = 2$.

Table 1. Comparative analysis of the proposed protocol.

Parameter	PIR Protocol			
	Computational [BDF00,SJ00]	With SC [SS00,SS01]	With SC [AF01,AF02]	The Proposed (With SC)
Response time	$O(1)$	$O(N)$	$O(1) = k$	$O(1) \leq k$
Extra Comm.	$O(N)$	no	no	no
Preprocessing	once, $O(N)$	once, $O(N)$	periodical, $O(N^2)$	periodical, $O(N^{1.5})$

```
Input: $DB[N]$: a database of $N$ records of size $L$; a parameter $p$
Output: a shuffled copy of $DB[N]$, each record is encrypted; $INDEX_{shuffl}$: an encrypted shuffling index
1: if Algorithm 3 has never been executed before then
2:     $excecute(Algorithm\ 3)$  {Split the records of the database in $p$ parts; $j$-th part of each record is saved in $DB_j[N]$, $\forall j, 1 \leq j \leq p$}
3: end if
4: $excecute(Algorithm\ 4)$ {Shuffle the $p$ databases using the SC: $DB_j \rightarrow DB_j^{shuffl}$ }
5: $excecute(Algorithm\ 5)$  {Assemble the shuffled records of the original database from their pieces}
```

Algorithm 2: The main preprocessing algorithm

```
Input: $DB[N]$: a database of $N$ records of size $L$; parameter $p$
Output: $\{DB_j[N]\}, j \in \{1, .., p\}$ : $p$ databases; an $x$-th record from the $y$-th database ($DB_y[x]$) is the $y$-th part of $DB[x]$
1: for $g = 1$ to $p$ do
2:     for $h = 1$ to $N$ do
3:         $read(Temp \Leftarrow DB[h][g])$          {Read the $g$-th part of $h$-th record}
4:         $write(DB_g[h] \Leftarrow Temp)$          {Produce the $h$-th record of $DB_g$}
5:     end for
6: end for
```

Algorithm 3: The database splitting algorithm

Using Algorithm 4, the SC shuffles all the databases based on the same shuffling vector ($DB_1 \rightarrow DB_1^{shuffl}$, $DB_2 \rightarrow DB_2^{shuffl}$,...,$DB_p \rightarrow DB_p^{shuffl}$) one by one. Each shuffling takes $(N/p)^2$ I/O's for a SC, because the SC can read p records into its memory at a time. This results in $p * (N/p)^2$ sequential reads of blocks of size L. Fig. 2(c) demonstrates an example of an output of this algorithm, for the input in Fig. 2(b).

Function *map* used in Algorithm 4 ensures that after each read of a database p (random) records remain in the memory of SC (in $Kept$ array) which are then written to secondary storage. Formally, at the end of the h-th read of the database DB_g, the array $Kept$ contains p records from the DB_g:

$$Kept[1] = DB_g[j_1], \quad ... \quad , Kept[p] = DB_g[j_p], \text{ where}$$

$$j_1 = V'[(h-1)p+1], \quad ... \quad , j_p = V'[(h-1)p+p] = V'[hp]$$

Algorithm 5 gathers the "pieces" of the each record into the one record, such that at query processing time one can access an entire record immediately, instead of accessing each part of the record from p different databases. As well as Algorithm 3, this algorithm can be performed by an UC, and takes $N * p$ reads.

The overall complexity of Algorithms 4,5 is $N^2/p + N * p$. Namely, N^2/p is complexity of the SC work, and $N * p$ complexity can be carried out by an

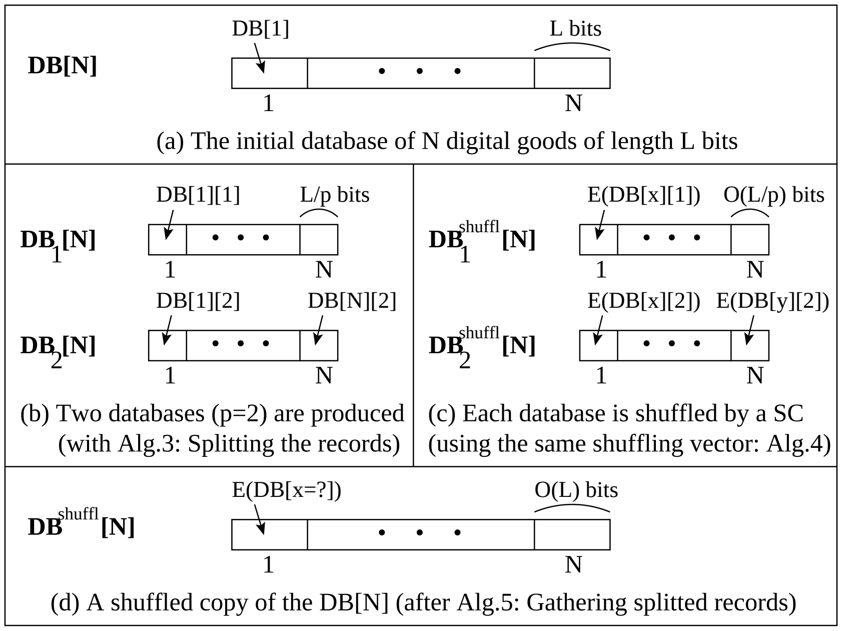
DB[N]
DB[1]
L bits
1
N
(a) The initial database of N digital goods of length L bits
DB1[N]
DB[1][1]
L/p bits
DB2[N]
DB[1][2]
DB[N][2]
(b) Two databases (p=2) are produced (with Alg.3: Splitting the records)
DB1shuffl[N]
E(DB[x][1])
O(L/p) bits
DB2shuffl[N]
E(DB[x][2])
E(DB[y][2])
(c) Each database is shuffled by a SC (using the same shuffling vector: Alg.4)
DBshuffl[N]
E(DB[x=?])
O(L) bits
(d) A shuffled copy of the DB[N] (after Alg.5: Gathering splitted records)

Fig. 2. An example of the preprocessing algorithm for $p = 2$.

UC. We omit $O(N * p)$ complexity of Algorithm 3 while calculating the overall preprocessing complexity, because this algorithm must be performed only once.

In the next section we determine the minimal complexity possible by varying parameter p.

3.2 Balancing The Preprocessing Complexity between SC and UC

In this section we study, in how many pieces to split the initial database records in order to gain optimal (i.e. minimal) complexity of the preprocessing algorithm (Alg. 2).

Theorem 1. *The minimal complexity of the Algorithm 2 is attained for* $p = \sqrt{N}$.

Proof. As follows from Section 3.1, the complexity of the preprocessing algorithm can be estimated by $O(N * p + N^2/p)$, given that the parameter $1 \leq p \leq L$ of the algorithm is specified.

To find the optimal p, we determine the minimum of the function $f(p) = N * p + N^2/p$.

$$f'(p) = (N * p + \frac{N^2}{p})' = N - \frac{N^2}{p^2}; \quad p_{opt} = \sqrt{N}.$$

Consequently, the optimal preprocessing complexity is $O(f(p_{opt})) = O(2N\sqrt{N}) = O(N\sqrt{N})$, as also shown in Fig 3 for $N = 10000$. □

```
Input: Output from Algorithm 3: $\{DB_j[N]\}, j \in \{1,..,p\}$: $p$ databases, each has N
    records of size $L/p$;
Output: $\{DB_j^{shuffl}[N]\}, j \in \{1,..,p\}$ : $p$ shuffled databases; $INDEX_{shuffl}$: an
    encrypted shuffling index, the same for all split databases
 1: $V = [1,...,N]$; $V' = shuffle(V)$          {Use the same index to shuffle databases}
 2: for $g = 1$ to $p$ do
 3:   for $h = 1$ to $N/p$ do
 4:     for $u = 1$ to $N/p$ do
 5:       $read\_into\_SC(Temp \Leftarrow DB_g[(u-1)p+1,...,(u-1)p+p])$          {Read $p$
                                      sequential records of size $L/p$ from $DB_g$}
 6:       $map(Kept \Leftarrow Temp)$          {Fill positions in $Kept$ in accordance with $V'$}
 7:     end for
 8:     $write\_from\_SC(DB_g^{shuffl}[(h-1)*p+1,...,(h-1)*p+1] \Leftarrow encrypt(Kept))$
        {Produce $p$ sequential records of size $L/p$ for $DB_g^{shuffl}$, encrypt separately}
 9:   end for
10: end for
11: $V'_{encrypted} = encrypt(V')$          {Encrypt the index with some key of the SC}
12: $write\_from\_SC(INDEX_{shuffl} \Leftarrow V'_{encrypt})$          {The encrypted shuffling index}
```

Algorithm 4: The algorithm for shuffling the split databases

```
Input: Output from Algorithm 4: $\{DB_j^{shuffl}[N]\}, j \in \{1,..,p\}$ : $p$ shuffled databases;
    the shuffling vector $V'$ is the same for all databases
Output: $\{DB^{shuffl}[N]$ : a shuffled copy of $DB[N]$; each record is encrypted
 1: for $h = 1$ to $N$ do
 2:   for $g = 1$ to $p$ do
 3:     $read(Temp \Leftarrow DB_g^{shuffl}[h])$ {Read the $g$-th part of $h$-th record for $DB_{shuffl}$}
 4:     $write(DB_g[h] \Leftarrow Temp)$ {Produce the $g$-th part for $h$-th record for $DB_{shuffl}$}
 5:   end for
 6: end for
```

Algorithm 5: The algorithm for assembling the shuffled database

The above result for optimal p is valid if $\sqrt{N} \leq L$, which is a rather reasonable assumption. Otherwise, if $\sqrt{N} > L$, the parameter should be assigned to the largest possible number, i.e. $p = L$.

The complexity of the work for the SC can be reduced further to $O(N)$ at the same time leading to a growth in the same order in the complexity of the work done by an UC, as shown in Fig.4.

4 Optimizing the Query Processing Complexity

According to the protocol in Section 2.1, k records must be read from the shuffled database in order to answer a query online, where k is the number of queries answered using the same shuffled copy of a database. This section suggests an approach to keep the query response time independent from k by employing preprocessing online.

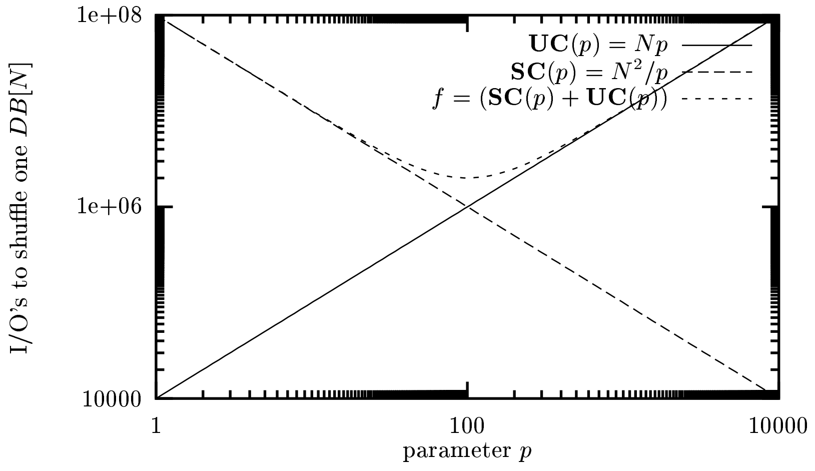

Fig. 3. The overall work done to shuffle one database (calculated as a sum of the number of I/O's for SC and UC) is not constant for different values of the parameter p.

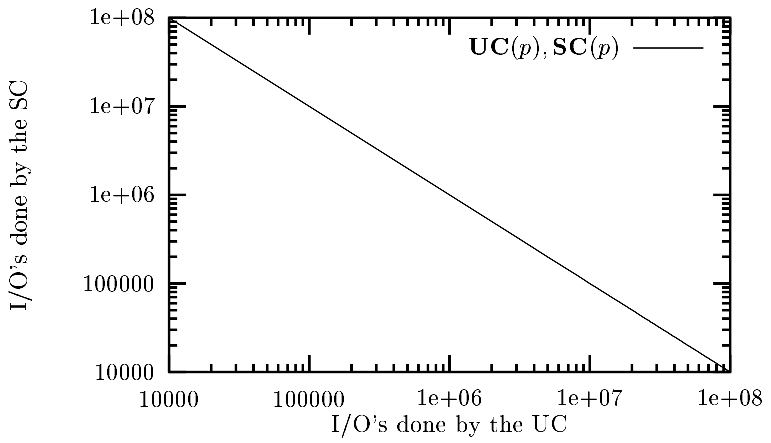

Fig. 4. At preprocessing, reducing the complexity of the SC results in the growth of complexity of the UC and vice versa.

4.1 A Straightforward Approach

The basic idea is to apply the protocol of Section 2.1 recursively [AS02].

After answering the k-th query, we propose to shuffle all the previously read records. The shuffled database would then consist of the group of previously unread records and the group of previously read records, now newly shuffled.

Processing the next query results in reading one record from each group [2], so that response time is determined by 2 accesses to secondary storage. If such "refreshment" of the shuffled database is done after each query, the query response time would remain being determined by 2 I/O's.

Periodical switching to a new shuffled database is inevitable, however. The reason is that the complexity of online shuffling of the previously accessed records grows with the number of queries answered, possibly resulting in delays for processing the next queries.

Note, that the shuffling of k records can be done using the preprocessing algorithm proposed in Section 3, providing the complexity $O(k\sqrt{k})$.

4.2 Interruptible Shuffling Protocol

The previously proposed secure shuffling algorithms (Alg. 1 from [AF01,AF02] and Alg. 2 introduced in Sect. 3.1) are non-interruptible. That is, if the execution of these algorithms is interrupted before it finishes, the output produced so far is of no use for the query processing protocol.

In case of using the preprocessing online, the amount of time available for preprocessing is unknown when preprocessing begins. The reason is that the arrival time of the next query is unknown. Thus, a preprocessing algorithm is preferred, that can be interrupted and whose output produced so far can be used to process the next query.

Our interruptible shuffling algorithm (Alg. 6) shuffles k records in $O(k^2)$ steps. In order to be interruptible, the algorithm works as follows. Given $(j-1)$ shuffled records and a j-th record, the algorithm produces j shuffled records in $O(j)$ time. The algorithm is interruptible, because it can be stopped at any time and the s shuffled records produced so far can be used to reduce the query response time from k to $(k-s+2)$ in way, very similar to that explained in Section 4.1.

5 Measuring the Complexity of the PIR Protocol

Above, we proposed several techniques to improve preprocessing and processing parts of the PIR protocol in [AF01,AF02]. In this section, we measure how these improvements influence the overall complexity of the protocol. For being able to do this, we first define a normalized complexity of a PIR protocol as following.

[2] The correctness of this protocol follows recursively from the formal proof of correctness of the protocol in Section 2.1, which can be found in [AF01,AF02].

```
Input: $j-1$ shuffled encrypted records $DB^{shuffl}[1,..,j-1]$; a record $R$
Output: $DB^{shuffl}[1,..,j]$ : $j$ shuffled encrypted records; the newly added record $R$
    can be at any position.
1: read_into_SC($Buf \Leftarrow R$)                              {Read the record $R$ into the SC}
2: for $g = 1$ to $j-1$ do
3:     read_into_SC($Temp \Leftarrow decrypt(DB^{shuffl}[g])$)     {Read the $g$-th record from
                                                              $DB_{shuffl}$, decrypt it}
4:     ifswap($Temp, Buf$) {Swap the arguments or wait a given time to pretend doing
                                                              swapping}
5:     write_from_SC($DB^{shuffl}[g] \Leftarrow encrypt(Temp)$)     {Write down the new $g$-th
                                                              record for $DB_{shuffl}$}
6: end for
7: write_from_SC($DB^{shuffl}[j] \Leftarrow encrypt(Buf)$)     {Write down the record from the
                                                              buffer of the SC}
```

Algorithm 6: The interruptible shuffling algorithm

5.1 A Normalized Measure for the Protocol Complexity

PIR protocols can be differentiated based on their preprocessing per query complexities and processing complexities. For example, a protocol in [AF01,AF02] preprocesses $P^{prep} = O(N^2/m)$ records per query, while exhibiting $P^{proc} = O(m)$ processing complexity. In the next subsection we must be able to compare, for example, this protocol to another one with $P^{prep} = \alpha(N,m)$, $P^{proc} = \beta(N,m)$. Below we define a normalized complexity that provides a single value measure for comparing the complexities of PIR protocols.

Let us fix the query response time of a PIR protocol to $O(m)$. Then, we define the normalized complexity P of a PIR protocol by the complexity of the preprocessing work done per query. Our definition of the normalized complexity does not depend on the communication complexity of the protocol, because we consider the only protocols with optimal communication.

For better understanding of the measurement, assume two PIR protocols. We adjust their processing complexities in such a way that they both exhibit a query response time $O(m)$. Then, the only difference in performance between them is how much time each of them spends on preprocessing per answered query. That is, we assume that at processing phase the same amount of work is done. Then, the protocols can be differentiated on how much preprocessing work per query is required.

5.2 The Measurement

In this section we measure and compare the normalized complexities of (i) the PIR protocol in [AF01,AF02], (ii) the same protocol improved as proposed in Sect. 3, (iii) the same protocol improved as proposed in Sect. 4. Finally, we consider the combination of both improvements and draw the conclusions.

1. The protocol presented in [AF01,AF02] requires the preprocessing of complexity N^2 to be done in order to answer each m queries, providing the

following normalized complexity:

$$P_1 = \frac{N^2}{m}$$

2. The protocol updated as proposed in Sect. 3 requires the preprocessing of complexity $N\sqrt{N}$ to be done in order to answer each m queries:

$$P_2 = \frac{N\sqrt{N}}{m}$$

3. The protocol updated as proposed in Sect. 4 have the same processing complexity in the worst case as the protocol in [AF02]. However, the idea explained in Sect. 4 can be applied to regenerating a usable shuffled database from an unusable one. Namely, after the m queries are executed, the shuffled database is not deleted. Instead, the accessed m records could be shuffled, providing a shuffled database being able to serve for answering additional $m/2$ queries. [3] This approach can be applied several times. The complexity of the protocol can be calculated as follows:

$$P_3 = \frac{N^2 + m^2 + 2(m/2)^2 + 4*(m/4)^2 + ... + 2m^2/m}{m + m/2 + m/4 + m/8 + ... + 1} =$$
$$\frac{N^2 + m^2(1 + 1/2 + 1/4 + ... + 2/m)}{m(1 + 1/2 + 1/4 + 1/8 + ... + 1/m)} \approxeq$$
$$\frac{N^2 + m^2(1 + 1/2 + 1/4 + ... + 1/m)}{m(1 + 1/2 + 1/4 + 1/8 + ... + 1/m)} =$$
$$\frac{N^2}{m(1 + 1/2 + 1/4 + 1/8 + ... + 1/m)} + m = \frac{N^2}{m(2 - 1/m)} + m \approxeq \frac{N^2}{2m} + m$$

Finally, we combine the two proposed modifications together and measure the normalized complexity of the resulting protocol. Namely, we assume that the shuffling complexity is $O(N\sqrt{N})$ (as proposed in Sect. 3), and that the used shuffled databases are refreshed as proposed in Sect. 4. Then, the complexity P_4 of the resulting protocol is calculated similar to P_3:

$$P_4 = \frac{N\sqrt{N} + m\sqrt{m} + 2(m/2)\sqrt{m/2} + 4*(m/4)\sqrt{m/4} + ... + 2m\sqrt{m}/m}{m + m/2 + m/4 + m/8 + ... + 1} \approxeq$$
$$\frac{N\sqrt{N}}{m(1 + 1/2 + 1/4 + 1/8 + ... + 1/m)} + \frac{3\sqrt{m}}{2} \approxeq \frac{N\sqrt{N}}{2m} + \frac{3\sqrt{m}}{2}$$

In summary, as demonstrated on Fig. 5, the approach proposed in Sect. 3 reduces the normalized complexity of the PIR protocol from N^2/m to $N\sqrt{N}/m$, whereas the approach proposed in Sect. 4 reduces the normalized complexity of the protocol by a constant factor.

[3] For details, please refer to Section 4.

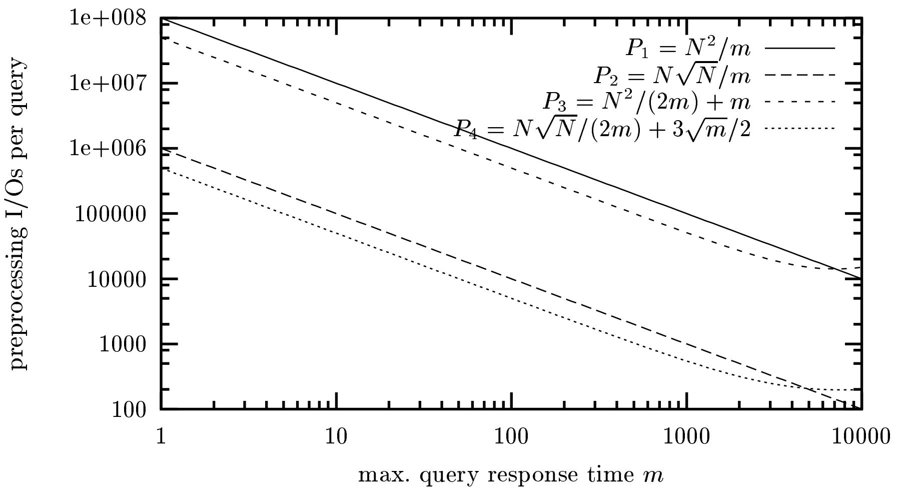

Fig. 5. The normalized complexities of the basic protocol [AF01,AF02], and the two modifications proposed in this paper are presented as functions of the maximal query response time m, for $N = 10000$.

6 Conclusions and Future Work

A recently proposed Private Information Retrieval (PIR) protocol acquire the query response time and the communication constant from the number of records in the database (N), thus being optimal for users [AF01,AF02]. The drawbacks of the protocol are that (i) k records must be read to answer k-th query, (ii) to start with $k = 1$, which is done periodically, the preprocessing of quadratic complexity must be performed. The latter means, N^2 times a block (of size of a database record) is read from secondary storage for the preprocessing to be complete.

We tackled the problem by proposing a preprocessing algorithm with $O(N\sqrt{N})$ complexity. We also show how to cut query response time from k to 2 by applying additional preprocessing online.

We emphasize, that although our preprocessing algorithm is of better complexity in comparison to the previous work, we only proved it's complexity to be optimal within the set of algorithms considered in this paper. Further improvements in the preprocessing complexity might be possible.

References

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

[AF02] D. Asonov and J.-C. Freytag. Almost optimal private information retrieval. In *Proceedings of 2nd Workshop on Privacy Enhancing Technologies (PET2002), San Francisco, USA*, April 2002.

[AS02] D. Asonov and S. Smith. Private communication, April 2002.

[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.

[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.

[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.

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

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

[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 PIR Protocols not Optimal for Users

The related work, that did not appear in Section 2, is presented in the following two sections. A more comprehensive overview on PIR can be found in [Aso01].

A.1 PIR with Secure Coprocessor

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

A secure coprocessor (a tamper-proof device) is used as a black box, where the selection of the requested record takes place. Although hosted at the server side, the SC is designed so that it prevents any one from accessing its memory from outside [SPW98]. A SC can prove what software is installed inside and whether it was changed in the past.

The basic protocol runs as shown in Fig. 6. The client encrypts the query "return 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.

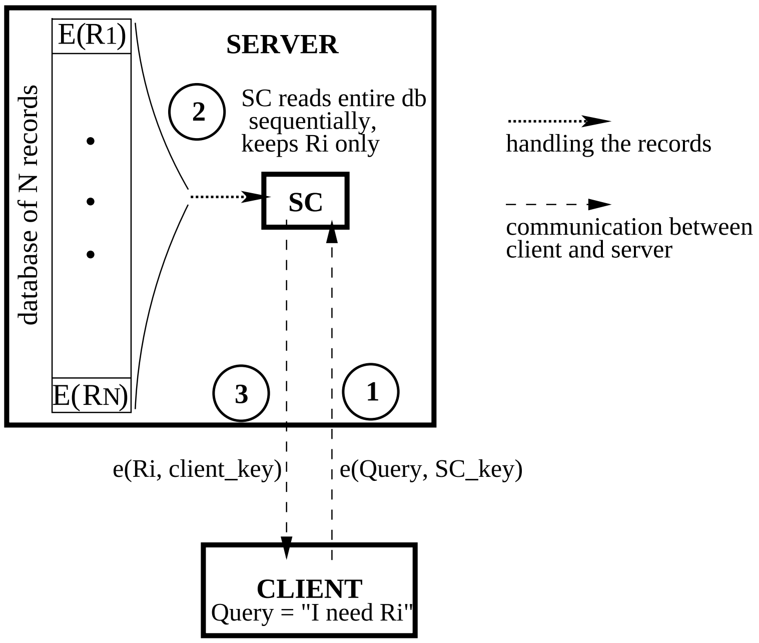

Fig. 6. I/O flows in the PIR with SC in [SS00] (Section A.1).

Pros and Cons. The main disadvantage of this PIR is the same as that of the PIR described in [KO97,CMS99]: $O(N)$ query response time, which is intolerable for large databases.

A.2 PIR with Preprocessing and Offline Communication

Although it does not seem feasible to attain less than $O(N)$ computation per PIR query, one could try to preprocess as much work as possible. Such that, when a query is submitted, it would require only $O(1)$ computation to answer it online.[4]

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.

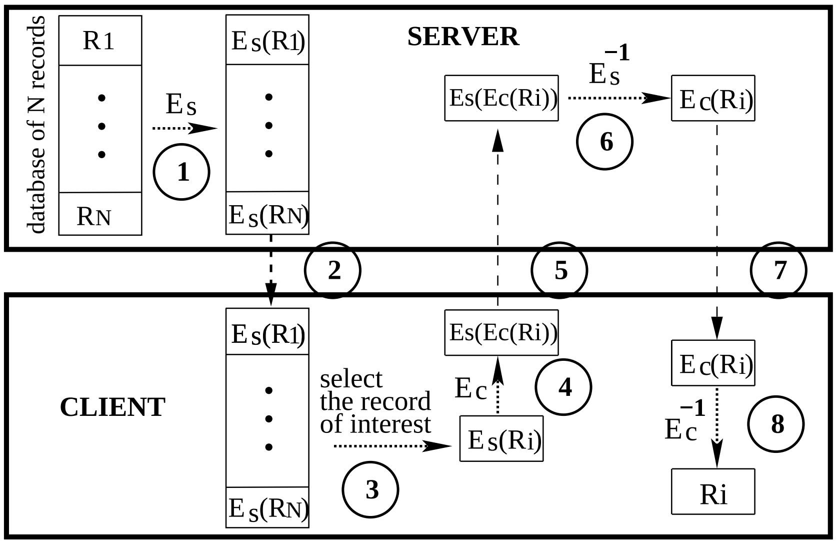

Fig. 7. An example of a PIR protocol with preprocessing and offline communication. Steps 1 and 2 are made offline once, and the other steps are performed online for every query submission.

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. 7).

Pros and Cons. Protocols in [BDF00,SJ00] attain $O(1)$ query response time. However, offline communication comparable to the size of the entire database must be done between client and server to start the protocol.

[4] 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].