Secure Coprocessor-based Private Information Retrieval without Periodical Preprocessing

Peishun Wang[1] **Huaxiong Wang**[2,3] **Josef Pieprzyk**[3]

[1] School of Computer Science and Software Engineering
University of Wollongong, NSW 2522, Australia
Email: peishun@uow.edu.au

[2] Center for Advanced Computing – Algorithms and Cryptography, Department of Computing
Macquarie University, NSW 2109, Australia
Email: hwang, josef@ics.mq.edu.au

[3] Division of Mathematical Sciences, School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore

Abstract

Early works on Private Information Retrieval (PIR) focused on minimizing the necessary communication overhead. They seemed to achieve this goal but at the expense of query response time. To mitigate this weakness, protocols with secure coprocessors were introduced. They achieve optimal communication complexity and better online processing complexity. Unfortunately, all secure coprocessor-based PIR protocols require heavy periodical preprocessing. In this paper, we propose a new protocol, which is free from the periodical preprocessing while offering the optimal communication complexity and almost optimal online processing complexity. The proposed protocol is proven to be secure.

Keywords: Private information retrieval, secure coprocessor.

1 Introduction

A private information retrieval (PIR) protocol allows a user to retrieve a data item of her choice from a database such that the database server does not learn any information on the identity of the item fetched. The problem was formulated by Chor et al. (Chor et al. 1998), and has attracted a considerable amount of attention. The efficiency of PIR protocols is typically measured by their communication complexity and computation overhead necessary to answer a query. Early works on PIR protocols (for both information theoretic and computational models) have been mainly focused on minimizing the communication complexity between the user and the database server. Many proposed PIR protocols (see, for example, (Ambainis 1997, Beimel et al. 2002, Chang 2004, Chor & Gilboa 1997, Gentry & Ramzan 2005, Woodruff & Yekhanin 2005)) succeeded in achieving this goal. However, these protocols have very high computation overheads, requiring the computation on the entire database in order to retrieve a single bit. This results in excessive query response time, and makes them impractical. Sion and Carbunar (Sion & Carbunar 2007) showed that the naive solution (*i.e.*, downloading the whole database) is more efficient than a carefully designed PIR protocol with sophisticated mathematical computation. To address this problem, several attempts have been made, and one of the most efficient solutions is based on a tamper-proof *secure coprocessor* (SC), which prevents anybody from accessing its memory from outside even if the adversary has direct physical access to the device (Smith et al. 1998). Being more specific, the database server installs a secure coprocessor, which works as a user extension at the server side. If the internal memory space of secure coprocessor was large enough to hold the entire database, the PIR problem would be solved easily. A user simply applies existing network protocols like HTTP and SSL to negotiate a secure session with the coprocessor, and makes a query. Since SC is physically protected, no one including the server should be able to observe what the query is. Unfortunately, its internal memory can only hold a fixed and small number of records at a time. Thus, secure coprocessor-based PIR protocols aim to provide private access to a large database while using only a small amount of coprocessor memory space.

Some secure coprocessor-based PIR protocols were proposed in (Asonov & Freytag 4/2002, 5/2002, Iliev & Smith 2003, 2004, 2005, Smith & Safford 2000, 2001, Wang et al. 2006, Yang et al. 2008). They achieved optimal communication complexity and good online processing complexity. However, all these protocols need a heavy preprocessing that periodically shuffles the database. In this paper, we propose a protocol without periodical preprocessing.

Our Contributions. We propose a new secure coprocessor-based PIR protocol that works in the two stages: offline shuffling and online retrieving. During the offline shuffle, the secure coprocessor double-encrypts all records and permutes the original database. During the online retrieval, the secure coprocessor usually reads two records from the shuffled database and writes two records back. We prove the security of the protocol. Unlike the previously published coprocessor-based PIR protocols, the proposed protocol uses two new ideas, namely, double-encryption and twin-writing, and consequently removes the need for periodical preprocessing. The analysis of efficiency indicates that the performance of the proposed protocol is better than the performance of previous coprocessor-based PIR protocols.

Organization. Section 2 reviews the related work. In Section 3 we provide the model. In Section 4 we construct a new protocol and show its security. In

Section 5 we analyze the performance. Finally Section 6 concludes the work.

2 Related Work

Smith and Safford (Smith & Safford 2000, 2001) came up with the idea of using a secure coprocessor (SC) for PIR. In their works, SC preprocesses the database, *i.e.*, it shuffles the records offline before the PIR protocol starts. More precisely, SC reads entire database n times record by record, and each time SC leaves a record in its secure memory and encrypts the record using a secret key known to SC but not to the server, then SC writes the encrypted record in the shuffled database. During the phase of online retrieval, when SC receives a query from a user, it reads through the entire shuffled database, and keeps the desired record in its internal memory. Then SC decrypts the record and sends it via a secure channel to the user. Obviously, the communication complexity is optimal, but the online computation complexity is $O(n)$ and the offline preprocessing complexity is $O(n^2)$.

To minimize the online computation, Asonov and Freytag (Asonov & Freytag 4/2002) modified Smith and Safford's protocol as follows. To answer the first query, SC accesses the desired encrypted record directly instead of reading the entire database. The encrypted record is decrypted inside the SC and sent to the user via a secure channel. To answer the k^{th} ($k \geq 2$) query, SC has to read the $k-1$ previously accessed records first, then the desired one. In case the k^{th} query requests the same record as one of the $k-1$ previous queries, SC reads a random (previously unread) record. Evidently, SC has to keep track of the accessed records. The database server can decide at which point $m = max(k)$ $(1 \leq m \leq n)$ to stop and to switch to shuffle the database again. Since m is a constant independent of n, if the maximal allowed query response time is fixed, m can be chosen without considering preprocessing. Thus, the server processes $O(1)$ records online in order to answer each query. Their protocol improves the online computation complexity from $O(n)$ to $O(n^{1/2})$ (even $O(1)$), and retains the optimal communication complexity. But the offline preprocessing complexity is still $O(n^2)$, and the reshuffling takes place when a fixed number of queries have been answered.

The above two protocols would be optimal from the user point of view, but they are still quite expensive for the server. For example, assuming that accessing one database record takes 0.01 second for SC and $n = 10000$, then we need $n^2 * 0.01 \approx 2$ weeks to prepare one shuffled database and an optimal tradeoff parameter $m = 141$. This means that we need to reshuffle the database once per 141 retrievals. To reduce the offline preprocessing, Asonov and Freytag (Asonov & Freytag 5/2002) modified their protocol. The server first splits each record in the database into p equal parts. As the result, the database is transformed into p share databases. Then SC shuffles all the share databases based on the same algorithm. The authors showed that for an optimal p, it is possible to reduce the preprocessing complexity to $O(n^{1.5})$. This means that $n^{1.5} * 0.01 \approx 3$ hours are needed to prepare a shuffled database. Iliev and Smith used the Beneš permutation networks as the shuffling algorithm to reduce the complexity of database shuffling. They presented protocols (Iliev & Smith 2003, 2005) with the preprocessing complexity of $O(n \lg n)$. Under the same condition, the preparation of one shuffled database needs $n \lg n * 0.01 \approx 23$ minutes. They modified their protocols and proposed a new variant (Iliev & Smith 2004), which requires a smaller internal storage space of size $O(\lg n)$ for shuffling. Furthermore, Wang *et al.* (Wang et al. 2006) and Yang *et al* (Yang et al. 2008) constructed protocols with new shuffling methods and reduced the computational cost for a query. Unfortunately, the problem of periodical switching to a new shuffled database still exists in these protocols.

3 Model

Throughout this paper, we use the following notation.

Let $a \xleftarrow{R} A$ denote choosing an element a uniformly at random from the set A. For an integer n, $[n]$ denotes the set of integers $\{1, 2, \ldots, n\}$. We write $x||y$ to denote the concatenation of the bit-strings x, y. By default, $\lg k \overset{def}{=} \lceil \log_2 k \rceil$. For a database DB with n records, DB_i $(i \in [n])$ denotes the i^{th} record in DB. As in (Asonov & Freytag 5/2002), we assume that it takes several orders of magnitude longer to operate on data in the external storage than to access main memory. Thus, the preprocessing complexity and online processing complexity of a PIR protocol are measured by the number of records accessed on the external storage, *i.e.*, by the number of I/Os.

3.1 The Model

Let's briefly recall the model of secure coprocessor-based PIR. It consists of a single server – the host H, which is connected with a secure coprocessor SC. A database DB is stored on a suitable high-performance storage medium that is a part of H and is separated from SC. DB has n records. If records are not all of the same length, each record is padded up to the same size (the padding contents are determined by the application). Before online queries start, DB is permuted to a new database D. To retrieve a record DB_i $(i \in [n])$, a user sends SC a query specifying the index i via a secure channel. SC then interacts with H, which accesses D instead of DB, to reply.

The adversary in the model attempts to derive information from the PIR protocol execution. Possible adversaries include outside attackers and the host, where the latter is able to not only observe all I/O operations performed for queries, but also can make queries as a legitimate user. The aim of SC is to retrieve a requested record, while hiding the identity of the record from any adversary. This means that the adversary should not be able to determine, which record in the original database has been or has not been requested by a user.

We follow three terms introduced in (Wang et al. 2006), access pattern, stained query and clean query. An access pattern for a time period is an array of the records of D read and written in the period, where the records are arranged chronologically. A stained query means that the plaintext and the index of the desired record are known to the adversary $\mathcal{A}$ without observing any access pattern. A clean query is the one the adversary $\mathcal{A}$ does not know before observing access patterns.

Let $Pr(Q \simeq DB_j)$ denote the probability of the event that the record DB_j is the one desired by the query Q. Note that, in information theoretic or computational PIR protocols, an adversary does not know anything about the queried record by observing access patterns, no matter whether the query is stained or clean. However, in all existing secure coprocessor-based PIR protocols, for a stained query, an adversary might know that the record read from D is the queried one, but for a clean query, she does not. Intuitively, the queried record is selected in the black box (*i.e.*, internal memory of SC). Now if every record of the database DB is in the SC's internal memory with nonzero probability, then any adversary cannot learn which record is the queried one. Based on this observation, we give a definition as follows.

Definition 3.1 *A SC-based PIR protocol is secure, if for a clean query Q and any possible access pattern $\mathcal{A}ccess$, the following relation holds*

$$0 < Pr(Q \simeq DB_j|\mathcal{A}ccess) \leq 1/M, \quad \forall j \in [n],$$

where M is the maximum number of records kept in the SC's internal memory.

Notice that the above definition of security is different from the security definition in the information theoretic or computational PIR protocols, which guarantees that $Pr(Q \simeq DB_j|\mathcal{A}ccess) = 1/n$ for any $j \in [n]$. However, in some coprocessor-based PIR protocols, the probability $Pr(Q \simeq DB_j|\mathcal{A}ccess)$ is not uniformly distributed for all $j \in [n]$. For example, in the protocols from (Asonov & Freytag 5/2002, Iliev & Smith 2005, Wang et al. 2006), if the l^{th} query ($1 \leq l \leq M$) is clean, from the view of an adversary, $Pr(Q \simeq DB_j|\mathcal{A}ccess) = \frac{1}{l(n-l+1)}$ for a record $DB_j \notin \{DB_{i_1}, DB_{i_2}, \cdots, DB_{i_{(l-1)}}\}$, where $\{DB_{i_1}, DB_{i_2}, \cdots, DB_{i_{(l-1)}}\}$ are the records read into SC's internal memory in the previous $l-1$ queries. A protocol satisfying the above definition is called a non-perfect PIR protocol. In the above definition, $Pr(Q \simeq DB_j|\mathcal{A}ccess) = 1/M$ means that DB_j is certainly in SC's internal memory, and $Pr(Q \simeq DB_j|\mathcal{A}ccess) > 0$ for all $j \in [n]$ implies that an adversary does not know, which record of DB is not in SC's internal memory.

4 The Proposed Protocol

As the previous works in (Asonov & Freytag 4/2002, 5/2002, Iliev & Smith 2003, 2004, 2005, Wang et al. 2006, Yang et al. 2008), our protocol consists of two phases: offline preprocessing and online retrieval. The database is permuted offline before any query starts. The offline preprocessing (permutation) is executed once only.

During the offline preprocessing phase, the coprocessor SC permutes the database DB into a new database D. Note that z ($1 < z \ll n$) records are kept in SC's internal memory C, and $n-z$ double-encrypted records are stored in D. SC maintains a table T in its internal memory C. The table T contains n rows and each row T_i ($i \in [n]$) keeps track of the record DB_i. More precisely, each row T_i consists of two fields: a single bit flag T_{i-flag} and an index $T_{i-index}$ that consists of $\lg(n-z)$ bits. There are three possible cases:

1. $T_{i-flag} = 0$ and $T_{i-index} = 0$ – this means that the i^{th} record DB_i is in the internal memory C (It does not exist in the database D);
2. $T_{i-flag} = 0$ and $T_{i-index} = k$ ($k \in [n-z]$) – this means that DB_i is stored in D as the record D_k and this record has never been accessed;
3. $T_{i-flag} = 1$ and $T_{i-index} = k$ ($k \in [n-z]$) – this means that DB_i is stored in D as the record D_k and this record has already been accessed.

During the online retrieval phase, on receiving a query on DB_i from a user, SC locates DB_i by checking T_i. There are three possible cases:

1. If DB_i is in C, then SC reads two records randomly chosen from D into C, where the first read record is accessed, and the second is not.
2. If DB_i is in D and unaccessed, then SC reads one random accessed record and $D_{T_{i-index}}$ into C.
3. If DB_i is in D and accessed, then SC reads $D_{T_{i-index}}$ and one random unaccessed record from D into C.

After sending the desired record DB_i to the user, SC chooses two records at random from C and writes the two records into D at the positions from which the previous two records were taken. Finally, SC changes the values of the corrsponding rows in T. For the first query, SC reads/writes one record from/into D, and for any other query, SC reads two records from D, which was called $twin-reading$ in (Yang et al. 2008), and writes two records into D, which we call $twin-$ *writing*. After $n-z$ queries, all records are treated as the unaccessed again. Note that there are always z records stored in C after SC finishes an online query.

4.1 Offline Preprocessing

SC applies the approach from (Iliev & Smith 2005) to permute the database DB, but the writing operation (*i.e.*, the record $DB_i (i \in [n])$ of the database DB is permuted to the record D_j ($j \in [n-z]$) of the database D) is modified as follows.

Step 1 SC generates two secret keys, k_0 and k_1, and creates a table T of $n \times (1 + \lg(n-z))$ bits, which has n rows $T_i = T_{i-flag}||T_{i-index}$ ($i = 1, \cdots, n$), where T_{i-flag} is a 1-bit flag and $T_{i-index}$ is $\lg(n-z)$ bits for storing an index. Each row T_i is initialized to the value 0, and the table is stored in C.

Step 2 SC uses k_1 to encrypt every record DB_i, appends k_1 to the encrypted record, and then encrypts it again under k_0.

Step 3 The encrypted-appended-encrypted (called double-encrypted) version of the record DB_i is written into the j^{th} entry in D as the record D_j;

Step 4 SC keeps $T_{i-flag} = 0$ and sets $T_{i-index} = j$, and then rewrites T_i in the table T.

Step 5 During the process of permutation, SC selects z records $\{DB_{i_j}\}_{j=1}^{z}$ ($i_j \in [n]$) uniformly at random and stores them and k_0, k_1 in C.

4.2 Online Retrieval

To present the protocol in a convenient way, let $\mathcal{G}$ denote a secret key generator that takes an old key k_s as input to generate a new secret key k_{s+1}. Now we describe two algorithms: **Reading**(j, k_0) and **Writing**(j, k_0, k_s, DB_r).

Reading(j, k_0) is a deterministic algorithm, which takes as input an index j ($j \in [n-z]$) and the secret key k_0, and outputs the corresponding record DB_l ($l \in [1, n]$) (*i.e.*, $T_{l-index} = j$). It works according to the following steps.

1. Reads the record D_j from D into C;
2. Decrypts D_j with k_0, gets a data and the appended secret key k_s ($s \geq 1$), and then decrypts the data with k_s to get the corresponding record DB_l.

Writing(j, k_0, k_s, DB_r) is a deterministic algorithm, which takes as input an index j ($j \in [n-z]$), two secret keys k_0 and k_s and a record DB_r ($r \in [n]$), and outputs a record D_j. It goes through the following steps.

1. Encrypts DB_r with k_s, appends k_s to the encrypted record, and then encrypts the appended-encrypted record with k_0;
2. Writes the double-encrypted version of the record DB_r at the j^{th} entry in D as the record D_j;

During the phase of online retrieval, on receiving the t^{th} query Q_t on the record DB_i from a user, if $t = x(n-z)+1$ for some $x \in \{0, 1, 2, \cdots\}$, SC carries out the **Algorithm 1** (below); otherwise, executes the **Algorithm 2** (below).

Algorithm 1: Single-Reading & Single-Writing

```
check $T_i$ in $T$;
if $T_i = T_{i-flag}||T_{i-index} = 0$ then
    choose $T_j \overset{R}{\longleftarrow} T$ such that $T_j \neq 0$;
    $l = j$
else
    $l = i$
endif
execute the algorithm Reading($T_{l-index}, k_0$) to
put a record $DB_l$ into $C$;
send $DB_i$ to the user as the answer to the query;
take a record $DB_r \overset{R}{\longleftarrow} C$;
$k_{t+1} = \mathcal{G}(k_t)$;
delete $k_t$;
execute Writing($T_{l-index}, k_0, k_{t+1}, DB_r$);
if $r \neq l$ then
    $T_{r-index} = T_{l-index}$;
    reset $T_l = 0$;
endif
set $T_{r-flag} = 1$;
set $T_r$ according to the new values of $T_{r-flag}$
and $T_{r-index}$.
```

Algorithm 2: Twin-Reading & Twin-Writing

```
check $T_i$ in $T$;
if $T_i = T_{i-flag}||T_{i-index} = 0$ then
    choose $T_{j_1}, T_{j_2} \overset{R}{\longleftarrow} T$ such that $T_{j_1-flag} = 1$,
$T_{j_2-flag} = 0$ and $T_{j_2-index} \neq 0$;
else if $T_{i-flag} = 0$ then
        choose $T_{j_1} \overset{R}{\longleftarrow} T$ such that $T_{j_1-flag} = 1$;
        $j_2 = i$;
    else
        choose $T_{j_2} \overset{R}{\longleftarrow} T$ such that $T_{j_2-flag} = 0$
        and $T_{j_2-index} \neq 0$;
        $j_1 = i$;
    endif
endif
execute the algorithm Reading($T_{j_1-index}, k_0$) to
put a record $DB_{j_1}$ into $C$;
execute the algorithm Reading($T_{j_2-index}, k_0$) to
put a record $DB_{j_2}$ into $C$;
send $DB_i$ to the user as the answer to the query;
take two records $DB_r, DB_s \overset{R}{\longleftarrow} C$;
$k_{t+1} = \mathcal{G}(k_t)$;
delete $k_t$;
execute Writing($T_{j_1-index}, k_0, k_{t+1}, DB_r$);
execute Writing($T_{j_2-index}, k_0, k_{t+1}, DB_s$);
take $T_{r-flag} = 1$ and $T_{r-index} = T_{j_1-index}$, and
reset $T_r$;
take $T_{s-flag} = 1$ and $T_{s-index} = T_{j_2-index}$, and
reset $T_s$;
if $r \neq j_1$ then reset $T_{j_1} = 0$;
endif
if $s \neq j_2$ then reset $T_{j_2} = 0$;
endif
if $t = x(n-z)$ for some integer $x$ then
reset $T_{l-flag} = 0$ for all $l \in [n]$;
endif
```

Now we consider the security of proposed protocol.

Theorem 1 *The proposed SC-based PIR protocol is secure according to Definition 3.1.*

Proof:

We use induction on the number N of queries to prove the security. Let $Pr(DB_i \simeq \{D_{j_1}, \ldots, D_{j_m}\})$ ($m \in [n-z]$) denote the probability of the event that DB_i is permuted to one of the records $\{D_{j_1}, \ldots, D_{j_m}\}$, $Pr(DB_i \simeq C)$ denote the probability of the event that DB_i is in C, and $\mathcal{A}ccess(Q)$ denote the access pattern (including reading and writing patterns) for the query Q. Note that, in the proposed protocol, the maximum number of records in C is $M = z+2$.

CASE $N = 1$: the 1^{st} online query Q_1 on DB_{i_1}.

$\mathcal{A}ccess(Q_1)$:

1. Reading Pattern: SC reads a record D_{x_1} from D into C.
2. Writing Pattern: SC chooses a random record DB_r from C and executes the algorithm **Writing**(x_1, k_0, k_1, DB_r) to write a record D_{x_1} into D. Note that the currently written record D_{x_1} is different from the just read record D_{x_1} even though they are permuted from an identical record in DB because of using different encryption keys.

Analysis :

1. Consider that the query is clean. D is permuted from DB in an oblivious way used in (Iliev & Smith 2005) during the offline preprocessing. According to the proof in (Iliev & Smith 2005), although the adversary $\mathcal{A}$ knows that D_{x_1} is read into C, she does not know which record in DB is or is not permuted to D_{x_1}. The original records in C are randomly and secretly selected by SC. So, from the view of $\mathcal{A}$, every record in DB can be located in C with the probability of $\frac{z+1}{n}$. Therefore,
$$Pr(Q_1 \simeq DB_j) = \frac{1}{z+1} \cdot \frac{z+1}{n} = \frac{1}{n}, \ \forall j \in [n].$$
2. By observing the access pattern $\mathcal{A}ccess(Q_1)$, $\mathcal{A}$ knows the following information about the queried record DB_{i_1},
$$Pr(DB_{i_1} \simeq \{D_{x_1}\}) = 1/(z+1) \text{ and}$$
$$Pr(DB_{i_1} \simeq C) = z/(z+1).$$
However, if the query is clean, the information gives $\mathcal{A}$ nothing to identify which record in DB is or not the queried record DB_{i_1}.

Conclusion : For a clean query Q_1, we have

$$0 < Pr(Q_1 \simeq DB_j | \mathcal{A}ccess) \leq 1/M, \quad \forall j \in [n],$$

where $\mathcal{A}ccess = \{\mathcal{A}ccess(Q_1)\}$ is the access pattern.

CASE $N = 2$: the 2^{nd} online query Q_2 on DB_{i_2}.

$\mathcal{A}ccess(Q_2)$:

1. Reading Pattern: SC reads the record D_{x_1} and another record D_{x_2} ($x_2 \neq x_1$) from D into C.

2. Writing Pattern: SC chooses two random records $\{DB_r, DB_s\}$ from C and executes **Writing(**x_1, k_0, k_2, DB_r**)** and **Writing(**x_2, k_0, k_2, DB_s**)** to write the records $\{D_{x_1}, D_{x_2}\}$ into D.

Analysis :

1. Consider that the query is clean. Whatever the first query is, D_{x_1} is read back into C. That means, DB_{i_1} is certainly in C (*i.e.* with the probability of 1). In addition, the records in $D \setminus \{D_{x_1}\}$ are permuted in an oblivious way during the offline preprocessing, $\mathcal{A}$ does not know which record in $DB \setminus \{DB_{i_1}\}$ is or is not permuted to D_{x_2}. So, from the view of $\mathcal{A}$, any other record (except DB_{i_1}) in DB is located in C with the probability of $\frac{z+1}{n-1}$. Therefore,
$$Pr(Q_2 \simeq DB_{i_1}) = \tfrac{1}{z+2}, \text{ and}$$
$$Pr(Q_2 \simeq DB_j) = \tfrac{z+1}{(z+2)(n-1)}, \quad \forall j \in [n] \setminus \{i_1\}.$$
2. By observing the $\mathcal{A}ccess(Q_1)$ and $\mathcal{A}ccess(Q_2)$, $\mathcal{A}$ knows the following information,
$$Pr(DB_{i_1} \simeq \{D_{x_1}, D_{x_2}\}) = 2/(z+2) \text{ and}$$
$$Pr(DB_{i_1} \simeq C) = z/(z+2),$$
$$Pr(DB_{i_2} \simeq \{D_{x_1}, D_{x_2}\}) = 2/(z+2) \text{ and}$$
$$Pr(DB_{i_2} \simeq C) = z/(z+2).$$
However, if the query $q = j$ $(j \in \{1,2\})$ is clean, the information gives $\mathcal{A}$ nothing to identify which record in DB is or not the queried record DB_{i_j}.

Conclusion : For a clean query Q_2, we have
$$0 < Pr(Q_2 \simeq DB_j | \mathcal{A}ccess) \leq 1/M, \quad \forall j \in [n],$$
where $\mathcal{A}ccess = \{\mathcal{A}ccess(Q_1), \mathcal{A}ccess(Q_2)\}$.

CASE $N = 3$: the 3^{rd} online query Q_3 on DB_{i_3}.

$\mathcal{A}ccess(Q_3)$:

1. Reading Pattern: SC reads a record $D_{y_3} \overset{R}{\leftarrow} \{D_{x_1}, D_{x_2}\}$ and another record D_{x_3} $(x_3 \notin \{x_1, x_2\})$ from D into C.
2. Writing Pattern: SC chooses two random records $\{DB_r, DB_s\}$ from C and executes **Writing(**y_3, k_0, k_3, DB_r**)** and **Writing(**x_3, k_0, k_3, DB_s**)** to write the records $\{D_{y_3}, D_{x_3}\}$ into D.

Analysis :

1. Consider that the query is clean.
   (a) The record $D_{y_3} \in \{D_{x_1}, D_{x_2}\}$ is read back into C, this means, DB_{i_1} is in C with the probability of $\frac{z+1}{z+2}$, and so $DB_{i_2} is$.
   (b) The records in $D \setminus \{D_{x_1}, D_{x_2}\}$ are permuted in an oblivious way during offline preprocessing, so $\mathcal{A}$ does not know which record in $DB \setminus \{DB_{i_1}, DB_{i_2}\}$ is or is not permuted to D_{x_3}. Hence, from the view of $\mathcal{A}$, every record in $DB \setminus \{DB_{i_1}, DB_{i_2}\}$ is in C with the probability of $\frac{z+1}{n-2}$.

   Therefore,
$$Pr(Q_3 \simeq DB_j) = \tfrac{z+1}{(z+2)^2}, \qquad \forall j \in \{i_1, i_2\}, \text{ and}$$
$$Pr(Q_3 \simeq DB_j) = \tfrac{z+1}{(z+2)(n-2)}, \quad \forall j \in [n] \setminus \{i_1, i_2\}.$$
2. By observing all access patterns $\mathcal{A}ccess(Q_1)$, $\mathcal{A}ccess(Q_2)$ and $\mathcal{A}ccess(Q_3)$, $\mathcal{A}$ knows the following information,
$$Pr(DB_{i_3} \simeq \{D_{y_3}, D_{x_3}\}) = 2/(z+2) \text{ and}$$
$$Pr(DB_{i_3} \simeq C) = z/(z+2).$$
For a previous query $q = j$ $(j \in \{1,2\})$, $\mathcal{A}$ knows
$$Pr(DB_{i_j} \simeq \{D_{y_3}, D_{x_3}\}) = \tfrac{2(z+1)}{(z+2)^2}, \text{ and}$$
$$Pr(DB_{i_j} \simeq C) = \tfrac{z(z+1)}{(z+2)^2}.$$
However, if the query $q = j$ $(j \in \{1,2,3\})$ is clean, the information gives $\mathcal{A}$ nothing to identify which one in DB is or not the queried record DB_{i_j}.

Conclusion : For a clean query Q_3, we have
$$0 < Pr(Q_3 \simeq DB_j | \mathcal{A}ccess) \leq 1/M, \quad \forall j \in [n],$$
where $\mathcal{A}ccess = \{\mathcal{A}ccess(Q_1), \mathcal{A}ccess(Q_2), \mathcal{A}ccess(Q_3)\}$.

Induction Step: Suppose that the same conclusion holds for **CASE** $N = t$ $(2 < t)$. This is, for the t^{th} query Q_t on DB_{i_t}, we have

Result 1 : For the current query Q_t, $\mathcal{A}$ knows the following information,
$$Pr(DB_{i_t} \simeq \{D_{y_t}, D_{x_t}\}) = 2/(z+2) \text{ and}$$
$$Pr(DB_{i_t} \simeq C) = z/(z+2),$$
where $y_t \in \{x_1, \ldots, x_{t-1}\}$.

For a previous query Q_j on DB_{i_j} $(j \in [t-1])$, $\mathcal{A}$ knows
$$Pr(DB_{i_j} \simeq \{D_{y_t}, D_{x_t}\}) > \tfrac{2z^{t-j}}{(z+2)^{t-j+1}} \text{ and}$$
$$Pr(DB_{i_j} \simeq C) > \left(\tfrac{z}{z+2}\right)^{t-j+1}.$$

Result 2 : If the query Q_t is clean, it holds that
$$0 < Pr(Q_t \simeq DB_j | \mathcal{A}ccess) \leq 1/M, \quad \forall j \in [n],$$
where $\mathcal{A}ccess = \{\mathcal{A}ccess(Q_1), \mathcal{A}ccess(Q_2), \ldots, \mathcal{A}ccess(Q_t)\}$.

Now, we proceed to prove that for $N = t+1$, *i.e.*, the $(t+1)^{th}$ online retrieval, if the query is clean, by observing all access patterns, $\mathcal{A}$ cannot determine which one in the original database DB is or not the queried record.

CASE $N = t+1$: the $(t+1)^{th}$ online query Q_{t+1} on $DB_{i_{t+1}}$.

$\mathcal{A}ccess(Q_{t+1})$:

1. Reading Pattern: SC reads a record $D_{y_{t+1}} \overset{R}{\leftarrow} \{D_{x_1}, \ldots, D_{x_t}\}$ and another record $D_{x_{t+1}}$ $(x_{t+1} \notin \{x_1, \ldots, x_t\})$ from D into C.
2. Writing Pattern: SC chooses two random records $\{DB_r, DB_s\}$ from C and executes **Writing(**$y_{t+1}, k_0, k_{t+1}, DB_r$**)** and **Writing(**$x_{t+1}, k_0, k_{t+1}, DB_s$**)** to write the records $\{D_{y_{t+1}}, D_{x_{t+1}}\}$ into D.

Analysis :

1. Consider that the query is clean.

Case A : $D_{y_{t+1}}$ is read from $\{D_{x_1}, \ldots, D_{x_{t-1}}\} \setminus \{D_{y_t}, D_{x_t}\}$ into C.

(a) Let's treat the dataset $D \setminus \{D_{x_t}\}$ as D, then, this case is the same as the case of $N = t$. According to the **Result 2**, we have
$0 < Pr(Q_{t+1} \simeq DB_j | \mathcal{A}ccess) \leq 1/M$, $\forall j \in [n] \setminus \{i_t\}$,
where $\mathcal{A}ccess = \{\mathcal{A}ccess(Q_1), \mathcal{A}ccess(Q_2), \ldots, \mathcal{A}ccess(Q_t)\}$.

(b) According to the **Result 1**, DB_{i_t} is located in C with the probability of $z/(z+2)$, so we have

$$Pr(Q_{t+1} \simeq DB_{i_t}) = \frac{z}{(z+2)^2}.$$

Case B : $D_{y_{t+1}}$ is read from $\{D_{y_t}, D_{x_t}\}$ into C.

(a) The records in $D \setminus \{D_{x_1}, \ldots, D_{x_t}\}$ are permuted in an oblivious way during offline preprocessing, so $\mathcal{A}$ does not know which record in $DB \setminus \{DB_{i_1}, \ldots, DB_{i_t}\}$ is or is not permuted to $D_{x_{t+1}}$. Hence, from the view of $\mathcal{A}$, every record in $DB \setminus \{DB_{i_1}, \ldots, DB_{i_t}\}$ is in C with the probability of $\frac{z+1}{n-(t \ mod \ n)}$.

(b) The record $D_{y_{t+1}}$ is read from $\{D_{y_t}, D_{x_t}\}$ into C, according to the **Result 1**, we know that, DB_{i_t} is in C with the probability of $\frac{z+1}{z+2}$, and the record DB_{i_j} $(j = 1, \ldots, t-1)$, which is queried by the previous query Q_j, is in C with the probability of at least $\frac{z^{t-j}(1+z)}{(z+2)^{t-j+1}}$.

Therefore,
$Pr(Q_{t+1} \simeq DB_{i_t}) = \frac{z+1}{(z+2)^2}$,
$Pr(Q_{t+1} \simeq DB_{i_j}) \geq \frac{z^{t-j}(1+z)}{(z+2)^{t-j+2}}$ and
$Pr(Q_{t+1} \simeq DB_{i_j}) \leq \frac{1}{z+2}$, $\forall j \in [t-1]$,
$Pr(Q_{t+1} \simeq DB_j) = \frac{z+1}{(z+2)(n-(t \ mod \ n))}$,
$\forall j \in [n] \setminus \{i_1, \ldots, i_t\}$.

2. By observing all access patterns $\mathcal{A}ccess(Q_1), \ldots, \mathcal{A}ccess(Q_{t+1})$, $\mathcal{A}$ knows the probabilities of the events that the current and previous queried records are in different locations, which, however, give $\mathcal{A}$ nothing to identify which one in DB is or not the queried record $DB_{i_{t+1}}$.

Conclusion : For a clean query Q_{t+1}, we have

$$0 < Pr(Q_{t+1} \simeq DB_j | \mathcal{A}ccess) \leq 1/M, \quad \forall j \in [n],$$

where $\mathcal{A}ccess = \{\mathcal{A}ccess(Q_1), \mathcal{A}ccess(Q_2), \ldots, \mathcal{A}ccess(Q_{t+1})\}$.

The proof is concluded.

5 Performance

Let YDDB, WDDB, IS, AF and SS denote the schemes of Yang *et al* (Yang et al. 2008), Wang *et al* (Wang et al. 2006), Iliev & Smith (Iliev & Smith 2003, 2004, 2005), Asonov & Freytag (Asonov & Freytag 4/2002, 5/2002) and Smith & Safford (Smith & Safford 2000, 2001), respectively. Now we give a comparison of our protocol against these secure coprocessor-based PIR protocols as follows.

Table 1. Comparison of Performance

Scheme	OCC	AOPC	OPPC	OSIM
Ours	yes	4	$O(n \lg n)$	4R+$n(1+\lg n)$
YDDB	yes	$O(n/M)$	$O(n \lg n)$	$\sqrt{n}$R+2P
WDDB	yes	$O(n/M)$	$O(n \lg n)$	$\sqrt{2n}$R+1P
IS	yes	$O(n \lg n/M)$	$O(n \lg n)$	$\sqrt{2n}$R+1P
AF	yes	$O(n/M)$	$O(n^{1.5})$	$\sqrt{2n}$R+1P
SS	yes	$O(n)$	$O(n^2)$	1R+1P

* OCC:	optimal communication complexity
* AOPC:	average online processing complexity for a query
* OPPC:	offline preprocessing complexity
* OSIM:	optimal size of SC's internal memory for minimizing query response time
* n:	the database size
* M:	the maximum of records kept in SC's internal memory, $1 < M \ll n$
* R:	a record
* P:	a permutation function

According to the above table, except the parameter OSIM, the performance of our protocol is better than the performance of other existing coprocessor-based PIR protocols. Note that, in our protocol before DB replies to a query, just two records are read. Although online processing complexity is four I/O operations, but for a single query, the user observes a delay equivalent to two I/O operations. Only in the environment of burst queries, the user observes a slightly longer delay.

Now consider the parameter OSIM. It is easy to construct a bijective permutation from a range to another range, so the permutation functions in the schemes of Wang *et al* (Wang et al. 2006), Iliev & Smith (Iliev & Smith 2003, 2004, 2005), Asonov & Freytag (Asonov & Freytag 4/2002, 5/2002) and Smith & Safford (Smith & Safford 2000, 2001) need a small storage space. However, one of two permutations in Yang *et al*'s scheme (Yang et al. 2008) is a bijective function from a set of random numbers to another set of random numbers. It is hard to construct and needs a big storage space. In practice, it is usually replaced with a matching table. So, Yang *et al*'s scheme is less efficient than ours in the parameter OSIM. Comparing with Wang *et al*'s scheme (Wang et al. 2006), if $R \geq \frac{n(1+\lg n)-1P}{\sqrt{2n}-4}$, then ($\sqrt{2n}$R+1P)≥(4R+$n(1+\lg n)$), this means, when the size of a record in the database DB is over $\frac{n(1+\lg n)-1P}{\sqrt{2n}-4}$ bits, our scheme is more efficient than Wang *et al*'s scheme considering the parameter OSIM. For example, assume that a database has one million records, *i.e.*, $n = 10^6$, and a permutation function has a size of 10^6 bits, then, when the record size is over 14175 bits, the internal storage capacity required in our scheme is smaller than that in Wang *et al*'s scheme.

To the best of our knowledge, except the inefficient schemes of Smith & Safford (Smith & Safford 2000, 2001), our protocol is the first one that does not need to reshuffle the database. Moreover, other SC-based PIR protocols have to make preprocessing periodically. Usually one cannot afford implementing inefficient PIR protocols, so something less secure but practical should be used. Comparing to the PIR protocols with uniform distribution, the proposed scheme is more efficient from implementation point of view.

6 Conclusions and Open Problem

In this paper we proposed a new secure coprocessor-based PIR protocol without reshuffling the database and showed its security. The protocol holds the optimal communication complexity, and its online processing complexity is almost optimal.

The *SC*-based PIR protocols assume that the *SC*'s secure memory can store a small number of data records. This assumption does not hold for multimedia databases, where records can be very long and storing even a single (and very long) record in *SC* can be a problem. Thus, task of designing a PIR protocol for databases with very long records is an interesting open problem.

Acknowledgements

Peishun Wang was supported by the RAACE scholarship, Macquarie University. Huaxiong Wang is supported in part by the Australian Research Council under ARC Discovery Project DP0665035 and the Singapore Ministry of Education under Research Grant T206B2204. Josef Pieprzyk was supported by the ARC grant DP0987734.

References

Ambainis, A. (1997), Upper bound on the communication complexity of private information retrieval, *in* 'Proc. of the 24th ICALP'.

Asonov, D., Freytag, J.-C. (4/2002), Almost optimal private information retrieval, *in* 'Proceedings of 2nd Workshop on Privacy Enhancing Technologies (PET2002)', San Francisco, USA.

Asonov, D., Freytag, J.-C. (5/2002), Private information retrieval, optimal for users and secure coprocessors, Technical Report HUB-IB-159, Humboldt University Berlin.

Beimel, A., Ishai, Y., Kushilevitz, E., Rayomnd, J.-F. (2002), Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval, *in* 'Proc. of the 43st IEEE Sym. on Found. of Comp. Sci.'

Chang, Y. (2004), Single Database Private Information Retrieval with Logarithmic Communication. *in* 'The 9th Australasian Conference on Information Security and Privacy (ACISP 2004)', LNCS, 3108, pp. 50-61, Springer-Verlag, Sydney, Australia.

Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M. (1998), Private information retrieval, *in* 'Journal of the ACM', 45. Earlier version in FOCS 95.

Chor, B., Gilboa, N. (1997), Computationally private information retrieval, *In* 'Proceedings of 29th STOC'.

Gentry, C., Ramzan, Z. (2005), Single-Database Private Information Retrieval with Constant Communication Rate, L. Caires et al. (Eds.): *in* 'ICALP 2005', LNCS 3580, pp.803–815.

Iliev, A., Smith, S. (2003), Privacy-enhanced credential services, *in* 'PKI Research Workshop', volume 2, Gaithersburg, MD. NIST.

Iliev, A., Smith, S. (2004), Private Information Storage with Logarithmic-space Secure Hardware, *in* 'Information Security Management, Education, and Privacy'. Kluwer, pp.201–216.

Iliev, A., Smith, S. (2005), Protecting Client Privacy with Trusted Computing at the Server, *in* 'IEEE Security & privacy'.

Sion, R., Carbunar, B. (2007), On the computational practicality of private information retrieval, *in* 'NDSS'.

Smith, S. W., Palmer, E. R., Weingart, S. H. (1998), Using a high-performance, programmable secure coprocessor, *in* 'Proceedings of the 2nd International Conference on Financial Cryptography'.

Smith, S. W., Safford, D. (2000), Practical private information retrieval with secure coprocessors. Technical report, IBM Research Division, T.J. Watson Research Center, 2000.

Smith, S. W., Safford, D. (2001), Practical server privacy with secure coprocessors, *in* 'IBM Systems Journal', 40(3).

Wang, S., Ding, X., Deng, R., and Bao, F. (2006), Private information retrieval using trusted hardware, *in* 'ESORICS', LNCS 4189, pp. 49-64.

Woodruff, D., Yekhanin, S. A. (2005), Geometric Approach to Information-theoretic Private Information Retrieval, *in* 'CCC', pp.275–284.

Yang, Y. Ding, X. Deng, R., Bao, F. (2008), An Efficient PIR Construction Using Trusted Hardware, *in* 'ISC', LNCS 5222, pp. 64–79.