International Journal of Information Security manuscript No.
(will be inserted by the editor)

Xuhua Ding · Yanjiang Yang · Robert H. Deng · Shuhong Wang

A New Hardware-assisted PIR with $\mathrm{O}(n)$ Shuffle Cost

the date of receipt and acceptance should be inserted later

Abstract Since the concept of private information retrieval (PIR) was first formalized by Chor et. al, various constructions have been proposed with a common goal of reducing communication complexity. Unfortunately, none of them is suitable for practical settings mainly due to the prohibitively high cost for either communications or computations. The booming of the Internet and its applications, especially, the recent trend in outsourcing databases, fuels the research on practical PIR schemes. In this paper, we propose a hardware-assisted PIR scheme with a novel shuffle algorithm. Our PIR construction entails $\mathrm{O}(n)$ offline computation cost, and constant online operations and $\mathrm{O}(\log n)$ communication cost, where n is the database size.

Keywords Algorithms · Privacy · Information Retrieval · Trusted Hardware

1 Introduction

Databases, as the storage systems of information, are one of the cornerstones of IT infrastructures. Across the network links of either intranets or the Internet are millions of database queries sent by various users around the world. While database services facilitate information sharing and retrieval, they open a door for attacks on user privacy. If no proper security measure is in place, a database query transaction not only reveals to the server the exact information accessed by the involved user, but allows the server to infer other information about a particular user or even a group of users. The former leakage obviously deprives a database client of her query privacy whereas the latter is likely to pose an bigger threat, as the server is able to derive a group of users' activity pattern by running statistical analyses on the transaction history. Such threats are particularly ominous for databases storing sensitive data, such as patents, medical records and stock quotes. For instance, a query on a patent may imply that the user is pursuing a related idea; stock queries to a database server may indicate that certain stock is exceptionally popular among a group of users sharing a common profile. A malicious server may exploit the privacy exposure due to database transactions to mount attacks against users' interests.

Recent works on keyword search on encrypted data [17,28] do not fully solve the above problem. Searchable encryption indeed helps to hide the data information requested by a user. Nonetheless, it is incapable of hiding users' access patterns. We also note that techniques for anonymous communications, e.g., onion routing [54], Crowds [55] and mix networks [22], do not resolve the above privacy threat either, since these schemes only protect user identities from communication's perspective. None of them addresses user privacy issues related to database transactions.

The right remedy is called Private Information Retrieval (PIR) that enables a user to retrieve data items from a database without revealing any information about her queries. The concept of PIR was first introduced by Chor et. al. in [25]. Since then, various schemes have been proposed. The main objective of previous studies on PIR is to reduce the communication complexity. In Section 2, we provide a systematic review on the security notion of PIR and the landmark results developed in the past decade. Despite all the efforts, a remaining open challenge in this line of research, as pointed out by the panelists in SECURECOMM'06, is how to design a *practical PIR* scheme. This problem remains unsolved as all existing PIR schemes require either impractically high communication cost or computation cost, though the results on asymptotic complexity are attractive. Sion and Carbunar even argue in [57] that a carefully designed PIR scheme with sophisticated cryptographic techniques on popular platforms costs more time delay than the triv-

Robert H. Deng · Xuhua Ding
School of Information Systems, Singapore Management University,

Yanjiang Yang
Institute of Infocomm Research, Singapore,

Shuhong Wang
Sumavision Technologies

ial solution, i.e., transferring the entire database, because of the capability difference between processors and network bandwidth. One of the conclusions of their analysis is that existing PIR solutions are far from being practical.

Our work presented in this paper aims to constructing a more efficient PIR scheme[1], which has not only the lowest communication and computational asymptotic complexities, but also the potential for practical deployment. Besides proposing the PIR protocol, we achieve the following theoretic results:

- We prove the semantic privacy of our construction. It is shown that our scheme exposes no extra user query information to a polynomially-bounded malicious server. The notion of semantic privacy is equivalent to the notion of indistinguishability of query distributions.
- Compared with existing PIR constructions, our scheme achieves the best performance in all aspects: $\mathrm{O}(\log n)$ communication complexity, $\mathrm{O}(1)$ online computation cost and $\mathrm{O}(n)$ offline computation cost, where n is the database size.

Organization

The rest of the paper is organized as follows. We provide a review on private information retrieval in the next section. In Section 3, we define the model and the architecture of our scheme. We then elaborate the algorithm details in Section 4, followed by a formal security analysis in Section 5 and a performance analysis in Section 6. Further discussions from a system perspective are provided in Section 7. We conclude this paper in Section 8.

2 A Review on PIR

The earliest references on "query privacy" date back to Blakely et al [15] and Feigenbaum [31]. Several subsequent papers such as [1,7,8] refined and extended the model in [31]. The first and currently commonly accepted formal notion of PIR was defined by Chor et al [25]. In their formalization, a database is modelled as a n-bit string $x = x_1 x_2 \cdots x_n$, and held by one or multiple servers. A use sends to the server(s) a query $q(i)$ to retrieve x_i. The privacy is defined as the indistinguishability of $q(i)$ and $q(j)$ for any two indexes i and j. Based on this formalization, many results have been produced in recent years.

We group existing PIR schemes roughly into three categories: information theoretical PIR, computational PIR and hardware-based PIR, according to the models in use. Information-theoretical PIR schemes provide the strongest security notion, where any two query distributions are identical. Such a security is at the cost of a high communication complexity, which can only be reduced by replicating the database to multiple servers and assuming no collusion among them. To further significantly reduce the asymptotic complexity, the concept of computational PIR was proposed whereby security is traded for efficiency. A computational PIR scheme ensures that the distributions of two queries cannot be differentiated within polynomial time. Though offering a less stronger privacy notion, the computational PIR schemes, using a single copy of the database, are comparatively more practical than their predecessors. Nonetheless, the computation complexity remains high. Hardware-based PIR schemes are one of the ongoing efforts aiming at further reducing the computation cost. We observe that all those studies have one common objective: to reduce the communication cost or computation cost, or both. The ultimate goal is to design a *practical* PIR scheme. We provide an abridged discussion on these three categories. Interested users are referred to [32] and [4] for more complete coverages.

2.1 Information-theoretical PIR

Information-theoretical PIR refers to PIR schemes that achieve *information theoretical privacy*, i.e., user privacy against computationally unbounded adversaries. A naive solution is for the server to return the entire database to the user. The downside of this solution is its $O(n)$ communication complexity, which is measured by the number of bits transmitted between the user and the server per query. Therefore, the major challenge in PIR design has been to minimize the communication cost. For a database of n bits, the upper bound of the communication, derived from the naive solution, is $\mathrm{O}(n)$. It also seems straightforward to derive the lower bound as $\mathrm{O}(\log n)$, since the user has to provide an index at least. In fact, Chor et. al. [25] proved that for any single-server information-theoretical PIR, the lower bound of communication complexity is also $\mathrm{O}(n)$.

One way to break the $\mathrm{O}(n)$ barrier is to have multiple servers, each holding a copy of the database with the assumption of no server collusion. In the landmark paper [25,26], Chor et al. presented several schemes for k database servers. They differentiate scenarios where k is small, e.g. $k = 2$ from those where k is large. When $k = 2$, their scheme has communication complexity of $\mathrm{O}(n^{1/3})$. To illustrate their idea, we start by explaining their basic two-server PIR model. In this construction, a user selects a uniformly random subset $S_0 \in_{\mathcal{R}} 2^{[1,..,n]}$, where $2^{[1,..,n]}$ denotes the power set of $\{1, 2, ..., n\}$. She also prepares another subset S_1 such that $S_1 = S_0 \cup \{i\}$ if $i \notin S_0$; Otherwise $S_1 = S_0 \setminus \{i\}$. S_0 is sent to the database D_0 and S_1 is sent to the second database D_1. For $b = 0, 1$, the database D_b replies with A_b: $A_b = x_{c_1} \oplus x_{c_2} \cdots \oplus x_{c_{k_b}}$, where k_b is the cardinality of set S_b and $c_j \in S_b$ for all $1 \leq j \leq k_b$. Namely, each database computes an exclusive-OR result over all the bits indicated in the received subset. The user computes x_i as $A_0 \oplus A_1$. It is

[1] A preliminary version of this work [60] appeared in ESORICS'06.

obvious that neither of the two servers has obtained any information about i, as each of them only obtains a uniformly distributed subset. Note that this scheme does not reduce the communication complexity since the user still needs to send O(n) bits. Nonetheless, it leads to a construction of a communication complexity $\mathrm{O}(kn^{1/\log k})$ for k servers. The key idea here is to represent the database as a $\log k$ dimensional cube and each dimension is treated as in the same aforementioned manner. With the same communication cost, the number of servers can be reduced by using *covering codes*, a technique from coding theory, which allows two servers to emulate 8 servers. Thus, the complexity for two-server model is further reduced to $\mathrm{O}(n^{1/3})$.

For general cases with large k, Chor et al constructed a scheme with communication complexity of $\mathrm{O}(k^2 \log kn^{1/k})$ by using low-degree polynomial interpolation. In this scheme, for each query on i, a client designs a polynomial function $F()$ of degree at most $k-1$ such that $F(i) = x_i$. Then the client gathers k points from the servers, which enable her to recover x_i. This complexity were reduced to $\mathrm{O}(2^{k^2} n^{1/(2k-1)})$ in [2] when k is not big. Further improvements in [41,42] cut the complexity to $\mathrm{O}(f(k).n^{1/(2k-1)})$, where $f(k)$ is a linear function of k. Beimel et al improved the k-database model in [10], where the communication complexity is only $n^{\mathrm{O}(\frac{\lg\lg k}{k \lg k})}$. The improvement comes from their recursive PIR construction. They first design a PIR protocol $\mathcal{P}$ with a key feature that each answer from a server is composed of multiple sub-answers and each sub-answer is known to several servers. Built on top of $\mathcal{P}$, they build a recursive PIR $\mathcal{P}'$ providing the claimed complexity. In $\mathcal{P}'$, a client queries the servers as in $\mathcal{P}$. However, the servers do not reply with long answers as in $\mathcal{P}$. Instead, the client, together with those servers with a common sub-answer, execute another PIR protocol so that the client is able to retrieve the bit from this sub-answer with less communication overhead. Recently, Woodruff and Yekhanin [63] further reduced the communication complexity to $\mathrm{O}(\frac{k^2}{t} \log kn^{1/\lfloor(2k-1)/t\rfloor})$, where t is the maximum number of malicious servers.

All the PIR schemes discussed above treat the database as a n-bit binary string. A different model is considered in [25] where the database is composed of blocks of equal length. Each time, a user retrieves an entire block at the communication cost of $\mathrm{O}(l(\frac{n}{l}+1)^{\frac{1}{k}})$, where l is the bit length of the block, n is the number of blocks in the database, and k is the number of database copies. In [24], the database is modeled as a set of n keywords. Though the structure of the database is unknown to users, a user is still able to perform a *private* search on the database with communication complexity $\mathrm{O}(n+l)$. The desired keyword is returned if only if it is stored in the database.

2.2 Computational PIR

The aforementioned information-theoretical PIR provides privacy against computationally unbounded adversaries. Nonetheless such an adversary model might be unnecessarily strong in practical settings. Another critical problem with information-theoretical PIR is its need for multiple database servers and the assumption that they do not collude. This assumption, unfortunately, is hard to hold in reality, as the copies of the database are often under the same administration. These standing issues motivate researches on *computational PIR* whereby the adversaries' computation power is polynomially-bounded and, usually, a single database is used.

Two computational PIR schemes were independently proposed in STOC'97. One is due to Chor and Gilboa [23]. This scheme is a 2-server construction with communication complexity $\mathrm{O}(n^\epsilon)$, for any $\epsilon > 0$, assuming the existence of pseudo-random generators [38], or equivalently the existence of general one-way functions. The other is due to Ostrovsky and Shoup [52]. This scheme allows both private read and write, using two servers which may keep *different* data. Assuming the existence of one-way trapdoor permutation, the communication complexity is $\mathrm{O}(g^{\mathrm{O}(1)}(\log n)^{\mathrm{O}(1)})$, where g is a security parameter.

The first single server computational PIR was proposed in [46] based on the assumption on the intractability of the quadratic residuosity problem [37]. The database in this scheme is viewed as an $r \times c$ bit matrix M. To retrieve the bit $M_{a,b}$ at position (a,b), a user runs the following algorithm.

1. The user generates a large RSA modulus N such that factorization of N is computationally hard. She then chooses c random numbers $y_1, y_2, ..., y_c \in \mathbb{Z}_N^*$ such that: y_b is a quadratic non-residue in $\mathbb{Z}_N^*$ whereas y_j is a quadratic residue in $\mathbb{Z}_N^*$ for all $1 \leq j \leq c$ and $j \neq b$. Both N and $\{y_1, \ldots, y_c\}$ are sent to the database while the trapdoor of factorization of N is kept secret to the user.
2. For the υ-th row, $1 \leq \upsilon \leq r$, the database computes $z_\upsilon \in \mathbb{Z}_N^*$ as follows: for $1 \leq j \leq c$, it computes $w_{\upsilon,j} = y_j^2 \bmod N$ if $M_{\upsilon,j} = 0$; otherwise $w_{\upsilon,j} = y_j$. Then it calculates $z_\upsilon = \prod_{j=1}^{c} w_{\upsilon,j} \bmod N$. Finally, $z_1, ..., z_r$ are sent to the user.
3. The user retrieves z_a and ignores the rest. She sets $M_{a,b} = 0$ iff z_a is a quadratic residue in $\mathbb{Z}_N^*$. Note that the user can compute $M_{a,b}$ efficiently since she knows the factorization of N.

The correctness of the protocol is straightforward. It is interesting to observe that the process of retrieving z_a from $\{z_1, \ldots, z_r\}$ is in fact a naive PIR scheme where the entire database $\{z_1, \cdots, z_r\}$ is transferred to the user. Based on this observation, the communication cost can be further reduced by recursively applying the same approach for retrieving z_a. This yields a more efficient PIR scheme with $O(n^\epsilon)$ communication complexity for any $\epsilon > 0$. The same method was used in [64] which has the same complexity based upon the hardness of subgroup membership problem. Assuming the intractability of Φ-hiding problem, Cachin et al. [21] obtained a probabilistically correct PIR scheme of polylogarithmic communication complexity. Note that this complexity is almost optimal, since even without the privacy requirement, the communication overhead is at least $\mathrm{O}(\log n)$. This result was further improved in [44] as an application of secure game with polynomial expressions, whereby the correctness of PIR is deterministic while the communication complexity remains in the order of polylogartithm. Another improvement on [21] is due to Limppa [47]. Limppa claimed that the scheme in [21] is only of theoretic virtue, since when $n \leq 2^{40}$, it requires even more communication than transferring the entire database. This problem is overcome in [47] by using the length-flexible additively homomorphic public-key cryptosystem [29]. The new scheme, with cheaper computation overhead as well, reduced the complexity to $\mathrm{O}(\log^2 n)$ for practically any value of n. Under the assumption of the existence of one-way permutations, which is much weaker than the assumptions in [21,44,47], Kushilevitz and Ostrovsky [45] demonstrated that there exists a single-server computational PIR with $\mathrm{O}(n - \frac{cn}{k})$ communication complexity, where c is a constant and k is the security parameter.

2.3 Variants of information-theoretic/computational PIR

We now review a number of variants of information-theoretic PIR and computational PIR schemes, which have various interesting features.

Resilient Information Retrieval Since many PIR schemes rely on database replication, it is worthwhile to consider the reliability of the server group. The so-called t-PIR schemes [25,41,16,9] deal with Byzantine failures where a collusion of (up to) t out of k database servers may manipulate their replies in order to compromise user privacy. The best communication performance, due to [9], is $\mathrm{O}(n^{1/\lfloor(2k-1)/t\rfloor})$. The non-Byzantine failure of servers was studied by Beimel and Stahl in [12] where a portion of servers could be faulty, e.g. one may fail to respond to a user's query. By making use of perfect hash families [48], Mehlhorn first showed how to transform a regular PIR scheme into a robust PIR. He then utilized Shamir's secret sharing scheme to construct a robust PIR with $\mathrm{O}(n^{1/3}\log k)$ communication complexity, in which at least 2-out-of-k servers are able to respond correctly.

Symmetric PIR Symmetric PIR are PIR schemes that not only preserve user privacy, but protect the secrecy of a database against users. Gertner et al. [34] showed that for any $k \geq 2$, there exists a k-server symmetric PIR protocol with communication complexity $\mathrm{O}(n^{1/(2k-1)})$ against honest-but-curious users; and a $\lceil \log n+1 \rceil$-server symmetric PIR protocol with communication complexity $\mathrm{O}(\log^2 n \log\log n)$ against dishonest users. Symmetric PIR was also studied in the setting of computational PIR model [46,51,49]. For example, Mishra et al. [49] showed that there exists a single-server symmetric PIR scheme against honest-but-curious users, with communication complexity $\mathrm{O}(n^\epsilon)$ where ϵ is a parameter of the underlying assumption on the intractability of quadratic residuosity problem. Remarkably, 1-out-of-n *Oblivious Transfer* (OT) (e.g., [19,20]) and symmetric PIR have the equivalent security implications, though they have different origins. PIR originates in the privacy concerns in database applications whereas OT was initially investigated as a cryptographic building block for secure multi-party computations.

PIRs with Preprocessing Another key performance metric that has not been discussed so far is computation complexity. All the aforementioned PIR schemes require high computation cost at the server end. Beimel et al. [11] proved that the expected computation of the server(s) is $\Omega(n)^2$. They further suggested to offload part of the computation workload to *offline* by means of preprocessing and pre-retrieving at the cost of additional storage space. In particular, they constructed for any $k \geq 2$ and $\epsilon > 0$: (1) a k-server protocol with $\mathrm{O}(n^{1/(2k-1)})$ communication, $\mathrm{O}(n/(\lg n)^{2k-2})$ computation, and $\mathrm{O}(n^{1+\epsilon})$ storage; (2) a k-server protocol with $\mathrm{O}(n^{1/k+\epsilon})$ computation and communication, and $n^{\mathrm{O}(1)}$ storage; (3) a computational k-server protocol with $\mathrm{O}(n^\epsilon)$ communication, $\mathrm{O}(n^{1/k+\epsilon})$ computation, and $n^{\mathrm{O}(1)}$ storage; (4) a protocol with a polylogarithmic number of servers, polylogarithmic communication and computation, and $\mathrm{O}(n^{1+\epsilon})$ storage.

Other Variants To reduce the security risks due to external attacks on replicated database servers, Gertner et al. [33] proposed a new paradigm of replication, where a combination of each server's database share yields the original database while individual shares reveal no useful information. DiCrescenzo et al. [30] considered minimizing *direct communications* between a user and the server(s) by introducing a third party to facilitate retrieval. This gives rise to information-theoretical PIR as well as computational PIR with communication complexity $\mathrm{O}(\log n)$ between the user and the server(s). Other results include quantum PIR as shown in [43,13]. Recently, Boneh et. al. [18] proposed a PIR scheme on encrypted data, by using a new homomorphic public key encryption scheme couple with Bloom filters.

[2] Ω is the notation for asymptotic lower bound. $f(n) = \Omega(g(n))$ if there exists a positive constant c and a positive integer n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$.

2.4 Hardware-based PIR

Hardware-based PIR is an alternative to the idea of preprocessing PIR aiming to improve performance of PIR. It is inspired by the seminal work ORAM [35]. The first hardware-based PIR was introduced by Smith and Safford in [58], where the database server is equipped with a tamper- resistant hardware such as an IBM 4758 secure coprocessor to assist handling user queries. Residing inside the server, the hardware functions in a self-contained execution environment with its own processor and secure storage. The scheme in [58] only managed to reduce the communication cost. Upon each query, the hardware reads all the data items from the database and returns the requested one to the user. Following this line, Asonov proposed a scheme in [5], which has the optimal communication cost and the online computation cost, and only requires $O(n\sqrt{n})$ time to shuffle the entire the database. Another improvement was due to Iliev and Smith [39,40]. In this scheme, the database is encrypted and secretly shuffled by the hardware, so that the server is unable to link an encrypted entry with its original form and position. Their scheme takes advantage of the co-processor's cache which is able to store at most β database entries. For each database read, the touched item is stored into the cache. For each user query, its online process only requires one database read. Specifically, if the requested item is in cache, a new random item is read into the cache; otherwise, the corresponding one is retrieved. Thus, only one read is needed in all cases. Nonetheless, after every β queries, the database has to be secretly reshuffled, which requires a computation cost $\mathrm{O}(n \log n)$. Note that the user's communication cost is optimal, i.e. $\mathrm{O}(\log n)$. Essentially, the approach in [39,40] offloads the normal computation cost offline in a batch process manner. The security of a hardware-based PIR depends on the strength of the underlying encryption scheme of the database. Therefore, computational security is offered in any practical setting.

Recently, Williams and Sion [61] made an algorithmic improvement on ORAM and constructed a PIR scheme with $\mathrm{O}(\log^2 n)$ computation complexity. The complexity was further improved to $\mathrm{O}(\log n \log\log n)$ in [62]. However, both schemes are at the cost of $\mathrm{O}(\sqrt{n})$ trusted storage. Moreover, the big-O notation of their complexity hides a big constant factor. If the database is not large, e.g. $n < 2^{20}$, the actual operation counts for both the poly-logarithm algorithm in [35] and its derivative in [61, 62] are even larger than the shuffle-based algorithms [39, 40, 5] and our scheme.

Remark

As evident from the discussions above, the main spirit of PIR research is to improve its efficiency. The researches on PIR over the past decade demonstrate a noticeable trend: the focus of research has drifted from reducing the asymptotic communication complexity to reducing the computation complexity. Apparently, the theoretical work on communication complexity seems fully developed, since the best result, polylogarithm complexity, is quite close to the optimal and is practical for real application settings. Nonetheless, the computation complexity remains too high for practical use. It is straightforward to observe that the computation complexity should be at least $\mathrm{O}(n)$ in the conventional PIR model. Otherwise, the server is able to compromise the privacy by analyzing those data items not involved in computation.

Sion et al in [57] study the computational practicality of PIR. They consider both the trivial PIR (i.e., transferring the whole database) and the one in [46], and estimate the actual *time expense* in seconds. Their results show that, in terms of time overhead, the existing computational PIR schemes are inferior to the trivial solution[3]. They even observe that this conclusion will still hold within at least two more decades. The key reason to this somewhat discouraging observation is the disparity between CPU performance in executing large number operations and network speed for data transmission. The analysis in [57] will drive more efforts to study how to reduce the computation cost. It remains an open problem to construct an efficient PIR scheme with practically affordable computation cost and communication cost.

In the end, we organize those major PIR constructions, as well as our scheme proposed in this paper, into a family tree shown in Figure 1. Unless explicitly specified, the complexity in the figure refers to communication cost.

3 Model and Architecture

3.1 Database Model and Permutation

A database $\mathcal{D}$ is modelled as an array of data items of equal length[4], represented by $\mathcal{D} = [d_1, d_2, \cdots d_n]$, where $\mathcal{D}[i] = d_i$ is the i-th item in its original form, for $1 \leq i \leq n$. We use π to denote a permutation of n integers $(1, 2, \cdots, n)$, and π^{-1} to denote its inverse. For $1 \leq i \leq n$, the image of i under π is denoted by $\pi(i)$. For example, let $n = 4$ and $\pi = (1324)$, which means $\pi(1) = 1, \pi(2) = 3, \pi(3) = 3, \pi(4) = 4$. Let $\mathcal{D}_{\pi,k}$ denote the resulting database by encrypting and permuting $\mathcal{D}$ with k and π. The relation between $\mathcal{D}_{\pi,k}$ and $\mathcal{D}$ is that the i-th element of $\mathcal{D}_{\pi,k}$ is the encryption of the $\pi(i)$-th element of $\mathcal{D}$. Specifically,

$$\mathcal{D}_{\pi,k}[i] = E_k(\mathcal{D}[\pi(i)]) = E_k(d_{\pi(i)})$$

[3] Note that this does not depreciate the value of the previous works in PIR research. Asymptotic complexity of a protocol only describes how the protocol cost rises when the problem size grows. It does not indicate its absolute cost.

[4] If necessary, we use padding for those data items with different length.

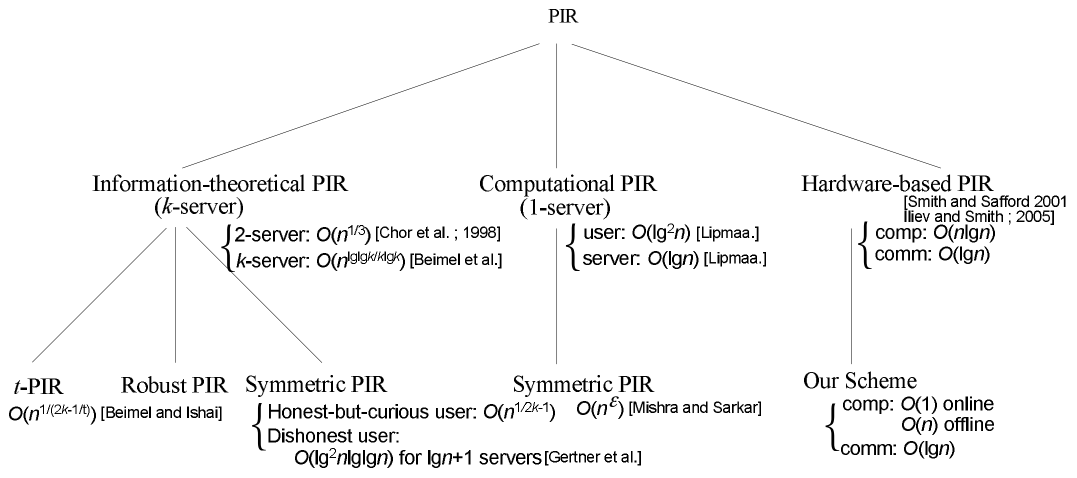
PIR
Information-theoretical PIR (k-server)
2-server: $O(n^{1/3})$ [Chor et al. ; 1998]
k-server: $O(n^{\lg\lg k/k\lg k})$ [Beimel et al.]
Computational PIR (1-server)
user: $O(\lg^2 n)$ [Lipmaa.]
server: $O(\lg n)$ [Lipmaa.]
Hardware-based PIR
[Smith and Safford 2001 Iliev and Smith ; 2005]
comp: $O(n\lg n)$
comm: $O(\lg n)$
t-PIR
$O(n^{1/(2k-1/t)})$ [Beimel and Ishai]
Robust PIR
Symmetric PIR
Honest-but-curious user: $O(n^{1/2k-1})$
Dishonest user:
$O(\lg^2 n\lg\lg n)$ for $\lg n+1$ servers [Gertner et al.]
Symmetric PIR
$O(n^{\varepsilon})$ [Mishra and Sarkar]
Our Scheme
comp: $O(1)$ online
$O(n)$ offline
comm: $O(\lg n)$

Fig. 1 A family tree of PIR schemes and their complexities

where $E_k()$ denotes a symmetric encryption function using a secret key k. In order to highlight the correlation between the two databases and to simplify the presentation, we use $\mathcal{D}_\pi[i] \simeq d_{\pi(i)}$ hereafter as an abbreviation of the previous equation by removing the notations for encryption. decryption will be applied to every database read, Further, we use the term *data items* to denote the original $d_1, d_2, \cdots d_n$ in $\mathcal{D}$ and *data records* to denote their encrypted forms in $\mathcal{D}_\pi$.

3.2 Architecture of Our PIR Scheme

Figure 2 depicts the architecture of our proposed PIR system, whose participants include multiple users and a single server hosting database $\mathcal{D} = [d_1, \cdots, d_n]$. Embedded in the server is a trusted hardware denoted by $\mathcal{TH}$, which consists of a processor and a memory chip. $\mathcal{TH}$ is capable of performing symmetric key encryptions, generating pseudo-random numbers and accessing the main memory of its host. A typical example is the latest secure co-processor IBM PCIXCC[5] [3]. The hardware is tamper-proof so that its internal states and computations is not accessible to adversaries. However, the hardware's accesses to the server's memory or disk can be observed by the server. To avoid confusion between the hardware's memory and the server's memory, we use *cache* to refer to the former. This trusted hardware

The users are interested in retrieving items from the database. The trusted hardware securely handles user queries by executing the PIR algorithm described in Section 4. In brief, the hardware maintains a permuted and encrypted database $\mathcal{D}_\pi$ at the server's memory. A user queries the database by interacting with the hardware via a secure channel, e.g. an SSL connection. To retrieve the i-th data item of $\mathcal{D}$, a user sends the index i to the hardware through the secure channel. Upon receiving i, the hardware computes the index for the corresponding record in the database $\mathcal{D}_\pi$ and retrieves it accordingly. (Note the different implication between a database item and a database record.)

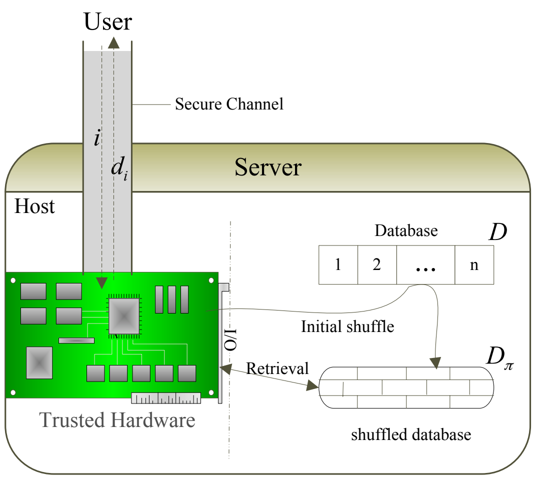

Fig. 2 A PIR system using trusted hardware

3.3 Security Definitions

Access Pattern: Essentially, the access pattern models all the information that is directly observed by the adversary from the query execution. The information includes the what is retrieved from the database and from what position. As in [35], the access pattern for a time period is defined as $\mathbb{A} = [a_1, \cdots, a_N]$, where N is the total number of database accesses during the period and a_i denotes the i-th retrieval from the shuffled database. Suppose that

[5] IBM PCIXCC is connected to a host through a PCI bus. It consists of a PowerPC 405GPR processor operating at 266MHz, 64MB DRAM and 16MB EPROM for persistent data. It can perform 128 AES encryptions at the rate of 185MB/s.

a_i is the second record in the encrypted and permuted database $\mathcal{D}_{\pi,k}$, namely $a_i = \mathcal{D}_{\pi,k}[2]$. By enclosing a_i in $\mathbb{A}$, we mean that the adversary observes that the second record is retrieved and what the record is. However, the adversary does not know what is encrypted within the record.

Note that for PIR schemes without using trusted hardware, a query's access pattern is independent of other queries. In contrast, for hardware-assisted PIR, such as our scheme, the access pattern of a query execution varies with preceding queries and all the past coin tosses by the hardware.

Stained Query and Clean Query: We differentiate two types of queries based on an adversary's prior knowledge of its private information. A query is stained if its content, e.g. the index of the requested data item in the original database $\mathcal{D}$, is known to the adversary without observing the access pattern. This may occur in several scenarios. For instance, a query is compromised or revealed accidentally; or the query could be originated from the adversary herself. On the other hand, a query is clean if the adversary does not know its content before observing the access pattern.

Adversary: We consider a *probabilistic polynomially bounded* adversary who attempts to derive private information from user queries. Possible adversaries include both outside attackers and the server (note that we do not assume any trust upon the server). The adversary in our model is more powerful than an eavesdropper. They are not only able to observe all the inputs and outputs of the database, but allowed to query the database in the same manner as a honest PIR user does. Note that the attacks upon the trusted hardware are dismissed since they are out of the scope of this paper.

Security Model: Following the security notion in Oblivious RAM [35], we measure the information leakage from PIR query executions. A secure PIR scheme ensures that a PPT adversary $\mathcal{A}$ does not gain advantages in determining the distribution of queries from the PIR protocol execution, where $\mathcal{A}$ is allowed to freely ask queries. We use $\mathcal{A}^{\mathsf{QUE}}$ to model that $\mathcal{A}$ is given access to a query oracle QUE, through which the adversary freely queries the database at any time. This implies that an arbitrarily mixed sequence of stained queries and clean queries are executed, and $\mathcal{A}$ observes all access patterns.

We consider a (original) database $\mathcal{D}$ and the corresponding shuffled and encrypted version $\mathcal{D}_\pi$. Let Q be the random variable representing a query, whose value is denoted by $q \in [1, n]$. Then $Q = q$ denotes that query Q is accessing on the q-th item in $\mathcal{D}$. Let A be the random variable representing a read access in $\mathcal{D}_\pi$ for replying Q, whose value is denoted by $a \in [1, n]$. Then $A = a$ denotes that the access is on the a-th record of $\mathcal{D}_\pi$.

Definition 1 Let κ be a security parameter. For a database $\mathcal{D} = [d_1, d_2, \cdots, d_n]$ and a corresponding shufffled and encrypted database $\mathcal{D}_\pi$, a PIR scheme in our model is secure, if and only if for any PPT $\mathcal{A}$, there exists a PPT $\mathcal{A}'$ such that for any clean query $Q = q \in [1, n]$, whose read access in $\mathcal{D}_\pi$ is $a \in [1, n]$, the following holds,

$$|\Pr[\mathcal{A}^{\mathsf{QUE}}(a) = q] - \Pr[\mathcal{A}'^{\mathsf{QUE}}(1^\kappa) = q]| < \epsilon(\kappa)$$

where $\epsilon(\kappa)$ is a negligible function on κ. Intuitively, the definition states that by observing access patterns, the adversary $\mathcal{A}$ gains negligible advantages in figuring out the item a clean query retrieves, over the guess based on his a-priori knowledge.

4 The PIR Scheme

We start by listing the notations used throughout this paper in Table 1.

Table 1 Notations

Notation	Description
$\mathcal{TH}$	The trusted hardware embedded in the server.
β	The maximum number of data items stored in the cache of $\mathcal{TH}$.
$\mathcal{D}$	The original database in the form of $(d_1, d_2, \cdots, d_n)$.
$\pi_0, \pi_1, \cdots$	A sequence of secret pseudorandom permutations of n elements $\{1, 2, \cdots, n\}$.
$\mathcal{D}_{\pi_s}$	A permuted/shuffled database of $\mathcal{D}$ using permutation π_s such that $\mathcal{D}_{\pi_s}[j] \simeq d_{\pi_s(j)}$, for $1 \leq j \leq n$, where $\mathcal{D}_{\pi_s}[j]$ denotes the j-th record in $\mathcal{D}_{\pi_s}$.
a_i	The retrieved data record by $\mathcal{TH}$ during its i-th access to a shuffled database.
$\mathcal{A}$	The access pattern comprising all the retrieved records $(a_1, \cdots, a_N)$ during a fixed time period.
$\mathcal{A}_s$	The access pattern comprising all the retrieved records during the s-th session.
Γ	The sorted list of (original) indices of all data items stored in the cache.

4.1 System setup

We consider applications where a trusted third party (TTP) is available to initialize the system. This TTP is involved only in the initialization phase and then stays offline afterwards. For those scenarios where a TTP is not available, an alternative solution is provided in Section 7.

TTP secretly selects a random permutation π_0 and a secret key sk_0. It permutes the original database $\mathcal{D}$ into $\mathcal{D}_{\pi_0}$, which is encrypted under sk_0, such that $\mathcal{D}_{\pi_0}[j] \simeq d_{\pi_0(j)}$ for $j \in [1, n]$. $\mathcal{D}_{\pi_0}$ is then delivered to the server. TTP secretly assigns π_0 and sk_0 to $\mathcal{TH}$, who then stores π_0 and sk_0 in its cache. TTP also initializes a set of SSL parameters including the SSL public/private key pair for $\mathcal{TH}$. This completes the system initialization.

4.2 Scheme Overview

The outline of our PIR scheme is as follows. Every β consecutive query executions are called a *session.* For the s-th session, $s \geq 0$, let $\pi_s, \mathcal{D}_{\pi_s}$ and sk_s denote the permutation, the shuffled database, and the encryption key respectively. Upon receiving a query from the user, $\mathcal{TH}$ retrieves a data record from $\mathcal{D}_{\pi_s}$, decrypts it with sk_s to get the data item, and stores the item in its cache. Then $\mathcal{TH}$ replies to the user with the desired data item. The detailed operations on data retrieval are described in Algorithm 1. After β queries are executed, $\mathcal{TH}$ generates a new random permutation π_{s+1} and an encryption key sk_{s+1}. It then reshuffles $\mathcal{D}_{\pi_s}$ into $\mathcal{D}_{\pi_{s+1}}$ by employing π_{s+1} and sk_{s+1}. Note that in the new produced database $\mathcal{D}_{\pi_{s+1}}$, all data records are encrypted under the new secret key sk_{s+1}. The details on database reshuffle are given in Algorithm 2. The old secrets π_s and sk_s are securely erased.

Caveat The original database $\mathcal{D}$ is not involved in any database retrieval operations. Since $\mathcal{TH}$ always performs a decryption for every *read* operation and an encryption for every *write* operation, we omit them in the algorithm description in order to keep our presentation concise.

4.3 Retrieval Query Processing Algorithm

The basic idea of our retrieval algorithm is the following. $\mathcal{TH}$ always reads a new record on every query and every record is accessed at most once. Thus, if the database is well permuted (in the sense of oblivious permutation), all database accesses within a session appear random to the adversary.

Without loss of generality, suppose that the user intends to retrieve d_i in $\mathcal{D}$ during the s-th session ($s \geq 0$). Upon receiving the query for index i, $\mathcal{TH}$ performs the following: It searches Γ for d_i. if d_i is not in its cache, it locates the corresponding record in the shuffled database $\mathcal{D}_{\pi_s}$ by computing the record index as $\pi_s^{-1}(i)$; Otherwise, it reads from $\mathcal{D}_{\pi_s}$ a random record which has not been accessed before[6]. The algorithm is elaborated in Figure 3.

Access Pattern. The access pattern $\mathcal{A}_s$ produced by Algorithm 1 is a sequence of data records which are retrieved from $\mathcal{D}_{\pi_s}$ during the s-th session. It is clear from Figure 3 that on each query, exactly one new data record is read from $\mathcal{D}_{\pi_s}$. Therefore, when the s-th session terminates, $\mathcal{A}_s$ has exactly β records.

Cache Management. All items and their indexes stored in the cache are organized in an array denoted by Γ. Depending on the structure of Γ, $\mathcal{TH}$ may apply different search techniques. Possible implementation include hash table and binary search trees. Note that the choice of searching algorithm is a trade-off between space and time. Since $\mathcal{TH}$ has limited cache size, a binary search is more suitable, which has O($\log k$) cost. Though the internal search time adds to the actual time for query execution, it does not affect the asymptotic complexity of the proposed PIR scheme in big-O notation. This is because the cache size k is a constant parameter that does not grow with the database size n.

```
Algorithm 1: Retrieving record $d_i$ using $\mathcal{D}_{\pi_s}$
1. $\mathcal{TH}$ decrypts the query and gets the requested index $i$.
2. If $i \notin \Gamma$
3.     $\mathcal{TH}$ reads $\pi_s^{-1}(i)$-th record of $\mathcal{D}_{\pi_s}$ and stores the item $d_i$ into its cache;
4.     $\Gamma = \Gamma \cup \{i\}$;
5. Else
6.     $\mathcal{TH}$ selects a random index $j$, $j \in_R \{1, \cdots, n\} \setminus \Gamma$.
7.     $\mathcal{TH}$ reads the $\pi_s^{-1}(j)$-th record from $\mathcal{D}_{\pi_s}$ and stores $d_j$ into its cache;
8.     $\Gamma = \Gamma \cup \{j\}$;
9. $\mathcal{TH}$ returns $d_i$ to the user.
```

Fig. 3 Retrieval Query Processing Algorithm

4.4 Reshuffle Process

After β retrievals, $\mathcal{TH}$'s cache reaches its limit, which demands a reshuffle of the database with a new permutation. Note that simply using *cache substitution* introduces a risk of privacy exposure. The reason is that when a discarded item is requested again, the adversary knows that a data record is retrieved more than once by $\mathcal{TH}$ from the same location. Therefore, a reshuffle procedure must be executed at the end of each session.

$\mathcal{TH}$ first secretly chooses a new random permutation π_{s+1}. The expected database $\mathcal{D}_{\pi_{s+1}}$ satisfies $\mathcal{D}_{\pi_{s+1}}[j] \simeq d_{\pi_{s+1}(j)}$, $j \in [1, n]$. The correlation between $\mathcal{D}_{\pi_s}$ and $\mathcal{D}_{\pi_{s+1}}$ is

$$\mathcal{D}_{\pi_{s+1}}[j] \simeq \mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)], \tag{1}$$

for $1 \leq j \leq n$, where $\pi_s^{-1} \circ \pi_{s+1}(j)$ means $\pi_s^{-1}(\pi_{s+1}(j))$.

The reshuffle algorithm is to generate $D_{\pi_{s+1}}$ using D_{π_s} and the cache. In a nutshell, $\mathcal{TH}$ fills in $D_{\pi_{s+1}}[1]$, $D_{\pi_{s+1}}[2], \cdots, D_{\pi_{s+1}}[n]$, sequentially. If the item to be written to $D_{\pi_{s+1}}[i]$ is in the cache, $\mathcal{TH}$ writes it directly from the cache; otherwise, $\mathcal{TH}$ fetches it from D_{π_s}. Therefore the challenge is how to *efficiently* and *obliviously* generate $D_{\pi_{s+1}}$, i.e. the adversary can not distinguish whether $D_{\pi_{s+1}}[i]$ is set by an item in cache or not.

The basic idea is as follows. We sort the items in $\mathcal{TH}$'s cache in ascending order based on their new positions in $\mathcal{D}_{\pi_{s+1}}$. Since the database allows index-based

[6] The operation should be implemented so that both "if" and "else" situations take the same amount time to stand against side-channel attack. This requirement is also applied at a similar situation in the reshuffle algorithm introduced later.

direct record retrieval, those un-cached items in $\mathcal{D}_{\pi_s}$ do not need to be physically sorted. Instead, they can be regarded as being virtually sorted as $\mathcal{TH}$ can calculate their new indexes in $\mathcal{D}_{\pi_{s+1}}$. The reshuffle process is similar to a merge-sort of two sorted sequences: the sorted items in the cache and those un-touched items. $\mathcal{TH}$ plays two roles: (1) participating in the merge-sort to initialize $\mathcal{D}_{\pi_{s+1}}$; (2) obfuscating the read/write pattern to protect the secrecy of π_{s+1}.

$\mathcal{TH}$ first sorts indices in Γ based on the ascending order of their images under π_{s+1}^{-1}. It assigns the entries in database $\mathcal{D}_{\pi_{s+1}}$ sequentially, starting from $\mathcal{D}_{\pi_{s+1}}[1]$. For the first $n-\beta$ assignments, $\mathcal{TH}$ always performs one read operation and one write operation per record; for the remaining β assignments, it always performs one write operation per record as they are in its cache. The initialization of $\mathcal{D}_{\pi_{s+1}}[j]$, $j \in [1, n]$, falls into one of the following two cases, depending on whether its corresponding item is in the cache or not.

Case (i) The corresponding item is not cached (i.e., $\pi_{s+1}(j) \notin \Gamma$): $\mathcal{TH}$ reads it (i.e., the record $\mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)]$) from $\mathcal{D}_{\pi_s}$ and writes it to $\mathcal{D}_{\pi_{s+1}}$ as $\mathcal{D}_{\pi_{s+1}}[j]$.

Case (ii) The corresponding item is in the cache (i.e., $\pi_{s+1}(j) \in \Gamma$): Before retrieving $\mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)]$ from the cache and writing it into $\mathcal{D}_{\pi_{s+1}}$ as $\mathcal{D}_{\pi_{s+1}}[j]$, $\mathcal{TH}$ also performs a read operation for two purposes: (a) to demonstrate the same reading pattern as in Case (i) so that the secrecy of π_{s+1} is protected; (b) to save the cost of future reads. Thus, instead of randomly reading a record from the $\mathcal{D}_{\pi_s}$, $\mathcal{TH}$ looks for the smallest index which has not been initialized and falls in Case (i). It then retrieves the corresponding data record from $\mathcal{D}_{\pi_s}$. Since Γ is sorted, this searching process costs β comparisons for the entire reshuffle process.

The details of the reshuffle algorithm are shown in Figure 4, where *min* denotes the head of sorted Γ. We use `sortdel`(i)/`sortins`(i) to denote the sorted deletion/insertion of index i from/to Γ and subsequent adjustments.

The reshuffle algorithm is secure and efficient. An intuitive explanation of its security is as follows. After a reshuffle, the new database is reset to its initial status. If an item has been accessed in the previous session, it is placed at a random position in the whole new database by the reshuffle. A record retrieved from $\mathcal{D}_{\pi_s}$ will be written out in one of the subsequent β writes. Since those items are not accessed in the current session, the information about their data items is not exposed. Furthermore, for every write operation in the reshuffle algorithm, the probability that the item is originally in cache, i.e. it has been queried before reshuffle, is always β/n, because the access pattern of the retrieval algorithm is uniformly random. Therefore, every newly encrypted record has the same probability in being queried before. In other words, the entire database appears uniform to the adversary.

Fig. 4 Database Reshuffle Algorithm

```
Algorithm 2: Reshuffle $\mathcal{D}_{\pi_s}$ into $\mathcal{D}_{\pi_{s+1}}$, executed by $\mathcal{TH}$

1.  secretly select a new random permutation $\pi_{s+1}$;
2.  sort indices in $\Gamma$ based on the order of their images under $\pi_{s+1}^{-1}$.
    set $j = 1; j' = 1$.
3.  while $1 \le j \le n - \beta$ do
4.    while $\pi_{s+1}(j') \in \Gamma$ do $j' = j' + 1$ end;
5.    set $r = \pi_s^{-1} \circ \pi_{s+1}(j')$; read $\mathcal{D}_{\pi_s}[r]$ from $\mathcal{D}_{\pi_s}$;
6.    if $j = j'$            /* Case (i): $\pi_{s+1}(j) \notin \Gamma$ */
7.      write $\mathcal{D}_{\pi_{s+1}}$ by setting $\mathcal{D}_{\pi_{s+1}}[j] \simeq \mathcal{D}_{\pi_s}[r]$;
8.    else                   /* Case (ii): $\pi_{s+1}(j) \in \Gamma$ */
9.      write $\mathcal{D}_{\pi_{s+1}}$ by setting $\mathcal{D}_{\pi_{s+1}}[j] \simeq d_{min}$;
10.     sortdel($j$); Insert $\mathcal{D}_{\pi_s}[r]$ into cache and sortins($j'$);
11.   $j = j + 1; j' = j' + 1$;
12. end{while};
13. while $n - k + 1 \le j \le n$ do
14.   set $\mathcal{D}_{\pi_{s+1}}[j] \simeq d_{min}$; sortdel($j$);
15.   $j = j + 1$;
16. end
```

Note that the increment of j' in the inner loop (Step 4) is executed at most $n-1$ times in total, since j' never decreases. Because Γ is a sorted list and the inserted and deleted indices are in an ascending order, the insertion and deletion are of constant cost. Totally n comparisons are needed for the whole execution. Therefore, the overall computation complexity of Algorithm 2 is O(n).[7]

Reshuffle Pattern The access pattern produced by Algorithm 2 is denoted by $\mathbb{R}_s$. We call it reshuffle pattern so as to differentiate it from the access pattern due to Algorithm 1. Since $\mathcal{TH}$ only reads $n-\beta$ data records, $\mathbb{R}_s$ has exactly $n-\beta$ elements. Note that the writing pattern is omitted because it is in a fixed order, i.e., sequentially writing from position 1 to position n. The writing pattern does not give the server extra advantages since the entire database is still in the server's storage.

5 Security Analysis

We now proceed to analyze the security of our scheme based on the notion defined in Section 3.3. We show in Theorem 1 that our scheme is secure with respect to Definition 1. The key components of the proof are two lemmas. Lemma 1 proves that the reshuffle procedure is oblivious in the same notion as in Oblivious RAM [35]. Thus after each reshuffle, the database is reset into the initial state such that the accesses among different sessions are not correlated. In Lemma 2, we show that each individual query session does not leak information of the query, which leads to the conclusion of the theorem.

[7] This complexity is exactly the same as the complexity of a merge-sort algorithm on two sorted arrays.

Recall that our architecture includes a secure communication channel between users and $\mathcal{TH}$. Therefore, we do not consider attacks on the communications.

Theorem 1 *For a database $\mathcal{D} = [d_1, d_2, \cdots, d_n]$ and a corresponding shufffled and encrypted database $\mathcal{D}_{\pi_s}$, $s \geq 0$, for any PPT adversary $\mathcal{A}^{\mathsf{QUE}}$ against our scheme, there exists a PPT $\mathcal{A}'^{\mathsf{QUE}}$, such that for any clean query $Q = q \in [1, n]$, whose read access in $\mathcal{D}_{\pi_s}$ is $a \in [1, n]$, we have*

$$|\Pr(\mathcal{A}^{\mathsf{QUE}}(a) = q) - \Pr(\mathcal{A}'^{\mathsf{QUE}}(1^\kappa) = q)| < \epsilon(\kappa) \quad (2)$$

Proof We prove the theorem by using a series of games [56] between $\mathcal{A}$ and a challenger who simulates an environment with respect to our scheme.

Game 0. Fix a PPT adversary $\mathcal{A}$. Game 0 is defined to be an attack game between $\mathcal{A}$ and the challenger, who runs an instance our scheme by simulating $\mathcal{TH}$. The game is conceptually equivalent to $\mathcal{A}$ attacking against our scheme as defined in Definition 1. In particular, the challenger sets up the system and simulates $\mathcal{TH}$, following our scheme: determines a database $\mathcal{D} = [d_1, d_2, \cdots, d_n]$; picks π_s and sk_s, and generates $\mathcal{D}_{\pi_s}$ $(s \geq 0)$, as per the reshuffle algorithm; replies to $\mathcal{A}$'s queries as per the retrieval query processing algorithm. At a certain time, the challenger challenges $\mathcal{A}$ with a read access a on $\mathcal{D}_{\pi_s}$ for a clean query Q retrieving d_q. At the end of the game, $\mathcal{A}$ outputs an index q'.

Let us define s_0 be the event that $q' = q$ in Game 0. Then it is evident that $\Pr(s_0) = \Pr(\mathcal{A}^{\mathsf{QUE}}(a) = q)$.

Game 1. We transform Game 0 into Game 1 by the following modification. Let $\prod_n$ be the set of all permutations over $[1, n]$. Instead of generating $\mathcal{D}_{\pi_s}, s \geq 0$, using a pseudorandom permutation π_s, the challenger picks a random $\Pi_s \in \prod_n$ and generates $\mathcal{D}_{\Pi_s}$; other steps remain unchanged.

Let s_1 be the event that $q' = q$ in Game 1. We claim that

$$|\Pr(s_1) - \Pr(s_0)| = \epsilon_{\text{prp}} \quad (3)$$

where ϵ_{prp} is the advantage of some PPT adversary in distinguishing between a pseudorandom permutation over $[1, n]$ and a random permutation in $\prod_n$.

Indeed, the following adversary "interpolates" between Game 0 and Game 1, and has an advantage equal to $|\Pr(s_1) - \Pr(s_0)|$:

```
Distinguisher $D^{\mathcal{O}}$
Sets up a system according to our scheme, with
the only exception that using $\mathcal{O}$ to generate a
shuffled and encrypted database $\mathcal{D}_{\mathcal{O}}$.
Challenges $\mathcal{A}^{\mathsf{QUE}}$ with a read access $a$ on $\mathcal{D}_{\mathcal{O}}$,
which corresponds to a clean query $Q = q$.
if $\mathcal{A}^{\mathsf{QUE}}(a) = q$
        then output 1
        else output 0.
```

It is clear that if $\mathcal{O}$ is a pseudorandom permutation (prp) over $[1, n]$, then Distinguisher D proceeds just as in Game 0, we thus have $\Pr(D^{\text{prp}} = 1) = \Pr(s_0)$. Otherwise, Distinguisher D proceeds as in Game 1 if $\mathcal{O}$ is a random permutation (rp) from $\prod_n$, and we have $\Pr(D^{\text{rp}} = 1) = \Pr(s_0)$.

Game 2. Game 2 proceeds identically as Game 1, except for the the following difference: the challenger generates shuffled and encrypted databases using perfect encryption (e.g., one-time pad), rather than semantically secure $E_{\mathsf{sk}_s}(.), s \geq 0$.

Let s_2 be the event that $q' = q$ in Game 2. We claim that

$$|\Pr(s_2) - \Pr(s_1)| = \epsilon_{\text{sem}} \quad (4)$$

where ϵ_{sem} is the advantage of some PPT adversary in distinguishing between semantically secure encryption and perfect encryption (i.e., breaking the semantic security of the encryption scheme). Similarly, it is straightforward to construct a distinguisher D as above (given oracle access to either semantically secure encryption or perfect encryption), "interpolating" between Game 1 and Game 2.

We further claim that $\Pr(s_2) = \Pr(\mathcal{A}'^{\mathsf{QUE}}(1^\kappa) = q)$, which will conclude the proof. To see this, by Lemma 1 we first show that our reshuffle algorithm is oblivious, so that database accesses in different sessions are independent from each other; then by Lemma 2, we show that observing the access patterns of database accesses provides the adversary no more knowledge to determine a clean query than a random guess, which implies $\Pr(s_2) = \Pr(\mathcal{A}'^{\mathsf{QUE}}(1^\kappa) = q)$. Note that the following analysis are in the context of Game 2, where random permutation and perfect encryption are used, hence the adversary computes its outputs based on the observation of the access patterns (caused by either the adversary's queries or other users' queries) and the reshuffle patterns.

Lemma 1 *The reshuffle algorithm in Figure 4 is oblivious, i.e., for all $s \geq 0$, all integer $j \in [1, n]$,*

$$\Pr(\mathcal{D}_{\Pi_{s+1}}[j] \simeq d_l \,|\mathbb{A}_0, \mathbb{R}_0, \cdots, \mathbb{A}_s, \mathbb{R}_s) = 1/n, \quad (5)$$

for all $l \in [1, n]$, where $\mathbb{A}_i$ and $\mathbb{R}_i$, $i \in [0, s]$, are the access pattern and reshuffle pattern for i-th session respectively.

Proof Given any $j \in [1, n]$, we prove Lemma 1 by induction on the session index s. Naturally, the proof applies to all $j \in [1, n]$.

I. $s = 0$. Since $\mathcal{D}_{\Pi_0}$, the initial shuffled database, is constructed in advance under a secret random permutation Π_0, the probability $\Pr(\mathcal{D}_{\Pi_0}[j] \simeq d_l \mid \emptyset) = 1/n$ holds for all $1 \leq l \leq n$.

II. Suppose Equation 5 holds for $s = i - 1$, i.e. $\Pr(\mathcal{D}_{\Pi_i}[j] \simeq d_l \,|\mathbb{A}_0, \mathbb{R}_0, \cdots, \mathbb{A}_{i-1}, \mathbb{R}_{i-1}) = 1/n$. We proceed to prove that it holds for $s = i$, i.e.

$$\Pr(\mathcal{D}_{\Pi_{i+1}}[j] \simeq d_l \,|\mathbb{A}_1, \mathbb{R}_1, \cdots, \mathbb{A}_i, \mathbb{R}_i) = 1/n,$$

for all $l \in [1, n]$.

In order to use the recursive assumption, we link the two databases $\mathcal{D}_{\Pi_{i+1}}$ and $\mathcal{D}_{\Pi_i}$ by the following conditional probability,

$$\begin{aligned}
&\Pr(\mathcal{D}_{\Pi_{i+1}}[j] \simeq d_l \mid \mathbb{A}_1, \mathbb{R}_1, \cdots, \mathbb{A}_i, \mathbb{R}_i) \\
&= \sum_{x=1}^{n} \{\Pr(\mathcal{D}_{\Pi_{i+1}}[j] \simeq \mathcal{D}_{\Pi_i}[x] \mid \mathbb{A}_1, \mathbb{R}_1, \cdots, \mathbb{A}_i, \mathbb{R}_i) \\
&\qquad \cdot \Pr(\mathcal{D}_{\Pi_i}[x] \simeq d_l \mid \mathbb{A}_1, \mathbb{R}_1, \cdots, \mathbb{A}_i, \mathbb{R}_i)\}.
\end{aligned}$$

Then the formula is evaluated depending on cases that whether or not l is stained and whether or not the item corresponding to x is in the cache. The conclusion is obtained by showing that the sum on the right hand side of the equation is $1/n$ in all cases.

For clarity, we define p_x and q_x as

$$p_x \triangleq \Pr(\mathcal{D}_{\Pi_{i+1}}[j] \simeq \mathcal{D}_{\Pi_i}[x] \mid \mathbb{A}_1, \mathbb{R}_1, \cdots, \mathbb{A}_i, \mathbb{R}_i),$$

and

$$q_x \triangleq \Pr(\mathcal{D}_{\Pi_i}[x] \simeq d_l \mid \mathbb{A}_1, \mathbb{R}_1, \cdots, \mathbb{A}_i, \mathbb{R}_i).$$

The objective now is to prove

$$\sum_{x=1}^{n} p_x q_x = 1/n.$$

Define $X = \{x \mid x \in [1, n],\ \Pi_i(x) \in \Gamma\}$ and $Y = \{x \mid x \in [1, n],\ \Pi_i(x) \notin \Gamma\}$. Note that $|X| = |\Gamma| = \beta$, $|Y| = n - \beta$ and $X \cup Y = [1, n]$. Thereafter,

$$\sum_{x=1}^{n} p_x q_x = \sum_{x \in X} p_x q_x + \sum_{x \in Y} p_x q_x$$

We observe that the adversary would have different projections on the new indices for those records in $\mathcal{D}_{\Pi_i}$. For those items in $\mathcal{TH}$'s cache, i.e. those whose indexes in $\mathcal{D}_{\Pi_i}$ are in X, the adversary obtains no information about their positions in $\mathcal{D}_{\Pi_{i+1}}$. On the other hand, for the other items, i.e. those whose indexes in $\mathcal{D}_{\Pi_i}$ are in Y, the adversary is certain that they would not be placed to positions which have been initialized before their retrievals from $\mathcal{D}_{\Pi_i}$. Moreover, the item retrieved by the first read in reshuffle will appear in one of the first $\beta + 1$ positions in $\mathcal{D}_{\Pi_{i+1}}$. Therefore, only for $x \in X$,

$$p_x = \Pr[\Pi_{i+1}(j) = \Pi_i(x)] = 1/n$$

But this does not hold for p_x, $x \in Y$. Consequently,

$$\sum_{x \in Y} p_x = 1 - \sum_{x \in X} p_x = (n - \beta)/n.$$

The computation of q_x is related to the stained queries. Let $\mathcal{C}$ denote the set of stained queries in the i-th session. We have two cases for $1 \le l \le n$:

- Case (1) $l \in \mathcal{C}$: $\sum_{x \in X} q_x = 1$, because in the adversary's perspective, there exists one and only one item in the cache which corresponds to query on d_l. For $x \in Y$, $q_x = 0$ because none matches. Thus

$$\sum_{x=1}^{n} p_x q_x = 1/n + 0 = 1/n, \text{ for } l \in \mathcal{C}.$$

- Case (2) $l \notin \mathcal{C}$: Suppose $\sum_{x \in X} q_x = \delta$ for $0 \le \delta \le 1$, then $\sum_{x \in Y} q_x = 1 - \delta$. Note that the execution of queries does not affect the adversary's observation on those not cached records, since they are not accessed. Therefore, by induction assumption: $\Pr(\mathcal{D}_{\Pi_i}[x] \simeq d_l \mid \mathbb{A}_0, \mathbb{R}_0, \cdots, \mathbb{A}_{i-1}, \mathbb{R}_{i-1}) = 1/n$, for all $l \in [1, n]$, we have $q_x = \frac{1-\delta}{n-\beta}$ due to equiprobability [8]. Hence,

$$\sum_{x=1}^{n} p_x q_x = \delta/n + (1 - \delta)/n = 1/n. \tag{6}$$

Combining Case 1 and Case 2, we have

$$\sum_{x=1}^{n} p_x q_x = 1/n, \text{ for all } 1 \le l \le n,$$

which concludes the proof. □

Lemma 1 implies that the reshuffle procedure resets the observed distribution of the data items. Therefore, the accesses occurring during separated sessions are independent of each other. Theorem 1 below proves the security of the proposed PIR scheme as a whole.

Lemma 2 *Given a time period, the observation of the access pattern $\mathbb{A} = (a_1, a_2, \cdots, a_N)$, $N > 0$, provides the adversary no more knowledge to determine any clean query Q than a random guess, i.e. for all $q \in [1, n]$,*

$$\Pr(Q = q \mid \mathbb{A}) = \Pr(Q = q) \tag{7}$$

where $\Pr(Q = q)$ is the a-priori probability of query Q being on index q.

Proof Since we have

$$\begin{aligned}
\Pr(Q = q \mid \mathbb{A}) &= \Pr(Q = q, \mathbb{A})/\Pr(\mathbb{A}) \\
&= \frac{\Pr(\mathbb{A} \mid Q = q) \cdot \Pr(Q = q)}{\Pr(\mathbb{A})}
\end{aligned}$$

Therefore, to prove Lemma 2 is equivalent to prove that $\Pr(\mathbb{A} \mid Q = q) = \Pr(\mathbb{A})$ for all possible access patterns $\mathbb{A}$.

For $1 < t \le N$, let $\Pr(a_t \mid a_1, \cdots, a_{t-1})$ denote the probability of the event that data a_t is accessed immediately after the access of $t - 1$ records. Let $\Pr(a_t \mid$

[8] Hint: Otherwise, for $x_0, x_1 \in Y$, $q_{x_0} \neq q_{x_1}$ implies $\Pr(\mathcal{D}_{\Pi_i}[x_0] \simeq d_l) \neq \Pr(\mathcal{D}_{\Pi_i}[x_1] \simeq d_l)$ where $d \notin \mathcal{C}$. This result is also provable by the indistinguishability for the adversary using two different permutations which are identical for $l \in \mathcal{C}$.

$a_1, \cdots, a_{t-1}, Q = q)$ denote the probability of the same event with additional knowledge that the requested index of the target query is q. Note that we do not assume any temporal order of the query q and the t-th query. We proceed to show below that

$$\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}, Q = q)$$

Without loss of generality, suppose a_t is read from $\mathcal{D}_{\Pi_s}$ during the s-th session. Consider the following two cases:

1. $a_t \in \mathbb{R}_s$, i.e. a_t is accessed during a reshuffle process: Obviously,

   $$\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}, Q = q)$$

   due to the fact that the access to a_t is completely determined by permutation Π_s and Π_{s+1}.
2. $a_t \in \mathbb{A}_s$, i.e. a_t is accessed during a query process: Let this query be the l-th query in this session, $l \in [1, \beta]$. Therefore, $l - 1$ data items are cached by $\mathcal{TH}$ before a_t is read. We consider two scenarios based upon Algorithm 1:
   (a) The requested data is cached in $\mathcal{TH}$: a_t is randomly chosen from those data items not cached in $\mathcal{TH}$. Thus, $\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \frac{1}{n-(l-1)}$.
   (b) The requested data is not cached in $\mathcal{TH}$: a_t is retrieved from D_{Π_s} based on the permutation Π_s. According to Lemma 1 (equiprobability, c.f. HINT 8), the probability that a_t is selected is $\frac{1}{n-(l-1)}$.

   Note that the compromise of a query, i.e. knowing $Q = q$, possibly helps an adversary to determine whether a_t is in case (2a) or (2b). Nonetheless, this information does not change $\Pr(a_t \mid a_1, \cdots, a_{t-1})$, since their values are $\frac{1}{n-(l-1)}$ in both cases. Thus, $\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}, Q = q)$ when $a_t \in \mathbb{A}_s$.

In total, we conclude that for any $t > 1$ and $Q = q$,

$$\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}, Q = q).$$

As a result,

$$\begin{aligned} &\Pr(\mathbb{A} \mid Q = q) = \Pr(a_1, \cdots, a_N \mid Q = q) \\ &= \Pr(a_N \mid a_1, \cdots, a_{N-1}, q) \cdot \Pr(a_1, \cdots, a_{N-1} \mid Q = q) \\ &= \Pr(a_1 \mid Q = q) \prod_{t=2}^{N} \Pr(a_t \mid a_1, \cdots, a_{t-1}, Q = q) \\ &= \Pr(a_1 \mid Q = q) \prod_{t=2}^{N} \Pr(a_t \mid a_1, \cdots, a_{t-1}) \end{aligned}$$

Since a_1 is independent of $Q = q$ (determined by initialization π_0), then $\Pr(a_1 \mid Q = q) = P(a_1)$. Thus, we have

$$\Pr(\mathbb{A} \mid Q = q) = \Pr(\mathbb{A}).$$

The result shows that, given the access pattern, the a-posteriori probability of a query equals to its a-priori probability, which concludes the proof for Lemma 2. □

Based on the above analysis, we conclude the proof for Theorem 1:

$$\begin{aligned} &|\Pr(\mathcal{A}^{\mathsf{QUE}}(a) = q) - \Pr(\mathcal{A}'^{\mathsf{QUE}}(1^\kappa) = q)| \\ &\qquad = |\Pr(s_0) - \Pr(s_2)| < \epsilon_{\text{prp}} + \epsilon_{\text{sem}} \end{aligned}$$

■

6 Performance

We proceed to analyze the communication and computation complexities of our PIR scheme. They are evaluated with respect to the database size n. Both our scheme and the hardware-based schemes in [58,39,40] belong to this category of models. We remark that we do not include the costs for secure channel establishment into the following performance calculation. After all, these costs are independent of the database size, and can be amortized if a user makes multiple queries at a time.

Communication: We consider the user/system communication cost per query. In our scheme, the user only inputs an index of the desired data item and $\mathcal{TH}$ returns exactly one data item. Therefore, its communication complexity per query is $\mathrm{O}(\log n)$. Note that $\mathrm{O}(\log n)$ is the lower bound of communication cost for all PIR constructions. The communications between the trusted hardware and the database is memory I/O and disk I/O. We treat them as part of the computation cost.

Computation: For simplicity purpose, each reading, writing, encryption, and decryption of a data item is treated as one *operation*. The computation cost is measured by the average number of operations per session and per query. We assumes that the costs of computing $\pi_s()$ and $\pi_s^{-1}()$ are independent of the database size. Our discussion in the next section will discuss the cost of computing the permutation function.

As evident in Figure 3 and Figure 4, it costs $\mathcal{TH}$ $\mathrm{O}(1)$ operations to process a query[9] and $\mathrm{O}(n)$ operations to reshuffle the database. Table 2 compares the computation cost of our scheme against those in [58] and [39,40]. Our scheme outperforms the other two hardware-based PIR schemes in all three metrics. The advantage originates in our reshuffle algorithm which utilizes the cache in a more efficient manner.

7 Discussion

7.1 Database Initialization Without TTP

For applications where no trusted third party exists, the trusted hardware can be used to initialize the database. $\mathcal{TH}$ first chooses a random permutation π_0. For $1 \leq i \leq$

[9] Note that it costs $\mathrm{O}(\log k)$ operations for internal search, which is applicable for [58,39,40] as well.

Table 2 Comparison of computation costs.

Schemes	**Total cost (per session of β queries)**	**Online cost (per query)**	**Average Cost (per query)**
Our scheme	O(n) We only take into account the main cost for the sake of simplicity and comparison clarity. The actual cost should be O(n) + O($\beta \log \beta$), where the second term quantity comes from sorting Γ, which costs $\beta \log \beta$ operations.	O(1)	O(n/β)
Scheme in [39,40]	O($n \log n$) Their actual cost is approximately O($n \log n$)+$O(n)$+$O(\beta \log \beta)$, where O(n)(+O($\beta \log \beta$)) comes from sorting $\bar{T}$ (Step (i) of [39]), $\beta \log \beta$ comes from sorting T (Step (ii)), and O($n \log n$) comes from Step (iii).	O(1)	O($\frac{n}{\beta} \log n$)
Scheme in [58]	O($\beta.n$)	O(n)	O(n)

n, it tags the i-th item d_i with its new index $\pi_0^{-1}(i)$. Using the merge-sort algorithm [27], $d_1, d_2, \cdots, d_n$ are sorted based on their new indexes. With the limited cache size in $\mathcal{TH}$, Batcher's odd-even merges sorter [6] is an appropriate choice which requires $(\log^2 n - \log n + 4)n/4 - 1$ comparisons. While Beneš network [59] and Goldstein et al's switch networks [36] incur less comparisons, the former however requires at least $n \log n$-bit ($>> \beta$) memory in the application domain and the latter has a prohibitively high setup cost. Note that encryption is applied during tagging and merging so that the process is oblivious to the server.

A simple example is presented in Figure 5. The database in the example has four items d_1, d_2, d_3, d_4. The permutation is $\pi_0 = (1324)$, i.e. $\pi_0(1) = 3, \pi_0(2) = 4, \pi_0(3) = 2$ and $\pi_0(4) = 1$. The circles denote $\mathcal{TH}$ and the squares denote encrypted data items. After initialization, the original four items are permuted as shown on the right end. All the encrypted items are stored in the host. In every operation, only two items are read into the cache of $\mathcal{TH}$ and then written back to the server.

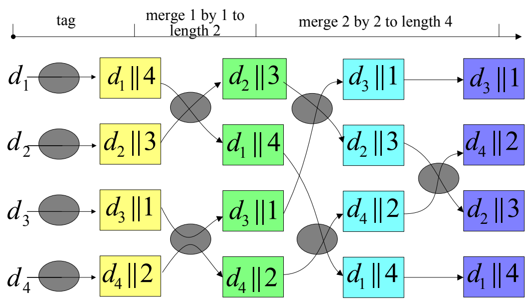

Fig. 5 Initial oblivious shuffle example using odd-even merges.

7.2 Instantiation of Encryption and Permutation Algorithms

An implicit assumption of our security proof in Section 5 is the semantic security of the encryption of the database. Otherwise, the encryption reveals the data information and consequently exposes user privacy. Our adversary model in Section 3 allows the adversary to submit queries and observe the subsequent access patterns and replies. Thereafter, the adversary is able to obtain β pairs of plaintext and ciphertext in maximum for each encryption key, since different random keys are used in different sessions. Thus, we require an encryption algorithm semantically secure under CPA (Chosen Plaintext Attack) model. In practice, CPA secure symmetric ciphers such as AES, are preferred over public key encryptions, since the latter have more expensive computation cost and higher storage space demand.

For the permutation algorithm, we argue that it is impractical for a hardware-based PIR to employ a *true* random permutation, since it requires $O(n \log n)$ bits of storage, comparable to the size of the whole database. As a result, we opt for a pseudo-random permutation with a light computation load.

A k-bit block cipher can be easily used to construct a pseudo-random permutation of $\mathbb{Z}_{2^k}$. Unfortunately, the block size of standard block ciphers, e.g. AES, is much larger than the bit-length of the database size n. Several studies have shown how to tackle with small domains. Black and Rogaway [14] proposed three methods to construct such ciphers using off-the-shelf block ciphers. The third method is based on the Feistel construction. Though efficient, its security is not strong enough when the domain is small. Pryamikov [53] proposed a new secure block cipher called TinyPRP whose block size is either 16-bit or 32-bit. He also proposed to use TinyPRP, together with the cycle-walking technique in [14], to build a pseudo-random permutation on arbitrary finite domain. Morris, Rogaway and Stegers [50] proposed a new scheme to encrypt messages in a small domain by using the Thorp shuffle.

When the database size n is not a power of 2, i.e. $n < 2^l$ and $l = \lceil \log n \rceil$, we can use the aforementioned techniques to construct a permutation on $\mathbb{Z}_{2^l}$. Using the cycle walking technique, if the image is greater than n, the permutation is repeated until the output is in $\mathbb{Z}_n$. Since $2^l < 2n$, the expected number of repeat is not greater than 2.

7.3 Service Continuity

According to our scheme, the database service is disrupted during the reshuffle process. The duration of a reshuffle cannot be viewed as non-negligible since it is an O(n) process. This problem is especially severe in the hardware-based schemes in [58,39,40], since a smaller cache requires more frequent reshuffle. Our scheme can mitigate this problem by maintaining two caches in $\mathcal{TH}$: one is for re-permuting the database while the other deals with user queries. This trivial solution may not be easy in the hardware-based model [58,39,40], due to the rigid and highly limited size of hardware: splitting the cache of the hardware into two halves with each having the capacity of storing $\beta/2$ items, will double the average computation cost in principle.

7.4 Update of data items

A byproduct of the reshuffle process is database update operations. To update d_i, $\mathcal{TH}$ reads d_i obliviously in the same way as handling a read request. Then, d_i is updated inside the cache and written into the new permuted database during the upcoming reshuffle process. Though the new value of d_i is not written immediately into the database, data consistency is ensured since $\mathcal{TH}$ returns the updated value directly from its cache upon user requests.

8 Conclusion

We present in this paper a novel hardware-assisted PIR system. The new PIR construction is proven secure. The execution of the PIR protocol does not expose information to the adversary about the query distribution. Namely, our PIR protocol ensures the distribution of any two queries are identical, provided that the underlying database encryption scheme is semantically secure under chosen plaintext attacks. The performance of our scheme beats previous constructions. A query execution costs O($\log n$) communication cost and O(1) amount of operations, including database access, encryption, decryption, and pseudo-random permutation.

Nonetheless, our scheme is still steps away from a truly practical scheme since it incurs O(n) amortized offline computation cost. The notion of informational secure PIR was relaxed to computational secure PIR in order to improve the communication complexity. It is an open problem how to trade security for a practical computation complexity.

Acknowledgement

This research is partly supported by the Office of Research, Singapore Management University. We would like to thank anonymous reviewers for their valuable suggestions and criticisms.

References

1. Abadi, M., Feigenbaum, J., Kilian, J.: On hiding information from an oracle. Journal of Computer and System Sciences **39** (1989)
2. Ambainis, A.: Upper bound on the communication complexity of private information retrieval. In: Proceedings of the 24th ICALP 1997
3. Arnold, T., Doorn, L.V.: The IBM PCIXCC: A new cryptographic coprocessor for the ibm eserver. IBM Journal of Research and Development **48** (2004)
4. Asonov, D.: Private information retrieval - an overview and current trends. Tagungsband der GI/OCG-Jahrestagung (2001)
5. Asonov, D.: Querying Databases Privately: A New Approach to Private Information Retrieval, LNCS 3218. Springer (2004)
6. Batcher, K.E.: Sorting networks and their applications
7. Beaver, D., Feigenbaum, J.: Hiding instances in multi-oracle queries. In: Proceedings of Symposium on Theoretical Aspects of Computer Science, 1990
8. Beaver, D., Feigenbaum, J., Kilian, J., Rogaway, P.: Locally random reductions: improvements and appliations. Journal of Cryptology **10** (1997)
9. Beimel, A., Ishai, Y.: Information-theoretic private information retrieval: a unified construction. In: Proceedings of ICALP 2001
10. Beimel, A., Ishai, Y., Kushilevitz, E., Raymond, J.F.: Breaking the $o(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In: Proceedings of IEEE FOCS 2002
11. Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers computation in private information retrieval: PIR with preprocessing. In: Proceedings of CRYPTO 2000
12. Beimel, A., Stahl, Y.: Robust information-theoretic private information retrieval. In: Proceedings of the 3rd Conference on Security in Communications Networks, 2002
13. Bennett, C., Brassard, G., Crepeau, C., Skubiszewska, M.H.: Practical quantum oblivious transfer protocols. In: Proceedings of Crypto '91
14. Black, J., Rogaway, P.: Ciphers with arbitrary finite domains. In: Proceedings of CT-RSA 2002
15. Blakely, G., Meadows, C.: A database encryption scheme which allows computation of statistics using encrypted data. In: Proceedings of IEEE Symposium on Security and Privacy, 1985
16. Blundo, C., Darco, P., , DeSantis, A.: A t-private k-database information retrieval scheme. International Journal of Information Security **1**(1) (2001)

17. Boneh, D., Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Proceedings of EUROCRYPTO 2004
18. Boneh, D., Kushilevitz, E., Ostrovsky, R., Skeith, W.: Public key encryption that allows PIR queries. In: Proceedings of CRYPTO (2007)
19. Brassard, G., Crpeau, C., Robert, J.M.: All-or-nothing disclosure of secrets. In: Proceedings of CRYPTO '86
20. Brassard, G., Crpeau, C., Santha, M.: Oblivious transfers and intersecting codes. IEEE Transactions on Information Theory **42**(6) (1996)
21. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylog communication. In: Proceedings of EUROCRYPTO 99
22. Chaum, D.: Untraceable electronic mail, return addresses and digital pseudonyms. Communications of the ACM **24**(2) (1981)
23. Chor, B., Gilboa, N.: Computationally private information retrieval. In: Proceedings of the 29th STOC, 1997
24. Chor, B., Gilboa, N., Naor, M.: Private information retrieval by keywords. Tech. rep., Israel Institute of Technology (1997)
25. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. In: Proceedings of IEEE FOCS 1995
26. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. Journal of the ACM **45**(6) (1998)
27. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Second Edition. The MIT Press (2003)
28. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. In: Proceedings of ACM CCS 2006
29. Damgård, I., Jurik, M.: A length-flexible threshold cryptosystem with applications. In: Proceedings of ACISP 2003
30. DiCrescenzo, G., Ishai, Y., Ostrovsky, R.: Universal service provides for private information retrieval. Journal of Cryptology **14**(1) (2001)
31. Feigenbaum, J.: Encrypting problem instances: Or ..., can you take advantage of someone without having to trust him. In: Proceedings of CRYPTO 1985
32. Gasarch, W.: A survey on private information retrieval. The Bulletin of the European Association for Theoretical Computer Science, Computational Complexity Column(82) (2004)
33. Gertner, Y., Goldwasser, S., Malkin, T.: A random server mode for private information retrieval or information theoretic PIR avoiding database replication. In: Proceedings of the 2nd RANDOM 1998
34. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. In: Proceedings of ACM STOC 1998
35. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. Journal of the ACM **43**(3) (1996)
36. Goldstein, J., Leibholz, S.: On the synthesis of signal switching networks with transient blocking. IEEE Transactions on Electronic Computers **16**(5) (1967)
37. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences **28**(2) (1984)
38. Hastand, J.: Pseudo-random generators with uniform assumptions. In: Proceedings of the 22nd ACM STOC, 1990
39. Iliev, A., Smith, S.: Private information storage with logarithm-space secure hardware. In: Proceedings of International Information Security Workshops, 2004
40. Iliev, A., Smith, S.: Protecting client privacy with trusted computing at the server. IEEE Security & Privacy **3**(2) (2005)
41. Ishai, Y., Kushilevitz, E.: Improved upper bounds on information-theoretic private information retrieval. In: Proceedings of the 31th ACM STOC 1999
42. Itoh, T.: Efficient private information retrieval. IEICE Transactions on Fundamentals, ES2-A(1), 1999
43. Kerenidis, I., de Wolf, R.: Exponential lower bound for 2-query locally decodeable codes via a quantum argument. In: Proceedings of the 35th ACM STOC 2003
44. Kiayias, A., Yung, M.: Secure games with polynomial expressions. In: Proceedings of the 28th ICALP 2001
45. Kushilevitz, E., Ostrovsky, R.: One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In: Proceedings of EUROCRYPTO 2000
46. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally private information retrieval. In: Proceeding of the 38th IEEE FOCS 1997
47. Lipmaa, H.: An oblivious transfer protocol with log-squared communication. In: Proceedings of ISC 2005
48. Mehlhorn, K.: Data structures and algorithms, volume 1. Sorting and Searching. Springer-Verlag (1984)
49. Mishra, S., Sarkar, P.: Symmetrically private information retrieval. In: Proceedings of 1st INDOCRYPT 2000
50. Morric, B., Rogaway, P., Stegers, T.: How to encipher messages on a small domain: Deterministic encryption and the thorp shuffle. In: Proceedings of CRYPTO (2009)
51. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: Proceedings of the 31st ACM STOC 1999
52. Ostrovsky, R., Shoup, V.: Private information storage. In: Proceedings of the 29th STOC, 1997
53. Pryamikov, V.: Enciphering with arbitrary finite domains. In: Proceedings of INDOCRYPT (2006)
54. Reed, M., Syverson, P., Goldschag, D.: Anonymous connections and onion routing. IEEE J. Selected Areas in Communications **16**(4) (1998)
55. Reiter, M., Rubin, A.: Crowds: anonymity for web transactions. ACM Transactions on Information System Security **1**(1) (1998)
56. Shoup, V.: Sequence of games: A tool for taming complexity in security proofs. Cryptology ePrint report 2004/332, November 30 (2004)
57. Sion, R., Carbunar, B.: On the computational practicality of private information retrieval. In: Proceedings of NDSS 2007
58. Smith, S., Safford, D.: Practical server privacy with secure coprocessors. IBM Systems Journal **40**(3) (2001)
59. Waksman, A.: A permutation network. Journal of the ACM **15**(1) (1968)
60. Wang, S., Ding, X., Deng, R., Bao, F.: Private information retrieval using trusted hardware. In: Proceedings of the 11th ESORICS, 2006
61. Williams, P., Sion, R.: Usable PIR. In: Proceedings of NDSS 2008
62. Williams, P., Sion, R., Carbunar, B.: Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In: Proceedings of ACM CCS 2008
63. Woodruff, D., Yekhanin, S.: A geometric approach to information-theoretic private information retrieval. SIAM Journal of Computing **47**(4) (2007)
64. Yamamura, A., Saito, T.: Private information retrieval based on subgroup membership problem. In: Proceedings of the 6th ACISP 2001