Private Information Retrieval Using Trusted Hardware

Shuhong Wang[1], Xuhua Ding[1], Robert H. Deng[1], and Feng Bao[2]

[1] School of Information Systems, SMU
{shwang,xhding,robertdeng}@smu.edu.sg
[2] Institute for Infocomm Research, Singapore
baofeng@i2r.a-star.edu.sg

Abstract. Many theoretical PIR (Private Information Retrieval) constructions have been proposed in the past years. Though information theoretically secure, most of them are impractical to deploy due to the prohibitively high communication and computation complexity. The recent trend in outsourcing databases fuels the research on practical PIR schemes. In this paper, we propose a new PIR system by making use of trusted hardware. Our system is proven to be information theoretically secure. Furthermore, we derive the computation complexity lower bound for hardware-based PIR schemes and show that our construction meets the lower bounds for both the communication and computation costs, respectively.

1 Introduction

Retrieval of sensitive data from databases or web services, such as patent databases, medical databases, and stock quotes, invokes concerns on user privacy exposure. A database server or web server may be interested in garner information about user profiles by examining users' database access activities. For example, a company's query on a patent from a patent database may imply that it is pursuing a related idea; an investor's query on a stock quote may indicate that he is planning to buy or sell the stock. In such cases, the server's ability of performing information inference is unfavorable to the users. Ideally, users' database retrieval patterns are not leaked to any other parties, including the servers.

A PIR (Private Information Retrieval) scheme allows a user to retrieve a data item from a database without revealing information about the data item. The earliest references of "query privacy" date back to Rivest et al [19] and Feigenbaum [9]. The first formal notion of PIR was defined by Chor et al [6]. In their formalization, a database is modelled as a n-bit string $x = x_1 x_2 \cdots x_n$, and a user is interested in retrieving one bit from x. With this formalization, many results have been produced in recent years. Depending on whether trusted hardware is employed or not, we classify PIR schemes into two categories: *traditional PIR* which does not utilize any trusted hardware and *hardware-based PIR* which employs trusted hardware in order to reduce communication and computation complexities.

The major body of PIR work focuses on the traditional PIR. Interested readers are referred to a survey [10] for a thorough review. A big challenge in PIR design is to minimize the communication complexity, which measures the number of bits transmitted between the user and the server(s) per query. A trivial solution of PIR is for the server to return the entire database. Therefore, the upper bound of communication complexity is $O(n)$ while the lower bound is $O(\log n)$, since by all means the user has to provide an index. For a single server PIR with information theoretic privacy, it is proven in [6] that the communication complexity is at least $O(n)$ and therefore confirming that $O(n)$ is the lower bound. Two approaches are used to reduce the communication cost. One is to duplicate the database in different servers, with the assumption that the servers do not communicate with each other. Without assuming any limit the servers' computation capability, PIR schemes with multiple database copies are able to offer information theoretic security with lower communication cost. The best known result is [3] due to Beimel et. al., with communication complexity $O(n^{\log\log\omega/\omega\log\omega})$, where ω is the number of database copies. The other approach still uses single server model but assumes that the server's computation capability is bounded. Schemes following this approach offer computational security with relatively low communication complexity. The best result to date is due to Lipmaa [16], where the user and the server communication complexity are $O(\kappa\log^2 n)$ and $O(\kappa\log n)$ respectively, with κ being the secure parameter of the underlying computationally hard problem.

Another key performance metric of PIR schemes is their computation complexity. All existing traditional PIR schemes require high computation cost at the server(s) end. Beimel et al [4] proved that the expected computation of the server(s) is $\Omega(n)$[3], which implies that any study on traditional PIR schemes is only able to improve its computation cost by a constant factor.

To the best of our knowledge, two *hardware-based PIR* constructions exist in literature. The earlier scheme [20] due to Smith and Safford is proposed solely to reduce the communication cost. On each query, a trusted hardware reads all the data items from an external database and returns the requested one to the user. The other hardware-based scheme [14, 15] is due to Iliev and Smith. The scheme facilitates an efficient online query process by offloading heavy computation load offline. For each query, its online process costs $O(1)$. Nonetheless, depending on the hardware's internal storage size, for every k ($k << n$) queries the external database needs to be reshuffled with a computation cost $O(n\log n)$. For convenience, we refer to the first scheme as SS01 and the latter as IS04. Both schemes have $O(\log n)$ communication complexity.

OUR CONTRIBUTIONS The contributions of this paper are three-fold: (1) We present a new PIR scheme using the same trusted hardware model as in [14] and prove that it is secure in the information theoretical sense; (2) Among all existing PIR constructions, our scheme achieves the best performance in all aspects:

[3] 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 $n \geq n_0$.

$O(\log n)$ communication complexity, $O(1)$ online computation cost and $O(n)$ offline computation cost; (3) We prove that our average computation complexity per query, $O(n/k)$, is the lower bound for hardware-based PIR schemes using the same model, where k ($k << n$) is the maximum number of data items stored by the trusted hardware.

2 Models and Definitions

Database Model and Its Permutation. We use π to denote a permutation of n integers: $(1, 2, \cdots, n)$. For $1 \leq i \leq n$, the image of i under π is denoted by $\pi(i)$. A database $\mathcal{D}$ is modelled as an array, represented by $\mathcal{D} = [d_1, d_2, \cdots d_n]$, where d_i is the i-th data item in its original form, for $1 \leq i \leq n$. A permuted $\mathcal{D}$ under π is denoted by $\mathcal{D}_\pi$ and its i-th data record is denoted by $\mathcal{D}_\pi[i]$, for $1 \leq i \leq n$. The database $\mathcal{D}$ is permuted into $\mathcal{D}_\pi$ by using π in such a way that *the i-th element of $\mathcal{D}_\pi$ is the $\pi(i)$-th element in $\mathcal{D}$*, i.e.

$$\mathcal{D}_\pi[i] = d_{\pi(i)} \tag{1}$$

In other words, $\mathcal{D}_\pi = \pi^{-1}(\mathcal{D})$. To protect the secrecy of π, the permutation is always coupled with encryption operations. In the rest of the paper, we use $\mathcal{D}_\pi[i] \simeq d_{\pi(i)}$ to denote that $\mathcal{D}_\pi[i]$ is the ciphertext of $d_{\pi(i)}$. To illustrate the idea, a trivial example is as follows. Let $\pi = (1324)$, which means $\pi(1) = 3, \pi(2) = 4, \pi(3) = 2, \pi(4) = 1$. Then for $\mathcal{D} = [d_1, d_2, d_3, d_4]$, we have $\mathcal{D}_\pi \simeq [d_3, d_4, d_2, d_1]$. To distinguish the entries in the original plaintext database and the permuted and encrypted database, we use the convention throughout the paper that data *items* refer to those in $\mathcal{D}$ and data *records* refer to those in $\mathcal{D}_\pi$.

Architecture. As shown in Figure 1 below, our hardware-based PIR scheme comprises of three types of entities: a group of *users*; a *server* and a *trusted hardware* denoted by TH. The server hosts a permuted and encrypted version of a database $\mathcal{D} = [d_1, \cdots, d_n]$, which consists of n items of equal length[4]. TH is a secure and tamper-resistant device residing on the server. With limited computation power and storage cache, TH shuffles the original database $\mathcal{D}$ into database $\mathcal{D}_\pi$ based on a permutation π; it remembers the permutation π and answers users' queries.

Each user interacts with TH via a secure channel, e.g. a SSL connection. When a user wants to retrieve the i-th data item of $\mathcal{D}$, she sends a query q to TH through the channel. On receiving q, TH accesses the permuted database $\mathcal{D}_\pi$ and retrieves the intended item d_i. Throughout the paper, by using "$q = i$", we mean the query q requests the i-th item of $\mathcal{D}$. By saying "the index of a (data) item", we refer to its index in the original database $\mathcal{D}$, by saying "the index of a (data) record", we refer to its index in the shuffled database in which the record locates.

[4] If necessary, we use padding for those data items with different length. Different to the bit model in [6], we extend it to the block model.

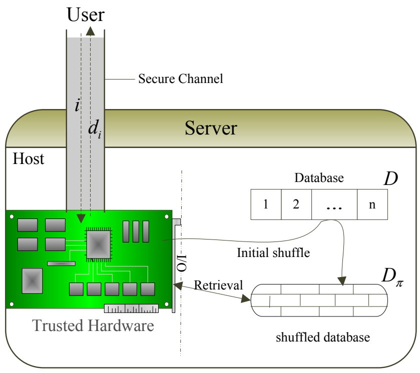

Fig. 1. Hardware-based PIR Model

Trusted Hardware TH is trusted in the sense that it honestly executes the PIR protocol. Neither outside adversaries nor the server is able to tamper its execution or access its private space. Its limited private cache is able to store up to k ($k << n$) data *items* along with their indices. The indices of those cached items are managed by TH using a list denoted by Γ. In other words, Γ stores the original indices. We assume TH is capable of performing CPA-secure (i.e., secure under Chosen Plaintext Attacks) symmetric key encryption/decryption and to generate random numbers or secret keys.

Access Pattern: As in [11], the access pattern for a time period is defined $\mathcal{A} = (a_1, \cdots, a_N)$, where a_i is the data record read in the i-th database access, for $i \in [1, N]$. We observe that a record in the access pattern is essentially a probabilistic result of both the current query and the query history. In fact, the latter results in the current state of TH and the database.

Adversary: We consider adversaries who attempt to derive non-trivial information from the user's queries. Possible adversaries include both outside attackers and the server. Note that we do not assume any trust on the server. The adversary is able to monitor all the input and output of TH. Moreover, the adversary is allowed to query the database at her will and receives the replies from TH.

Stained Query and Clean Query: A query is stained if its content, e.g. the index of the requested data, is known to the adversary without observing the access pattern. This may occur in several scenarios. For instance, a query is compromised or revealed accidently; 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.

Security Model: Our security model follows the security notion in ORAM [11]. We measure the information leakage from PIR query executions. A secure PIR scheme ensures that the adversary does not gain additional information to determine the distribution of queries. Formally, we define it as follows.

Definition 1. *A hardware-based PIR scheme is secure, if given a* clean *query q and an access pattern $\mathcal{A}$, the conditional probability for the event that the query is on index j (i.e., q=j) is the same as its a-priori probability, i.e.*

$$\Pr(q = j|\mathcal{A}) = \Pr(q = j), \text{ for all } j \in [1, n].$$

The equation implies that the access pattern $\mathcal{A}$ reveals to the adversary no information on the target query q's content.

Table 1 below highlights the notations used throughout this paper.

Table 1. Notations

Notation	Description
k	The maximum number of data items cached in 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 random permutations of n elements $\{1, 2, \cdots, n\}$.
$\mathcal{D}_{\pi_s}$	A permuted 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 TH during its i-th access to a shuffle 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*, as defined in Section 3.
Γ	The list of (original) indices of all data items stored in TH's cache.

3 The PIR Scheme

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 other scenarios where the TTP is not available, an alternative solution is provided in Section 6.

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 TH, which completes the system initialization.

The outline of our PIR scheme is as follows. Every k 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 database, and the encryption key respectively. On receiving a query from the user, 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 private cache. Then

TH replies to the user with the desired data item. The detailed operations on data retrieval are shown in Algorithm 1. After k queries are executed, TH generates a new random permutation π_{s+1} and an encryption key sk_{s+1}. It reshuffles $\mathcal{D}_{\pi_s}$ into $\mathcal{D}_{\pi_{s+1}}$ by employing π_{s+1} and sk_{s+1}. Note that in the newly shuffled database $\mathcal{D}_{\pi_{s+1}}$, all data items are encrypted under the secret key sk_{s+1}. The details on database reshuffle are given in Algorithm 2.

The original database $\mathcal{D}$ is not involved in any database retrieval operations. Since 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 the presentation compact and concise.

Retrieval Query Process

The basic idea of our retrieval algorithm is the following. TH always reads a different record on every query and every record is accessed at most once. Thus, if the database is well permutated (in the sense of oblivious permutation), all database accesses within the 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 ($s \geq 0$). On receiving the query for index i, TH performs the following: If the requested d_i is not in TH's cache, it locates the corresponding record in the permutated database $\mathcal{D}_{\pi_s}$ by computing the record index as $\pi_s^{-1}(i)$. If the requested item resides in TH, it reads from $\mathcal{D}_{\pi_s}$ a random record which is not accessed before [5]. The algorithm is elaborated in Figure 2 below.

`Algorithm 1: Retrieving record` d_i `using` $\mathcal{D}_{\pi_s}$

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

Fig. 2. Retrieval Query Processing Algorithm

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 2 that on each query, TH reads exactly one data record from $\mathcal{D}_{\pi_s}$. Therefore, when the s-th session terminates, $\mathcal{A}_s$ has exactly k records.

[5] The operation should be coded so that both "if" and "else" situations take the same time to stand against side-channel attack. This requirement is applied at the similar situation of reshuffle algorithm later.

Reshuffle Process

After k retrievals, TH's private 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 TH from the same location. Therefore, a reshuffle procedure must be executed at the end of each session.

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

$$\mathcal{D}_{\pi_{s+1}}[j] \simeq \mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)], \tag{2}$$

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

The basic idea of our reshuffle algorithm is as follows. We sort the items in TH's cache in ascending order based on their new positions in $\mathcal{D}_{\pi_{s+1}}$. We observe that those un-cached items in $\mathcal{D}_{\pi_s}$ will also be logically sorted in the ascending order based on their new indexes, because the database supports index-based record retrieval. The reshuffle process is similar to a merge-sort of the sorted cached items and un-cached items. 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}.

TH first sorts indices in Γ based on the ascending order of their images under π_{s+1}^{-1}. It assigns database $\mathcal{D}_{\pi_{s+1}}$ sequentially, starting from $\mathcal{D}_{\pi_{s+1}}[1]$. For the first $n-k$ assignments, TH always performs one read operation and one write operation; for the other k assignments, TH always performs one write operation. 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., $\pi_{s+1}(j) \notin \Gamma$): 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 $\mathcal{D}_{\pi_{s+1}}[j]$, TH also performs a read operation for two purposes: (a) to demonstrate the same reading pattern as if in Case (i) so that the secrecy of π_{s+1} is protected; (b) to save the cost of future reads. Moreover, instead of randomly reading a record from the $\mathcal{D}_{\pi_s}$, 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 totally costs k comparisons for the entire reshuffle process. The benefit of this approach coupled with a sorted Γ is that both the costs for testing if $\pi_{s+1}(j) \in \Gamma$ and the item retrieval from the cache are $O(1)$; Otherwise, both cost $O(k)$ per item and $O(nk)$ in total.

The details of the reshuffle algorithm are shown in Figure 3, where *min* denotes the head of sorted Γ. We use `sortedel`(i)/ `sortins`(i) to denote the sorted deletion/insertion of index i from/to Γ and subsequent adjustments.

Fig. 3. Database Reshuffle Algorithm

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

1. secretly select a new random permutation π_{s+1};
2. sort indices in Γ based on the order of their images under π_{s+1}^{-1}.
   set $j = 1; j' = 1$.
3. **while** $1 \leq j \leq n - k$ **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 $\mathcal{D}_{\pi_{s+1}}[j] \simeq d_{min}$; [a]
10.   `sortedel`(j); Insert $\mathcal{D}_{\pi_s}[r]$ into cache and `sorteins`(j');
11. $j = j + 1; j' = j' + 1$;
12. **end**{while};
13. **while** $n - k + 1 \leq j \leq n$ **do**
14.   set $\mathcal{D}_{\pi_{s+1}}[j] = d_{min}$; `sortdel`$(j)$;
15. $j = j + 1$;
16. **end**

[a] d_{min} is exactly $\mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)]$ since $\mathcal{D}_{\pi_{s+1}}$ is filled in by an increasing order.

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 by the reshuffle. Other items are indeed relatively linkable between sessions. However, the linkage does not provide the adversary any advantage since they have not been accessed at all. Note that the addition in the inner loop 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 at most n comparisons are needed for the whole execution. Therefore, the overall computation complexity of Algorithm 2 is $O(n)$.

Reshuffle Pattern The access pattern produced by Algorithm 2 is denoted by $\mathcal{R}_s$. Since TH only reads $n - k$ data records, $\mathcal{R}_s$ has exactly $n - k$ 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.

4 Security

We now proceed to analyze the security of our scheme based on the notion defined in Section 2. Our proof essentially goes as follows. Lemma 1 proves that the reshuffle procedure is oblivious in the same notion as in Oblivious RAM [11]. Thus after each reshuffle, the database is reset into the initial state such that the accesses between different sessions are not correlated. Then in Theorem 1 we show that each individual query session does not leak information of the query, which leads to the conclusion on user privacy across all sessions.

Lemma 1. *The reshuffle algorithm in Figure 3 is oblivious. For any non-negative integer s, any integer $j \in [1, n]$,*

$$\Pr(\mathcal{D}_{\pi_s}[j] = d_l \mid \mathcal{A}_0, \mathcal{R}_0, \cdots, \mathcal{A}_{s-1}, \mathcal{R}_{s-1}) = 1/n, \tag{3}$$

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

Proof We prove Lemma 1 for a fixed j by induction on the session index s. 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 $\Pr(\mathcal{D}_{\pi_0}[j] = d_l \mid \emptyset) = 1/n$ holds for all $1 \leq l \leq n$.

II. Suppose the lemma is true for $s = i$, that is, $\Pr(\mathcal{D}_{\pi_i}[j] = d_l \mid \mathcal{A}_0, \mathcal{R}_0, \cdots, \mathcal{A}_{i-1}, \mathcal{R}_{i-1}) = 1/n$. We proceed to prove that it holds for $s = i + 1$, i.e.

$$\Pr(\mathcal{D}_{\pi_{i+1}}[j] = d_l \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i) = 1/n, \tag{4}$$

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,

$$\begin{aligned} &\Pr(\mathcal{D}_{\pi_{i+1}}[j] = d_l \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i) \\ &= \sum_{x=1}^{n} \Pr(\mathcal{D}_{\pi_{i+1}}[j] = \mathcal{D}_{\pi_i}[x] \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i) \cdot \Pr(\mathcal{D}_{\pi_i}[x] = d_l \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i). \end{aligned} \tag{5}$$

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 right hand side of the equation sums to be $1/n$ in any case.

For clarity, we define p_x and q_x by

$$p_x = \Pr(\mathcal{D}_{\pi_{i+1}}[j] = \mathcal{D}_{\pi_i}[x] \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i),$$

and

$$q_x = \Pr(\mathcal{D}_{\pi_i}[x] = d_l \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i).$$

The objective now is to prove

$$\sum_{x=1}^{n} p_x q_x = 1/n. \tag{6}$$

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

$$\sum_{x=1}^{n} p_x q_x = \sum_{x \in X} p_x q_x + \sum_{x \in Y} p_x q_x. \tag{7}$$

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 TH's cache, i.e. those whose indices 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 indices 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 $k+1$ positions in $\mathcal{D}_{\pi_{i+1}}$. Therefore, only for $x \in X$,

$$p_x = \Pr(\pi_{i+1}(j) = \pi_i(x)) = 1/n \tag{8}$$

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

$$\sum_{x \in Y} p_x = 1 - \sum_{x \in X} p_x = (n-k)/n. \tag{9}$$

The computation of q_x is related to the stained queries. Recall that our adversary is allowed to explore the protocol by sending queries and receiving replies. Therefore, queries in TH could be stained. 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

$$\sum_{x=1}^{n} p_x q_x = 1/n + 0 = 1/n, \text{ for } l \in \mathcal{C}. \tag{10}$$

- 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: $\Pr(\mathcal{D}_{\pi_i}[x] = d_l \mid \mathcal{A}_0, \mathcal{R}_0, \cdots, \mathcal{A}_{i-1}, \mathcal{R}_{i-1}) = 1/n$, for all $l \in [1,n]$, we have $q_x = \frac{1-\delta}{n-k}$ due to equiprobability [6]. Hence,

$$\sum_{x=1}^{n} p_x q_x = \delta/n + (1-\delta)/n = 1/n. \tag{11}$$

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

Combining Case 1 and Case 2, we have

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

which concludes the proof. □

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

Theorem 1. *Given a time period, the observation of the access pattern* $\mathcal{A} = (a_1, a_2, \cdots, a_N)$*,* $N > 0$*, provides the adversary no additional knowledge to determine any* clean *query* q*, i.e. for all* $j \in [1, n]$*,*

$$\Pr(q = j | \mathcal{A}) = \Pr(q = j) \tag{12}$$

where $\Pr(q = j)$ *is an a-priori probability of query* q *being on index* j*.*

Proof For $1 < t \leq N$, 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 $\Pr(a_t \mid a_1, \cdots, a_{t-1}; q = j)$ denote the probability of the same event with additional knowledge that the requested index of query q is j. Note that we do not assume any temporal order of the query q and the t-th query. We proceed to show below that

$$\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}; q = j) \tag{13}$$

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

   Note that the compromise of a query, i.e. knowing $q = j$, 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, $\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}, q = j)$ when $a_t \in \mathcal{A}_s$.

In total, we conclude that for any t and $q = j$,

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

As a result,

$$\begin{aligned}
\Pr(\mathcal{A} \mid q = j) &= \Pr(a_1, \cdots, a_N \mid q = j) \\
&= \Pr(a_N \mid a_1, \cdots, a_{N-1}; q = j) \cdot \Pr(a_1, \cdots, a_{t-1} \mid q = j) \\
&= \prod_{t=1}^{N} \Pr(a_t \mid a_1, \cdots, a_{t-1}; q = j) \\
&= \prod_{t=1}^{N} \Pr(a_t \mid a_1, \cdots, a_{t-1}) \\
&= \Pr(\mathcal{A}). \qquad (14)
\end{aligned}$$

Then,

$$\begin{aligned}
\Pr(q = j \mid \mathcal{A}) &= \Pr(q = j, \mathcal{A})/\Pr(\mathcal{A}) \\
&= \frac{\Pr(\mathcal{A} \mid q = j) \cdot \Pr(q = j)}{\Pr(A)} \\
&= \Pr(q = j).
\end{aligned}$$

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

5 Performance

We proceed to analyze the communication and computation complexity of our PIR scheme. They are evaluated with respect to the database size n. Both complexities of our scheme reach the respective lower bounds for hardware-based PIR schemes.

Communication: We consider the user/system communication cost per query. In our scheme, the users only inputs an index of the desired data item and TH returns exactly one data item. Therefore, its communication complexity per query is $O(\log n)$. Note that $O(\log n)$ is the lower bound of communication cost for all PIR constructions.

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 also measure the online cost which excludes the expense of offline reshuffle operations.

As evident in Figure 2 and 3, it costs the trusted hardware $O(1)$ operations to process a query and $O(n)$ operations to reshuffle the database. Table 2 compares the computation cost of our scheme against [20] and [14, 15].

Table 2. Comparison of Computation Cost of Three Hardware-based PIR Schemes.

Schemes	**Total cost per session (k queries)**	**Online cost per query**	**Average Cost per query**
Our scheme	$O(n)$ [a]	$O(1)$	$O(n/k)$
IS04 [14, 15]	$O(n \log n)$ [b]	$O(1)$	$O(\frac{n}{k} \log n)$
SS01 [20]	$O(kn)$	$O(n)$	$O(n)$

[a] We only take count of the main cost for simplicity and comparison clarity. The actual cost should be $O(n) + O(k \log k)$, where the later quantity comes from sorting T, once each session, which costs $k \log k$ operations.

[b] Their actually cost is approximately $O(n \log n) + O(n) + O(k \log k)$, where $O(n)(+O(k \log k))$ comes from sorting $\bar{T}$ (Step (i) of [14]), $k \log k$ comes from sorting T (Step (ii)), and $O(n \log n)$ comes from Step (iii). In fact, our whole reshuffle only costs what their Step (i) costs.

Our scheme outperforms the other two hardware-based PIR schemes in all three metrics. The advantage originates from our reshuffle algorithm which utilizes the hardware's cache in a more efficient manner. Moreover, we prove that the average cost per retrieval of our scheme reaches the lower bound for all information-theoretic PIR schemes with the same trusted hardware system model. Our result is summarized in the following theorem.

Theorem 2. *For any information-theoretically secure PIR scheme with a trusted hardware storing maximum k data items, the average computation cost per retrieval is $\Omega(n/k)$.*

Proof: Our proof is constructed in a similar manner to the proof in [4] which shows the computational lower bound for traditional PIR schemes.

Fix a PIR scheme, let B_i denote the set of all indices that the hardware reads in order to return d_i. B_i is essentially a random variable probabilistically determined by both the user query on d_i and the items that the hardware has already stored inside. Consequently, $\mathrm{E}(|B_i|)$ denotes the expect number of data items to read by the hardware to process a query on index i. We evaluate $\mathrm{E}(|B_i|)$ as the computation cost for PIR schemes.

For $1 \leq l \leq n$, let $\Pr(l \in B_i)$ be the probability that the hardware reads d_l in order to answer the query on index i. We define $\mathrm{P}(l)$ as the maximum of these probabilities for all $1 \leq i \leq n$, i.e.

$$\mathrm{P}(l) = \max_{1 \leq i \leq n} \{\Pr(l \in B_i)\}.$$

Note that for an information-theoretically secure PIR scheme, user privacy implies that $B_1, B_2, \cdots, B_n$ have the identical distribution. Therefore, for convenience purpose, let

$$\mathrm{P}(l) = \Pr(l \in B_1).$$

Due to Lemma 5 of [4][7],

$$\mathrm{E}(|B_i|) = \sum_{l=1}^{n} \mathrm{P}(l). \tag{15}$$

Our target now is to show $\mathrm{E}(|B_i|) = \Omega(n/k)$. We prove it by contradiction.

Suppose that among $\mathrm{P}(1), \cdots, \mathrm{P}(n)$, there at least exist $k+1$ of them whose values are less than $1/(k+1)$. Without loss of generality, let the $k+1$ probabilities be $\mathrm{P}(1), \mathrm{P}(2), \cdots, \mathrm{P}(k+1)$. Now consider the probability $\Pr(1 \notin B_1 \cap 2 \notin B_2 \cdots \cap (k+1) \notin B_{k+1})$. We have,

$$\begin{aligned}
&\Pr(1 \notin B_1 \cap 2 \notin B_2 \cdots \cap (k+1) \notin B_{k+1}) \\
&= 1 - \Pr(1 \in B_1 \cup 2 \in B_2 \cdots \cup (k+1) \in B_{k+1}) \\
&\geq 1 - \sum_{l=1}^{k+1} \Pr(l \in B_l) = 1 - \sum_{l=1}^{k+1} \mathrm{P}(l) \\
&> 1 - (k+1)\frac{1}{k+1} = 0.
\end{aligned}$$

On the other hand, note that TH only caches k data items in maximum. As a consequence, there always exists one data item which must be read from the database during the $k+1$ queries on $1, 2, \cdots, k+1$. Thus, the event that $1 \notin B_1 \cap 2 \notin B_2 \cdots \cap (k+1) \notin B_{k+1}$ never occurs, i.e. $\Pr(1 \notin B_1 \cap 2 \notin B_2 \cdots \cap (k+1) \notin B_{k+1}) = 0$, which contradicts the probability computation above.

Thus, at most k elements in $\{\mathrm{P}(1), \cdots, \mathrm{P}(n)\}$ whose values are less than $1/(k+1)$. As a result,

$$\sum_{l=1}^{n} \mathrm{P}(l) \geq (n-k) \cdot \frac{1}{k+1}, \tag{16}$$

which shows the lower bound for average computation cost is $\Omega(n/k)$. □

6 Discussion

Database Initialization Using TH

For applications where no trusted third party exists, TH can be used to initialize the database. TH first chooses a random permutation π_0. For $1 \leq i \leq n$, TH tags the i-th item d_i with its new index $\pi_0^{-1}(i)$. Using the merge-sort algorithm [8], $d_1, d_2, \cdots, d_n$ are sorted based on their new indices by TH. With the limited cache size in TH, Batcher's odd-even merges sorter [1] is an appropriate choice which requires $(\log^2 n - \log n + 4)n/4 - 1$ comparisons. One may argue that Beneš network [22] and Goldstein et al's switch networks [12] incur less comparisons. Unfortunately, neither is feasible in our system since the first one requires at

[7] It can be proved by defining the random variables $Y_1, \cdots, Y_n$ where $Y_l = 1$ if $l \in B_i$ and $Y_l = 0$ otherwise.

least $n \log n$-bit ($>> k$) memory in TH while 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 Fig. 4. The database in the example has 4 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 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 on the host. In every operation, only two items are read into TH's cache and then written back to the server.

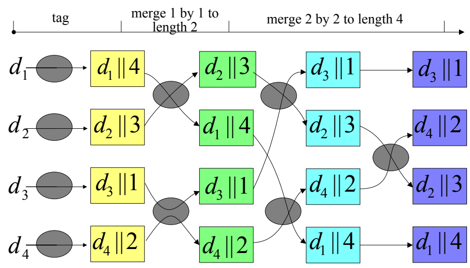

Fig. 4. Initial oblivious shuffle example using odd-even merges.

Instantiation of Encryption and Permutation Algorithms

An implicit assumption of our security proof in Section 4 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 2 allows the adversary to submit queries and observe the subsequent access patterns and replies. Thereafter, the adversary is able to obtain k pairs of plaintext and ciphertext in maximum for each encryption key, since different random keys are selected in different sessions. Thus, it is demanded to have 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.

Since a cipher secure under CPA is transformed into an invertible pseudo-random permutation, we choose a CPA secure block cipher, e.g. AES, to implement the needed pseudo-permutation. With a block cipher, a message is encrypted by blocks. When $n \neq 2^{\lceil \log n \rceil}$, the ciphertext may be greater than n.

In that case, the encryption is repeated until the output is in the appropriate range. Since $2^{\lceil \log n \rceil} \leq 2n$, the expected number of encryptions is less than 2. Black and Rogaway's result in [5] provides more information on ciphers with arbitrary finite domains.

Service Continuity

The database service is disrupted during the reshuffle process. The duration of a reshuffle is non-negligible since it is an $O(n)$ process. A trivial approach to maintaining the continuity of service is to deploy two pieces of trusted hardware. While one is re-permuting the database, the other deals with user queries. In case that installing an extra hardware is infeasible, an alternative is to split the cache of the hardware into two halves with each having the capacity of storing $k/2$ items. Consequently, the average computation cost will be doubled.

Update of data items

A byproduct of the reshuffle process is database update operations. To update d_i, the trusted hardware reads d_i obliviously in the same way as handling a read request. Then, d_i is updated inside the hardware's 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 the hardware returns the updated value directly from the cache upon user requests.

7 Conclusion

In summary, we present in this paper a novel PIR scheme with the support of a trusted hardware. The new PIR construction is provably secure. The observation of the access pattern does not offer additional information to adaptive adversaries in determining the data items retrieved by a user.

Similar to other hardware-based PIR schemes, the communication complexity of our scheme reaches its lower bound, $O(\log n)$. In terms of computation complexity, our design is more efficient than all other existing constructions. Its online cost per query is $O(1)$ and the average cost per query is only $O(n/k)$, which outperforms the best known result by a factor $O(\log n)$ (though using a big-O notation, the hidden constant factor is around 1 here). Furthermore, we prove that $O(n/k)$ is the lower bound of computation cost for PIR schemes with the same trusted hardware based architecture.

The Trusted Computing Group (TCG) [21] defines a set of Trusted Computing Platform (TCP) specifications aiming to provide hardware-based root of trust and a set of primitive functions to propagate trust to application software as well as across platforms. How to extend and improve our proposed PIR scheme based on trusted computing technologies will be one of our future research directions.

Brief introduction to Trusted Computing

The root of trust in TCP is a hardware component on the motherboard called the Trusted Platform Module (TPM). TPM provides protected data by never releasing root keys outside of the TPM. In addition, TPM provides some primitive cryptographic functions, such as random number generation, RSA key pair generation, RSA algorithms and hash function. Most importantly, TPM provides mechanism for integrity measurement, storage, and reporting of a platform, from which trust and attestation capabilities can be built. In addition to TCG compliant TPM, Intel's LaGrande Technology (LT) [13] includes an extended CPU enabling software domain separation and protection. Beyond the hardware layer there is a domain manager supporting protected execution environments by domain separation, including separation of processes, memory pages, and device drivers. This emergence of industry standard trusted computing technologies promise to establish an adequate foundation for building a practical trusted platform for our proposed PIR scheme.

References

1. Kenneth E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.
2. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-key Encryption Schemes. In *Proceedings of Crypto '98*, LNCS 1462, pages 26–45, Berlin, 1998.
3. Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $o(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *FOCS*, pages 261–270. IEEE Computer Society, 2002.
4. Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *CRYPTO*, pages 55–73, 2000.
5. John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In Bart Preneel, editor, *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2002.
6. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.
7. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
8. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms, Second Edition. ISBN 0-262-03293-7.
9. Joan Feigenbaum. Encrypting problem instances: Or ..., can you take advantage of someone without having to trust him? In Hugh C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 477–488. Springer, 1985.
10. William Gasarch. A survey on private information retrieval. *The Bulletin of the European Association for Theoretical Computer Science*, Computational Complexity Column(82), 2004.
11. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

12. J. L. Goldstein and S. W. Leibholz. On the synthesis of signal switching networks with transient blocking. IEEE Transactions on Electronic Computers. vol.16, no.5, 637-641, 1967.
13. LaGrande technology architecrure. Intel Developer Forum, 2003.
14. Alexander Iliev and Sean Smith. Private information storage with logarithm-space secure hardware. In *International Information Security Workshops*, pages 199–214, 2004.
15. Alexander Iliev and Sean W. Smith. Protecting client privacy with trusted computing at the server. *IEEE Security & Privacy*, 3(2):20–28, 2005.
16. Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2005.
17. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
18. Jacques Patarin. Luby-rackoff: 7 rounds are enough for $2^{n(1-\epsilon)}$ security. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 513–529. Springer, 2003.
19. Ronald L. Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms.
20. Sean W. Smith and David Safford. Practical server privacy with secure coprocessors. *IBM Systems Journal*, 40(3):683–695, 2001.
21. TCG Specification Architecture Overview. Available from http://www.trustedcomputinggroup.org.
22. Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968.