# Private Information Retrieval by Keywords

Benny Chor* Niv Gilboa† Moni Naor‡

**Abstract**

Private information retrieval (PIR) schemes enable a user to access one or more servers that hold copies of a database and *privately* retrieve parts of the $n$ bits of data stored in the database. This means that the queries give each individual database no partial information (in the information theoretic or computational sense) on the identity of the item retrieved by the user.

All known PIR schemes assume that the user knows the *physical address* of the sought item. This is usually not the case when accessing a public database that is not managed by the user. Such databases are typically presented with keywords, which are then internally translated (at the database end) to physical addresses, using an appropriate search structure (for example, a hash table or a binary tree). In this note we describe a simple, modular way to privately access data by keywords. It combines *any* conventional search structure with *any* underlying PIR scheme (including single server schemes). The transformation requires no modification in the way that the search structure is maintained. Therefore the same database will support both private and regular (non private) searches.

*Computer Science Dept., Technion, Haifa, Israel. Supported by the fund for the promotion of research at the Technion. Email: benny@cs.technion.ac.il

†Computer Science Dept., Technion, Haifa, Israel. Email: gilboa@cs.technion.ac.il

‡Computer Science and Applied Math. Dept., Weizmann Institute of Science, Rehovot, Israel. Email: naor@wisdom.weizmann.ac.il

# 1 Introduction

The problem of Private information retrieval ($\mathcal{PIR}$) combines two major needs in the computer world. One is to access electronically and remotely information that is located in appropriately organized data bases. The second is to access these databases privately. Privacy is an increasing concern in open networks like the Internet. This concern has led in the last years to a number of works, both in the research community and in commercial applications.

Private information retrieval schemes enable a user to access (a single or replicated copies of) databases and retrieve information without revealing the identity of the sought items to any individual database [3, 1, 2, 11]. A major drawback of all $\mathcal{PIR}$ schemes known today is the assumption that the user knows the *physical address* of the sought item. This is usually not the case in most databases in use. Instead, the user typically holds a keyword (for example, the name of a specific company traded in the stock market), and the database internally converts from this keyword to physical addresses.

In this work we describe a simple, modular way to privately access data by keywords. Our scheme combines $\mathcal{PIR}$ solutions together with data structures that support search operations, in order to retrieve information privately in the keyword model. We present a general transformation from $\mathcal{PIR}$ schemes to retrieval by keywords schemes, using a large class of data structures. We also describe specific instantiations of this transformation.

The main idea in constructions is the following: the databases insert $s_1, \ldots, s_n$ into a data structure, which supports search operations on strings. The user conducts an oblivious walk on the data structure until either the word $w$ is found, or $\mathcal{U}$ is assured of the fact that $w$ is not one of $s_1, \ldots, s_n$. A typical search in the data structure involves a sequence of operations, where each operation consists of fetching the contents of a word from memory, performing a "local" computation, which depends on the keyword and the fetched contents, and either determining a new address based on the computation, or terminating the search (successfully or unsuccessfully). This sequence of operations can be viewed as a walk on the data structure. By employing a $\mathcal{PIR}$ scheme, we transform this walk into an *oblivious walk* on the data structure, namely a walk where each server gets no information on the walk (and, therefore, on the desired keyword itself).

## 1.1 Related Work

The work that introduced the problem of retrieving information privately is [3]. A user wishes to privately retrieve the $i$-th bit in the database, without revealing any information about $i$. A distributed setting was suggested in which:

- There are $k$, $k > 1$, databases which hold copies of the same string $x$.
- The databses are allowed to communicate with the user, but not with one another.

In [1] Ambainis generalized several of the results in [3]. The two works combined present the following schemes:

1. A $k$ database scheme, for any constant $k \geq 2$, with communication complexity $O(n^{\frac{1}{2k-1}})$.

2. An $O(\log n)$ database scheme with communication complexity $O(\log^2 n \cdot \log\log n)$.

A related problem that was studied in [3], and is especially important in the current work of private retrieval by keywords, is the problem of private block retrieval. In this scenario, each database holds, instead of a string of $n$ bits, $n$ blocks of $\ell$ bits each. The user's goal is to privately retrieve one of the $n$ blocks. Further details on this problem and possible solutions appear in section 3.

Both [3] and [1] require the privacy of the user to be protected in an *information theoretic* sense. In [2, 11] and partially in [14], this is replaced with *computational privacy.* It is assumed that the databases are computationally bounded. Under appropriate intractability assumptions, the databases cannot gain information about $i$. The advantage of [2, 11, 14] is that their schemes have significantly lower communication complexity than those in [1, 3].

A computationally private,two database scheme is presented in [2]. The cryptographic assumption in this work is that pseudo-random generators exist (alternatively that one way functions exist). The communication complexity of the scheme is $O(n^\epsilon)$ for any constant $\epsilon > 0$.

In [3] it was proved that if there is only one database, and *information theoretic privacy* is required, then the communication complexity of a $\mathcal{PIR}$ scheme is $\Omega(n)$. However, Kushilevitz and Ostrovsky construct in [11] a *computationally private*, one database scheme, with communication complexity $O(n^\epsilon)$ for any constant $\epsilon > 0$. They assume that the problem of determining quadratic residuosity modulo composite numbers [9] is intractable.

In [14] Ostrovsky and Shoup construct computationally private schemes that are based on the *oblivious RAM* technique [5, 8, 13], with poly-logarithmic communication complexity. The assumption used in this work is that trap door permutations exist. The scheme is executes in $O(\log n)$ rounds of communication, compared with 1 round for all other $\mathcal{PIR}$ schemes.

Gertner *et. al.* [7] have extended the privacy requirement so that the database's privacy is protected too. Further details on that problem, and its application to the current paper appear in subsection 6.2.

### Organization

In section 2 we introduce basic definitions and notations. In section 3 we briefly discuss the various known $\mathcal{PIR}$ schemes, in which a block of $\ell$ bits is retrieved in an efficient manner. In section 4 we show general reductions from each of the problems: $\mathcal{PIR}$ and private retrieval by keywords, to the other. In section 5 we present three examples of the general reduction from $\mathcal{PIR}$ to private retrieval by keywords, using specific data structures. In section 6 we discuss two unrelated subjects. The first is how to reduce the communication complexity of our schemes, at the the cost of introducing errors (with low probability). The second is a scheme which maintains the privacy of both the user and the databases.

## 2 Definitions and Notations

In this work we are interested in two related problems: private information retrieval (PIR) and private retrieval by keywords. In both cases there are is a single user, denoted by $\mathcal{U}$ and $k \geq 1$

databases, denoted by $\mathcal{DB}_\infty, \dots, \mathcal{DB}_{||}$, which hold copies of the same data. The first problem was defined in [3] in the following way:

**Definition 1:** Suppose that each database holds $x \in \{0,1\}^n$, and that the user holds $i \in \{1, \dots, n\}$. A solution to the *private information retrieval* problem is a protocol, which allows the user to retrieve the value of the $i$-th bit in $x$ without leaking information about $i$ to any individual database.

In [3] the authors also introduced a generalization of the PIR problem in which blocks of bits are retrieved. In this setting the databases hold $n$ blocks of $\ell$ bits each, and the user wishes to privately retrieve the $i$-th block. This problem is denoted by $\mathcal{PIR}(\ell, n, k)$. If $\ell = 1$, the problem of retrieving blocks is simply the PIR problem and is denoted by $\mathcal{PIR}(n, k)$. The second problem we deal with in this work is defined as follows:

**Definition 2:** Suppose that each of the databases holds the $n$ binary strings $s_1, \dots, s_n \in \{0,1\}^\ell$, and that $\mathcal{U}$ holds a binary string $w \in \{0,1\}^\ell$. A solution to the *private retrieval by keywords* problem is a protocol which allows the user to find out if for some $j \in \{0,1\}^n$, $w = s_j$, without leaking any information about $w$ to the databases.

We denote the PrivatE Retrieval by KeYwords problem with parameters $\ell, n, k$ (length of strings, number of strings and number of databases respectively) by $\mathcal{PERKY}(\ell, n, k)$.

Given a scheme, $\mathcal{P}$, that solves the problem $\mathcal{PIR}(\ell, n, k)$, we denote its communication complexity by $C(\ell, n, k)$. We use the notation $C(n, k)$ to denote $C(1, n, k)$. Its round complexity is denoted by $R(\ell, n, k)$ or $R(n, k)$. We use $\sigma$ to denote the security parameter in computationally private schemes.

# 3 Private Retrieval of Blocks

The model of private information retrieval of a single bit is interesting from a theoretic point of view. A more realistic model [3] assumes that the data is partitioned into blocks (or records), and a whole block has to be retrieved. For simplicity, we assume that each block/record contains the same number of bits, $\ell$. Clearly $\mathcal{PIR}(\ell, n, k)$ can be solved by $\ell$ invocations of $\mathcal{PIR}(n, k)$ (namely $\ell$ iterations of a solution to $\mathcal{PIR}(n, k)$). We are interested in more efficient solutions. It is evident that for any $\mathcal{PIR}$ scheme, $C(\ell, n, k) \geq \ell$, since $\ell$ bits have to be retrieved. We thus divide the retrieved blocks into two categories:

1. "Large" blocks, which can be retrieved with a constant communication overhead for each bit, total communication complexity $O(\ell)$.

2. "Small" blocks, which are retrieved with communication complexity $\omega(\ell)$.

This distinction depends on the $\mathcal{PIR}$ setting: the number of databases and the required type of privacy. A large block in one setting may be small in another.

In the *information-theoretic* setting of [3] the following results are known:

**Claim 1:**

1. For any constant $k$, $k \geq 2$ and for any $\ell$, $\ell \geq n^{\frac{1}{k-1}}$, there exists a $\mathcal{PIR}(\ell, n, k)$ scheme with communication complexity $O(\ell)$.
2. For any constant $k$, $k \geq 2$ and for any $\ell$, there exists a $\mathcal{PIR}(\ell, n, k)$ scheme with communication complexity $O(n^{\frac{1}{2k-1}} \cdot \ell^{\frac{k}{2k-1}})$.

The first part of the claim, which deals with large blocks, is proved in [3]. It is explicitly stated, in that work, only for $k = 2$. However, the polynomial interpolation scheme which is constructed in [3] as a solution to $\mathcal{PIR}(n, k)$, for any $k \geq 2$, achieves the result in the claim.

The second part of the claim is proved by combining two techniques: the efficient $\mathcal{PIR}(n, k)$ schemes of [1], and a generalization of the balancing technique of [2]. The full proof appears in [6].

In the one database, computational setting of [11] we do not know of any $\ell$ such that blocks of size greater than $\ell$ can be retrieved with communication comlexity $O(\ell)$. We do have the following result:

**Claim 2:** Assume the intractability of the quadratic residuosity problem, [9], and let the security parameter be $\sigma$. For any block size $\ell$, and constant $d$ there exists a computational $\mathcal{PIR}(\ell, n, 1)$ scheme with communication complexity $\sigma n^{1/d} + \ell \sigma^d$.

In the $\mathcal{PIR}$ scheme of [11] a single bit is retrieved with $\sigma \cdot n^{1/d}$ bits sent by the user and $\sigma^d$ by the database. Combining the above with a balancing technique shown in [3] proves the claim.

In the two database, computational setting of [2] it is possible to construct $\mathcal{PIR}(n, 2)$ schemes in which the user sends $O(n^{1/d})$ bits, for some constant $d$, and each database replies with a single bit. Therefore:

**Claim 3:** Assume the existence of pseudo-random generators. For any constant $d$, and block size $\ell$, $\ell \geq n^{1/d}$, there exists a computational $\mathcal{PIR}(\ell, n, 2)$ scheme with communication complexity $O(n^{1/d})$.

# 4 General Solutions to $\mathcal{PERKY}(\ell, n, k)$

We begin by proving a couple of simple theorems.

**Theorem 1:**

1. The problem $\mathcal{PERKY}(\ell, n, 1)$ can be solved with communication complexity $O(\ell n)$.
2. The problem $\mathcal{PERKY}(\ell, n, k)$ can be solved with the same communication complexity as a solution to $\mathcal{PIR}(1, 2^\ell, k)$.

**Proof:**

1. The database sends all the strings it holds to $\mathcal{U}$.

2. The $n$ strings $s_1, \ldots, s_n$ are replaced with their incidence vector: a $2^\ell$ bit string in which the $j$-th bit is 1 iff the $j$-th $\ell$ bit string, in the lexicographic order, is one of $s_1, \ldots, s_n$. Suppose that the word $\mathcal{U}$ holds, $w$, is the $i$-th word in the lexicographic order. $\mathcal{U}$ and $\mathcal{DB}_1, \ldots, \mathcal{DB}_k$ combine in a $\mathcal{PIR}(1, 2^\ell, k)$ scheme in which the user retrieves the $i$-th bit.

$\square$

**Theorem 2:** The problem $\mathcal{PIR}(n, k)$ can be solved with the same communication complexity as a solution to $\mathcal{PERKY}(\log n + 1, n, k)$.

**Proof:** The databases replace the $n$ bit string they hold, $x = x_1, \ldots, x_n$, with $n$ strings of length $\log n + 1$ bits each. If $j \in \{1, \ldots, n\}$, and its binary expansion is $j_1, \ldots, j_{\log n}$, then the $j$-th string that each database now holds is $j_1, \ldots, j_{\log n}, x_j$. Suppose that the user is interested in the $i$-th bit. That bit is 1 iff the $i$-th string held by the databases is $i_1, \ldots, i_{\log n}, 1$. $\mathcal{U}$ and $\mathcal{DB}_\infty, \ldots, \mathcal{DB}_{||}$ combine in a $\mathcal{PERKY}(\log n + 1, n, k)$ protocol in which the word that $\mathcal{U}$ holds is $i_1, \ldots, i_{\log n}, 1$. $\square$

The above reduction indicates that we cannot hope to find $\mathcal{PERKY}$ schemes which are significantly more efficient than $\mathcal{PIR}$ schemes. Our aim is now to use $\mathcal{PIR}$ schemes in order to construct $\mathcal{PERKY}$ schemes that are more efficient than what can be obtained by theorem 1. The main idea in all of our subsequent $\mathcal{PERKY}$ constructions is the following: the databases insert $s_1, \ldots, s_n$ into a data structure which supports search operations on strings. The user conducts an oblivious walk on the data structure until either the word $w$ is found, or $\mathcal{U}$ is assured of the fact that $w$ is not one of $s_1, \ldots, s_n$. Typically, a successful search yields an address which contains data pertaining to the keyword.

A typical search in the data structure involves a sequence of operations, where each operation consists of fetching the contents of a word from memory, performing a "local" computation, which depends on the keyword and the fetched contents, and either determining a new address based on the computation, or terminating the search (successfully or unsuccessfully). This sequence of operations can be viewed as a walk on the data structure. We now describe a general outline of transforming this walk into an *oblivious walk* on the data structure, namely a walk where each server gets no information on the walk (and, therefore, on the desired keyword itself). For the sake of simplicity, we assume that the data structure has a fixed *root*, a word with known address that is always accessed at the first operation (regardless of the sought keyword).

In the oblivious walk, the operations of the original data structure are divided between the user and the server(s). Each server maintains the original data structure, without any modification. The server supports the user in fetching words with known addresses from the memory, by performing an agreed upon $\mathcal{PIR}$ scheme. The local computations are performed exclusively by the user. The result of each local computation determines the address of the next word to be fetched. If the data structure requires (in the worst case) $d$ memory accesses , the user will invoke $\mathcal{PIR}$ $d$ times successively. If for some given keyword fewer than $d$ invocations are required, the user will still execute $d$ $\mathcal{PIR}$ schemes, with arbitrary (dummy) addresses in the last operations. Otherwise the server learns the search length for this specific keyword.

This discussion leads to the following theorem:

**Theorem 3:** Let $\mathcal{DS}$ be a data structure which enables the search for $n$ keywords, each of length $\ell$ (bits), and has the following properties:

1. Each word in the data structure is of length $m$ bits.

2. There are $t$ such words in the data structure.

3. The worst case search length is $d$.

4. In the $q$-th operation on the data structure, $1 \leq q \leq d$, a contiguous memory block containing $t_q$ words ($m$ bit each) is accessed.

Let $\mathcal{P}$ be a $\mathcal{PIR}(\ell, n, k)$ scheme with communication complexity $C(\ell, n, k)$ and round complexity $R(\ell, n, k)$. Then $\mathcal{PERKY}(\ell, n, k)$ can be solved with communication complexity $\sum_{q=1}^{d} C(m, t_q, k)$, and round complexity $\sum_{q=1}^{d} R(m, t_q, k)$.

**Proof:** **An initial scheme:**

1. The databases insert the $n$ strings into the agreed upon data structure, $\mathcal{DS}$. Each data item (i.e $m$ bit word) contains information about which is the next data item to be retrieved in a particular search operation. The next item is one of $t$ items in the data structure.

2. $\mathcal{U}$ and $\mathcal{DB}_\infty, \ldots, \mathcal{DB}_{||}$ combine in $d$ executions of $\mathcal{P}$. In the first execution the desired address is pre-determined to be the root of the data structure. In the $q$-th execution, $1 < q \leq d$, the address is determined by the answer to the previous query. Since a single item, whose address is known, is privately retrieved out of $t$ items, each of length $m$, the total communication complexity is $d \cdot C(m, t, k)$, and the round complexity is $d \cdot R(m, t, k)$.

**An improved scheme:** We can improve the complexity of the scheme by utilizing property 4 of the data structure.

1. The databases insert the $n$ strings into $\mathcal{DS}$. Each database constructs $d$ arrays, based on the data structure. The $q$-th array, $1 \leq q \leq d$, contains the $t_q$ items on which the $q$-th query(operation) can take place (in some sequence of queries). Since the $q$-th query determines on which item the next query will take place, each of the $t_q$ items may have to contain addresses of items in array number $q + 1$.

2. $\mathcal{U}$ and $\mathcal{DB}_\infty, \ldots, \mathcal{DB}_{||}$ combine in $d$ executions of $\mathcal{P}$ in similar fashion to the initial scheme. The difference is that in the $q$-th execution the total number of items is $t_q$ instead of $t$. $\square$

Most of the known $\mathcal{PIR}$ schemes are executed in one round of communication (one query and one answer). Therefore, the above $\mathcal{PERKY}$ scheme can be carried out in $d$ rounds of communication.

We remark that such schemes support the simultaneous implementation of private as well as "regular" (non-private) retrieval, which is certainly a desirable property from a practical point of view.

It should be pointed out that all the information which enables the search should now be explicitly described in the server(s), and the resulting memory overhead should be accounted for. This is particularly relevant in cases where the data structure is constructed using probabilistic methods. In the next section we describe in detail three data structures that are useful in our context: A binary tree, a trie, and a perfect hash function. The implementation of the perfect hash function uses probabilistic methods.

# 5 Specific Implementations

In the following subsections we give three examples of $\mathcal{PERKY}$ schemes, using a different data structure each time. We use a $\mathcal{PIR}(\ell, n, k)$ scheme, $\mathcal{P}$, as in theorem 3. We denote the communication complexity of $\mathcal{P}$ by $C(\ell, n, k)$ and its round complexity by $R(\ell, n, k)$.

## 5.1 Binary Search Tree

**Claim 4:** There exists a $\mathcal{PERKY}(\ell, n, k)$ scheme with communication complexity $\sum_{q=1}^{\log n} C(\ell, 2^{q-1}, k)$, and round complexity $\sum_{q=1}^{\log n} R(\ell, 2^{q-1}, k)$.

**Proof:** **The Scheme:** We assume that for some positive integer $d$, $n = 2^d$.

1. The databases sort the $n$ strings $s_1, \ldots, s_n$ according to some total order. The strings are then partitioned into $\log n$ arrays of different sizes. Array number $q$ $(1 \leq q \leq \log n)$ contains the $2^{q-1}$ strings that appear in positions $\frac{1}{2^q}n, \frac{3}{2^q}n, \ldots, \frac{2^q-1}{2^q}n$.

2. During the excution of this $\mathcal{PERKY}$ scheme, we refer to an index $j$ $(j \in \{1, \ldots, n\})$ as *possible* for the user, if it is still possible, given the information that the user knows, that $w = s_j$. The user conducts a binary search invoking the $\mathcal{PIR}$ scheme, $\mathcal{P}$, $\log n$ times. In the beginning all of the indices $1, \ldots, n$ are possible. In the $q$-th execution of $\mathcal{P}$ $(1 \leq q \leq \log n)$ the user privately retrieves a single string out of the $2^{q-1}$ strings that form the $q$-th array. The index of the retrieved string is the median of the indices, which are still possible. By construction, this string does appear in the $q$-th array. After each $\mathcal{PIR}$ execution, the number of possible indices is reduced by half, and after $\log n$ executions of $\mathcal{P}$, $\mathcal{U}$ knows whether $w \in \{s_1, \ldots, s_n\}$. Furthermore, if for some $j$, $w = s_j$, the user also discovers what $j$ is.

**Communication Complexity:** The $\mathcal{PERKY}$ scheme we outlined above is made up of $\log n$ $\mathcal{PIR}$ schemes. In the $q$-th $\mathcal{PIR}$ scheme, a string of size $\ell$ is retrieved out of an array of $2^{q-1}$

such strings. Therefore, the total communication complexity is $\sum_{q=1}^{\log n} C(\ell, 2^{q-1}, k)$, and the round complexity is $\sum_{q=1}^{\log n} R(\ell, 2^{q-1}, k)$. □

## 5.2 Trie

**Definition 3:** Let $s$ be a binary string of $q$ bits, $0 \leq q \leq \ell$. We say that $s$ is a *legal prefix*, if it is a prefix of some $s_j$ $(1 \leq j \leq n)$.

**Claim 5:** There exists a $\mathcal{PERKY}(\ell, n, k)$ scheme, which achieves the following:

- Communication complexity $O(\ell \cdot C(\log n + \log \ell, n/\ell, k) + C(\ell, n, k))$
- Round complexity $O(\ell \cdot R(\log n + \log \ell, n/\ell, k) + R(\ell, n, k))$.

**Proof:** **An initial scheme:** We use a *trie* data structure [10, page 481] and hereby describe the arrays that are used to implement it in our context. There are $\ell$ arrays with indices $q = 0, 1, \ldots, \ell - 1$. The $q$-th array represents all the *legal prefixes* of length $q$. Suppose $y \in \{0,1\}^q$, $q = 0, 1 \ldots \ell - 1$ is such a prefix. It is represented by one item in the $q$-th array, which contains two addresses. They are the addresses, in array number $q + 1$, associated with the prefixes $y0, y1 \in \{0,1\}^{q+1}$. If $y0$ (or $y1$) is not legal, the item associated with $y$ contains a string of zeroes instead of the address of $y0$ (or $y1$). In the last array, number $\ell - 1$, each item contains two bits instead of two addresses. The bits denote whether the two possible extensions of an $\ell - 1$ bit prefix to an $\ell$ bit string, are among the strings $s_1, \ldots, s_n$. The number of items in each array depends on the strings $s_1, \ldots, s_n$, but is always bounded by $n$. Since in most $\mathcal{PIR}$ schemes the user has to know the number of items that each database holds, in order to correctly execute the scheme, all the arrays are assumed to contain exactly $n$ items. Shorter arrays are padded with dummy items ( containing strings of zeroes).

The $\mathcal{PERKY}$ scheme is executed in the following manner. Suppose that $\mathcal{U}$ holds the word $w = w_1 \ldots w_\ell$. In the beginning of the $q$-th execution of $\mathcal{P}$ $(0 \leq q \leq \ell - 1)$ the user knows what address in the $q$-th array is associated with the prefix $w_1 \ldots w_q$ (if $q = 0$ it is the first address, which contains the only "real" item in array 0). $\mathcal{U}$ wishes to retrieve one of the two addresses in the item associated with $w_1 \ldots w_q$. It is the address of the item associated with $w_1 \ldots w_q w_{q+1}$ in array number $q + 1$. $\mathcal{U}$ and $\mathcal{DB}_\infty, \ldots, \mathcal{DB}_{||}$ execute $\mathcal{P}$ in order to privately retrieve the desired address. The communication complexity of the scheme is $O(\ell \cdot C(\log 2n, 2n, k))$, and the round complexity is $\ell \cdot R(\log 2n, 2n, k)$.

**An improved scheme:** In the previous data structure there is a cetain amount of redundancy. For any legal prefix $y$, there is an item that contains the addresses for both possible extensions of $y$ ($y0$ and $y1$). Counting over all the arrays, there are $\ell n$ such items. However, there are only $n - 1$ prefixes $y$ for which *both* extensions $y0$ and $y1$ are legal. We use this fact to constuct a slightly different set of arrays, which leads to a more efficient $\mathcal{PERKY}$ scheme.

This time we use $\ell + 1$ arrays, and enumerate them by $1, \ldots, \ell, \ell + 1$. The $q$-th array, $(1 \leq q \leq \ell)$, contains an item for any prefix $y = y_1 \ldots y_{q-1} y_q$ for which both $y_1 \ldots y_{q-1}0$ and

$y_1 \dots y_{q-1}1$ are legal. The last array, number $\ell + 1$, contains $n$ items, which are simply the strings $s_1, \dots, s_n$. An item in the first $\ell$ arrays, say array number $q$, is composed of a pair of elements $(r, a)$:

1. $r$ is a pointer that points to one of the succeeding arrays, namely $r \geq q + 1$.

2. $a$ is an index in the $r$-th array.

Suppose that the item $(r, a)$ is associated with the prefix $y = y_1 \dots y_q$. One of the following conditions holds:

1. If $r \leq \ell$ there is only one legal extension of $y$ to $r - 1$ bits, but there are two legal extensions of $y$ to $r$ bits. One with 0 at the $r$-th bit, and one with 1 at the $r$-th bit. The address of the first extension in the $r$-th array is $a$, and that of the second extension is $a + 1$.

2. If $r = \ell + 1$ there is just one legal extension of $y$ to $\ell$ bits ($y$ is a prefix of exactly one of the strings $s_1, \dots, s_n$). $a$ is the address of that string in array number $\ell + 1$.

The $\mathcal{PERKY}$ scheme that we derive from this data structure can be divided into three stages.

1. In the first stage $\mathcal{DB}_\infty$ sends some general information about the data structure to $\mathcal{U}$. It sends the number of items in each of the first $\ell$ arrays, and the index of the first array in which there is an item (if for instance all the strings $s_1, \dots, s_n$ begin with a 0 bit, the first array is empty).

2. In the second stage, the participants (user and databases) execute $\mathcal{P}$, $\ell$ times, one for each of the arrays $1, \dots, \ell$. $\mathcal{U}$ retrieves the first item in a non-empty array. Given an item $(r, a)$, $\mathcal{U}$ decides what item to retrieve from the $r$-th array. If $w_r = 0$ ($\mathcal{U}$ holds $w = w_1 \dots w_\ell$), $\mathcal{U}$ retrieves the item at address $a$ in the $r$-th array. Otherwise it retrieves the item at address $a + 1$.

3. After the second stage of the scheme, there remains at most one string among $\{s_1, \dots, s_n\}$, which may be equal to $w$, and the user knows its address in array number $\ell + 1$. The last stage consists of a simple $\mathcal{PIR}(\ell, n, k)$ scheme in which the user privately retrieves out of array number $\ell + 1$ the only string which may be identical to $w$.

As in previous schemes, the user has to take some care here that privacy is maintained. Disregarding privacy, the user may have been able to forgo the retrieval of items from several of the arrays. Retrieving an item from the $q$-th array is necessary only if there is an item in the array, which is associated with a prefix of $w$. However, in order to maintain privacy, the user has to execute $\mathcal{P}$, and retrieve some arbitrary item from the $q$-th array, even if it does not contain a "relevant" item.

**Communication Complexity:** We denote the number of items in the $q$-th array, $1 \leq q \leq \ell$, by $n_q$. In the first stage $\mathcal{DB}_\infty$ sends $\log \ell + \sum_{q=1}^{\ell} \log n_q$ bits to $\mathcal{U}$. In the second stage $\mathcal{P}$ is executed $\ell$ times. In the $q$-th execution an item of size $\log n + \log \ell$ is retrieved out of $n_q$ such items. The communication complexity of this stage is therefore $\sum_{q=1}^{\ell} C(\log n + \log \ell, n_q, k)$. The communication complexity of the third stage is $C(\ell, n, k)$.

An item appears in one of the arrays iff it is associated with a prefix $y = y_1 \dots y_{q-1} y_q$ such that both $y_1 \dots y_{q-1}0$ and $y_1 \dots y_{q-1}1$ are legal prefixes. There are only $n$ such prefixes $y$ since there are only $n$ strings $(s_1, \dots, s_n)$. Consequently $\sum_{q=1}^{\ell} n_q = n$. The worst case communication complexity will result if the $n$ items are spread evenly over the arrays, i.e for all $q, n_q = \frac{n}{\ell}$. The worst case total communication complexity of this $\mathcal{PERKY}(\ell, n, k)$ scheme is:

$$\log \ell + \ell \log(\frac{n}{\ell}) + \ell C(\log n + \log \ell, n/\ell, k) + C(\ell, n, k) \ .$$

The communication complexity of a scheme that solves $\mathcal{PIR}(\log n + \log \ell, n/\ell, k)$ is at least $\log n + \log \ell$. Therefore, we can derive an asymptotic upper bound for the communication complexity:

$$O(\ell C(\log n + \log \ell, n/\ell, k) + C(\ell, n, k)) \ .\square$$

## 5.3 Perfect Hashing

Suppose the databases can find a hash function $h : \{0,1\}^\ell \longrightarrow \{1, \dots, t\}$, where $t \geq n$, such that $h$ is *perfect* for $\{s_1, \dots, s_n\}$. A hash function is perfect for $\{s_1, \dots, s_n\}$ if its restriction to $\{s_1, \dots, s_n\} \subseteq \{0,1\}^\ell$ is one-to-one. In that case the following $\mathcal{PERKY}(\ell, t, k)$ scheme can be implemented:

1. $\mathcal{DB}_\infty$ computes the function $h$ and sends a description of it to $\mathcal{U}$. The user forwards this description to all the other databases.

2. The databses construct an array of $t$ strings. For every $j \in \{1, \dots, n\}$, the string $s_j$ appears in position $h(s_j)$. There might be empty positions in the array, in which strings of zeroes are inserted.

3. $\mathcal{U}$ and $\mathcal{DB}_\infty, \dots, \mathcal{DB}_{||}$ combine in a $\mathcal{PIR}(\ell, t, k)$ scheme, in which the user privately retrieves the item in the array whose address is $h(w)$. Then $w \in \{s_1, \dots, s_n\}$ iff that string is $w$.

**Claim 6:** There exists a $\mathcal{PERKY}(\ell, n, k)$ scheme, which achieves the following:

- Communication complexity $C(\ell, n^2, k) + 2\ell$.

- Round complexity $R(\ell, n^2, k) + 1$.

**Proof:** In order to find a hash function whose restriction is bijective we use techniques introduced in [15] and [4]. We define a family, of functions, $\mathcal{H}$, such that $\forall h \in \mathcal{H}, h : \{0,1\}^\ell \longrightarrow \{1, \dots, t\}$ and $|\mathcal{H}| = 2^{2\ell}$. For any $(a, b) \in \{0,1\}^\ell \times \{0,1\}^\ell$ we define a function $h_{a,b}$ by the rule $h_{a,b}(x) = ax + b$. $a$ and $b$ are viewed as elements of the finite field $GF(2^\ell)$, and addition and multiplication are with respect to this field. $\mathcal{H}$ is a universal family of hash functions. In other words, for any $x, y \in \{0,1\}^\ell, x \neq y$, if $h_{a,b}$ is chosen uniformly at random from $\mathcal{H}$, then $Pr[h_{a,b}(x) = h_{a,b}(y)] = \frac{1}{2^\ell}$.

We are interested in functions that map $\{0,1\}^\ell$ to $\{1,\ldots,t\}$, and therefore, for every $h_{a,b} \in \mathcal{H}$, we will take $h_{a,b}(x)$ to be only the first $\log t$ bits of $ax+b$ (for our purposes it is sufficient to assume that $t$ is a power of 2). The family $\mathcal{H}$ remains universal with $Pr[h_{a,b}(x) = h_{a,b}(y)] = \frac{1}{t}$. We choose uniformly at random a function $h \in \mathcal{H}$, and use it to map $n$ field elements $s_1,\ldots,s_n$. Denote the number of pairs $s_i \neq s_j$ such that $h(s_i) = h(s_j)$ by $F$. The expected value of $F$ over all choices of $h$ is:

$$E[F] = \sum_{s_i \neq s_j} Pr[h(s_i) = h(s_j)] = \binom{n}{2} \cdot \frac{1}{t}$$

If we choose $t = n^2$ then $E[F] \leq \frac{1}{2}$, and since $F$ is an integer, at least half the functions in $\mathcal{H}$ are 1-to-1 over $s_1,\ldots,s_n$ (a function is 1-to-1 iff $F=0$).

The pair $(a,b) \in \{0,1\}^\ell \times \{0,1\}^\ell$ is a unique description of the function $h = ax+b$, and therefore the communication complexity between the user and any single database in the first round of communication, is $2\ell$. We have thus shown a $\mathcal{PERKY}(\ell,n,k)$ scheme with one round of communication in addition to $R(\ell,n,k)$ ($h$ is sent back and forth in the first round), and communication complexity $C(\ell,n^2,k)+2\ell$. □

In terms of the number of communication rounds, this scheme is superior to all our previous suggestions. However, in terms of communication complexity, and in particular in terms of a database's space complexity, this scheme is not efficient, because an array of size $n^2$ is used.

**An improved scheme:** We reduce the number of entries in the array to $O(n)$ by using the perfect hashing technique introduced in [4]. To make this presentation self contained, we give a brief description of the technique.

The main idea is to map the strings $s_1,\ldots,s_n$ to an array of length $O(n)$ in two stages. In the first stage the functions in $\mathcal{H}$ are regarded as mapping strings of length $\ell$ to $\{1,\ldots,n\}$. We choose a function $h \in \mathcal{H}$ that has the following property: $F$, the number of pairs $s_i \neq s_j$ such that $h(s_i) = h(s_j)$, is at most $n$. As we noted previously $E[F] = \binom{n}{2} \cdot \frac{1}{t}$. Since $t=n$ we have $E[F] \leq \frac{n}{2}$, and hence $|F| \leq n$ for at least half the functions in $\mathcal{H}$. Therefore, a function which satisfies the requirements for $h$ can be found (in an efficient manner).

In the second stage we choose $n$ functions: $h_1,\ldots,h_n \in \mathcal{H}$. We denote the number of strings $h$ maps to $i$ by $m_i$, and assume that these strings are $s'_1,\ldots,s'_{m_i}$. In order to choose $h_i$ ($1 \leq i \leq n$) we regard the functions in $\mathcal{H}$ as mapping strings of length $\ell$ to $\{1,\ldots,m_i^2\}$. The function $h_i \in \mathcal{H}$ is required to be a 1-to-1 mapping of $\{s'_1,\ldots,s'_{m_i}\}$ to $\{1,\ldots,m_i^2\}$. At least half the functions in $\mathcal{H}$ satisfy this requirement, and therefore $h_i$ can be found.

Two arrays are now constructed. In the first there are $n$ entries, and in each entry two data elements are stored: $h_i$ and $\sum_{q=1}^{i-1} m_q^2$. In the second array there are $\sum_{q=1}^{n} m_q^2$ entries. The strings $\{s'_1,\ldots,s'_{n_i}\}$ are stored in this array as follows: if $h(s_j) = i$, then $s_j$ is stored in cell number $\sum_{q=1}^{i-1} m_q^2 + h_i(s_j)$ in the array. By construction no two strings are stored in the same cell.

We now show that the size of the second array is at most $3n$. The number of pairs $s_i \neq s_j$ such that $h(s_i) = h(s_j)$ is $|F| = \sum_{q=1}^{n} \binom{m_q}{2}$. On the other hand, by the choice of $h$ we have

$|F| \leq n$. Therefore:

$$\begin{aligned}\sum_{q=1}^{n} m_q^2 &= \sum_{q=1}^{n} m_q(m_q - 1) + \sum_{q=1}^{n} m_q \\ &= 2\,|F| + n \\ &\leq 3n\end{aligned}$$

Our aim is to transform the above construction into a $\mathcal{PERKY}$ scheme. The databases construct the two arrays, and the user retrieves the string at the correct cell. In order to achieve this goal the databases have to agree on $n+1$ hash functions, $h, h_1, \ldots, h_n$ instead of just one in the previous hashing scheme. Having one database choose those functions , and then distribute them through $\mathcal{U}$, is impractical. Since the number of these functions is $n$, and each one is encoded by $2\ell$ bits, the communication complexity is even larger than just having one of the databases send $s_1, \ldots, s_n$ to $\mathcal{U}$. Below we present several solutions to this problem.

1. The simplest solution is to use a scheme that solves $\mathcal{PIR}(\ell, n, 1)$. If there is a single database, [11], the problem of database coordination does not even arise.

2. The second type of solution is to slightly extend the scope of the $\mathcal{PIR}$, or $\mathcal{PERKY}$ model. For instance, the databases can be allowed to have a source of shared randomness. This is already assumed if the model of communication and privacy being used is that of Symmetric $\mathcal{PIR}$ [7].

3. Another solution is to have all the databases go over the functions in $\mathcal{H}$ in deterministic fashion, for example in lexicographic order. The first function that satisfies the conditions required of $h_i$ is chosen as $h_i$. This search will always end successfully, because $\mathcal{H}$ contains functions of the desired type. However, the computational complexity may become exponential in $\ell$ in the worst case, as opposed to an expected time of $O(\ell n)$. This may pose no problem when the only measure of efficiency we use is communication complexity. It does cause difficulties if the databases are assumed to be computationally bounded, as in [2, 11].

Given one of these possible solutions, we have the following:

**Claim 7:** There exists a $\mathcal{PERKY}(\ell, n, k)$ scheme, which achieves the following:

- Communication complexity $O(C(\ell, n, k))$.
- Round complexity $R(2\ell + \log n, n, k) + R(\ell, 3n, k) + \frac{1}{2}$.

**The scheme–**

1. The databases agree on the functions $h, h_1, h_2, \ldots, h_n$. $\mathcal{DB}_\infty$ sends $h$ to the user.

2. The databases construct the first of the two arrays we described previously. The array has $n$ entries. In entry number $i$ ($1 \leq i \leq n$) the databases store two data elements: A description of the function $h_i$ and the number $\sum_{q=1}^{i-1} m_q^2$. The user and databases execute a $\mathcal{PIR}$ scheme by which $\mathcal{U}$ retrieves the contents of entry number $h(w)$ in the array.

3. The databases construct the second of the two arrays. The array has $3n$ entries. Each of the strings $\{s_1, \ldots, s_n\}$ is stored in a (unique) cell in the array. In every cell which remains empty a string of $\ell$ zeroes is stored. If $h(s_j) = i$ then $s_j$ appears in cell number $\sum_{q=1}^{i-1} m_q^2 + h_i(s_j)$ in the array. The participants combine in a $\mathcal{PIR}(\ell, 3n, k)$ scheme, in which the user privately retrieves the string at the address to which $w$ would be mapped (i.e if $h(w) = i$, the address is $\sum_{q=1}^{i-1} m_q^2 + h_i(w)$). The string $w$ is in $\{s_1, \ldots, s_n\}$ if and only if $w$ is equal to the retrieved string.

**Correctness and complexity:** As we remarked previously, if we use the functions of $\mathcal{H}$ to map $n$ items to an array of size $n$, for at least half of the functions in $\mathcal{H}$ the number of collisions (the size of $F$) is less than $n$. Therefore, the expected number of trials the databases have to make until choosing $h$ is 2 (if the choice of $h$ is random and uniform). Similarily, if the functions of $\mathcal{H}$ are used to map $m_q$ items to an array of size $m_q^2$, at least half the functions in $\mathcal{H}$ are bijective. Therefore, for every $q, 1 \leq q \leq n$ the expected time for choosing $h_q$ is also 2.

In stage 1 the communication complexity is $2\ell$, which is twice the trivial lower bound on $\mathcal{PIR}(\ell, n, k)$ ($\ell$ bits have to be communicated even without a privacy requirement). The round complexity of this stage is $\frac{1}{2}$. In stage 2 the communication complexity is $C(2\ell + \log n, n, k)$. Since $\log n \leq \ell$, asymptotically we have $O(C(\ell, n, k))$. The communication complexity in stage 3 is at most $C(\ell, 3n, k)$, and the total communication complexity of the scheme is $O(C(\ell, n, k))$. $\square$

# 6 Other $\mathcal{PERKY}$ Topics

## 6.1 Reducing Communication Complexity

In all of the $\mathcal{PERKY}$ protocols we presented, the communication complexity is a function of the form $\ell^\epsilon \cdot f(n)$, where $\epsilon > 0$ is some constant and $f$ is some sublinear function. In certain contexts the length of the keywords, $\ell$, may be very large.

Following the execution of each of the previous $\mathcal{PERKY}$ protocols, the user always knows whether the word it holds, $w$, is one of the $n$ words held by the databases. If we allow errors, a certain tradeoff is possible between the communication complexity of a protocol, and its probability of error. That is the probability, that at the end of the protocol $\mathcal{U}$ obtains a wrong answer as to whether $w$ is one of $s_1, \ldots, s_n$ or not. This tradeoff is possible through the application of the string equivalence technique, described in [12, pp. 30–31].

**Claim 8:** Given a scheme, $\mathcal{P}$, that solves $\mathcal{PERKY}(\ell, n, k)$ with communication complexity $C(\ell, n, k)$, and a prime number $p$, there exists a scheme, $\mathcal{P}'$, That solves $\mathcal{PERKY}(\ell, n, k)$ with communication complexity $C(\log p, n, k)$, and error probability at most $\frac{n\ell}{p}$.

**Proof:** **The scheme $\mathcal{P}'$–**

1. $\mathcal{U}$ chooses $p$ and $x$. $p$ is a prime number (not , and $x$ is chosen uniformly at random in the range $0, \ldots, p-1$. $\mathcal{U}$ sends $p$ and $x$ to the databases.

2. All the participants in the scheme consider the binary strings they hold as polynomials over $GF(p)$. That is, a string $s = b_0 \dots b_{\ell-1}$, $b_0, \dots, b_{\ell-1} \in \{0,1\}$ represents the polynomial $s(x) = \sum_{j=0}^{\ell-1} b_j x^j$. The databases construct a new set of $n$ strings. The $i$-th string is the binary representation of the field element $s_i(x)$.

3. The user and databases execute the scheme $\mathcal{P}$, in which the $n$ strings each database holds are $s_1(x), \dots, s_n(x)$, and the string that $\mathcal{U}$ holds is $w(x)$.

The size of all the strings is reduced to $\log p$ bits each, and therefore the communication complexity is $C(\log p, n, k)$. The probability that for some $i, 1 \leq i \leq n$, $s_i \neq w$, but $s_i(x) = w(x)$ is at most $\frac{\ell-1}{p}$. Therefore, the probability that $w \notin \{s_1, \dots, s_n\}$ but $w(x) \in \{s_1(x), \dots, s_n(x)$ is less than $\frac{n\ell}{p}$. Obviously, if $w \in \{s_1, \dots, s_n\}$, then $w(x) \in \{s_1(x), \dots, s_n(x)\}$. $\square$

## 6.2 Symmetric $\mathcal{PERKY}$

In [7] the notion of *symmetric* PIR, $\mathcal{SPIR}$, was introduced. The paper dealt with the problem of retrieving a bit while keeping the information of both the user *and the databases* private. The privacy of the database is protected in a very strict sense. The user is allowed to learn a single *physical* bit of the database, and nothing else. An important requirement of the $\mathcal{SPIR}$ schemes, in which at least two databases participate, is a source of shared randomness. This source (say a random string), is shared by all the databases, but is inaccessible to the user. In fact, it is proven in the paper that without some sort of interaction between the databases, no $\mathcal{SPIR}$ scheme is possible, regardless of its communication complexity.

Given a $k$, $k \geq 1$, database $\mathcal{PIR}$ scheme, $\mathcal{P}$, with communication complexity $C(1, n, k)$, the main constructions of [7] are (for an honest user):

- A $k+1$ database $\mathcal{SPIR}$ scheme with communication complexity $O(C(1, n, k+1))$.
- A $k$ database $\mathcal{SPIR}$ scheme with communication complexity $O(C(1, n, k))$, in case $\mathcal{P}$ is one of the $\mathcal{PIR}$ schemes presented in [1, 2, 3, 11].

If the user is dishonest and tries to learn extra information by not adhering to the required protocol, the $\mathcal{SPIR}$ schemes have to be changed slightly. The communication complexity of these more reselient $\mathcal{SPIR}$ schemes increases by a multiplicative factor of $\log n$ compared to the communication complexity of the respective honest-user $\mathcal{SPIR}$ schemes.

We define symmetric $\mathcal{PERKY}$ in an analogous fashion to $\mathcal{SPIR}$.

**Definition 4:** A *symmetric* $\mathcal{PERKY}$ scheme is a $\mathcal{PERKY}$ scheme, in which all the databases can access the same random string not known to the user, such that in a single invocation the user cannot obtain any information, which does not follow from knowing whether $w \in \{s_1, \dots, s_n\}$ or not.

An obvious solution is similar to theorem 1. Replace the $n$ strings $s_1 \dots s_n$ with their incidence vector ($2^\ell$ bits long), and execute a $\mathcal{SPIR}$ scheme to retrieve the desired bit. A more efficient solution is based on the initial trie scheme of subsection 5.2. It seems that converting the $\mathcal{PERKY}$ schemes that are based on other data structures into symmetric $\mathcal{PERKY}$ schemes is more difficult.

**Claim 9:** Let $\mathcal{P}$ be a one round, $k$ database $\mathcal{SPIR}$ scheme (for dishonest users) with communication complexity $C(1, n, k)$. There exists a symmetric $\mathcal{PERKY}(\ell, n, k)$ scheme (for dishonest users), $\mathcal{Q}$, with communication complexity $\ell \cdot C(2 \log n, 2n, k)$.

**Proof:**

As a first step we construct a symmetric $\mathcal{PIR}(m, n, k)$ scheme, $\mathcal{P}'$. We require that in this type of scheme the user obtains a single block of $m$ bits and nothing else. In particular, the user cannot obtain bits from different blocks. The protocol is essentially the same as the general solution to $\mathcal{PIR}(\ell, n, k)$ presented in [3]. The string $x$, which each database holds, is viewed as an $m \times n$ matrix, where each column is one block of $m$ bits. $\mathcal{U}$ sends a single query to each database, as his part of the scheme $\mathcal{P}$, when retrieving a single bit out of $n$ bits. Each database executes its part of $\mathcal{P}$, $m$ times in parallel. In the $j$-th execution, $1 \leq j \leq m$, the $j$-th row is regarded as the database contents, while the query is that same query which $\mathcal{U}$ sent. The databases conclude by sending their $m$ answers to $\mathcal{U}$. A block of $m$ bits can be retrieved by $\mathcal{P}'$, because $\mathcal{P}$ is a one round $\mathcal{PIR}$ scheme. Nothing else can be obtained by the user, because $\mathcal{P}$ is symmetric, and the physical bits, which $\mathcal{U}$ obtains from each row, are all in the same column(block).

The communication complexity of $\mathcal{P}'$ is at most $m \cdot C(n)$. However, in many cases the amount of communication can be reduced by using balancing techniques, see [2, 6].

We construct the symmetric $\mathcal{PERKY}$ scheme, $\mathcal{Q}$, in similar fashion to the first trie based $\mathcal{PERKY}$ scheme (see subsection 5.2). The data structure is again made up of $\ell$ arrays, with indices $0, \ldots, \ell - 1$. There is one data item in array number 0, and $n + 1$ data items in each of the other arrays (compared to $n$ in the previous scheme). The items in the array number $q$ $(1 \leq q \leq \ell - 1)$ are divided into 3 categories:

1. There is an item in the $q$-th array for every legal prefix of length $q$.
2. If the number of distinct legal prefixes of length $q$ is $j$, there are $n - j$ items in the $q$-th array which are "dummy" items, and are simply random strings of the appropriate length.
3. There is one special item which represents all the illegal prefixes.

Recall that in the original trie scheme the dummy items were strings of zeroes. Using the same idea here might allow a dishonest user to gain some partial information. For instance, suppose the user deviates from the protocol and retrieves a dummy item in array number $q$. If the item is a string of zeroes, $\mathcal{U}$ can identify it as a dummy item and learn that there are less than $n$ distinct prefixes of length $q$. However, if that item is a random string, the dishonest user learns nothing.

The item which represents illegal prefixes is needed in order to prevent an honest user from learning partial information. In the original trie scheme if the prefix $y_1 \ldots y_q$ is legal, but the prefix $y_1 \ldots y_q y_{q+1}$ is not, the user may learn which prefix is legal and which is not. Thereby $\mathcal{U}$

gains more information than is allowed in a symmetric $\mathcal{PERKY}$ scheme. Using the extra item ensures that the user cannot distinguish if a prefix of length $q < \ell$ is legal or not legal.

Each item in array number $q$ is divided into two data elements. The two elements represent the two possible extensions of the prefix identified with the item to a prefix of length $q+1$. The first element represents the extension of the current prefix by 0, and the second represents the extension by 1. Each element is a pair $(r, a)$, where $r, a \in \{0,1\}^{\log(n+1)}$.

In the first array (number 0) there is one data item, which is made up of two pairs $(r_0, a_0)$ and $(r_1, a_1)$. $a_0$ is the address in the second array (number 1) of the one bit prefix 0. Similarily, $a_1$ is the address of the one bit prefix 1. $r_0$ and $r_1$ are two random strings. It is possible that one of these prefixes (0 or 1) is not legal. If, for instance, 0 is not legal, then $a_0$ is the address of the special item in array number 1, the item which represents the illegal prefixes.

Let $1 \leq q \leq \ell - 2$ and suppose that $y_1 \ldots y_{q-1} y_q y_{q+1}$ is a legal prefix. Suppose, furthermore, that the prefixes $y_1 \ldots y_{q-1}$, $y_1 \ldots y_q$ and $y_1 \ldots y_{q+1}$ are represented by the items $(r_1, a_1)$, $(r_2, a_2)$ and $(r_3, a_3)$ respectively. The three pairs $(r_1, a_1)$, $(r_2, a_2)$, and $(r_3, a_3)$ appear in arrays number $q-1$, $q$ and $q+1$ respectively. The bitwise xor $r_1 \bigoplus a_2$ is the address in array number $q+1$ in which the item that contains $(r_3, a_3)$ is held.

Dividing an address into two parts, one in array number $q-1$, and one in array number $q$ thwarts attempts by a dishonest user to learn partial information, which is supposed to remain secret. In the original trie scheme a user can deviate from the protocol by retrieving the "wrong" element from the $q$-th array. For instance, $\mathcal{U}$ may be able to learn that there exist $i, j$ such that $w_1 \ldots w_q$ is a prefix of $s_i$, $y_{q+1} \ldots y_\ell$ is a suffix of $s_j$, and $s_i, s_j$ are not necessarily the same string. In the current data structure this problem does not arise. Suppose $\mathcal{U}$ retrieved the two pairs $(r_1, a_1)$ and $(r_2, a_2)$ in arrays number $q-1$ and $q$ respectively. The element that the user is supposed to retrieve in array number $q+1$ is the one at address $r_1 \bigoplus a_2$. If $\mathcal{U}$ retrieves a different element $(r, a)$, that element includes (from the user's point of view) two completely random strings. The user learns nothing by seeing these strings, and in fact can learn nothing further in subsequent phases of the scheme.

Let $1 \leq q \leq \ell - 2$ and let $y_1 \ldots y_q$ be a $q$ bit prefix. The item that represents $y_1 \ldots y_{q-1}$ contains an element $(r_1, a_1)$, and the item that represents $y_1 \ldots y_q$ contains an element $(r_2, a_2)$ such that $r_1 \bigoplus a_2$ is the address of the item that represents $y_1 \ldots y_q y_{q+1}$. Assume that this last prefix $y_1 \ldots y_q y_{q+1}$ is not legal. In that case $r_1 \bigoplus a_2$ is the address of the special item in array number $q+1$ (that is the item that represents all the illegal prefixes).

As in the scheme of subsection 5.2, in the last array (number $\ell - 1$) each item is a pair of bits, which show whether the corresponding $\ell$ bit strings appear in the database or not.

In order to round off the description of the database, we remark that the order of the items in each array is chosen at random. In other words, the address, in which the item that represents $y_1 \ldots y_q$ is held, is independent of the value of $y_1 \ldots y_q$. In addition, the databases regard queries in round $q$ as refering to array number $q$.

The databases make joint decisions on random strings, random positions of items etc. through the mechanism of shared randomness.

The $\mathcal{PERKY}$ scheme $\mathcal{Q}$ is as follows: The user and databases execute the symmetric $\mathcal{PIR}(2\log n, 2(n+1), k)$ scheme, $\mathcal{P}'$, $\ell - 1$ times. In the $q$-th execution, $0 \leq q \leq \ell - 2$, one element (i.e a pair

$(r, a)$) is retrieved from the $q$-th array. Given the elements retrieved from arrays $q-1$ and $q$, the user can obtain the address of the element it wishes to retrieve in array $q+1$. Finally, the participants in $\mathcal{Q}$ execute the $\mathcal{SPIR}$ scheme $\mathcal{P}$ to retrieve a single bit out of $2n+2$ bits in array number $\ell - 1$.

If $\mathcal{U}$ and the databases execute the protocol correctly, the user knows at the end of $\mathcal{Q}$ whether $w \in \{s_1, \ldots, s_n\}$ in similar fashion to the trie based $\mathcal{PERKY}$ scheme. The communication complexity of $\mathcal{Q}$ is less than $\ell$ times the communication complexity of $\mathcal{P}'$, and is therefore less than $2\ell \cdot \log n \cdot C(1, 2n+2, k)$.

We now prove that database privacy is maintained. An honest user, who follows the protocol, learns nothing it is not allowed to know, because database privacy is maintained in $\mathcal{P}$ and $\mathcal{P}'$. Even a dishonest user can only retrieve one physical element from each array, because $\mathcal{P}'$ is being used. If it doesn't retrieve the specified element, it obtains a completely random string of length $2\log n$, and can subsequently learn nothing of value in that invocation of the scheme. $\square$

# 7 Concluding Remarks

In this work a solution to the $\mathcal{PERKY}$ problem is defined as a protocol by which the user learns whether $w \in \{s_1, \ldots, s_n\}$ or not. In many situations a user would actually like to know the address $i$ for which $w = s_i$. After finding out the address, further queries about the keyword $w$ or the data it represents can be accomplished by using $\mathcal{PIR}$ schemes, which are more efficient than their $\mathcal{PERKY}$ counterparts. If $w \in \{s_1, \ldots, s_n\}$ finding out $i$ is simple, given any $\mathcal{PERKY}$ scheme, $\mathcal{P}$.

The databases construct $\log n$ sets of strings, which contain $n/2$ items each. Suppose the binary representation of $i$ is $i_1 \ldots i_{\log n}$, and the $i$-th keyword is $s_i$. The keyword $s_i$ appears in the $q$-th set ($1 \leq q \leq \log n$) if and only if $i_q = 1$. The user and databases execute the $\mathcal{PERKY}$ scheme $\mathcal{P}$ $\log n$ times in parallel. In the $q$-th execution the user finds out if $w$ appears in the $q$-th set constructed by the databases.

The communication complexity of the above protocol is $\log n$ times the communication complexity of $\mathcal{P}$. However, in all the schemes shown in this work the user can find what the address $i$ is in a more efficient manner. In the binary tree, trie and perfect hashing schemes the user and databases execute several $\mathcal{PIR}$ schemes. In the last execution the databases construct an array such that by retrieving the data in the $j$-th cell (for some $1 \leq j \leq n$ ) the user finds out if $w \in \{s_1, \ldots, s_n\}$. Adding to the $j$-th cell the address $i$, which is the position of $w$ in the original database, allows the user to discover what $i$ is without further ado. In each of the cases (tree, trie and hashing) the communication complexity increases by a constant factor at most.

# References


[1] A. Ambainis, *"An upper bound for Private Information Retrieval"*, In Proc. of $24^{th}$ ICALP, to appear,1997.

[2] B. Chor, N. Gilboa, *“Computationally Private Information Retrieval“* Proc. of $29^{th}$ STOC (1997), pp. 304–313.

[3] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan, *“Private Information Retrieval”* , Proc. of $36^{th}$ FOCS, 1995, pp. 41–50.

[4] M. Fredman, J. Komlos, E. Szemeredi, *“Storing a sparse table in O(1) worst case access time”* , Journal of the ACM, Vol. 31, 1984, pp. 538–544.

[5] O. Goldreich *”Towards a theory of software protection and simulation by oblivious RAMs”*, Proc. of $19^{th}$ STOC, 1987.

[6] N. Gilboa, *”Balancing and Private retrieval of blocks”*, in preparation.

[7] Y. Gertner, Y. Ishai, E. Kushilevitz, T. Malkin, *” Protecting Data Privacy in private Information Retrieval Schemes”*, Manuscript, 1997.

[8] O. Goldreich, R. Ostrovsky, *”Software Protection and Simulation on Oblivious RAMs”*, JACM, Vol. 43, No. 3, 1996, pp. 431-473.

[9] S. Goldwasser, S. Micali, *”Probabilistic Encryption”*, JCSS, Vol. 28, No. 2, 1984, pp. 270-299. Previous version in STOC 1982.

[10] D. Knuth, The art of computer programming, Vol. 3, Addison Wesley, 1973.

[11] E. Kushilevitz, R. Ostrovsky, *”Single-database computationally private information retrieval”* Proc. of $38^{th}$ FOCS, 1997.

[12] E. Kushilevitz, N. Nisan, *”Communication Complexity”* Cambridge University Press, 1997.

[13] R. Ostrovsky *”Software protection and simulation on oblivious RAMs”*, M.I.T. Ph.D. Thesis in Computer Science, June 1992. Preliminary version in Proc. $22^{nd}$ STOC, 1990.

[14] R. Ostrovsky, V. Shoup, *“Private Information Storage”*, these proceedings. Proc. of $29^{th}$ STOC, 1997, pp. 294–303.

[15] M. Wegman, J. Carter, *“New hash functions and their use in authentication and set equality”*, Journal of Computer and System Sciences, Vol. 22, No. 3, 1981, pp. 265–279.