One-way Trapdoor Permutations Are Sufficient for Single-Server Private Information Retrieval

(extended abstract)

Eyal Kushilevitz * Rafail Ostrovsky†

April 23, 1999

Abstract

We show that general one-way trapdoor permutations are sufficient to privately retrieve an entry from a database of size $n$ with total communication complexity strictly less than $n$. More specifically, if $K$ is the security parameter of the trapdoor permutations, we show a protocol where the user sends $O(K^2)$ bits and the server sends $\frac{n \log{n}}{K}$ bits (for any constant $c$). Thus, for sufficiently large databases (e.g., when $K = n^\epsilon$ for some small $\epsilon$) our construction breaks the information-theoretic lower-bound (of $n$ bits); this demonstrates the feasibility of basing private information retrieval on general assumptions, resolving a major open problem in this area.

1 Introduction

Private information retrieval (PIR, for short) is a communication protocol between a user and a server. In this protocol the user wishes to retrieve an item from a database stored in the server without revealing to the server which item is being retrieved. For concreteness, the database is viewed as an $n$-bit string $x$ and the entry to be retrieved is the $i$-th bit of $x$. This problem was introduced by Chor et al. [6] and various aspects of it were further studied in [1, 2, 4, 5, 7, 8, 9, 10, 17, 20, 21, 24, 28]. A naive solution for hiding which particular item is being retrieved (i.e., the index $i$) is to retrieve the entire database $x$. The communication complexity of this solution is $n$ bits. Solutions that are more efficient than the naive one were found by [6] and later in [1, 17]. These works consider a setting where there are several identical copies of the database stored in several servers. The user can make queries to different servers and use the answers to reconstruct the bit $x_i$; assuming that the servers do not communicate with each other, then privacy can be achieved with a cost which is much less than $n$ (e.g., $O(n^{1/3})$ when two such servers are available). Moreover, Chor et al. [6] have shown that if there is only a single server, then getting “perfect” (i.e., information-theoretic) privacy with communication of less than $n$ bits is impossible.

Kushilevitz and Ostrovsky [20] have shown a way to get around this impossibility results. Namely they show that, assuming the hardness of some number-theoretic problem (specifically, the quadratic residuosity problem), it is possible to design a private information retrieval protocol with a single server and communication complexity of $O(n^\epsilon)$ (for any constant $\epsilon > 0$). Their result heavily relies on the

*IBM T.J. Watson Research Center, and Department of Computer Science, Technion. E-mail: eyalk@watson.ibm.com and eyalk@cs.technion.ac.il. Supported in part by the Mitchell-Schoerf program at the Technion.
†Bellcore. E-mail: rafail@bellcore.com

In [5] it is shown that, in the scenario where there are several servers storing identical database $x$, then intractability assumptions might be of help in constructing PIR protocols.
algebraic properties of the quadratic residuosity problem. Micali et al [4] have shown that under the so-called \( \phi \)-hiding (number-theoretic) assumption one can achieve even more efficient poly-logarithmic (in \( n \)) communication. Other single-server PIR protocols which are based on specific (number-theoretic and/or algebraic) intractability assumptions were presented in [21, 28]. All these PIR protocols exploit specific algebraic structures related to the specific intractability assumption in use. In this paper, we address the question whether PIR protocols can be based on some “general” (preferably, the weakest possible) assumption.

Starting with the work of Yao [29], the program of identifying the weakest possible assumptions to reach various cryptographic tasks was lunched. This program enjoyed a great success and for most cryptographic primitives we have very good grasp of both necessary and sufficient conditions; see, e.g., [14, 27]. What about private information retrieval? On the lower-bound front, in addition to the information-theoretic lower-bound of [6], recent work has established that single-server private information retrieval with less than \( n \) communication (even \( n - 1 \) bits!) already requires one-way functions [2]; in fact single-server private information retrieval with less than \( n \) communication even implies oblivious transfer (OT) [8]. The most general assumption based on which it is (currently) known how to construct OT is that one-way trapdoor permutations exist [13].

Thus, in a sense, the most general assumption one can hope to use for constructing single-server private information retrieval protocols is the assumption that one-way trapdoor permutations exist (or trapdoor functions with polynomial pre-image size; see [3]).

In this paper, we show that this is indeed feasible. That is, we show that under the sole assumption that one-way trapdoor permutations exist (without relying on special properties of any specific assumption) then single-server private information retrieval with less than \( n \) communication (or more precisely, of communication \( n - \frac{cn \log n}{K} + O(K^2) \), where \( K \ll n \) is the security parameter and \( c \) is some constant). We note however that, while the communication complexity is below the lower bounds provided by [6, 2, 8], it is nowhere close to what can be achieved based on specific assumptions. This quantitative question remains for future study.

**Organization:** Section 2 includes some definitions that are used in this paper. In particular, it describes several tools from the literature that are used by our constructions; these include some facts about hard-core predicates from [12], some properties of universal one-way hash functions from [23], and the notion of interactive hashing [25]. In Section 3 we describe our basic PIR protocols based on one-way trapdoor permutations. These protocol is further extended in Section 4 to deal with faulty behavior by the server.

## 2 Preliminaries

### 2.1 Notation

We use the following notations throughout the paper. The data string is denoted by \( x \), its length is denoted by \( n \). The index of the bit that the user wishes to retrieve from this string is denoted by \( i \). We use \( K \) to denote a security parameter.

For a finite set \( A \), we denote by \( a \in_R A \) the experiment of choosing an element of \( A \) according to the uniform distribution (and independently of all other random choices made).

---

\(^2\)Impagliazzo and Rudich [16] have shown that OT is unlikely to be implemented based one one-way functions only (i.e. without trapdoor) since the proof of security (using black-box reductions) would yield a proof that \( P \) is not equal to \( NP \). Also, Impagliazzo and Luby [15] have shown that oblivious transfer protocols already imply the existence of one-way functions. (In fact, OT was shown to be complete for any two-party computation [18, 19].)
2.2 Definitions

In this section we define the notions of one-way trapdoor permutations and of hard-core predicates. The reader is referred to [11] for an extended background related to these definitions.

**Definition 1.** A collection of functions \( \mathcal{G} = \{G_K\} \) is called a collection of one-way trapdoor permutations if the following hold:

- There exists a probabilistic polynomial-time generating algorithm, \( I \), that on input \( 1^K \) outputs a pair \((g, g^{-1})\) where \( g \) is an index of a function in \( \mathcal{G}_K \) and \( g^{-1} \) is a string called the “trapdoor for \( g \).”
- Each function \( g \in \mathcal{G}_K \) is a permutation over \( \{0,1\}^K \) and is computable in polynomial time (that is, there exists an algorithm that given \( g \in \mathcal{G} \), and \( x \in \{0,1\}^n \), computes the value of \( g(x) \) in time polynomial in \(|x|\)).
- Each \( g \) is easy to invert given its trapdoor \( g^{-1} \). That is, there exists an algorithm that given \( y \in \{0,1\}^K \) and the string \( g^{-1} \) computes the (unique) value \( x \) such that \( g(x) = y \) (i.e., \( x = g^{-1}(y) \)) in time polynomial in \( K \).
- It is hard to invert the functions in \( \mathcal{G} \) without having the trapdoor. Formally, for every probabilistic polynomial-time algorithm \( B \), every integer \( c \), and sufficiently large \( K \)

\[
\Pr_{g \in I_G(1^K), \sigma \in \{0,1\}^K} \left( B(g, \sigma) = g^{-1}(\sigma) \right) < \frac{1}{K^c},
\]

where “\( g \in I_G(1^K) \)” denotes choosing a function \( g \) according to the probability distribution induced by the generating algorithm \( I \).

Next, we will need the notion of hard-core predicates. Specifically, we will use the Goldreich-Levin hard-core predicates [12]. For a string \( r \in \{0,1\}^K \) let us denote \( r(x) = \langle r, x \rangle \), where \( \langle \cdot, \cdot \rangle \) is the standard inner-product modulo 2. The Goldreich-Levin Theorem [12] states that if \( g \) is a one-way permutation then there is no algorithm that can compute \( r(x) \) given \( g(x) \) and \( r \). Formally, for every probabilistic polynomial-time algorithm \( B \), every integer \( c \), and sufficiently large \( K \)

\[
\Pr_{g \in I_G(1^K), x \in \{0,1\}^K, r \in \{0,1\}^K} \left( B(g(x), r) = r(x) \right) < \frac{1}{2} + \frac{1}{K^c}.
\]

**Remark:** the above definitions concentrate on the case of one-way permutations; however, they can be easily generalized to deal with more general notions. In particular, the Goldreich-Levin Theorem [12] applies to any one-way function.

2.3 Some Useful Machinery

Let \( \mathcal{G} \) be some arbitrary family of one-way trapdoor permutations over \( \{0,1\}^K \). It is sometimes convenient to view strings in \( \{0,1\}^K \) as elements of the field \( \mathbb{GF}[2^K] \). With this view in mind, let

\[
\mathcal{H} = \{ h_{a,b} : \mathbb{GF}[2^K] \to \mathbb{GF}[2^K] \mid h(x) = ax + b, \ a, b \in \mathbb{GF}[2^K], \ a \neq 0 \}.
\]

Given \( \mathcal{G} \) and \( \mathcal{H} \), Naor and Yung [23] define the following family of functions

\[
\mathcal{F} = \{ f : \{0,1\}^K \to \{0,1\}^{K-1} \mid g \in \mathcal{G}, h \in \mathcal{H}, f(x) = \text{chop}(h(g(x))) \}.
\]
where the chop operator takes a string and chops its last bit. 
For a function \( f \in \mathcal{F} \) we sometimes denote \( f = (g, h) \) to indicate the functions \( g \in \mathcal{G}, h \in \mathcal{H} \) based on which \( f \) is defined. Moreover, if \( I \) is the generating algorithm for \( \mathcal{G} \) then we denote by \( I_{x} \) a generating algorithm for \( \mathcal{F} \) that generates \((g, g^{-1})\) by applying \( I \), generates \( h \in \mathcal{H} \) according to the uniform distribution and let \( f = (g, h) \).

The following are basic properties of \( \mathcal{F} \).

1. Each function \( f \in \mathcal{F} \) is \( 2 \to 1 \). In other words, for every \( x \in \{0, 1\}^{K} \) there is a (unique) string, denoted \( x^{*} \), such that \( f(x^{*}) = f(x) \) and \( x^{*} \neq x \).

2. Every function \( f = (g, h) \) in \( \mathcal{F} \) is efficiently computable. Moreover, given the trapdoor \( g^{-1} \) it is easy to compute, for every \( y \in \{0, 1\}^{K-1} \) the two strings \( x, x^{*} \) such that \( f(x) = f(x^{*}) = y \).

3. Collisions are hard to find for \( \mathcal{F} \) [23] (i.e., given \( x \) and \( f(x) \) it is hard to find the string \( x^{*} \)). Formally, for every \( x \), for every probabilistic polynomial-time algorithm \( B \), every integer \( c \), and sufficiently large \( K \)

\[
\Pr_{x \in \{0, 1\}^{K}} \left( B(x) = x^{*} \right) < \frac{1}{K^c}.
\]

Note that property 3 does not guarantee that specific bits of \( x^{*} \) are hard to find. Instead we will make use of hard-core bits.

We shall also use an interactive hashing protocol, originally invented in [25] and later used in [22, 26]. This is a protocol between two players Alice and Bob, where both Alice and Bob are probabilistic polynomial-time machines. Alice is given as an input \( 1^{K} \), a function \( g \in \mathcal{G}_{K} \) and an input \( x \in \{0, 1\}^{K} \); Bob is given \( 1^{K} \). The interactive hashing protocol proceeds as follows:

- Bob chooses uniformly at random \( K - 1 \) vectors \( H_{1}, \ldots, H_{K-1} \) in \( \{0, 1\}^{K} \) subject to the constraint that these \( K - 1 \) vectors are linearly independent (viewing them as elements of the linear space \( \mathbb{Z}_{2}^{K} \)).
- The players interact in \( K - 1 \) rounds where in round \( i \) they do the following:
  - Bob sends to Alice \( H_{i} \)
  - Alice sends to Bob \( \langle H_{i}, g(x) \rangle \) (the inner product of \( H_{i} \) and \( g(x) \)).

The communication in this protocol, consisting of the strings \( H_{1}, \ldots, H_{K-1} \) sent by Bob and the bits \( \langle H_{1}, g(x) \rangle, \ldots, \langle H_{K-1}, g(x) \rangle \), define \( K - 1 \) linear equations and since all the \( H_{i} \)'s are linearly independent these equations admit two solutions, denoted \( \{y, y^{*}\} \) (we use the same notation as was used above for the pre-images of \( f \in \mathcal{F} \) to stress the analogy between these two tools; this analogy will also be used in our protocols). We now state several facts regarding interactive hashing that make it useful for our purposes:

- If Alice follows the protocol then one of \( \{y, y^{*}\} \) is \( g(x) \) (recall that \( x \) is an input to Alice).
- Bob sends total of \( O(K^{2}) \) bits to Alice. Alice sends total of \( K - 1 \) bits in response.
- It is hard for Alice to find inverses of both \( y, y^{*} \), even if Alice does not follow the protocol. Formally, for every probabilistic polynomial-time algorithm \( A' \), for every integer \( c \) and sufficiently large \( K \), if \( g \) is chosen according to \( I_{g}(1^{K}) \) then after \( A' \) executes the protocol with Bob, the probability that \( A' \) outputs \( x_{0}, x_{1} \) such that both \( g(x_{0}) = y \) and \( g(x_{1}) = y^{*} \) is less than \( \frac{1}{K^{c}} \).

\[\text{Note that every } h \in \mathcal{H} \text{ is } 1 \to 1 \text{ and easy to invert; therefore, given } y \text{ one can try the two options for the chopped bit, invert } h \text{ and then invert } g \text{ using the trapdoor. We also note that this property was not considered in [23] since they deal with arbitrary one-way permutations and not only with trapdoor permutations.}\]
This protocol, as described up to this point, works with any one-way permutations; in this paper we will apply it with one-way trapdoor permutations; this modifications gives one additional property which is simple yet crucial for our application:

- Given the trapdoor for \( g \) (i.e., the string \( g^{-1} \)) and the communication (i.e., the strings \( H_1, \ldots, H_{K-1} \) and the bits \( \langle H_1, g(x) \rangle, \ldots, \langle H_{K-1}, g(x) \rangle \)) Bob can compute both \( x_0 \) and \( x_1 \) (i.e., the strings such that \( g(x_0) = y \) and \( g(x_1) = y' \)).

### 2.4 PIR Protocols

A Private Information Retrieval (PIR) is a protocol for two players: a server \( S \) who knows an \( n \)-bit string \( x \) (called the database), and a user \( U \) holding an index \( i \in [n] \) and interested in retrieving the value \( x_i \). When considering the privacy requirement of PIR protocols there are several possible types of “faulty” behaviors by the server: the server might be honest-but-curious or it might be malicious. Below we detail the definition for each of these types; we note however that the difference is especially important when dealing with multi-round protocols (as those described in this work).

An honest-but-curious server is a one that behaves according to the pre-defined protocol and just tries to deduce information about \( i \) from the communication it sees. This is formulated as follows: Fix a data string \( x \); for every \( i, i' \in [n] \) (where \( i \neq i' \)) the distribution of communications generated by the protocol when the user is interested in bit \( i \) is indistinguishable from the distribution generated when the user is interested in bit \( i' \).\(^4\) We stress here that \( x \) is fixed and the server is not allowed to change it during the protocol’s execution.

A malicious server is a one that does not necessarily follow the protocol. It should be immediately noticed that there are several “bad” behaviors by a malicious server which cannot be avoided; e.g., the server may refuse to participate in the protocol or it may change the content of the database (say, it can act as if \( x = 0^n \)). The privacy requirement in this case makes sure however that, no matter what the server does, the identity of the index \( i \) is not revealed. Formally, for every \( i, i' \in [n] \) (where \( i \neq i' \)) no probabilistic polynomial-time server \( S' \) can distinguish executions of the protocol when the user’s index is \( i \) from executions of the protocol when the user’s index is \( i' \). We stress that here, the server is allowed to modify its messages in an arbitrary manner during the protocol execution in order to be able to distinguish.

### 3 A PIR Protocol with respect to a Honest-but-Curious Server

In this section we present the honest-but-curious PIR protocol which proves that it is possible to construct a PIR protocol from any family of one-way trapdoor permutations, with communication complexity smaller than \( n \). (Later we describe some simple improvements on this protocol.)

**Theorem 1.** If one-way trapdoor permutations exist then there exists honest-but-curious single-server PIR protocol whose communication complexity is at most

\[
n - \frac{n}{2K} + O(K).
\]

(More precisely, the user sends \( O(K) \) bits and the server sends at most \( n - \frac{n}{2K} \) bits.)

(Some slightly better bounds are mentioned in Section 3.2 below).

\(^4\)For lack of space we omit the formal definition of indistinguishability which is a standard one.
Let $\mathcal{G}$ be a collection of one-way trapdoor permutations, as guaranteed by the theorem, and let $\mathcal{F}$ be a family of $2 \rightarrow 1$ functions constructed based on $\mathcal{G}$, as described in Section 2.3. Assume, without loss of generality, that $n$ is divisible by $2K$ and let $\ell = n/(2K)$. The protocol works as follows.

1. The user picks two functions $f_L = (g_L, h_L)$ and $f_R = (g_R, h_R)$ (including the corresponding trapdoors $g_L^{-1}$ and $g_R^{-1}$) using the generating algorithm $I_\mathcal{F}(1^K)$. It sends the functions $f_L, f_R$ to the server (without the trapdoors).

2. The server and the user view $x$ as if it is composed of $2\ell$ sub-strings $z_{1,L}, z_{1,R}, z_{2,L}, z_{2,R}, \ldots, z_{\ell,L}, z_{\ell,R}$ each of size $K$ (we refer to these strings as “blocks”). The server now applies $f_L$ to each block $z_{i,L}$ and applies $f_R$ to each block $z_{i,R}$. It sends all the outcomes

$$f_L(z_{1,L}) \quad f_R(z_{1,R})$$
$$f_L(z_{2,L}) \quad f_R(z_{2,R})$$
$$\vdots \quad \vdots$$
$$f_L(z_{\ell,L}) \quad f_R(z_{\ell,R})$$

3. The user, having the trapdoors for both $f_L$ and $f_R$, can compute for each block $z$ the two possible pre-images $\{z, z^*\}$. Assume that the bit $x_i$ is in some block $z_{i,L}$, for some $s$. The user picks random $r_L, r_R \in \{0, 1\}^K$ such that the hard-core predicates corresponding to $r_L, r_R$ satisfy

$$r_L(z_{i,L}) \neq r_L(z_{i,L}^*) \quad \text{and} \quad r_R(z_{i,R}) = r_R(z_{i,R}^*).$$

It sends $r_L, r_R$ to the server. (If the index $x_i$ is in block $z_{i,R}$ then $r_L, r_R$ are chosen subject to the constraint $r_R(z_{i,R}) \neq r_R(z_{i,R}^*)$ and $r_L(z_{i,L}) = r_L(z_{i,L}^*)$.)

4. For each $j = 1, \ldots, \ell$ the server computes and sends the bit $b_j = r_L(z_{j,L}) \oplus r_R(z_{j,R})$.

5. By the choice of $r_L, r_R$ the bit $b_j$ allows the user to compute the value of $z_{j,L}$ (or the value of $z_{j,R}$ depending on the way that $r_L, r_R$ were chosen).\(^5\) This gives the user the bit $x_i$ (as well as all other bits in the corresponding block).

**Correctness:** The correctness follows from the description of the protocol and the basic properties of $\mathcal{F}$. The idea here is that for the pair of blocks in which the user is interested, $z_{s,L}, z_{s,R}$, the hard-core predicates are chosen in a way that they are sensitive on the block which the user wishes to retrieve, and are constant on the other block. This allows the user to distinguish the target $z$ from $z^*$.

**Communication complexity:** The only messages sent by the user are those for specifying $f_L, f_R, r_L, r_R$; all together $O(K)$ bits. The server, on the other hand, sends for each pair of blocks $2(K-1)$ bits in Step 2 and an additional bit in Step 4. All together, $\ell \cdot (2K - 1) = n - \frac{n}{2K}$ bits. Therefore, the communication complexity is as claimed by the theorem.

\(^5\)The user ignores all the other bits $b_j$, for $j \neq s$. 
3.1 Proof of Security

The only information that the user sends which depends on the index it is interested in is the choice of \( r_L, r_R \) (Step 3). We need to show that these strings maintain the privacy of the user's index. For this we introduce some notation. We say that a block \( z_{i,L} \) (resp. \( z_{i,R} \)) is of type “E” (equal) if \( r_L(z_{i,L}) = r_L(z_{i,L}^*) \) (resp., if \( r_R(z_{i,R}) = r_R(z_{i,R}^*) \)); similarly, we say that a block \( z_{i,L} \) (resp. \( z_{i,R} \)) is of type “N” (not equal) if \( r_L(z_{i,L}) \neq r_L(z_{i,L}^*) \) (resp., if \( r_R(z_{i,R}) \neq r_R(z_{i,R}^*) \)). Hence, the choice of \( r_L, r_R \) defines a sequence of \( i \) pairs in \( \{E, N\} \) with the only restriction being that the pair in which the index \( i \) resides must be either \((N, E)\) or \((E, N)\) (depending on whether \( i \) is in the left block or the right block). We also use \( * \) to denote a “don’t-care”. So if, for example, the user wishes to retrieve the first block it picks \( r_L, r_R \) subject to the constraint that the corresponding sequence is \((N, E), (*, *), \ldots, (*, *)\).

Using the above notation, we will now prove that the server cannot distinguish any pair of indices \( i, i' \) the user may wish to retrieve. Obviously, if \( i, i' \) are in the same block then the user behaves in an identical way in both cases and there is no way for the server to distinguish the two cases. The next case is where \( i, i' \) are in the same pair of blocks; say, \( i \) is in \( z_{i,L} \) and \( i' \) in \( z_{i,R} \). For simplicity of notations assume \( s = 1 \) then in the first case \( r_L, r_R \) are chosen uniformly from those that induce the sequence

\[
(N, E), (*, *), \ldots, (*, *)
\]

while in the second case \( r_L, r_R \) are chosen from those that induce the sequences

\[
(E, N), (*, *), \ldots, (*, *). 
\]

We omit the details for this case since it is a degenerate case of the more general scenario where, say, \( i \) is in \( z_{i,L} \) and \( i' \) in \( z_{i,R} \). Again, for simplicity of notations assume \( s = 1, s' = 2 \); then, we have to distinguish the following two types of sequences:

\[
(N, E), (*, *), (*, *), \ldots, (*, *)
\]

and

\[
(*, *), (E, N), (*, *), \ldots, (*, *). 
\]

(Note that if, for example, the user can tell that for some \( s \) the corresponding pair is of type, say, \((E, E)\) then it can conclude that none of the blocks \( z_{i,L}, z_{i,R} \) is of interest for the user.) We now show that if the server is able to distinguish these two types of sequences it can also predict the hard-core predicate associated with the family \( \mathcal{F} \).

The first step uses a hybrid argument to claim that if one can distinguish the two distribution of \( r_L, r_R \) as above (given \( z, f_1 \) and \( f_2 \)) then it can also distinguish two adjacent distributions among the following list of distributions:

\[
\begin{align*}
\Pi_1 & : & (N, E), (*, *), (*, *), \ldots, (*, *) \\
\Pi_2 & : & (*, E), (*, *), (*, *), \ldots, (*, *) \\
\Pi_3 & : & (*, *), (*, *), (*, *), \ldots, (*, *) \\
\Pi_4 & : & (*, *), (E, *), (*, *), \ldots, (*, * ) \\
\Pi_5 & : & (*, *), (E, N), (*, *), \ldots, (*, *)
\end{align*}
\]

(If each pair of adjacent distributions is indistinguishable then so are \( \Pi_1 \) and \( \Pi_5 \), contradicting the assumption that the server can distinguish.) Suppose, for example, that one can distinguish \( \Pi_1 \) and \( \Pi_2 \)
(other cases are similar or even simpler; they might require flipping the roles of \( f_L \) and \( f_R \). Then, it is also possible to distinguish \( \Pi_1 \) and

\[
\Pi'_1 : (E, E), (\ast, \ast), (\ast, \ast), \ldots, (\ast, \ast).
\]

To make the distinguishing property more concrete assume, without loss of generality, that for some data string \( x \),

\[
Pr_{f_L, f_R \in I_F(1^K), (r_L, r_R) \in \Pi_1}(D(x, f_L, f_R, r_L, r_R) = 1) \leq \frac{1}{2} - \epsilon
\]

and

\[
Pr_{f_L, f_R \in I_F(1^K), (r_L, r_R) \in \Pi'_1}(D(x, f_L, f_R, r_L, r_R) = 1) \geq \frac{1}{2} + \epsilon.
\]

We use this algorithm \( D \) to construct an algorithm \( B \) that on input \( g \in I_G(1^K), y \in R \{0, 1\}^K \) and \( r \in R \{0, 1\}^K \) predicts the hard-core predicate \( r(g^{-1}(y)) \), with probability \( 0.5 + \epsilon \). This contradicts the Goldreich-Levin Theorem [12] (See Section 2.2). Algorithm \( B \) works as follows:

1. Choose \( h_L \) at random subject to the constraint \( h_L(y) = h_L(g(z_{1,L})) \). Let \( f_L = (g, h_L) \) and \( r_L = r \). (Note that, with respect to \( f_L \) we have \( z_{1,L}^* = g^{-1}(y) \). Also crucial is the fact that \( \mathcal{H} \) is a pairwise independent family; this guarantees that such \( h_L \) exists and that the induced distribution is otherwise random.)

2. Choose a function \( f_R \in I_F(1^K) \) (including the corresponding trapdoor!) and compute the string \( z_{1,R}^* \) (by using the trapdoor). Pick a random \( r_R \) subject to the constraint that \( r_R(z_{1,R}^*) = r_R(z_{1,L}^*) \).

3. Invoke \( D \) on input \( (x, f_L, f_R, r_L, r_R) \). If the output is \( \ast \) (in which case the input is more likely to be from \( \Pi'_1 \); i.e., \( r_L(z_{1,L}) \) and \( r_L(z_{1,L}^*) \) are more likely to be not-equal) then \( B \)'s output is \( 1 - r_L(z_{1,L}) \). If the output is \( 0 \) (in which case the input is more likely to be from \( \Pi_1 \); i.e., \( r_L(z_{1,L}) \) and \( r_L(z_{1,L}^*) \) are more likely to be equal) then \( B \)'s output is \( r_L(z_{1,L}) \). (Note that while \( B \) does not know what \( z_{1,L}^* \) is, it knows \( z_{1,L} \) and hence can apply \( r_L \) to it.

It can be verified that the distribution of inputs provided to \( D \) is exactly what is needed and hence the correctness of \( B \) follows.

### 3.2 Some Improvements

We tried to make the description of the protocol above as simple as possible. There are however certain modifications that one can apply to it in order to slightly improve the efficiency. One such improvement is instead of using two functions \( f_L, f_R \) one can use \( d \) such functions \( f_1, \ldots, f_d \) (where \( d \) may depend on \( K \) and/or \( n \)). Then, the user can choose hard-core predicates \( r_1, \ldots, r_d \) such that the one corresponding to the index \( i \) gets two different values (on the corresponding \( z, z^* \)) while each of the other hard-core predicates get the same value (on \( z, z^* \)). Then, when the server returns the exclusive-or of the \( d \) bits this allows the user to reconstruct the block of interest.

A second (more significant) modification that one can make is, instead of defining \( F \) with one bit chopped from \( h(g(x)) \), to define it so that \( c \log n \) bits are chopped (for some constant \( c \); obviously \( K > c \log n \)). This way the number of pre-images for every \( f \in F \) is polynomial in \( n \). Now, in Step 2 of the protocol the server needs to send only \( K - c \log n \) bits per block, and in Step 4 (if combined with the previous modification) for each \( d \) blocks it needs to send back \( c \log n \) bits. This gives a complexity of \( n - \frac{(d-1)n \log n}{dK} \) bits from the server to the user and \( O(dK) \) bits from the user to the server.
4 A PIR Protocol with respect to a Malicious Server

In this section we deal with the case where the server is malicious. It is instructive to consider first the protocol of Section 3 and examine the possibilities of a malicious server to violate the privacy of the protocol. Suppose that the server after receiving the functions \( f_L, f_R \) from the user (in Step 1) can find a pair of strings \( \alpha_1, \alpha_2 \in \{0,1\}^K \) such that \( f_L(\alpha_1) = f_L(\alpha_2) \) (note that the properties of \( \mathcal{F} \) guarantee that for every \( x \) and a randomly chosen \( f \in \mathcal{F} \) it is hard to find \( x^* \); but it does not guarantee that after choosing \( f \) one cannot find a pair \( x, x^* \) with respect to this \( f \); this is exactly the weakness that we wish to use). Then, the server can replace say \( z_{1,L} \) by \( \alpha_1 \). Now, when getting \( r_L, r_R \) from the user (in Step 3) it can tell whether the first block is of type "E" or "N" (since it knows both \( z_{1,L} \) and \( z_{1,L}^* \) which are just \( \alpha_1 \) and \( \alpha_2 \)). So, for example, if the block is of type "E" then it follows that \( i \) is not in the first block. This violates the privacy of \( i \).

To overcome the above difficulties, we replace the use of the family \( \mathcal{F} \) by the use of interactive hashing. While the two tools have several similarities, interactive hashing is the right tool to make sure that the server cannot, for example, force both \( \alpha_1 \) and \( \alpha_2 \) to be mapped in the same way. However, there is another technical difficulty in generalizing the honest-but-curious case to the malicious case. Consider the proof of security in Section 3.1. A crucial point in that proof is that we can make \( z_{1,L} \) (which is fixed) and \( g^{-1}(y) \) be mapped to the same value. In the malicious case this cannot be done because the server need not fix the database and may choose it in some arbitrary way (possibly depending on the communication). Intuitively, this means that the fact that the distinguisher can tell blocks of type "E" (equal) from blocks of type "N" (not equal) does not necessarily help us in predicting the hard-core bit. This will require us to come up with some extra new machinery (see the definition of \( \mathcal{G} \) below). We prove the following theorem:

**Theorem 2.** If one-way trapdoor permutations exist then there exists malicious single-server PIR protocol whose communication complexity is at most

\[
n - \frac{n}{6K} + O(K^2).
\]

(More precisely, the user sends \( O(K^2) \) bits and the server sends at most \( n - \frac{n}{6K} \) bits. Also, with exponentially small probability, in \( n/K \), the protocol fails; i.e., the user do not get the bit \( x \); but its privacy is still maintained.\(^6\))

Let \( \mathcal{G} \) be a collection of one-way trapdoor permutations, as guaranteed by the theorem. As a first step we construct, based on \( \mathcal{G} \), a new family of one-way trapdoor permutations \( \mathcal{G} \) which is defined as follows. Each function \( \hat{g} \in \mathcal{G}_K \) is defined using 4 functions \( g_{00}, g_{01}, g_{10}, g_{11} \in \mathcal{G}_{K-2} \). Let \( x \) be a string in \( \{0,1\}^K \) and write \( x = b_1 b_2 w \), where \( b_1, b_2 \in \{0,1\} \) and \( w \in \{0,1\}^{K-2} \). We define

\[\hat{g}(x) = b_1 b_2 g_{b_1 b_2}(w).\]

Clearly each such \( \hat{g} \) is a permutation over \( \{0,1\}^K \). The trapdoor \( \hat{g}^{-1} \) corresponding to \( \hat{g} \) consists of the corresponding 4 trapdoors; i.e., \( (g_{00}^{-1}, g_{01}^{-1}, g_{10}^{-1}, g_{11}^{-1}) \). The generating algorithm for \( \mathcal{G} \), denoted \( I_{\mathcal{G}}(1^K) \) simply works by applying \( I_{\mathcal{G}}(1^{K-2}) \) four times for generating \( g_{00}, g_{01}, g_{10}, g_{11} \) (with their trapdoors).

As before assume, without loss of generality, that \( n \) is divisible by \( 2K \) and let \( \ell = n/(2K) \). The protocol works as follows.

---

\(^6\)As usual, the probability of error can be avoided at the cost of making the claim about the expected communication complexity (rather than about the worst case complexity).
1. The user picks two functions $\hat{g}_L$ and $\hat{g}_R$ (including the corresponding trapdoors $\hat{g}_L^{-1}$ and $\hat{g}_R^{-1}$) using the generating algorithm $I_\hat{G}(1^K)$. It sends the functions $\hat{g}_L, \hat{g}_R$ to the server (without the trapdoors).

2. As before the server and the user view the string $x$ as if it is composed of $2\ell$ “blocks” $z_{1,L}, z_{1,R}, z_{2,L}, z_{2,R}, \ldots, z_{i,L}, z_{i,R}$ each of size $K$.

Now the server and the user play $2\ell$ interactive hashing protocols as follows. First, the user chooses $K - 1$ linearly independent vectors in $\{0, 1\}^K$ denoted $(H_1^L, \ldots, H_{K-1}^L)$. Now, for each $t$ from 1 to $K - 1$ (in rounds) do:

- The user sends to the server $H_t^L$.
- The server sends to the user the bits $\langle H_t^L, \hat{g}_L(z_{1,L}) \rangle, \ldots, \langle H_t^L, \hat{g}_L(z_{t,L}) \rangle$.

The same is repeated for the “right” blocks. That is, the user chooses another set of $K - 1$ linearly independent vectors $H_1^R, \ldots, H_{K-1}^R$ and (in rounds) get from the server the values $\langle H_t^R, \hat{g}_R(z_{1,R}) \rangle, \ldots, \langle H_t^R, \hat{g}_R(z_{t,R}) \rangle$.

3. The user, having the trapdoors for both $\hat{g}_L$ and $\hat{g}_R$, can compute for each block $z$ the two possible pre-images $\{z, z^*\}$. We call a block bad if the first two bits of $z, z^*$ are equal; otherwise it is called good. If more than $1/3$ of the blocks are bad then the protocol halts (it is important to note that the functions in $\hat{G}$ do not change the first two bits; therefore both players, including the server who does not have the trapdoor, can tell which block is bad and which is not). We call a pair of blocks $z_{i,L}, z_{i,R}$ good if both blocks are good; otherwise the pair is bad.

4. Dealing with bad pairs of blocks:

The user chooses two more vectors $H_K^L$ (independent of $H_1^L, \ldots, H_{K-1}^L$) and $H_K^R$ (independent of $H_1^R, \ldots, H_{K-1}^R$). It sends these vectors to the server. In return, for each bad pair $z_{i,L}, z_{i,R}$, the server sends $\langle H_K^L, \hat{g}_L(z_{i,L}) \rangle$ and $\langle H_K^R, \hat{g}_R(z_{i,R}) \rangle$. In this case both $z_{i,L}, z_{i,R}$ become known to the user.

5. Dealing with good pairs of blocks:

Assume that the bit $x_i$ is in some block $z_{i,L}$, for some good pair $z_{i,L}, z_{i,R}$ (if $i$ is in a pair where at least one of the blocks is bad then in fact the user already knows the block from the previous step and can continue in an arbitrary manner). The user picks random $r_L, r_R \in \{0, 1\}^K$ such that

$$r_L(z_{i,L}) \neq r_L(z_{i,L}^*) \quad \text{and} \quad r_R(z_{i,R}) = r_R(z_{i,R}^*).$$

(If the index $x_i$ is in block $z_{i,L}$ then $r_L, r_R$ are chosen subject to the constraint $r_R(z_{i,R}) \neq r_R(z_{i,R}^*)$ and $r_L(z_{i,L}) = r_L(z_{i,L}^*)$.)

(a) The user sends $r_L, r_R$ to the server.

(b) For every good pair $z_{i,L}, z_{i,R}$ the server computes and sends the bit $b_i = r_L(z_{i,L}) \oplus r_R(z_{i,R})$.

(c) By the choice of $r_L, r_R$ the bit $b_i$ allows the user to compute the value of $z_{i,L}$ (or the value of $z_{i,R}$ depending on the way that $r_L, r_R$ were chosen). This gives the user the bit $x_i$ (as well as all other bits in the corresponding block).

Remark: Improvements similar to those described in Section 3.2 are possible in this case as well; details are omitted for lack of space.
Correctness: The correctness is similar to the correctness of the protocol in Section 3; one difference, which is not crucial for the correctness argument, is the use of the interactive hashing (i.e., \(H^1, \ldots, H^{K-1}\) and \(H^R, \ldots, H^{R}_{K-1}\)) instead of “standard hashing” (i.e., apply the functions \(h_L, h_R \in \mathcal{H}\) and chop the last bit). The second difference, is the treatment of bad pairs; however, from the point of view of correctness this is an easy case since both blocks of each such pair become known to the user. The only significant difference is the fact that the protocol may halt without the server retrieving \(z_i\) (Step 3). However, the properties of interactive hashing guarantee that the probability of each block being bad (i.e., both pre-images start with the same 2 bits) is 1/4; hence, the probability that at least 1/3 of the blocks are bad is exponentially small in the number of blocks (i.e., \(2^\ell = n/K\)).

Communication complexity: The only messages sent by the user are those for specifying the vectors \(H^1, \ldots, H^L, H^R, \ldots, H^{R}_{K-1}\) as well as \(\hat{g}_L, \hat{g}_R, r_L, r_R\); all together \(O(K^3)\) bits. The server, on the other hand, sends for each pair of blocks \(2(K - 1)\) bits in the interactive hashing protocol (Step 2); then, for each bad pair it sends two more bits (and at most 2/3 of the pairs are bad) and for each good pair it sends only one additional bit (and at least 1/3 of the pairs are good). All together, at most \(n - n/6K\) bits. Therefore, the communication complexity is as claimed by the theorem.

4.1 Proof of Security (sketch)

Here we provide the high level ideas for the proof of security in the malicious case. Suppose that the malicious server can distinguish two indices \(i\) and \(i'\). The first (simple-yet-important) observation is that if the index that the user wishes the retrieve happens to be (in a certain execution) in a bad pair of blocks then all the messages sent by the user during this execution are independent of the index. This allows us to concentrate on the good pairs only.

Using the same notation as in the honest-but-curious case (Section 3.1), and repeating a similar hybrid argument we conclude that there is some distinguisher that can tell pairs \(r_L, r_R\) which are drawn from the distribution

\[
\Pi_1 : (N, E), (\dagger, \dagger), (\dagger, \dagger), \ldots, (\dagger, \dagger)
\]

and pairs which are drawn from the distribution

\[
\Pi_2 : (E, E), (\dagger, \dagger), (\dagger, \dagger), \ldots, (\dagger, \dagger).
\]

This again is turned into a predictor for the Goldreich-Levin hard-core predicate. Specifically, let \(D\) be the distinguisher between \(\Pi_1\) and \(\Pi_2\). Our prediction algorithm \(B\) on input \(g \in I_\mathcal{G}(1^{K-2}), w \in R\{0,1\}^{K-2}\) construct an input for \(D\) as follows: As before it chooses \(\hat{g}_R \in I_\mathcal{G}(1^K)\), including its trapdoor (the corresponding \(r_R\) is chosen at random, based on the transcript of the interactive hashing, subject to the constraint that \(r_R(z^1_{1, R}) = r_R(z_{1, R})\)). Next, \(B\) chooses 3 functions \(g', g'', g''' \in I_\mathcal{G}(1^{K-2})\) and uses them together with \(g\) (in a random order) to define a function \(\hat{g} \in \hat{\mathcal{G}}\) (note that \(\hat{g}\) is distributed as if it was chosen directly from \(I_\mathcal{G}(1^K)\)). Suppose that \(g\) is \(g_{i_1 i_2}\) with respect to \(\hat{g}\). Next \(B\) makes sure that in the interactive hashing protocol corresponding to block \(z_{1, L}\) one of the two pre-images will be \(b_1 b_2 g^{-1}(w)\) (the properties of interactive hashing guarantee that this is possible; this is done by standard “rewinding” techniques, see [25, 22]). Now, there are two cases: either the first block is bad (in which case, as explained above, it cannot be of help for the distinguisher \(D\)) or the block is good. If the block is good then this means that one of the two pre-images is \(b_1 b_2 g^{-1}(w)\) and the other is \(b_1 b_2 g_{i_1 i_2}^{-1}(w)\), for some function \(g_{i_1 i_2}^{-1}\) different than \(g\) (by the definition of the block being good). Since for each function other than \(g\), the algorithm \(B\) knows the trapdoor then obtaining from \(B\) the information whether the block is of type "E" or type "N" suffices for computing \(r_L(g^{-1}(w))\) as required.
In the final version of this paper we will provide a full formal construction of $B$ together with its detailed proof.

**Acknowledgment**

We thank LeGamin bistro for good snacks that fueled this research.

**References**


