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

Eyal Kushilevitz[†] Rafail Ostrovsky[‡]

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, we present a protocol in which the user sends $O(K^2)$ bits and the server sends $n - \frac{cn}{K}$ bits (for any constant c), where K is the security parameter of the trapdoor permutations. Thus, for sufficiently large databases (e.g., when $K = n^{\epsilon}$ for some small ϵ) our construction breaks the information-theoretic lower-bound (of at least n bits). This demonstrates the feasibility of basing single-server private information retrieval on general complexity assumptions.

An important implication of our result is that we can implement a 1-out-of-n Oblivious Transfer protocol with communication complexity strictly less than n based on any one-way trapdoor permutation.

*B. Prneel (ED.): EUROCRYPT 2000, LNCS 1807, pp. 104-121, 2000. Springer-Verlag.

[†]IBM T.J. Watson Research Center, New-York and Computer Science Department, Technion, Haifa 32000, Israel. e-mail: eyalk@cs.technion.ac.il, eyalk@watson.ibm.com

[‡]Telcordia Technologies Inc., 445 South Street, Morristown, New Jersey 07960-6438, USA. e-mail rafail@research.telcordia.com

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. [9] and various aspects of it were further studied in [1, 8, 32, 27, 11, 15, 16, 28, 39, 12, 2, 7, 24, 30]. 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, in a setting where there are identical copies of the database stored in several servers, were found by [9] and later in [1, 24]. In this setting, 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, [9] have shown that if there is only a single server, then getting information-theoretic privacy with communication of less than n bits is impossible, hence motivating the use of replication.

Kushilevitz and Ostrovsky [27] 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 $\epsilon > 0$). [1] Their result strongly relies on the algebraic properties of the quadratic residuosity problem. Other single-server PIR protocols which are based on specific (number-theoretic and/or algebraic) intractability assumptions were subsequently presented in [28, 39, 7]. In particular, Cachin, Micali and Stadler [7] have shown that under the so-called ϕ-hiding (number-theoretic) assumption one can achieve even more efficient poly-logarithmic (in n) communication with a single server. (This is almost optimal since even without the privacy requirement the communication complexity must be at least $\log n$.) 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 [40], the program of identifying the weakest possible assumptions to reach various cryptographic tasks was launched. 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. [21, 38, 36]. What about private information retrieval? On the lower-bound front, in addition to the information-theoretic lower-bound [9], recent work has established that single-server private information retrieval with less than n communication (even $n-1$ bits) already implies the existence of one-way functions [2] and, more gener-

[1] In [8] it is shown, in the setting where there are several servers storing identical database x, that intractability assumptions might be of help in constructing efficient PIR protocols.

ally, the existence of *Oblivious Transfer* (OT) protocols [12] (the connection between PIR and OT is discussed in more details below). The most general assumption based on which it is (currently) known how to construct OT is that one-way trapdoor permutations exist [20]. [2] 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, under the sole assumption that one-way trapdoor permutations exist (without relying on special properties of any specific assumption), that single-server private information retrieval with strictly less than n communication is possible (or more precisely, of communication $n - \frac{cn}{K} + O(K^2)$, where $K \ll n$ is the security parameter and c is some constant[3]). We note however that, while the communication complexity is below the information-theoretic lower bounds of [9], it is nowhere close to what can be achieved based on specific assumptions. This quantitative question remains for future study.

As we already mentioned, single-server private information retrieval has a close connection to the notion of Oblivious Transfer (OT), introduced by Rabin [37]. A different variant of Oblivious Transfer, called 1-out-of-2 OT, was introduced in [13] and, more generally, 1-out-of-n OT was considered in [4].[4] All these notions were shown to be equivalent [5] and complete for all two party computations [25]. As mentioned, communication-efficient implementation of 1-out-of-n OT can be viewed as a single-server PIR protocol with an additional guarantee that only one (out of n) secrets is learned by the user. This notion (in the setting of several non-communicating servers) was first considered in [16] and called *Symmetric Private Information Retrieval (or SPIR)*. Kushilevitz and Ostrovsky [27] noted that in a setting of single-server PIR their protocol can be made into 1-out-of-n OT protocol (i.e., SPIR) with communication complexity $O(n^\epsilon)$ for any $\epsilon > 0$ (again, based on a specific algebraic assumption). Naor and Pinkas [30] have subsequently shown how to turn any PIR protocol into SPIR protocol with one invocation of PIR protocol and logarithmic number of invocations of 1-out-of-2 (string) OT. Combining our results with the results of [30] and with known implementations of OT based on any one-way trapdoor permutation [20], we get 1-out-of-n OT (i.e., SPIR) protocol based on any one-way trapdoor permutation whose

---

[2] Impagliazzo and Rudich [23] 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 [22] 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 [25, 26].) We also note that there are known constructions for OT which are based on *concrete* assumptions, such as the Diffie-Hellman assumption; in this case a trapdoor may not be required.

[3] Further improvements are possible; see Section 3.2.

[4] Loosely speaking, 1-out-of-n OT is a protocol for 2 players: A *sender* who initially has n secrets $x_1, \ldots, x_n$ and a *receiver* who initially holds an index $1 \le i \le n$. At the end of the protocol the receiver knows x_i but has no information about the other secrets, while the sender has no information about the index i. Note that OT is different from PIR in that there is no communication complexity requirement (beyond being polynomially bounded) but, on the other hand, "secrecy" is required for *both* players.

communication complexity is strictly less than n.

Organization and Techniques: Section 2 includes some definitions that are used in this paper. In addition, it describes several tools from the literature that are used by our constructions. These include some facts about the Goldreich-Levin *hard-core predicates* [19], some properties of *universal one-way hash functions*, introduced by Naor and Yung [31], and properties of *interactive hashing* protocol, introduced by Ostrovsky, Venkatesan and Yung [33] [5]. In Section 3 we describe our basic PIR protocols based on one-way trapdoor permutations. This 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).

2.2 Definitions

In this section we define the notions of *one-way trapdoor permutations* and of *hard-core predicates*. The reader is referred to [17] for an extended background related to these definitions.

Definition 1. A collection of functions $\mathcal{G} = (\mathcal{G}_K)$ is called a collections 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 $x \in \{0,1\}^*$ computes the value of $g(x)$ in time polynomial in $|x|$).

[5] Interactive hashing has found many applications in cryptography (cf. [33, 29, 14, 34, 35, 18, 10, 6]) since, in some settings, it can replace collision-resistant hash-functions but it can be implemented from general cryptographic assumptions. The drawback of this primitive is its high round-complexity (our protocol for a malicious server inherits this drawback; the question of how to reduce the round-complexity of this protocol is an interesting open problem).

- 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

$$\Pr_{g \in I_{\mathcal{G}}(1^K), y \in_R \{0,1\}^K}(B(g,y) = g^{-1}(y)) < \frac{1}{K^c} ,$$

  where "$g \in I_{\mathcal{G}}(1^K)$" denotes choosing a function g according to the probability distribution induced by the generating algorithm I.

Remark: There are definitions of one-way trapdoor permutations that give more power to the adversary. For example, the adversary may *adaptively* ask for many inverses of his choosing and only then try to invert the given permutation on a randomly chosen point. Another strengthening of the adversary, which is of interest in some cases, is requiring that it can recognize if g is "well-formed". The way in which we use the trapdoor permutations in our protocols, none of these issues come up and so we stick to the above simpler definition.

Next, we will need the notion of hard-core predicates. Specifically, we will use the Goldreich-Levin hard-core predicates [19]. 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 [19] 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

$$\Pr_{g \in I_{\mathcal{G}}(1^K), x \in_R \{0,1\}^K, r \in_R \{0,1\}^K}(B(g(x), r) = r(x)) \quad < \quad \frac{1}{2} + \frac{1}{K^c} .$$

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 [19] 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 $\mathbf{GF}[2^K]$. With this view in mind, let

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

Given $\mathcal{G}$ and $\mathcal{H}$, Naor and Yung [31] define the following family of functions

$$\mathcal{F} = \left\{ f : \{0,1\}^K \to \{0,1\}^{K-1} \;\mid\; g \in \mathcal{G}, h \in \mathcal{H}, f(x) = \text{chop}(h(g(x))) \right\}$$

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

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 $x, x^\star$ such that $f(x) = f(x^\star) = y$. [6]

3. Collisions are hard to find for $\mathcal{F}$ [31] (i.e., given x and $f(x)$ it is hard to find the string $x^\star$). Formally, for every x, for every probabilistic polynomial-time algorithm B, every integer c, and sufficiently large K

$$\Pr_{f=(g,h)\in I_{\mathcal{F}}(1^K)} (B(x, f(x)) = x^\star) \quad < \quad \frac{1}{K^c} .$$

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

We shall use in an essential way an *interactive hashing* protocol of Ostrovsky, Venkatesan and Yung [33]. Interactive hashing found many applications in cryptography (cf. [33, 14, 29, 34, 35, 18, 10, 6]). 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 Z_2^K).
- The players interact in $K-1$ rounds where in round i they do the following:
  - Bob sends to Alice H_t
  - Alice sends to Bob $\langle H_t, g(x) \rangle$ (the inner product of H_t and $g(x)$).

[6] Note that every $h \in \mathcal{H}$ is $1 \rightarrow 1$ and easy to invert; therefore, given y one can try the two options for the chopped bit, invert h and then invert g using the trapdoor. We also note that this property was not considered in [31] since they deal with arbitrary one-way permutations and not only with *trapdoors* permutations.

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_t's are linearly independent these equations admit two solutions, denoted $\{y, y^\star\}$ (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 $\{y, y^\star\}$ 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 $y, y^\star$, 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 $I_{\mathcal{G}}(1^K)$ 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 $g(x_1) = y^\star$ is less than $\frac{1}{K^c}$.

Interactive hashing, as described up-to this point, works with any one-way permutation. In [33] one more property was used, which is needed in the current paper as well. Specifically, we will apply interactive hashing with one-way *trapdoor* permutations; this modifications gives the following crucial property:

- 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 $g(x_1) = y^\star$).

2.4 PIR Protocols

A *Private Information Retrieval* (PIR) is a protocol for two players: a server $\mathcal{S}$ who knows an n-bit string x (called the *database*), and a user $\mathcal{U}$ 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 index i'.[7] We stress here that x is fixed and the server is not allowed to change it during the protocol's execution.

[7] For lack of space we omit the formal definition of *indistinguishability* which is a standard one [40].

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 $\mathcal{S}'$ 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).

Let $\mathcal{G}$ be a collection of one-way trapdoor permutations, as guaranteed by the theorem, and let $\mathcal{F}$ be a family of $2 \to 1$ 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. Both the server and the user view x as if it is composed of 2ℓ 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 $z_{j,L}$ and applies f_R to each block $z_{j,R}$. It sends all the outcomes

$$\begin{array}{cc} f_L(z_{1,L}) & f_R(z_{1,R}) \\ f_L(z_{2,L}) & f_R(z_{2,R}) \\ \vdots & \vdots \\ f_L(z_{\ell,L}) & f_R(z_{\ell,R}) \end{array}$$

to the user.

3. The user, having the trapdoors for both f_L and f_R, can compute for each block z the two possible pre-images $\{z, z^\star\}$. Assume that the bit x_i is in some block $z_{s,L}$, 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

$$r_L(z_{s,L}) \neq r_L(z^\star_{s,L}) \quad \text{and} \quad r_R(z_{s,R}) = r_R(z^\star_{s,R}).$$

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

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_s allows the user to compute the value of $z_{s,L}$ (or the value of $z_{s,R}$ depending on the way that r_L, r_R were chosen).[8] 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 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 $z^\star$.

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.

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 $z_{s,L}$ (resp.

[8] The user ignores all the other bits b_j, for $j \neq s$.

$z_{s,R}$) is of type "E" (equal) if $r_L(z_{s,L}) = r_L(z^\star_{s,L})$ (resp., if $r_R(z_{s,R}) = r_R(z^\star_{s,R})$); similarly, we say that a block $z_{s,L}$ (resp. $z_{s,R}$) is of type "N" (not equal) if $r_L(z_{s,L}) \neq r_L(z^\star_{s,L})$ (resp., if $r_R(z_{s,R}) \neq r_R(z^\star_{s,R})$). Hence, the choice of r_L, r_R defines a sequence of ℓ pairs in $\{E, N\}^2$ 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 $\star$ 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 $(N, E), (\star, \star), \ldots, (\star, \star)$.

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 $z_{s,L}$ and i' in $z_{s,R}$. 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

$$(N, E), (\star, \star), \ldots, (\star, \star)$$

while in the second case r_L, r_R are chosen from those that induce the sequence

$$(E, N), (\star, \star), \ldots, (\star, \star).$$

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

$$(N, E), (\star, \star), (\star, \star), \ldots, (\star, \star)$$

and

$$(\star, \star), (E, N), (\star, \star), \ldots, (\star, \star).$$

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

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 x, f_L and f_R) then it can also distinguish two adjacent distributions among the following list of distributions:

$$\begin{array}{ll}
\Pi_1: & (N, E), (\star, \star), (\star, \star), \ldots, (\star, \star) \\
\Pi_2: & (\star, E), (\star, \star), (\star, \star), \ldots, (\star, \star) \\
\Pi_3: & (\star, \star), (\star, \star), (\star, \star), \ldots, (\star, \star) \\
\Pi_4: & (\star, \star), (E, \star), (\star, \star), \ldots, (\star, \star) \\
\Pi_5: & (\star, \star), (E, N), (\star, \star), \ldots, (\star, \star)
\end{array}$$

(If each pair of adjacent distributions is indistinguishable then so are Π_1 and Π_5, contradicting the assumption that the server can distinguish.) Suppose, for example, that one can distinguish Π_1 and Π_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 Π_1 and

$$\Pi_2' : \quad (E,E),(\star,\star),(\star,\star),\ldots,(\star,\star).$$

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

$$Pr_{f_L,f_R\in I_{\mathcal{F}}(1^K),(r_L,r_R)\in\Pi_1}(D(x,f_L,f_R,r_L,r_R)=1) \quad \leq \quad \frac{1}{2}-\epsilon$$

and

$$Pr_{f_L,f_R\in I_{\mathcal{F}}(1^K),(r_L,r_R)\in\Pi_2'}(D(x,f_L,f_R,r_L,r_R)=1) \quad \geq \quad \frac{1}{2}+\epsilon.$$

We use this algorithm D to construct an algorithm B that on input $g \in I_{\mathcal{G}}(1^K), y \in_R \{0,1\}^K$ 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 [19] (See Section 2.2). Algorithm B works as follows:

1. Choose h_L at random subject to the constraint $\text{chop}(h_L(y)) = \text{chop}(h_L(g(z_{1,L})))$.[9] Let $f_L = (g,h_L)$ and $r_L = r$. (Note that, with respect to f_L we have $z^\star_{1,L} = g^{-1}(y)$. Also crucial is the fact that since D does not have y (only B does) the distribution of h_L looks random to D).

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

3. Invoke D on input (x,f_L,f_R,r_L,r_R). If the output is "1" (in which case the input is more likely to be from Π_2'; i.e., $r_L(z_{1,L})$ and $r_L(z^\star_{1,L})$ 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 Π_1; i.e., $r_L(z_{1,L})$ and $r_L(z^\star_{1,L})$ are more likely to be equal) then B's output is $r_L(z_{1,L})$. (Note that while B does not know what $z^\star_{1,L}$ 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.

[9] Specifically, in the unlikely event that $g(z_{1,L}) = y$ we are done; otherwise, choose $v \in \{0,1\}^K$ at random and let v' be identical to v with the last bit flipped. Then, we solve the system of equations $a \cdot y + b = v$ and $a \cdot g(z_{1,L}) + b = v'$ to find a, b (i.e., h_L). In particular $a = (v-v')/(y-g(z_{1,L}))$ (note that this is well defined since $y \neq g(z_{1,L})$ and different than 0 since $v \neq v'$).

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 to 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 $z, z^\star$) while each of the other hard-core predicates get the same value (on $z, z^\star$). 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 using $\mathcal{F}$ as above, where each $f \in \mathcal{F}$ is obtained by chopping a single bit from $h(g(x))$, we can chop some s bits (specifically, $s = O(\log\log n)$). Now, in Step 2 of the protocol the server needs to send only $K - s$ bits per block. In Step 3 the user can pick s strings r's that will allow him to retrieve only the block of interest. Finally, in Step 4 (if combined with the previous modification) for each d blocks it needs to send back s bits. This gives a complexity of $n - \frac{(d-1)cn\log\log n}{dK}$ bits from the server to the user (for any constant c) and $O(Kd\log\log n)$ 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 $x^\star$; but it does *not* guarantee that after choosing f one cannot find a pair $x, x^\star$ 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 α_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 $z^\star_{1,L}$ which are just α_1 and α_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 α_1 and α_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 $\hat{\mathcal{G}}$ 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, if the server is honest then with a negligible probability the protocol fails; i.e., the user does not get the bit x_i but its privacy is still maintained.[10])

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 $\hat{\mathcal{G}}$ which is defined as follows. Each function $\hat{g} \in \hat{\mathcal{G}}_K$ 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 $\hat{\mathcal{G}}$, denoted $I_{\hat{\mathcal{G}}}(1^K)$ 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.

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 $I_{\hat{\mathcal{G}}}(1^K)$. 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 2ℓ "blocks" $z_{1,L}, z_{1,R}, z_{2,L}, z_{2,R}, \ldots, z_{\ell,L}, z_{\ell,R}$ each of size K.
   Now the server and the user play 2ℓ 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.

[10] As pointed out in Section 2.4, a "bad" server can always refuse to let the user retrieve the bit; hence, this is not considered a violation of the correctness requirement.

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

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 $\langle H_t^R, \hat{g}_R(z_{1,R})\rangle\rangle, \ldots, \langle H_t^L, \hat{g}_R(z_{\ell,R})\rangle\rangle$.

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 $\{z, z^\star\}$. We call a block *bad* if the first two bits of $z, z^\star$ 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 $\hat{\mathcal{G}}$ 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 $z_{j,L}, z_{j,R}$ *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 $z_{j,L}, z_{j,R}$, the server sends $\langle H_K^L, \hat{g}_L(z_{j,L})\rangle\rangle$ and $\langle H_K^R, \hat{g}_R(z_{j,R})\rangle\rangle$. In this case both $z_{j,L}, z_{j,R}$ become known to the user.

5. *Dealing with good pairs of blocks:*
Assume that the bit x_i is in some block $z_{s,L}$, for some good pair $z_{s,L}, z_{S,R}$ (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

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

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

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

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

   (c) By the choice of r_L, r_R the bit b_s allows the user to compute the value of $z_{s,L}$ (or the value of $z_{s,R}$ 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., $H_1^L, \ldots, H_{K-1}^L$ and $H_1^R, \ldots, H_{K-1}^R$) 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 user retrieving x_i (Step 3). However, the properties of interactive hashing guarantee that if the server plays honestly, then the probability of each block being bad (i.e., both pre-images start with the same 2 bits) is 1/4. Hence, by Chernoff bound, the probability in the case of *honest* server that at least 1/3 of the blocks are bad is exponentially small in the number of blocks (i.e., $2\ell = n/K$). (Note that if the server is dishonest in a way that makes more than 1/3 of the blocks bad then the protocol is aborted.)

Communication complexity: The only messages sent by the user are those for specifying the vectors $H_1^L, \ldots, H_K^L$ and $H_1^R, \ldots, H_K^R$ as well as $\hat{g}_L, \hat{g}_R, r_L, r_R$; all together $O(K^2)$ bits. The server, on the other hand, sends for each pair of blocks $2(K-1)$ bits in the interactive hashing protocol (Step 2). If the protocol halts in Step 3 (either because the server is dishonest or just because of "bad luck") then there is no more communication. Otherwise, for each bad pair the server 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 $n - \frac{n}{6K}$ 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 (in a typical case) there is a distinguisher that can tell pairs r_L, r_R which are drawn from the distribution

$$\Pi_1: \quad (N, E), (\star, \star), (\star, \star), \ldots, (\star, \star)$$

and pairs which are drawn from the distribution

$$\Pi_2': \quad (E, E), (\star, \star), (\star, \star), \ldots, (\star, \star).$$

This again is turned into a predictor for the Goldreich-Levin hard-core predicate. Specifically, let D be the distinguisher between Π_1 and Π'_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 $\hat{g}_R \in I_{\hat{\mathcal{G}}}(1^K)$, including its trapdoor (the corresponding r_R is chosen at random, based on the transcript of the interactive hashing, subject to the constraint that $r_R(z^{\star}_{1,R}) = r_R(z_{1,R})$). 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 $I_{\hat{\mathcal{G}}}(1^K)$). Suppose that g is $g_{b_1 b_2}$ 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 [33, 29]). 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 $b'_1 b'_2 g^{-1}_{b'_1 b'_2}(w')$, for some function $g_{b'_1 b'_2}$ 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 D the information whether the block is of type "E" or type "N" suffices for computing $r_L(g^{-1}(w))$ as required.

5 Concluding Remarks

In this paper we show how based on one-way trapdoor permutations, one can get single-server PIR protocols with communication complexity smaller than n, hence overcoming impossibility results that show that no such protocols exist under certain weaker assumptions [9, 2, 12]. A major open problem is to lower the communication complexity so that it will be comparable to what can be achieved based on specific assumptions [27, 7].

Another interesting observation is that combining our results with results of Naor and Pinkas [30], one can obtain a single-server SPIR protocol [16, 27] (i.e., a 1-out-of-n OT with "small" communication complexity) based on any one-way trapdoor permutations whose communication complexity is strictly smaller than n. In contrast, all previous communication-efficient SPIR protocols required specific algebraic assumptions [27, 39, 7, 30]. Specifically, [30] show how to implement SPIR based on a single invocation of PIR and an additional $\log n$ invocations of 1-out-of-2 OT on K-bit strings (their construction uses pseudo-random functions, however those can be implemented from any one-way function [21]). Since implementing 1-out-of-2 OT based on one-way trapdoor permutations can be done with communication complexity which is polynomial in K [20], the total communication complexity of our SPIR protocol is still smaller than n (for sufficiently small K) and we need only the assumption of a one-way trapdoor permutation. This result can also be easily extended to 1-out-of-n *string* Oblivious Transfer with total communication less than the total size of all the secrets.

Acknowledgment

We thanks the anonymous referees for some useful comments. We also thank LeGamin bistro for good snacks that fueled this research.

References

[1] A. Ambainis. Upper bound on the communication complexity of private information retrieval. In *Proc. of 24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 401–407, 1997.

[2] A. Beimel, Y. Ishai, E. Kushilevitz, and T. Malkin. One-way functions are essential for single-server private information retrieval. In *Proc. of the 31th Annu. ACM Symp. on the Theory of Computing*, 1999.

[3] M. Bellare, S. Halevi, A. Sahai, and S. Vadhan. Many-to-one trapdoor functions and their relation to public-key cryptosystems. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 283–298. Springer-Verlag, 1998.

[4] G. Brassard, C. Crepeau and J.-M. Robert All-or-nothing disclosure of secrets In *Advances in Cryptology: Proceedings of Crypto '86* Springer-Verlag, 1987, pp. 234-238.

[5] C. Crépeau. Equivalence between two flavors of oblivious transfers. In *Proc. of CRYPTO '87*, pages 350–354, 1988.

[6] C. Cachin, C. Crepeau, and J. Marcil. Oblivious transfer with a memory-bounded receiver. In *Proc. 39th IEEE Symposium on Foundations of Computer Science* (FOCS), pp. 493-502, 1998.

[7] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - EUROCRYPT '99*, 1999.

[8] B. Chor and N. Gilboa. Computationally private information retrieval. In *Proc. of the 29th Annu. ACM Symp. on the Theory of Computing*, pages 304–313, 1997.

[9] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proc. of the 36th Annu. IEEE Symp. on Foundations of Computer Science*, pages 41–51, 1995. Journal version in JACM, Vol. 45(6), 1998, pp. 965-981.

[10] I. Damgard. Interactive hashing can simplify zero-knowledge protocol design without computational assumptions. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[11] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for database private information retrieval. In *Proc. of the 17th Annu. ACM Symp. on Principles of Distributed Computing*, pages 91–100, 1998.

[12] G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *Advances in Cryptology - EUROCRYPT 2000*, 2000.

[13] S. Even, O. Goldreich and A. Lempel A Randomized Protocol for Signing Contracts Communications of the ACM, Vol 28, 1985, pp. 637-447.

[14] J. Feigenbaum and R. Ostrovsky A note on one-prover, instance-hiding zero-knowledge proof systems In Advances in Cryptology – Asiacrypt'91, Lecture Notes in Computer Science, volume 739, Springer, Berlin, 1993, pp. 352–359.

[15] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In M. Luby, J. Rolim, and M. Serna, editors, *RANDOM '98, 2nd International Workshop on Randomization and Approximation Techniques in Computer Science*, volume 1518 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 1998.

[16] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proc. of the 30th Annu. ACM Symp. on the Theory of Computing*, pages 151–160, 1998.

[17] O. Goldreich. *Foundations of Cryptography (fragments of a book).* Electronic Colloquium on Computational Complexity, 1995. Electronic publication: http://www.eccc.uni-trier.de/eccc-local/ECCC-Books/eccc-books.html.

[18] O. Goldreich, S. Goldwasser, and N. Linial. Fault-tolerant computation in the full information model: the Two-party Case In *Proc. of the 32nd Annu. IEEE Symp. on Foundations of Computer Science*, pages 447–457, 1991. Journal Version in *SIAM J. on Computing*, Vol. 27(3), 1998, pp. 505-544.

[19] O. Goldreich and L. Levin. A hard predicate for all one-way functions. In *Proc. of the 21st Annu. ACM Symp. on the Theory of Computing*, pages 25–32, 1989.

[20] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. of the 19th Annu. ACM Symp. on the Theory of Computing*, pages 218–229, 1987.

[21] J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby. Construction of a pseudo-random generator from any one-way function. Technical Report TR-91-068, International Computer Science Institute, 1991.

[22] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proc. of the 30th Annu. IEEE Symp. on Foundations of Computer Science*, pages 230–235, 1989.

[23] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proc. of the 21st Annu. ACM Symp. on the Theory of Computing*, pages 44–61, 1989.

[24] Y. Ishai and E. Kushilevitz. Improved upper bounds on information theoretic private information retrieval. In *Proc. of the 31th Annu. ACM Symp. on the Theory of Computing*, 1999.

[25] J. Kilian. Basing cryptography on oblivious transfer. In *Proc. of the 20th Annu. ACM Symp. on the Theory of Computing*, pages 20–31, 1988.

[26] J. Kilian. A general completeness theorem for two-party games. In *Proc. of the 23th Annu. ACM Symp. on the Theory of Computing*, pages 553–560, 1991.

[27] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proc. of the 38th Annu. IEEE Symp. on Foundations of Computer Science*, pages 364–373, 1997.

[28] E. Mann. Private access to distributed information. Master's thesis, Technion - Israel Institute of Technology, Haifa, 1998.

[29] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung. Perfect zero-knowledge arguments for NP can be based on general complexity assumptions. In E. F. Brickell, editor, *Advances in Cryptology - CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992. Final version in *J. Cryptology*, 11(2):87–108, 1998.

[30] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. of the 31th Annu. ACM Symp. on the Theory of Computing*, pages 245–254, 1999.

[31] M. Naor and M. Yung. Universal one-way functions and their cryptographic applications. In *Proc. of the 21st Annu. ACM Symp. on the Theory of Computing*, pages 33–43, 1989.

[32] R. Ostrovsky and V. Shoup. Private information storage. In *Proc. of the 29th Annu. ACM Symp. on the Theory of Computing*, pages 294–303, 1997.

[33] R. Ostrovsky, R. Venkatesan, and M. Yung. Fair games against an all-powerful adversary. Presented at DIMACS Complexity and Cryptography workshop, October 1990, Princeton. Prelim. version in *Proc. of the Sequences II workshop* 1991, Springer-Verlag,

pp. 418-429. Final version in *AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 13 *Distributed Computing and Cryptography*, Jin-Yi Cai, editor, pp. 155-169. AMS, 1993.

[34] R. Ostrovsky, R. Venkatesan, and M. Yung. Secure Commitment Against Powerful Adversary: A Security Primitive based on Average Intractability. In Proceedings of 9th Symposium on Theoretical Aspects of Computer Science (STACS-92) (LNCS 577 Springer Verlag Ed. A. Finkel and M. Jantzen) pp. 439-448 February 13-15 1992, Paris, France.

[35] R. Ostrovsky, R. Venkatesan, and M. Yung. Interactive hashing simplifies zero-knowledge protocol design. In *Advances in Cryptology - EUROCRYPT '93*, Lecture Notes in Computer Science. Springer-Verlag, 1993.

[36] R. Ostrovsky and A. Wigderson One-Way Functions are Essential for Non-Trivial Zero-Knowledge. In *Proceedings of the Second Israel Symposium on Theory of Computing and Systems* (ISTCS-93) pp. 1-10., IEEE 1993.

[37] M. O. Rabin How to exchange secrets by oblivious transfer Technical Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[38] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. of the 22nd Annu. ACM Symp. on the Theory of Computing*, pages 387–394, 1990.

[39] J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *ASIACRYPT '98*, 1998.

[40] A. C. Yao. Theory and application of trapdoor functions. In *Proc. of the 23th Annu. IEEE Symp. on Foundations of Computer Science*, pages 80–91, 1982.