A Survey of Single-Database Private Information Retrieval: Techniques and Applications

Rafail Ostrovsky* and William E. Skeith III**

[1] Computer Science Department and Department of Mathematics, UCLA, 90095
rafail@cs.ucla.edu
[2] Department of Mathematics, UCLA
wskeith@math.ucla.edu, wskeith@ucla.edu

Abstract. In this paper we survey the notion of Single-Database Private Information Retrieval (PIR). The first Single-Database PIR was constructed in 1997 by Kushilevitz and Ostrovsky and since then Single-Database PIR has emerged as an important cryptographic primitive. For example, Single-Database PIR turned out to be intimately connected to collision-resistant hash functions, oblivious transfer and public-key encryptions with additional properties. In this survey, we give an overview of many of the constructions for Single-Database PIR (including an abstract construction based upon homomorphic encryption) and describe some of the connections of PIR to other primitives.

1 Introduction

A Single-Database Private Information Retrieval (PIR) scheme is a game between two players: a user and a database. The database holds some public data (for concreteness, an n-bit string). The user wishes to retrieve some item from the database (such as the i-th bit) without revealing to the database which item was queried (i.e., i remains hidden). We stress that in this model the database data is public (such as stock quotes) but centrally located; the user, without a local copy, must send a request to retrieve some part of the central data[1]. A naive solution is to have the user download the entire database, which of course preserves privacy. However, the total communication complexity in this solution, measured as the number of bits transmitted between the user and the database is n. Private Information Retrieval protocols allow the user to retrieve data from a public database with communication strictly smaller than n, i.e., with smaller communication then just downloading the entire database.

* Supported in part by IBM Faculty Award, Xerox Innovation Group Award, NSF Cybertrust grant no. 0430254 and U.C. MICRO grant.
** Supported in part by NSF grant no. 0430254, and U.C. Presidential Fellowship.

[1] PIR should not be confused with a private-key *searching on encrypted data* problem, where user uploads his own encrypted data to a remote database and wants to privately search over that encrypted data without reveling any information to the database. For this model, see the discussion in [9,18] and references therein.

T. Okamoto and X. Wang (Eds.): PKC 2007, LNCS 4450, pp. 393–411, 2007.

1.1 Single-Database PIR

PIR was introduced by Chor, Goldreich, Kushilevitz and Sudan [8] in 1995 in the setting where there are many copies of the same database and none of the copies are allowed to communicate with each other. In the same paper, Chor at. al. [8] showed that single-database PIR does not exist (in the information-theoretic sense.) Nevertheless, two years later, (assuming a certain secure public-key encryption) Kushilevitz and Ostrovsky [23] presented a method for constructing single-database PIR. The communication complexity of their solution is $O(2^{\sqrt{\log n \log\log N}})$ which for any $\epsilon > 0$ is less then $O(n^{\epsilon})$. Their result relies on the algebraic properties Goldwasser-Micali Public-Key encryption scheme [17]. In 1999, Cachin, Micali and Stadler [7] demonstrated the first single database PIR with polylogarithmic communication, under the so-called ϕ-hiding number-theoretic assumption. Chang [6], and Lipmaa [25] showed $O(\log^2 n)$ communication complexity PIR protocol (with a multiplicative security parameter factor), using a construction similar to the original [23] but replacing the Goldwasser-Micali homomorphic encryption with the Damgård, M. Jurik variant of the Pailler homomorphic encryption [10]. Gentry and Ramzan [15] also showed the current best bound for communication complexity of $O(\log^2 n)$ with an additional benefit that if one considers retrieving more then one bit, and in particular many consecutive bits (which we call blocks) then ratio of block size to communication is only a small constant. The scheme of Lipmaa [25] has the property that when acting on blocks the ratio of block size to communication actually approaches 1, yet the parameters must be quite large before this scheme becomes an advantage over that of [15]. In general, the issue of *amortizing* the cost of PIR protocol for many queries has received a lot of attention. We discuss it separately in the next subsection.

All the works mentioned above exploit some sort of algebraic properties, often coming from homomorphic public-key encryptions. In [24] work, Kushilevitz and Ostrovsky have shown how to construct Single Database PIR without the use of any algebraic assumptions, and instead relying on the existence of one-way trapdoor permutations. However, the use of the more general assumption comes with a performance cost: they show how to achieve $(n - O(\frac{n}{k} - k^2))$ communication complexity, and additionally, the protocol requires more than one round of interaction.

In this survey, we give the main techniques and ideas behind all these constructions (and in fact, show a generic construction from any homomorphic encryption scheme with certain properties) and attempt to do so in a unified manner.

1.2 Amortizing Database Work in PIR

Instead of asking to retrieve blocks, one can ask what happens if one wants to retrieve k out of n bits of the database (not necessarily consecutive). Indeed, this was considered by Ishai, Kushilevitz, Ostrovsky and Sahai [20]. In this setting, in addition to communication complexity (of retrieving k out of n bits) there is another important consideration: the total amount of *computation* needed to be performed by the database to compute all k PIR answers. (Observe that for a single PIR query the amount of computation required by the database must be linear: if this is not the case, the database will not touch at least one bit, and hence the database can safely deduce that the ”untouched” bits are not the

ones being retrieved, violating user's privacy.) Now, what is the total computation required to retrieve k different bits? A naive solution is to just run one of the PIR solutions k times. It is easy to see that using *hashing* one can do better: The user, with indices $i_1, \ldots, i_k$, picks at random a hash function h that sends all n entries of the database to k buckets and where the selection of h is made independently from $i_1, \ldots, i_k$. The user sends h to the database. Note that the expected size of each bucket is about n/k. The database partitions its database into buckets according to h (that is gets from the user), and treats every bucket as a new "tiny" database. For an appropriate choice of a hash family, this ensures that with probability $1 - 2^{-\Omega(\sigma)}$, the number of items hashed to any particular bucket is at most $\sigma \log k$. Now the user can apply the standard PIR protocol $\sigma \log k$ times to each bucket. Except for $2^{-\Omega(\sigma)}$ error probability, the user will be able get all k items. Note that the cost is much smaller then the naive solution. In particular, counting the length of all PIR invocations the total size of all databases on which we run standard PIR is $\sigma \log k \cdot n$, instead of naive kn. This idea is developed further, and in fact the error-probability is removed, and better performance is derived via explicit *batch codes* [20] instead of hashing.

Note however, that this approach requires that it is *the same* user that is interested in all k queries. What happens if the users are different? In this case, assuming the existence of *anonymous communication*, nearly-optimal PIR in all parameters can be achieved in the multi-user case [21].

1.3 Connections: Single Database PIR and OT

Single-database PIR has a close connection to the notion of Oblivious Transfer (OT), introduced by Rabin [35]. A different variant of Oblivious Transfer, called 1-out-of-2 OT, was introduced by Even, Goldreich and Lempel [14] and, more generally, 1-out-of-n OT was considered in Brassard, Crepeau and Robert [3]. Informally, 1-out-of-n OT is a protocol for two players: A *sender* who initially has n secrets $x_1, \ldots, x_n$ and a *receiver* who initially holds an index $1 \leq i \leq n$. 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, while for the PIR it is required only for the user. All Oblivious Transfer definitions are shown to be equivalent [5]. 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 and the remaining $n-1$ remain hidden. In [23], it is noted that their protocol can also be made into a 1-out-of-n OT protocol[2], showing the first 1-out-of-n OT with sublinear communication complexity. Naor and Pinkas [27] have subsequently shown how to turn any PIR protocol into 1-out-of-n protocol with one invocation of a Single-Database PIR protocol and logarithmic number of invocations of 1-out-of-2 OT.

[2] 1-out-of-n OT in the setting of *multiple copies of the database* where none of the copies are allowed to talk to each other was treated in [16] and renamed *Symmetric Private Information Retrieval* (SPIR), though for Single-dabatase PIR, the defitnion SPIR is identical to the more established notion of 1-out-of-n OT.

DiCresenzo, Malkin and Ostrovsky [12] showed that any single database PIR protocol implies OT. In fact, their result holds even if PIR protocol allows the communcation from database to the user to be as big as $n - 1$. Thus, [12] combined with [27] tells us that any Single-Database PIR implies 1-out-of-n OT. In [24], it is shown how to build 1-out-of-n OT based on any one-way trapdoor permutation with communication complexity strictly less than n.

1.4 Connections: PIR and Collision-Resistant Hashing

Ishai, Kushilevitz and Ostrovsky [19] showed that any one-round Single-Databe PIR protocol is also a collision-resistant hash function. Simply pick an index i for the PIR query at random, and generate a PIR query. Such a PIR query is the description of the hash function. The database contents serves as the input to the hash function and the evaluation of the PIR query on the database is the output of the hash function. It is easy to see that the PIR function is both length-decreasing and collision-resistant. It is length-decreasing by the non-triviality of PIR protocol, since it must return the answer with length which is less then the size of the database. Is it collision resistant since if the adversary can find two different databases that produce the same PIR answer, then these two databases must differ in at least one position, say j. Finding such a position tells us that $j \neq i$, hence it reveals information about i. This violates the PIR requirement that no information about i should be revealed.

1.5 Connections: PIR and Function-Hiding PKE

A classic view of a Public-Key Encryption/Decryption paradigm is that of an identity map: it takes a plaintext message m and creates a ciphertext which can be decrypted back to m. However, in many applications, instead of an identity map, there is a need for a Public-Key Encryption to perform some *secret computation* during encryption. That is, the key-generation algorithm takes as an additional input a function specification $f(\cdot) \in \mathcal{F}$ from some class $\mathcal{F}$ of functions and produces a Public Key. The resulting Public-Key is not much bigger then the description of a typical $f' \in \mathcal{F}$, yet the public-key should not reveal which f from $\mathcal{F}$ have been used during the key-generation phase. The encryption/decryption maps m to $f(m)$. The definition becomes nontrivial (in the sense that one can not push all the work of computing $f(\cdot)$ to the decryption phase) when for all $f \in \mathcal{F}$ it holds that $|f(m)| < |m|$, and we insist that the cyphertext size must be smaller than the size of m.

Any single-round PIR can be used to achieve this notion for the class of Encryption functions that encrypt a single bit out of the message, hiding *which* bit they encrypt: simply publish in your public key both the PIR query and an additional Public-Key Encryption (with small ciphertext expansion, compared to the plaintext, such as [34,10]). When encrypting the message, first compute PIR answer, and then encrypt the resulting answer with the Public-Key Encryption. (Some specific PIR constructions do not need this additional layer of encryption).

What makes the Function-Hiding PKE notion interesting, is that there are many examples of functions beyond PIR-based projection map. For example, as was shown by Ostrovsky and Skeith [31] that one can construct an encryption scheme which takes

multiple documents, and encrypts only a subset of these documents – only those that contain a set of hidden keywords, where the public-key encryption function does not reveal which keywords are used as selectors of the subset.

1.6 Connections: PIR and Complexity Theory

Dziembowski and Maurer have shown the danger of mixing computational and information-theoretic assumptions in the bounded-storage model. The key tool to demonstrate an attack was a computationally-private PIR protocol [13]. The compressibility of NP languages was shown by Harnick and Naor to be intimately connected to computational PIR [22]. In particular, what they show that if certain NP language is compressible, then one can construct a single-database PIR protocol (and a collision-resistant hash function) that can be built (in a non-black-box way) based on any one-way function. Naor and Nissim [28] have shown how to use computational PIR (and Oblivious RAMs [18]) to construct communication-efficient secure function evaluation protocols.

There is an interesting connection between zero-knowledge arguments and Single-Database PIR. In particular, Tauman-Kalai and Raz have shown (for a certain restricted class) an extremely efficient zero-knowledge argument (with pre-processing) assuming Single-Database PIR protocols [36].

Another framework of constructing efficient PIR protocols is with the help of additional servers, such that even if some of the servers leak information to the database, the overall privacy is maintained [11]. The technique of [11] is also used to achieve PIR *combiners* [26], where given several PIR implementations, if some are faulty, they can still be combined into one non-faulty PIR.

1.7 Public-Key Encryption That Supports PIR Read and Write

Consider the following problem: Alice wishes to maintain her email using a storage-provider Bob (such as Yahoo! or hotmail e-mail account). She publishes a public key for a semantically-secure public-key Encryption scheme, and asks all people to send their e-mails encrypted under her Public Key to the intermediary Bob. Bob (i.e. the storage-provider) should allow Alice to collect, retrieve, search and delete emails at her leisure. In known implementations of such services, either the content of the emails is known to the storage-provider Bob (and then the privacy of both Alice and the senders is lost) or the senders can encrypt their messages to Alice, in which case privacy is maintained, but sophisticated services (such as search by keyword, and deletion) cannot be easily performed by Bob. Recently, Boneh, Kushilevitz, Ostrovsky and Skeith [2] (solving the open problem of [1]) have shown how to create a public key that allows arbitrary senders to send Bob encrypted e-mail messages that support PIR queries over these messages and the ability to modify (i.e. to do PIR writing) Bob's database, both with small communication complexity (approximately $O(\sqrt{n})$). It may be interesting to note, however, that manipulating the algebraic structures of currently available homomorphic encryption schemes cannot achieve PIR writing with communication better than $\Omega(\sqrt{n})$, as shown in the recent work of Ostrovsky and Skeith [32].

1.8 Organization of the Rest of the Paper

In the rest of the paper we give an overview of the basic techniques of single database PIR. It is by no means a complete account of all of the literature, but we hope that it rather serves as an introduction, and a clear exposition of the techniques that have proved themselves most useful. We begin with what we feel are the most natural and intuitive settings, which are based upon homomorphic encryption, and we attempt to give a fairly unified and clear account of this variety of PIR protocols. We then move to PIR based on the Φ-Hiding assumption, and to a construction based upon one-way trapdoor permutations. Throughout, our focus is primarily on the intuition behind these schemes; for complete technical details, one can of course follow the references.

1.9 Balancing the Communication Between Sender and Receiver

Virtually every computationally private information retrieval protocol is somewhat comparable to every other in that they all:

- Adhere to a strict definition of privacy
- Necessarily have $\Omega(n)$ computational complexity (where n is the size of the database).[3]

As such, it is the case that the primary metric of value or quality for a PIR protocol is the total amount of *communication* required for its execution. Therefore, it may be useful to examine a somewhat general technique for minimizing communication complexity in certain types of protocols, which we'll be able to apply to computational PIR. Suppose that a protocol $\mathcal{P}$ is executed between a user $\mathbf{U}$ and a database $\mathbf{DB}$, in which $\mathbf{U}$ should privately learn some function $f(X)$ where $X \in \{0,1\}^n$ is the collection of data held by $\mathbf{DB}$. By "privately", we mean that $\mathbf{DB}$ should not gain information regarding certain details of f. Let $g(n)$ represent the communication from $\mathbf{U}$ to $\mathbf{DB}$ and $h(n)$ be the communication from $\mathbf{DB}$ to $\mathbf{U}$ involved in the execution of $\mathcal{P}$. So, $g, h : \mathbb{Z}^+ \longrightarrow \mathbb{Z}^+$. As a simplifying assumption to illustrate the idea, suppose that:

1. The function of the database $f(X)$ that $\mathbf{U}$ wishes to compute via the protocol depends only on a single bit of X.
2. g, h can be represented, or at least estimated by polynomial (or rational) functions in n.

If all of these conditions are satisfied, then we'll often have a convenient way to take the protocol $\mathcal{P}$, and derive a protocol $\mathcal{P}'$ with lower communication which will just execute $\mathcal{P}$ as a subroutine. The idea is as follows: since the function of X we are computing is highly local (it depends only on a single bit of X) we can define $\mathcal{P}'$ to be a protocol that breaks down the database X into y smaller pieces (of size n/y) and executes $\mathcal{P}$ on each smaller piece. Then, the desired output will be obtained in one of the y executions of $\mathcal{P}$. Such a protocol will have total communication $T_n(y) = g(n/y) + yh(n/y)$. It may be the case that this will increase the communication of $\mathbf{U}$ or $\mathbf{DB}$, but will reduce the total communication involved. If indeed all functions

[3] In order to preserve privacy, the database's computation must involve every database element.

are differentiable as we've assumed, then we can use standard calculus techniques to minimize this function (for any positive n) with respect to y. For example, suppose that the user's communication is linear, and the database's communication is constant. For example, let $g(n) = rn + s$ and $h(n) = c$, so that $T_n(y) = yc + s + \frac{rn}{y}$. Solving the equation $\frac{d}{dy}T_n(y) = 0$ on $(0, \infty)$ gives

$$y = \frac{\sqrt{crn}}{c}$$

This value of y is easily verified to be a local minimum, and we see that by executing the protocol $\mathcal{O}(\sqrt{n})$ times on pieces of size $\mathcal{O}(\sqrt{n})$ we can minimize the total communication.

More generally, similar techniques can of course be applied when the function f depends on more than one bit of X, as long as there is a uniform way (independent of f) to break down the database X into pieces that contain the relevant bits. These techniques can be applied to more general situations still, in which the function depends on many database locations; however, in this case one will need a method of reconstructing the output from the multiple protocol returns (in our simple example, the method is just selecting the appropriate value from all the returns). Also, for this technique to be of value in such a situation, it will generally be necessary to have a uniform way to describe the problem on smaller database pieces.

2 PIR Based on Group-Homomorphic Encryption

The original work on computational PIR by Kushilevitz and Ostrovsky [23] presented a private information retrieval protocol based upon homomorphic encryption. Such techniques are often very natural ways to construct a variety of privacy-preserving protocols. It is often the case with such protocols based upon homomorphic encryption, that although the protocol is designed with a specific cryptosystem, there is a more fundamental, underlying design that could be instantiated with many different cryptosystems in place of the original, and furthermore this choice of cryptosystem can have a very non-trivial impact on performance. For example, the work of [23] used the homomorphic cryptosystem of Goldwasser and Micali [17] to create a PIR protocol, and in the following years, many other similar protocols were developed based upon other cryptosystems, e.g., the work of Chang [6] which is based upon the cryptosystem of Paillier [34], and also the work of Lipmaa [25]. However, the method of [23] was actually quite generic, although it was not originally stated in generality. In this section, we'll present an abstract construction based upon any group homomorphic encryption scheme which has [23] and [6] as special cases, as well as capturing the work of [25]. Hopefully, this section will provide the reader with general intuition regarding private information retrieval, as well as a pleasant way to understand the basics of a moderate amount of the literature in computational PIR.

2.1 Homomorphic Encryption Schemes

Let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a cryptosystem, the symbols representing the key generation, encryption, and decryption algorithms respectively. Generally, we say that such a cryptosystem

is secure if it is secure against a chosen plaintext attack, i.e., if $\mathcal{E}$ produces distributions that are computationally indistinguishable, regardless of the input. Roughly speaking, this means that it is not feasible to extract any information from the output of $\mathcal{E}$. For example, even if a (computationally bounded) adversary knows that there are only two possible messages a and b, he still cannot tell $\mathcal{E}(a)$ apart from $\mathcal{E}(b)$, even if he repeatedly executes the (randomized) algorithm $\mathcal{E}$ on inputs of his choice.

To construct our PIR protocol, we only need a secure cryptosystem that is homomorphic over an abelian group, G. I.e., if the cryptosystem $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ has plaintext set G, and ciphertext set G', where G, G' are groups, then we have that

$$\mathcal{D}(\mathcal{E}(a) \star \mathcal{E}(b)) = a * b$$

where $a, b \in G$, and $*, \star$ represent the group operations of G, G' respectively. So, the cryptosystem allows for oblivious distributed computation of the group operation of G. (Note that we reduce the equivalence to hold modulo decryption since the encryption algorithm $\mathcal{E}$ must be probabilistic in order to satisfy our requirements for security.)

For such a cryptosystem to be of any conceivable use, we of course have that $|G| > 1$. Hence, there is at least one element $g \in G$ of order greater than 1. Suppose that $\mathrm{ord}(g) = m$. If the discrete log problem[4] in G is easy (as will often be the case e.g., when G is an additive group of integers) then we can represent our database as $X = \{x_i\}_{i=1}^n$ where each $x_i \in \{0, ..., m-1\}$. Otherwise, we will just restrict the values of our database to be binary, which is the traditional setting for PIR. I.e., $x_i \in \{0, 1\}$ for all $i \in [n]$. As it turns out, a homomorphic encryption protocol alone is enough to create a PIR protocol.

2.2 Basic Protocols from Homomorphic Encryption

In what follows, we will provide a sequence of examples of PIR from homomorphic encryption, each becoming slightly more refined and efficient. Let us suppose that the i^* position of the database is desired by a user $\mathcal{U}$. Keeping the notation established above, let G, G' be groups which correspond to the plaintext and ciphertext of our homomorphic cryptosystem (resp.) and let $g \in G$ be a non-identity element. As a first attempt at a PIR protocol, a user $\mathcal{U}$ could send queries of the form $Q = \{q_i\}_{i=1}^n$ where each $q_i \in G'$ such that

$$\mathcal{D}(q_i) = \begin{cases} g & \text{if } i = i^* \\ \mathrm{id}_G & \text{otherwise} \end{cases}$$

Then, the database can respond with

$$R = \sum_{i=1}^{n} x_i \cdot q_i$$

[4] We will refer to the problem of inverting the $\mathbb{Z}$-module action on an abelian group G as the "discrete log problem in G".

using additive notation for the operation of G' (and of G from this point forward) and using $\cdot$ to represent the $\mathbb{Z}$-module action. Now $\mathcal{U}$ can recover the desired database bit as follows, computing

$$\mathcal{D}(R) = \mathcal{D}\left(\sum_{i=1}^{n} x_i \cdot q_i\right) = \sum_{i=1}^{n} x_i \cdot \mathcal{D}(q_i) = x_{i^*} \cdot \mathcal{D}(q_{i^*}) = x_{i^*} \cdot g$$

and hence $\mathcal{U}$ determines $x_{i^*} = 1$ if and only if $\mathcal{D}(R) = g$. Or, in the case where the discrete log is easy in G, $\mathcal{U}$ would compute x_{i^*} as the log of $\mathcal{D}(R)$ to the base g (just by division, in the case of an additive group of integers). This protocol is clearly correct, but it is also easily seen to be private. The only information received by $\mathcal{DB}$ during the protocol was an array of ciphertexts, which by our assumptions on the cryptosystem, each come from (computationally) indistinguishable distributions, and hence contain no information that can be efficiently extracted. For a formal proof, one can apply a standard hybrid argument.

However, although our protocol is both correct and private, it unfortunately requires the communication of information proportional in size to the entire database in order to retrieve a single database element. This could have just as easily been done by sending the entire database to the user, which would also maintain the user's privacy. Setting $k = \log |G'|$ as a security parameter, the user must communicate $\mathcal{O}(nk)$ bits. Fortunately, this can be modified into a more communication-efficient protocol without much effort. To begin, one can organize the database as a square, $X = \{x_{ij}\}_{i,j=1}^{\sqrt{n}}$, and if the (i^*, j^*) position of the database is desired, the user can send a query of the form $Q = \{q_i\}_{i=1}^{\sqrt{n}}$ defined just as before (we will ignore the j^* index for reasons that will become clear shortly). Then, the database can compute $R_j = \sum_{i=1}^{\sqrt{n}} x_{ij} \cdot q_i$ for each j and send $\{R_j\}_{j=1}^{\sqrt{n}}$ back to the user as the query response. Now as we've seen, from $\mathcal{D}(R_j)$, $\mathcal{U}$ can recover $\{x_{i^* j}\}_{j=1}^{\sqrt{n}}$ just as before. In particular, $\mathcal{U}$ can compute $x_{i^* j^*}$ (even though much more information is actually received). Note that the total communication involved in the protocol has now become non-trivially small: it is now proportional to $\sqrt{n}$ for each party as opposed to the $\mathcal{O}(n)$ communication required by our original proposal and the trivial solution of communicating the entire database to $\mathcal{U}$.

However, we can make further improvements still. Let

$$\phi : G' \hookrightarrow \mathbb{Z}^l$$

be an injective map such that for all $y \in G'$, each component of $\phi(y)$ is less than $\mathrm{ord}(g)$. I.e., one can think of the map as $\phi : G' \hookrightarrow \mathbb{Z}_{\mathrm{ord}(g)}^l$. Any such map with do-we only require that both ϕ and ϕ^{-1} are efficiently, publicly computable. Note that in general, we will always have $l > 1$ since $\mathrm{ord}(g) \leq |G|$ and $|G| < |G'|$, the latter inequality following from the fact that the encryption scheme is always probabilistic ($\mathcal{D}$ is never injective, but of course is always surjective). Again, note that we do not ask for any algebraic conditions from the map ϕ; it can be any easily computed injective set map. (For example, we could just break down a binary representation of elements of G' into sufficiently small blocks of bits to obtain the map ϕ.) Now, we can refine our query,

and send $Q = \left[\{q_i\}_{i=1}^{\sqrt{n}}, \{p_j\}_{j=1}^{\sqrt{n}}\right]$ where q_{i^*} and p_{j^*} are set to encryptions of g, but all others encrypt id_G. Then, the database will initially proceed as before, computing

$$R_j = \sum_{i=1}^{\sqrt{n}} x_{ij} \cdot q_i$$

but then further computing

$$\overline{R}_t = \sum_{j=1}^{n} \phi(R_j)_t \cdot p_j$$

where $\phi(R_j)_t$ represents the t-th component of $\phi(R_j)$. This is sent as the query response to $\mathcal{U}$. To recover the desired data, $\mathcal{U}$ computes for every $t \in [l]$

$$\mathcal{D}(\overline{R}_t) = \phi(R_{j^*})_t \cdot g$$

from which $\phi(R_{j^*})$ can be computed. Then since ϕ^{-1} is efficiently computable, $\mathcal{U}$ can recover $R_{j^*} = \sum_{i=1}^{\sqrt{n}} x_{ij^*} \cdot q_i$, and as we have seen $x_{i^*j^*}$ is easily recovered from $\mathcal{D}(R_{j^*})$.

So now what amount of communication is required by the parties? The database sends $\mathcal{O}(l \log(|G'|)) = \mathcal{O}(lk)$ bits of information, meanwhile the user $\mathcal{U}$ sends $\mathcal{O}(2k\sqrt{n})$ bits of information which will generally be a large improvement on the database side. We can naturally extend this idea to higher dimensional analogs. Representing the database as a d-dimensional cube (we have just seen the construction for $d = 1, 2$), we accomplish the following communication complexity: $\mathcal{O}(kd\sqrt[d]{n})$ for the user's query, and $\mathcal{O}(l^{d-1}k)$ communication for the database's response.

The preceding construction is essentially that of [23] and of [6]. Both are simply special cases of what has been described above:

The work of [23] is based upon the cryptosystem of [17], which is homomorphic over the group $\mathbb{Z}_2$, having ciphertext group $\mathbb{Z}_N$ for a large composite N. In this case, it is simply the binary representation of a group element that plays the role of the map $\phi : G \hookrightarrow \mathbb{Z}^l$. I.e., we have $l = k$, the security parameter, and $\phi : \mathbb{Z}_N \hookrightarrow \mathbb{Z}^k$ takes an element $h \in \mathbb{Z}_N$ and maps it to a sequence of k integers, each in $\{0, 1\}$ corresponding to a binary representation of h.

The work of [6] is also a special case of this construction. Here, the protocol is based upon the cryptosystem of [34], and we have $G = \mathbb{Z}_N$ and $G' = \mathbb{Z}_{N^2}^*$, hence we can greatly reduce the parameter l in comparison to the work of [23]. In this case, it is easy to see that we only need $l = 2$, which is in fact minimal, as we have discussed before. The author of [6] uses the map $\phi : \mathbb{Z}_{N^2}^* \hookrightarrow \mathbb{Z}_N$ defined by the division algorithm, dividing by N to obtain a quotient and remainder of appropriate size. Roughly speaking, (and using C-programming notation) he uses the map $x \mapsto (x/N, x\%N)$. However, as we have seen before, this is not necessary- any map could have been used (appropriately partitioning bits, etc.). Note also that since the discrete log in G is not hard (as we have defined it) we do not need to restrict our database to storing bits. Database elements could be any numbers in $\mathbb{Z}_N$.

These quite generic methods also capture the work of [25], as long as the appropriate cryptosystem is in place.

Consider a "length-flexible" cryptosystem, for example, that of Damgård and Jurik [10]. Such a cryptosystem has the property that given a message of arbitrary length, and given a *fixed* public key, one can choose a cryptosystem from a family of systems based on that key, so that the message *fits in one ciphertext block* regardless of the key and the message length. Using this, we can further reduce the database's communication in our PIR protocol, using essentially the very same generic technique described above. We'll demonstrate the following:

Theorem 1. *For all $d \in \mathbb{Z}^+$ there exists a PIR protocol based on the homomorphic cryptosystem of Damgård and Jurik with user communication of $\mathcal{O}(kd^2 \sqrt[d]{n})$ and database communication of $\mathcal{O}(kd)$ where k is a security parameter and n is the database size.*

What the Damgård and Jurik system affords us is the following: instead of having only one plaintext and ciphertext group G, G', we now have a countable family at our disposal:

$$\{G_i, G_i'\}_{i=1}^{\infty}$$

all of which correspond to a *single* public key. These groups are realized by $G_i \simeq \mathbb{Z}_{N^i}, G_i' \simeq \mathbb{Z}_{N^{i+1}}^*$, and hence we have natural inclusion maps of $G_i' \hookrightarrow G_{i+1}$. This, along with the observation that G_i is cyclic for all i, are essentially the only important facts regarding this system that we'll utilize. So, G_i is always cyclic, and we have a natural (although not algebraic) map

$$\psi_i : G_i' \hookrightarrow G_{i+1}$$

This is all we need to modify our generic method. We will just replace the map ϕ with the maps ψ_i, and accordingly, we will modify our query so that the vector for the i-th dimension encrypts id_{G_i} in all positions except for the index of interest, which will encrypt a generator of G_i. With only these minor substitutions to the abstract construction, the protocol will follow exactly as before. This will give us a protocol with communication complexity for the user $\mathcal{U}$ of

$$\sum_{i=1}^{d} ik \sqrt[d]{n} = \mathcal{O}(kd^2 \sqrt[d]{n})$$

and for the database, we require only

$$\mathcal{O}(kd)$$

as opposed to the previous exponential dependence on the dimension d of the cube used! Optimizing the parameters, setting $d = \frac{\log(n)}{2}$, we have $\mathcal{O}(k \log^2(n))$ communication for the user and $\mathcal{O}(k \log(n))$ for the database. So, as one can see, even a completely generic method can be quite useful, producing a near optimal, poly-logarithmic protocol.

3 PIR Based on the Φ-Hiding Assumption

Cachin, Micali, and Stadler recently developed a new cryptographic assumption called the Φ-Hiding Assumption, and successfully used this assumption to build a PIR protocol with logarithmic communication. Roughly, this assumption states that given two primes p_0, p_1 and a composite $m = pq$ such that either $p_0|\phi(m)$ or $p_1|\phi(m)$, it is hard to distinguish between the two primes. (Here, $\phi(m)$ is the Euler-ϕ function, so that $\phi(m) = (p-1)(q-1)$.) The assumption also of course states that given a small prime p, it is computationally feasible to find a composite m such that $p|\phi(m)$. Such an m is said to *ϕ-hide* p. A query for the i-th bit of the database essentially contains input to a prime sequence generator, a composite m that ϕ-hides p_i (the i-th prime in the sequence) and a random $r \in \mathbb{Z}_m^*$. The database algorithm returns a value $R \in \mathbb{Z}_m^*$ such that with very high probability, R has p_i-th roots if and only if the database bit at location i was 1.

3.1 Preliminaries

To understand the protocol, let us start with some very basic algebraic observations. Let G be a finite abelian group, and let $k \in \mathbb{Z}^+$. Consider the following map:

$$\varphi_k : G \to G \quad \text{defined by} \quad x \mapsto x^k$$

Since G is abelian, it is clear that φ_k is a homomorphism for all $k \in \mathbb{Z}^+$. What is $\varphi_k(G)$? Clearly it is precisely the set of all elements in G that posses a k-th root in G. I.e.,

$$\mathrm{Im}(\varphi_k) = \{x \in G \mid \exists y \in G \ni x = y^k\}$$

We will denote this set by $H_k = \mathrm{Im}(\varphi_k)$. Clearly it is a subgroup since it is the homomorphic image of a group. The size of this subgroup of course depends on k and G. If, for example, $(k, |G|) = 1$, then it is easy to see that $\varphi_k(G) = G$, since if $\mathrm{Ker}(\varphi_k) \neq \{e\}$ then there are non-identity elements of order dividing k, which is clearly impossible. In the case that $(k, |G|) > 1$, how big is $\mathrm{Ker}(\varphi_k)$? It is at least as big as the largest prime divisor of $(k, |G|) > 1$, by Cauchy's theorem if you like. For example, if k is a prime such that $k \mid |G|$, then the map φ_k is at least a k to 1 map.

Finally, let's take a look at the subgroups $H_k = \varphi_k(G) = \mathrm{Im}(\varphi_k)$. We will just need the following observation:

$$\forall k \in \mathbb{Z} \quad H_k \lhd\!\lhd G$$

Here the symbol $H_k \lhd\!\lhd G$ signifies that H_k is a *characteristic* subgroup of G, which is to say that the subgroups H_k are fixed by *every* automorphism of G. (Compare with *normal* subgroups which are those fixed by every *inner* automorphism of G.) Note that for any finite G, if $H \lhd\!\lhd G$ and $\varphi \in \mathrm{Aut}(G)$ then $\varphi(x) \in H \iff x \in H$ for all $x \in G$.

Let us summarize the few facts that will be of importance to us, and also narrow our view to correspond more directly to what we will need. Suppose that $p \in \mathbb{Z}$ is a prime, and define the maps φ_p as before. Then,

1. $\varphi_p \in \mathrm{Aut}(G) \iff p \nmid |G|$
2. φ_p is at least a p to 1 map if $p \mid |G|$.

3. $\forall p, \quad H_p \lhd\!\lhd G$ (although this is trivial in the case that $p \nmid |G|$ and hence $H_p = G$). So for any $\varphi \in \mathrm{Aut}(G)$ and $x \in G$ we have $\varphi(x) \in H_p \iff x \in H_p$.

3.2 A Brief Description of the Protocol

We now have enough information for a basic understanding of how and why the PIR protocol of [7] works. First, we will begin with the "how". Continuing with our preceding notation, suppose that $X = \{x_i\}_{i=1}^n$ is our database, with each $x_i \in \{0, 1\}$, and again, suppose that the index of interest to $\mathcal{U}$ is i^*. The protocol executes the following steps, involving a database $\mathcal{DB}$ and a user $\mathcal{U}$.

1. $\mathcal{U}$ sends a random seed for a publicly known prime sequence generator to $\mathcal{DB}$, the primes being of intermediate size[5].
2. $\mathcal{U}$ computes p_{i^*}, the i^*-th prime in the sequence based on the random seed.
3. $\mathcal{U}$ finds a composite number m that ϕ-hides p_{i^*}, and sends m to $\mathcal{DB}$. In particular, we have that $p_{i^*} \mid \phi(m)$. Recall that $\phi(m) = |\mathbb{Z}_m^*|$.
4. $\mathcal{DB}$ selects $r \in \mathbb{Z}_m^*$ at random, and computes $R \in \mathbb{Z}_m^*$ as follows:

$$\begin{aligned} R &= \varphi_{p_n^{x_n}} \circ \varphi_{p_{n-1}^{x_{n-1}}} \circ \cdots \circ \varphi_{p_1^{x_1}}(r) \\ &= r^{\prod_{i=1}^n p_i^{x_i}} \bmod m \end{aligned}$$

5. $\mathcal{U}$ receives R from $\mathcal{DB}$ as the response, and determines that $x_{i^*} = 1$ if and only if $R \in H_{p_{i^*}}$.

These steps are essentially the entire protocol at a high level. However, it may not be immediately obvious that the statement $R \in H_{p_{i^*}}$ has much to do with the statement $x_{i^*} = 1$. But using the 3 facts we established early on, it isn't too hard to see that these are in fact equivalent with very high probability.

From our first fact, we know that $\varphi_{p_i} \in \mathrm{Aut}(G)$ whenever $i \neq i^*$ with overwhelming probability, since the only way for this to not be the case is if $p_i \mid \phi(m)$. However, due to the fact that there are at most only a logarithmic number of prime divisors of $\phi(m)$ out of many choices, this event will be extremely unlikely[6]. So, all of the φ_{p_i} are automorphisms, except for $\varphi_{p_{i^*}}$.

From our next fact, we know that with very high probability $r \notin H_{p_{i^*}}$, where $r \in \mathbb{Z}_m^*$ was the element randomly chosen by $\mathcal{DB}$. Since the map is at least p_{i^*} to 1, the entire group is at least p_{i^*} times the size of $H_{p_{i^*}}$. So, if we were to pick an element at random from $\mathbb{Z}_m^*$, there is at best a $\frac{1}{p_{i^*}}$ chance that it will be in $H_{p_{i^*}}$. So, in the length of our primes p_i, there is an exponentially small probability that a random r will be in $H_{p_{i^*}}$.

Finally, we noted that the subgroups H_{p_i} are characteristic subgroups, and hence our fixed by every automorphism of $\mathbb{Z}_m^*$. In particular, $H_{p_{i^*}} \lhd\!\lhd \mathbb{Z}_m^*$. So, all of the

[5] Revealing a large prime dividing $\phi(m)$, $(p > \sqrt[4]{m})$ enables one to factor m, so the primes must be chosen to be small.

[6] According to the prime number theorem, there are approximately $\frac{N}{2 \log N}$ primes of bit length equal to that of N. Our chances of picking m such that another p_i inadvertently divides $\phi(m)$ are approximately $\frac{\mathrm{polylog}(m)}{m}$ which is negligibly small as the length of m in bits (i.e., $\log m$, the security parameter) increases.

automorphisms $\{\varphi_{p_i}\}_{i \neq i^*}$ will preserve this group: things outside will stay outside, and things inside will stay inside, and of course $\varphi_{p_{i^*}}$ moves every element into $H_{p_{i^*}}$. I.e.,

$$\varphi_{p_{i^*}}(x) \in H_{p_{i^*}} \quad \forall x$$

and if $i \neq i^*$, then

$$\varphi_{p_i}(x) \in H_{p_{i^*}} \iff x \in H_{p_{i^*}}$$

We can trace the path that r takes to become R and see what happens: We have that the element r begins outside of the subgroup $H_{p_{i^*}}$ and then r is moved by many maps, all of which come from the set

$$\{\varphi_1, ...\varphi_{i^*-1}\} \cup \{\mathrm{Id}\}$$

depending on whether or not $x_i = 1$. But what is important is that all of these maps are automorphisms, which therefore fix $H_{p_{i^*}}$. So, no matter what the configuration of the first $i^* - 1$ elements of the database, r will have not moved into $H_{p_{i^*}}$ at this point. Next, we conditionally apply the map $\varphi_{p_{i^*}}$ depending on whether or not $x_{i^*} = 1$, which conditionally moves our element into $H_{p_{i^*}}$. This is followed by the application of more automorphisms, which as we have seen have no effect on whether or not the response R will be in $H_{p_{i^*}}$. So, since $H_{p_{i^*}}$ is fixed by every automorphism, the only chance that r has to move from outside $H_{p_{i^*}}$ to inside $H_{p_{i^*}}$ is if the map $\varphi_{p_{i^*}}$ is applied, which happens if and only if $x_{i^*} = 1$. Hence, we have that (with overwhelming probability) $R \in H_{p_{i^*}}$ if and only if $x_{i^*} = 1$.

The privacy this protocol can be proved directly from the Φ-Hiding assumption, although it may be more pleasant to think of this in terms of the indistinguishability of the subgroup H_{p_i} to a party not knowing the factorization of m. Now, let us take a look back and examine the communication to see why this was useful. The challenge of creating PIR protocols is usually to minimize the amount of communication. A PIR protocol with linear communication is quite trivial to construct: just transfer the entire database. This is of course not very useful. The PIR protocol we have described above, however, has nearly optimal communication. The database's response is a single element $R \in \mathbb{Z}_m^*$ which has size proportional to the security parameter alone (which must be at least logarithmic in n), and the user's query has the size of the security parameter, and the random input to a prime sequence generator, which could also be as small as logarithmic in n. So, we have constructed a PIR protocol with only logarithmic communication, which is of course optimal: If $\mathcal{DB}$ wants to avoid sending information proportional to the size of the database, then $\mathcal{U}$ must somehow communicate information about what index is desired, which requires at least a logarithmic amount of communication. However, with the recommended parameters for security, the total communication is approximately $\mathcal{O}(\log^8 n)$.

3.3 Generalizations: Smooth Subgroups

More recently, Gentry and Ramzan [15] have generalized some of the fundamental ideas behind these methods, creating protocols based on *smooth subgroups*, which are those that have many small primes dividing their order. Somewhat similar to CMS [7],

a list of primes is chosen corresponding to the positions of the database, and a query for position i essentially consists of a description of a group G such that $|G|$ is divisible by p_i. However, the work of [15] is designed to retrieve blocks of data at a single time (CMS [7] must be repeatedly executed to accomplish this functionality). Rather than repeatedly exponentiating by all of the primes, the database is represented as an integer e such that when reduced mod p_i, the value is the i-th block of the database (such an integer always exists of course by the Chinese Remainder Theorem). Now to recover the data (which is just $e \bmod p_i$), a discrete log computation can be made in the (small) subgroup of order p_i.

4 PIR from Any Trapdoor Permutation

In 2000, Kushilevitz and Ostrovsky [24] demonstrated that the existence of one-way trapdoor permutations suffices to create a non-trivial PIR, where non-trivial simply means that the total communication between the parties is strictly smaller than the size of the database. Although the protocol requires multiple rounds of interaction, the basic construction remains fairly simple in the case of an honest but curious server. In case of a malicious server the construction is more complicated and the reader is referred to the original paper for details. Here, we only illustrate the basic idea of the honest-but-curious case.

4.1 Preliminaries

For this construction, the existence of one way trapdoor permutations (f, f^{-1}) is assumed, as well as Goldreich-Levin hard core bits.

Another tool (used in the honest-but-curious case) is the universal one way hashing of Naor and Yung [29]. For the dishonest case, universal way-way hash functions are replaced with an *interactive hashing* protocol [33], and on top of that some additional machinery is needed. However, for the honest but curious case the proof is far more simple. Recall that universal one way hash functions satisfy a slightly weaker type of collision-resistance. Basically, if one first picks any input x from the domain, and a then independently a hash function h from a universal one way family, it is computationally infeasible to find $x' \neq x \in h^{-1}(h(x))$.

The PIR protocol we'll discuss here uses universal one way hash functions which are 2 to 1 (i.e., for all y in the codomain, $|h^{-1}(y)| = 2$) and each function will map $\{0,1\}^k \longrightarrow \{0,1\}^{k-1}$ for some integer k.

4.2 Outline of the Protocol

At a very high level, the protocol revolves around the following idea: The server takes an n bit database and partitions it into consecutive blocks of length k (k will be the input length to a trapdoor permutation f). It collapses every block of the database by one bit, and sends this (slightly) reduced-size database back to the user. The user then selects and sends to the server some information that will allow him to determine the one missing bit of information for the block in which he or she is interested. Now, using

communication balancing techniques similar to what we've described in the introduction, we can hold on to the constant advantage (below n) given to us by the server collapsing the one bit of every database block. The trick, of course, is to avoid revealing information about which block the user is interested in when recovering this last bit. The solution is quite simple. As mentioned, the database collapses a bit of each database block before sending this information to the user. There are many obvious ways to do this, for example just sending all but one bit of each block. However, in these situations, the database knows exactly the two possibilities that arise from the collapsed data sent to the user, as well as knowing the actual value in the database. This would seemingly make it quite difficult for the user to determine which of the two possibilities exist in the database without the database gaining information. So instead, a method is devised in which the database collapses a bit of each block *without knowing the other possibility*. This will enable the user to determine which possibility exists for a given block without revealing what block he or she is interested in.

4.3 Sketch of Protocol Details

As we alluded to in the outline, we need to provide a way for the database to collapse a bit of each block, but without knowing the other possibility. This is accomplished precisely via a family $\mathcal{F}$ of universal one way hash functions, and in fact, the orginal construction of such a family by Naor and Yung [29] is used. The important point, is that the only assumption needed to build this family of universal one way hash functions was the existence of one-way permutations, and furthermore, because they were constructed via one-way permutations, a party holding the trapdoor *can find collisions*. To summarize, here are the important properties we need from the family $\mathcal{F}$:

1. Each function of $\mathcal{F}$ is efficiently computable.
2. Each function has the property of being 2 to 1.
3. Given only $x, f(x)$ for $f \in \mathcal{F}$, it is computationally infeasible to find $x' \neq x \in f^{-1}(f(x))$ without trapdoor information.
4. With trapdoor information, it is feasible to find collisions in every function $f \in \mathcal{F}$.

The protocol proceeds as follows:

The database is divided into blocks of size K, one of which the user is interested in. Furthermore, the database is organized into pairs of blocks, denote them by $z_{i,L}$ and $z_{i,R}$ (L, R standing for "left" and "right"). A query consists of two descriptions of universal one way hash functions, f_L, f_R, to which the user has the trapdoors. Upon receipt of the query, the database computes the values of $f_L(z_{i,L})$ and $f_R(z_{i,R})$ for each block of the database, and returns these values to the user. The user, who has trapdoors, can compute both possible pre-images (z, z') that may correspond to the block of the database of interest. It only remains to have the database communicate which one, while maintaining privacy. This is accomplished via hardcore predicates. Without loss of generality, suppose the user wishes to retrieve the left block, say $z_{s,L}$. Then, the user selects two hardcore predicates, r_L, r_R according to the conditions that $r_L(z_{s,L}) \neq r_L(z'_{s,L})$, yet $r_R(z_{s,R}) = r_R(z'_{s,R})$. These predicates are sent to the database, who responds with $r_L(z_{i,L}) \oplus r_R(z_{i,R})$ for every pair of blocks. Now, regardless of the possibilities of the

right block, the hardcore predicates will be the same, hence the user can solve for the left hardcore predicate, and hence the left block, as we assumed the predicates evaluated on the two choices to be distinct. This completes a basic description of the protocol.

The descriptions of f_L, f_R, r_L, r_R are all $\mathcal{O}(K)$, which is the only communication from the user to the database. The communication from the database to the user is easily seen to be $n - \frac{n}{2K}$ bits in the initial round, and one more bit in the final response. Hence, the protocol does achieve smaller than n communication, for $n > O(k^2)$. Next we argue that the protocol is also secure. The only information sent to the database which contains any information about what block the user is interested in, is that of the hardcore predicates, r_L, r_R. The value of the hardcore predicates *on the two possible pre-images of a hash value* is exactly what gives us the information regarding the user's selection. We only need to show that given such predicates, they do not reveal information about the selected block. Informally, this is the right approach, as the definition of hardcore predicate states that the outcomes are hard to predict better than random when only given the output of a function. Indeed, as fairly straightforward hybrid argument shows, this is the case.

5 Conclusions

In this paper, we have given a general survey of Single-Database PIR and it's many connections to other cryptographic primitives. We also discussed several implementations of single-database PIR, including a very generic construction from homomorphic encryption. As well-studied as single database PIR seems to be, many open problems remain. For example, reducing the communication complexity of a PIR protocol based on general trapdoor permutations, as well as exploring the connections PIR has to other communication-efficient protocols both in cryptography and complexity theory.

References

1. D. Boneh, G. Crescenzo, R. Ostrovsky, G. Persiano. Public Key Encryption with Keyword Search. EUROCRYPT 2004: 506-522
2. D. Boneh E. Kushilevitzy R. Ostrovsky, W. Skeith Public Key Encryption that Allows PIR Queries IACR E-print archive, 2007.
3. 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.
4. 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.
5. C. Crépeau. Equivalence between two flavors of oblivious transfers. In *Proc. of CRYPTO '87*, pages 350–354, 1988.
6. Y. C. Chang. Single Database Private Information Retrieval with Logarithmic Communication. ACISP 2004
7. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.

8. 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: *J. of the ACM*, 45:965–981, 1998.
9. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security* CCS 2006, pages 79-88, 2006.
10. I. Damgård, M. Jurik. A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System. In Public Key Cryptography (PKC 2001)
11. G. DiCrescenzo, 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. Full version in Journal of Cryptology 14(1): 37-74 (2001).
12. G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *Advances in Cryptology - EUROCRYPT 2000*, 122-138.
13. S. Dziembowski, U. Maurer On Generating the Initial Key in the Bounded-Storage Model. EUROCRYPT 2004: 126-137
14. S. Even, O. Goldreich and A. Lempel A Randomized Protocol for Signing Contracts Communications of the ACM, Vol 28, 1985, pp. 637-447.
15. C. Gentry and Z. Ramzan. Single Database Private Information Retrieval with Constant Communication Rate. ICALP 2005, LNCS 3580, pp. 803815, 2005.
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. S. Goldwasser and S. Micali. Probabilistic encryption. In J. Comp. Sys. Sci, 28(1):270–299, 1984.
18. O. Goldreich, R Ostrovsky. Software Protection and Simulation on Oblivious RAMs. J. ACM 43(3): 431-473 (1996)
19. Y. Ishai, E. Kushilevitz, R. Ostrovsky Sufficient Conditions for Collision-Resistant Hashing. TCC 2005: 445-456
20. Y. Ishai, E. Kushilevitz, R. Ostrovsky and A. Sahai. Batch codes and their applications. STOC 2004: 262-271
21. Y. Ishai, E. Kushilevitz, R. Ostrovsky and A. Sahai. Cryptography from Anonymity. FOCS 2006: 239-248
22. D. Harnik, M Naor On the Compressibility of NP Instances and Cryptographic Applications. FOCS 2006: 719-728
23. 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.
24. E. Kushilevitz and R. Ostrovsky. One-Way Trapdoor Permutations Are Sufficient for Non-trivial Single-Server Private Information Retrieval. EUROCRYPT 2000: 104-121
25. H. Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. ISC 2005: 314-328
26. R. Meier, B. Przydatek On Robust Combiners for Private Information Retrieval and Other Primitives. CRYPTO 2006: 555-569
27. 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.
28. M. Naor, K. Nissim: Communication Complexity and Secure Function Evaluation Electronic Colloquium on Computational Complexity (ECCC) 8(062): (2001)
29. M. Naor, M. Yung. Universal One-Way Hash Functions and their Cryptographic Applications In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing. (May 15–17 1989: Seattle, WA, USA)*

30. 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.
31. R. Ostrovsky and W. Skeith. Private Searching on Streaming Data. In *Advances in Cryptology – CRYPTO 2005*
32. R. Ostrovsky and W. Skeith. Algebraic Lower Bounds for Computing on Encrypted Data. In ECCC, Electronic Colloquium on Computational Complexity
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. P. Paillier. Public Key Cryptosystems based on CompositeDegree Residue Classes. Advances in Cryptology - EUROCRYPT 99, LNCS volume 1592, pp. 223-238. Springer Verlag, 1999.
35. M. O. Rabin How to exchange secrets by oblivious transfer Technical Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.
36. Y. Tauman Kalai, R. Raz: Succinct Non-Interactive Zero-Knowledge Proofs with Preprocessing for LOGSNP. FOCS 2006: 355-366