Breaking the $O(n^{1/(2k-1)})$ Barrier for Information-Theoretic Private Information Retrieval[*]

Amos Beimel[†] Yuval Ishai[‡] Eyal Kushilevitz [§] Jean-François Raymond[¶]

Abstract

Private Information Retrieval (PIR) protocols allow a user to retrieve a data item from a database while hiding the identity of the item being retrieved. Specifically, in information-theoretic, *k-*server *PIR protocols the database is replicated among k servers, and each server learns nothing about the item the user retrieves. The cost of such protocols is measured by the* communication complexity *of retrieving one out of n bits of data. For any fixed k, the complexity of the best protocols prior to our work was $O(n^{\frac{1}{2k-1}})$ (Ambainis, 1997). Since then several methods were developed in an attempt to beat this bound, but all these methods yielded the same asymptotic bound.*

In this work, this barrier is finally broken and the complexity of information-theoretic k-server PIR is improved to $n^{O(\frac{\log\log k}{k\log k})}$. The new PIR protocols can also be used to construct k-query binary locally decodable codes *of length $\exp(n^{O(\frac{\log\log k}{k\log k})})$, compared to $\exp(n^{\frac{1}{k-1}})$ in previous constructions. The improvements presented in this paper apply even for small values of k: the PIR protocols are more efficient than previous ones for every $k \geq 3$, and the locally decodable codes are shorter for every $k \geq 4$.*

1. Introduction

A Private Information Retrieval (PIR) protocol allows a user to retrieve a data item of its choice from a database while preventing the server storing the database from gaining information about the identity of this item. This problem was introduced by Chor, Goldreich, Kushilevitz, and Sudan [11] and since then has attracted a considerable amount of attention (see below). In formalizing the problem, it is convenient to model the database by an n-bit string x where the user, holding some *retrieval index* i, wishes to learn the i-th data bit x_i. A trivial solution to the PIR problem is to send the entire database x to the user. However, while being perfectly private, the *communication complexity* of this solution may be prohibitively large. Indeed, the most significant goal of PIR-related research has been to minimize the communication complexity of PIR protocols. Unfortunately, if the server is not allowed to gain *any* information about the identity of the retrieved bit, then the linear communication complexity of the trivial solution is optimal [11]. To overcome this problem, Chor et al. [11] suggested that the user accesses k replicated copies of the database stored at different servers, requiring that each individual server gets absolutely no information about i. PIR in this setting is referred to as *information-theoretic* PIR.

The best known complexity for information-theoretic PIR protocols prior to the current work is $O(n^{1/(2k-1)})$. This was first obtained for $k = 2$ in [11] and generalized to any fixed value of k by Ambainis [1]. This upper bound remained the best known until this work, in spite of various attempts to improve it [20, 21, 5]. While these attempts resulted in finding new, very different, PIR protocols, they all ended up with the same $O(n^{1/(2k-1)})$ bound. (The constants, which depend on k, were significantly improved; this is in addition to asymptotic improvements for some extensions of the basic problem.) Note that the number of servers, k, is usually considered to be "small" and, in particular, independent of the length of the database, n; for larger values of k, there is a construction ([11] and implicitly in [3, 4]) that gives an $O(\log n)$-server protocol with $O(\log^2 n \log\log n)$ communication bits or alternatively (with different parameters) an $O(\log n/\log\log n)$-server protocol with $\mathrm{poly}(\log n)$ communication.

Other than the interest in PIR protocols for their own sake, they also found various applications (see, e.g., [15, 28, 9]). One particularly interesting application of PIR is for the construction of so-called *locally decodable codes*. A k-query Locally Decodable Code (LDC) allows to encode a database $x \in \{0,1\}^n$ into a string y, such that even if a large fraction of y is adversarially corrupted, each bit of x can still be decoded *with high probability* by probing k, randomly selected, locations in y. (See Section 4 for a more precise definition.) Katz and Trevisan [23] have shown an intimate relation between such codes and information-theoretic PIR. In particular, any information-theoretic PIR protocol can be

---

[*] Some preliminary results of the current work appeared in [31].

[†] CS Dept., Ben-Gurion University. E-mail: beimel@cs.bgu.ac.il.

[‡] CS Dept., Technion. E-mail: yuvali@cs.technion.ac.il. Work done while at Princeton University, AT&T Labs – Research, and DIMACS.

[§] CS Dept., Technion. E-mail: eyalk@cs.technion.ac.il. This research was supported by the fund for the promotion of research at the Technion.

[¶] Accenture, Paris France. E-mail: jean_francois_r@hotmail.com. Work done in part while at McGill University's School of CS. Partly funded by a NSERC PGS-A scholarship.

converted into an LDC of related efficiency. The best previously known upper bound on the length of a k-query *binary* LDC was $m(n) = 2^{O(n^{1/(k-1)})}$. This bound was obtained from PIR protocols with a single answer bit per server.

Our results. We improve over the previous upper bounds for information-theoretic PIR and LDC. Our main contribution is a k-server PIR protocol whose communication complexity is $O(n^{\frac{c \log\log k}{k \log k}})$ for some constant c. (More specifically, our analysis shows that $c = 2$ can be used for every $k \geq 3$.) This protocol can be transformed in a *generic* way [18, 23] into a k-query binary LDC of length $\exp(n^{\frac{c' \log\log k}{k \log k}})$. However, we also provide a direct construction which is significantly better for small values of k. Our protocol is recursive and its analysis is obtained via the solution of a certain recurrence. As mentioned, the most interesting values of k are small ones. Hence, for several such values, we present in Figure 1 an analysis of the communication complexity where the exponent is determined exactly. The results in this figure show that our bounds are better than the previous ones for values which are as small as $k = 3$ for the case of PIR and $k = 4$ for the case of LDC.

	communication of k-server PIR		length of k-query binary LDC	
k	previous	new	previous	new
2	$O(n^{1/3})$	-	$2^{O(n)}$	-
3	$O(n^{1/5})$	$O(n^{1/5.25})$	$2^{O(n^{1/2})}$	-
4	$O(n^{1/7})$	$O(n^{1/7.87})$	$2^{O(n^{1/3})}$	$2^{O(n^{3/10})}$
5	$O(n^{1/9})$	$O(n^{1/10.83})$	$2^{O(n^{1/4})}$	$2^{O(n^{1/5})}$
6	$O(n^{1/11})$	$O(n^{1/13.78})$	$2^{O(n^{1/5})}$	$2^{O(n^{1/7})}$

Figure 1. Upper bounds for small values of k.

Techniques. Our construction borrows some ideas from previous work on PIR. These include the idea of representing the database using polynomials (as in [11, 2] and especially [5]), the notion of "blocks" from [20], and the idea of recursively retrieving bits from the servers' answers (instead of sending the whole answers) as a way to reduce communication. Recursion was used previously in PIR protocols [1, 10, 25]; however, our recursion is somewhat more sophisticated. Assume that we have a PIR protocol $\mathcal{P}$ with the following three properties:

- The queries are short, however, the answers are long.
- The user only needs few bits from each answer.[1]
- There is an overlap between the answers that different servers send to the user. More precisely, each answer consists of several sub-answers and each sub-answer is known to several servers.

This protocol leads to a recursive protocol $\mathcal{P}'$ as follows: The user sends its queries as in $\mathcal{P}$, and each server computes its answer. However, the servers do not send their long answers to the user; instead the user and each subset of servers that hold a common sub-answer execute a PIR protocol in which the user retrieves the bits it needs from this sub-answer. The difficulty of constructing an appropriate protocol $\mathcal{P}$, to be used in the recursion, is in the somewhat contradicting goals of the above description. On one hand, we want the number of sub-answers and their size to be as small as possible. On the other hand, we want the "replication" (i.e., the overlap between sub-answers) to be as large as possible. Organizing the answers appropriately into sub-answers with good parameters according to the paradigm suggested above does not seem to be straightforward. Most of the technical work in this paper shows, in a sense, how to construct a protocol $\mathcal{P}$ with such properties.

Related work. Several extensions of the basic PIR model were studied. These include extensions to t-private protocols, in which the user is protected against collusions of up to t servers [11, 20, 5]; extensions which protect the servers holding the database (in addition to the user), termed symmetric PIR (SPIR) [17, 29]; and other extensions [30, 16, 13, 6, 9, 7]. PIR was also studied in a *computational* setting where privacy should only hold against computationally bounded servers; computational PIR was studied in both the multi-server setting [10] and a single server setting [25, 27, 32, 8, 26, 14, 24]. In contrast to information-theoretic PIR, computational PIR protocols with sublinear communication exist even in the single-server case (under standard cryptographic assumptions).

From a practical point of view, single-server PIR protocols are preferable to multi-server ones for obvious reasons: they avoid the need to maintain replicated copies of the database or to compromise the user's privacy against several colluding servers. Moreover, single-server protocols from the literature obtain better asymptotic communication complexity than information-theoretic protocols with a constant number of servers. However, for typical real-life parameters the known single-server protocols are less efficient than known multi-server (even 2-server) protocols. Furthermore, single-server protocols have some *inherent* limitations which can only be avoided in a multi-server setting. For instance, it is impossible for a (sublinear-communication) single-server PIR protocol to have very short queries (say, $O(\log n)$ bits long) sent from the user to the server, or very short answers (say, one bit long) sent in return. These two extreme types of protocols, which can be realized in the information-theoretic setting, have various applications [13, 6]. Finally, the close relation between information-theoretic PIR and locally decodable codes [23] further motivates the study of PIR in this setting.

No strong general lower bounds on PIR are known. Mann [27] obtained a constant-factor improvement over the trivial $\log_2 n$ bound, for any constant k. In the 2-server case, much stronger lower bounds can be shown under the restriction that the user reconstructs x_i by computing the exclusive-or of a *constant* number of bits sent by the

---

[1] The user cannot reveal to the servers which bits it needs since this information might disclose the index i it is interested in.

servers [19]. Other lower bounds for restricted PIR protocols are given by Itoh [22]. Lower bounds for locally decodable codes appear in [23, 12]. These results still leave an exponential gap between known upper bounds and lower bounds in the general (unrestricted) case.

Organization. In Section 2 we provide some necessary definitions. In Section 3 we describe a concrete PIR protocol with the promised complexity. In Section 4 we describe PIR protocols with short answers and their applications to locally-decodable codes. In Section 5 we describe an abstract framework which generalizes the concrete protocol. Finally, in Appendix A we give a high-level description of our protocol explaining why it saves communication.

2. Preliminaries

We use in our protocols multivariate polynomials. By default, all polynomials are over GF(2). Variables of such polynomials are denoted with capital letters, e.g., Z_h; assignments to these variables are in small letters, e.g., z_h. The term *degree-d* polynomial refers to a polynomial whose *total* degree is *at most d*. For an integer t, $[t]$ denotes the set $\{1, \ldots, t\}$. Finally, $\log r$ should be read as $\log_2 r$.

A k-server PIR protocol involves k servers $\mathcal{S}_1, \ldots, \mathcal{S}_k$, each holding the same n-bit string x (the database), and a user $\mathcal{U}$ who knows n and wants to retrieve some bit x_i, $i \in [n]$, without revealing i. We restrict our attention to *one-round*, 1-private, information-theoretic PIR protocols.

Definition 2.1 (PIR) *A PIR protocol is a triplet of algorithms $\mathcal{P} = (\mathcal{Q}, \mathcal{A}, \mathcal{C})$. At the beginning of the protocol, the user $\mathcal{U}$ invokes $\mathcal{Q}(k, n, i)$ to pick a (randomized) k-tuple of* queries $(q_1, q_2, \ldots, q_k)$*, along with an auxiliary information string* aux. *It sends each server $\mathcal{S}_j$ the query q_j and keeps* aux *for a later use. Each server $\mathcal{S}_j$ responds with an* answer $a_j = \mathcal{A}(k, j, x, q_j)$. *(We can assume without loss of generality that the servers are deterministic; hence, each answer is a* function *of the query and the database.) Finally, $\mathcal{U}$ computes its output by applying the reconstruction algorithm $\mathcal{C}(k, n, a_1, \ldots, a_k, \text{aux})$. We view the number of servers k as constant, and require all algorithms to be efficient in the data length n. A protocol $\mathcal{P}$ restricted to a fixed k will be referred to as a k-*server *protocol. A protocol as above should satisfy the following requirements:*

Correctness. *For any $k, n, x \in \{0,1\}^n$ and $i \in [n]$, the user outputs the correct value of x_i with probability 1 (where the probability is over the randomness of $\mathcal{Q}$).*

Privacy. *Each server learns no information about i. Formally, for any k, n, $i_1, i_2 \in [n]$, and server $j \in [k]$, the distributions $\mathcal{Q}_j(k, n, i_1)$ and $\mathcal{Q}_j(k, n, i_2)$ are identical, where $\mathcal{Q}_j$ denotes the j-th output of $\mathcal{Q}$.*

The *communication complexity* of a PIR protocol $\mathcal{P}$, denoted $\mathrm{C}_{\mathcal{P}}(n, k)$, is a function of k and n measuring the total number of bits communicated between the user $\mathcal{U}$ and the k servers maximized over all choices of $x \in \{0,1\}^n$, $i \in [n]$, and random inputs. The *query length* of $\mathcal{P}$, denoted $\mathrm{Q}_{\mathcal{P}}(n, k)$, is the maximal number of bits sent from $\mathcal{U}$ to any single server, and the *answer length*, denoted $\mathrm{A}_{\mathcal{P}}(n, k)$, is the maximal number of answer bits sent by any server.

Finally, we say that a PIR protocol $\mathcal{P}$ is *linear* (over GF(2)) if $\mathcal{U}$ recovers x_i by taking the exclusive-or of some subset of the answer bits determined by aux. All protocols constructed in this work are linear.

3. A Concrete Protocol

We present below a PIR protocol that achieves the desired upper bound. It builds upon several ideas that are borrowed from [20, 5]; however, for self containment, the presentation assumes no knowledge of these works.

The protocol is based on representing the n-bit database x by a multivariate polynomial $P_x(Z_1, \ldots, Z_m)$ over GF(2). The polynomial P_x will be defined in Section 3.1; for the time being we only describe its important features. In this representation we carefully control two parameters: the degree d and the number of variables m which is chosen such that $m = \Theta(n^{1/d})$.[2] The polynomial P_x represents x in the following sense: with every $i \in [n]$ we associate a distinct assignment (also referred to as "encoding") $E(i) \in \{0,1\}^m$; the polynomial P_x satisfies

$$\forall i \in [n], \quad P_x(E(i)) = x_i \tag{1}$$

(we do not care about the value $P_x(\vec{z})$ for assignments $\vec{z}$ which are not of the form $E(i)$, for some i). Each coefficient of P_x is determined by x and hence each server $\mathcal{S}_j$ can compute it. The user $\mathcal{U}$, on the other hand, does not know x. It has an index i, pointing to the bit from x it is interested in, and it can compute $E(i)$. Hence, the PIR problem is reduced to the problem of evaluating $P_x(E(i))$ while keeping $E(i)$ secret from each server. To this end, $\mathcal{U}$ chooses at random $\vec{y}_1, \ldots, \vec{y}_k \in \{0,1\}^m$ subject to the constraint

$$E(i) \;=\; \sum_{j=1}^{k} \vec{y}_j \tag{2}$$

and sends to each server $\mathcal{S}_j$ all the $\vec{y}$'s except $\vec{y}_j$. Note that since each $k-1$ of the $\vec{y}$'s are uniformly and independently distributed, a single server can learn no information about i. The user's goal is to evaluate $P_x(\sum_{j=1}^{k} \vec{y}_j) = P_x(E(i)) = x_i$. (Each $\vec{y}_j$ consists of m values $y_{j,h}$, for $j \in [k], h \in [m]$.) Equivalently, we can think of each variable Z_h of P_x as the sum of k variables: $Z_h = \sum_{j=1}^{k} Y_{j,h}$. The value $P_x(E(i))$ is obtained by assigning the value $y_{j,h}$ to each variable $Y_{j,h}$. Let Q_x be the polynomial obtained by viewing P_x as a polynomial in the variables $\{Y_{j,h}\}_{j \in [k]}^{h \in [m]}$. This is a degree-$d$ polynomial in mk variables. Consider

---

[2] To be more precise $m = \Theta(dn^{1/d})$. As we treat d and k as constants, we will ignore constants depending on d and k throughout the paper.

a monomial M of this polynomial; M depends on at most d variables. Since each variable is known to $k-1$ of the servers (i.e., only one server does not know it) then there exists a server that is missing at most $\lfloor d/k \rfloor$ of the variables of M; we assign M to this server (if there is more than one server with this property we pick one arbitrarily).

Suppose, for the moment, that $d = k-1$. In this case $\lfloor d/k \rfloor = 0$; i.e., the server to which M is assigned knows the assignment $y_{j,h}$ for all the variables $Y_{j,h}$ in M and can actually compute the value of M. The PIR protocol therefore consists of $\mathcal{U}$ picking values $\vec{y}_j$ as in (2), sending each $\vec{y}_j$ to all servers except $\mathcal{S}_j$, and each server answering $\mathcal{U}$ with the sum (in GF(2)) of all monomials assigned to it. By the above discussion, the sum of these answers equals x_i. The communication complexity of this protocol is $O(m) = O(n^{1/d}) = O(n^{1/(k-1)})$ bits. More specifically, we have shown:

Claim 3.1 ([13, 20, 5]) *There exists a k-server PIR protocol with query length $O(n^{1/(k-1)})$ and answer length* 1.

Next, consider the case $d = 2k-1$. Again, assign each monomial M to a server that misses $\lfloor d/k \rfloor = 1$ of the variables of M. Each server $\mathcal{S}_j$ can therefore substitute the values for all but (at most) one of the variables of each monomial M assigned to it. After substituting these values, the sum of the monomials assigned to $\mathcal{S}_j$ can be expressed as a degree-1 polynomial $P_j(Y_{j,1}, \ldots, Y_{j,m})$, whose variables $Y_{j,1}, \ldots, Y_{j,m}$ are precisely those whose values are unknown to $\mathcal{S}_j$. Note, however, that if the user could learn all polynomials P_j, then by substituting the correct values $y_{j,h}$ for all their variables and summing up the values of the k polynomials it will get

$$\sum_{j=1}^{k} P_j(y_{j,1}, \ldots, y_{j,m}) = P_x(\sum_{j=1}^{k} \vec{y}_j) = P_x(E(i)) = x_i.$$

The PIR protocol starts as before, but this time $\mathcal{S}_j$ sends the $m+1$ *coefficients* (a single bit each) of the degree-1 polynomial P_j. The communication complexity of this protocol is therefore still $O(m) = O(n^{1/d})$ which, by the choice of d, equals $O(n^{1/(2k-1)})$ bits. To summarize the discussion so far, we have shown how to obtain a PIR protocol with the best known complexity prior to the current work:

Claim 3.2 ([1, 20, 5]) *There exists a k-server PIR protocol with communication complexity $O(n^{1/(2k-1)})$.*

Next, it is useful to note that just further increasing the value of d is of no use. While in such a case each polynomial P_j as above indeed has less variables, it is of a higher degree (i.e., $\lfloor d/k \rfloor$); hence the list of coefficients is no shorter than what we get by choosing $d = 2k-1$, as above. We emphasize that the amount of information that the user needs about each polynomial P_j is very small (i.e., the value $P_j(\vec{y}_j)$); however, it cannot reveal $\vec{y}_j$ to $\mathcal{S}_j$ as this will expose the value $E(i)$ and hence i.

The contribution of this paper starts with the following idea to go around the above difficulty. Suppose that we can choose the parameters in a way that each polynomial which the user wants to evaluate is known to several servers. Rather than asking the servers to send the coefficients, the user can *recursively* retrieve the value of this polynomial by using a PIR protocol among the servers sharing the polynomial. Assume that we can express

$$P_x(\sum_{j=1}^{k} \vec{y}_j) = \sum_{V \subseteq [k]} P_V(z_V), \tag{3}$$

where each polynomial P_V is known to every server in the set V, and z_V is an assignment known to the user. The polynomials P_V may have higher degree (than the degree-1 polynomials that we have in the $d = 2k-1$ case), yet we hope to avoid sending the list of coefficients by the servers and instead let the user get each value $P_V(z_V)$ by applying, recursively, a PIR protocol with the servers of V. Note that the number of servers in each such V is smaller than k, which is a disadvantage compared to the number of servers that we have, say, for retrieving the value $P_x(E(i))$. One may hope, however, that P_V will have a low degree and a small number of variables.

We do not know how to *directly* construct such polynomials (with good parameters); instead, we use a recursive PIR in the following way. Each P_V will be such that, knowing z_V, it suffices to get a small number of its coefficients in order to obtain $P_V(z_V)$; the identity of these coefficients may reveal information about i and hence each coefficient will be retrieved using a PIR protocol. Before showing how to construct the polynomials P_V, we specify their properties that imply the complexity of the overall solution. We use two parameters λ and k'. The parameter k' is a lower bound on the size of the sets V we will use (except for the sets V of size 1 which will also be used). Each polynomial P_V consists of monomials M in which each of the $|V|$ servers misses at most λ of the variables. Therefore, all but at most $\lambda|V|$ variables of M are known to all servers in V and so after substituting the values known to all servers in V the degree of P_V will be at most $\lambda|V|$. The number of variables on which P_V depends is m (as in P_x) and, in fact, the user will seek the value of $P_V(E(i))$. The motivation for doing so has to do with our choice of encoding $E(\cdot)$; in our encoding most bits of $E(i)$ are set to 0 and thus most monomials of $P_V(E(i))$ are set to 0 (and so their coefficients are of no interest). The user therefore needs to retrieve only the coefficients of those monomials where all variables are set to 1. The number of these coefficients is small (at most 2^d). Since P_V has $O(m^{\lambda|V|})$ coefficients, the user can retrieve the value $P_x(E(i))$ using 2^d executions of a $|V|$-server PIR protocol with database of size $O(m^{\lambda|V|}) = O(n^{\lambda|V|/d})$.

Assuming we can indeed find such polynomials P_V with the above properties and that we have a PIR protocol $\mathcal{P}$ with communication complexity $\mathrm{C}_{\mathcal{P}}(n,k)$, we get a protocol $\mathcal{P}'$ with communication complexity

$$\mathrm{C}_{\mathcal{P}'}(n,k) \leq O_k\left(n^{1/d} + \sum_{\ell=k'}^{k} \binom{k}{\ell} \mathrm{C}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell)\right). \tag{4}$$

(The notation O_k indicates that the constant depends on k.) An appropriate choice of parameters will ensure, in particular, that $\lambda k/d < 1$ and so $\mathcal{P}$ is applied to shorter strings.

3.1. Constructing the Polynomials

To complete the description of the protocol, we provide specific implementations for the encoding $E(\cdot)$, the polynomials P_x and P_V, and the values z_V that together satisfy Equation (3). More precisely, we describe an encoding E of length $m = \Theta(n^{1/d})$, polynomials P_x, P_V as above and polynomials P_j of degree 1, such that for every i

$$\begin{aligned} x_i &= P_x(E(i)) \\ &= \sum_{V\subseteq[k],|V|\geq k'} P_V(E(i)) \;+\; \sum_{j=1}^{k} P_j(\vec{y}_j). \quad (5) \end{aligned}$$

(For each P_V we use $z_V = E(i)$ and for each P_j we use $\vec{y}_j$.) Furthermore, each polynomial P_V can be computed from P_x and $\{\vec{y}_j\}_{j\notin V}$ (this holds for $V = \{j\}$ as well).

The construction of E and P_x proceeds as follows. Let $E(1), \ldots, E(n)$ be n distinct binary vectors (strings) of length m and weight d. Such vectors exist if $\binom{m}{d} \geq n$, i.e., $m = \Theta(n^{1/d})$ variables are sufficient.[3] Define

$$P_x(Z_1, \ldots, Z_m) \stackrel{\text{def}}{=} \sum_{i=1}^{n} x_i \prod_{E(i)_\ell = 1} Z_\ell,$$

($E(i)_\ell$ is the ℓth bit of $E(i)$). Since each $E(i)$ is of weight d then the degree of P_x is d. Each assignment $E(i)$ to the variables $Z_1, \ldots, Z_m$ satisfies exactly one monomial in P_x (whose coefficient is x_i); thus, $P_x(E(i)) = x_i$.

Consider the polynomial $Q_x(\{Y_{j,h}\}_{j\in[k]}^{h\in[m]}) \stackrel{\text{def}}{=} P_x(\sum_{j=1}^{k} Y_{j,1}, \ldots, \sum_{j=1}^{k} Y_{j,m})$. This is a polynomial with mk variables and degree d. That is, Q_x is obtained from P_x by setting $Z_h = \sum_{j=1}^{k} Y_{j,h}$. For every monomial M, consider the set $V(M) \subseteq [k]$ that contains all servers that appear at most λ times in M. The first attempt to define P_V is by assigning all monomials with $V(M) = V$ to V and obtaining P_V by substituting $\{\vec{y}_j\}_{j\notin V}$ in these monomials. The resulting polynomial P_V has small degree, namely $\lambda|V|$, and has few variables, namely $m|V|$. However, in this case P_V should be evaluated at the point $\langle \vec{y}_j \rangle_{j\in V}$; this point may be of arbitrary weight and hence we do not know how to apply the recursion.

Before constructing the polynomials P_V, we choose the "correct" value of d, i.e., the maximal value that guarantees that for any monomial M of degree at most d either there is a server $\mathcal{S}_j$ that knows all but at most one variable in the monomial (in this case this monomial contributes to the corresponding P_j), or the set $V(M)$ has size at least k'.

Claim 3.3 *Let $\lambda, k' \leq k$ be parameters, and $d \leq (\lambda+1)k - (\lambda-1)k' + (\lambda-2)$. Furthermore, let M be a monomial of degree at most d in the variables $Y_{j,h}$, where $j \in [k]$ and $h \in [m]$. Then, either there is a server that misses at most one variable, or $|V(M)| \geq k'$.*

Proof: Assume that the claim does not hold. That is, every $\mathcal{S}_j$ misses at least two variables in the monomial M (i.e., at least two of the variables $\{Y_{j,h}\}_{h\in[m]}$ appear in M) and at most $k'-1$ servers miss at most λ variables of M (equivalently at least $k-(k'-1)$ servers miss at least $\lambda+1$ variables). Therefore, the number of variables in the monomial is at least $(k-(k'-1))(\lambda+1) + (k'-1)\cdot 2 \geq d+1$, contradicting the choice of d. ☐

Following is the main technical claim underlying our construction.[4]

Claim 3.4 *Let k, λ, k' and d be as in Claim 3.3, $P_x(Z_1, \ldots, Z_m)$ a polynomial of degree at most d, and $\vec{z} = \sum_{j=1}^{k} \vec{y}_j$. Then, there are polynomials $P_V(Z_1, \ldots, Z_m)$ for every $V \subseteq [k]$, where $|V| \geq k'$, and polynomials $P_j(Z_1, \ldots, Z_m)$ for $j \in [k]$, such that*

1. *Each polynomial P_V is of degree $\lambda|V|$ and can be computed from P_x and $\{\vec{y}_j\}_{j\notin V}$;*
2. *Each polynomial P_j is of degree 1 and can be computed from P_x and $\{\vec{y}_{j'}\}_{j'\neq j}$;*
3. *$P_x(\vec{z}) = \sum_{V\subseteq[k],|V|\geq k'} P_V(\vec{z}) + \sum_{j\in[k]} P_j(\vec{y}_j)$.*

Proof: It suffices to prove the claim for polynomials that consist of a single monomial (and then summing over all monomials in P_x). Hence, consider, w.l.o.g., $P(Z_1, \ldots, Z_m) = Z_1 Z_2 \cdots Z_d$ (instead of P_x). Let

$$\begin{aligned} Q(\{Y_{j,h}\}_{j\in[k]}^{h\in[m]}) &\stackrel{\text{def}}{=} P(\sum_{j=1}^{k} Y_{j,1}, \ldots, \sum_{j=1}^{k} Y_{j,d}) \\ &= (\sum_{j=1}^{k} Y_{j,1})(\sum_{j=1}^{k} Y_{j,2}) \cdots (\sum_{j=1}^{k} Y_{j,d}). \end{aligned}$$

The polynomial Q has k^d monomials of degree d each. Each monomial M is of the form $Y_{j_1,1} \cdots Y_{j_d,d}$. Denote

$$T(M) \stackrel{\text{def}}{=} \prod_{j_q \in V(M)} Z_q \prod_{j_q \notin V(M)} Y_{j_q,q}.$$

Note that $T(M)$ is expressed in terms of both Z's and Y's and it should be interpreted as follows: (a) when $T(M)$ is part of a polynomial in $\{Y_{j,h}\}$ then it should be interpreted as the polynomial obtained by substituting each Z_q with $\sum_{j=1}^{k} Y_{j,q}$. (b) when $T(M)$ is part of a polynomial in $\{Z_j\}$ then it should be interpreted as a single monomial of degree $\leq \lambda|V(M)|$ whose coefficient $\prod_{j_q\notin V(M)} Y_{j_q,q}$ is known to all servers in $V(M)$.

---

[3] Alternatively, one can use an encoding with strings of weight *at most* d; however, for small values of d (e.g., constant) this yields only minor efficiency improvements. Another option is to use the encoding of [11, 20] (called **E3** in [5]). This can improve the efficiency of our protocols by a factor of 2^d in some cases, but will further complicate the presentation.

[4] In the full version of this paper we will present an alternative proof of this claim based on the inclusion-exclusion principle.

Below is an algorithm to construct the polynomials P_V as in the claim. The algorithm maintains a polynomial Q' (initially $Q' = Q$) with all the monomials that we need to take care of (monomials may be repeatedly taken out and inserted into Q'). Denote by $\delta(M)$ the number of variables $Y_{j_q,q}$ in M with $j_q \in V(M)$.

```
1. Set $Q' = Q$ and for all $V$ set $P_V(Z_1, \ldots, Z_m) = 0$.
2. Find a set $V$ such that $V = V(M)$ for some monomial $M$ (currently) in $Q'$, and such that $V$ is of the largest size among all sets $V(M)$ for such $M$'s.
   If $|V| < k'$ then STOP.
3. While there is a monomial $M$ s.t. $V(M) = V$:
     • among these $M$'s pick $M$ that maximizes $\delta(M)$;
     • add $T(M)$ to $P_V$, set $Q' = Q' - T(M)$.
4. GOTO 2.
```

To argue the correctness of the algorithm, view any $T(M)$ added to P_V in Step 3 as a sum of $k^{\delta(M)}$ monomials in the $\{Y_{j,h}\}$ variables. Clearly, the P_V's are of the desired degree and their sum evaluated at $\vec{z}$ is $P(\vec{z}) - Q'(\vec{z})$ (for Q' that the algorithm halts with). We need to argue that the algorithm halts.

We say that two monomials $M_1 = Y_{j_1,1} \cdots Y_{j_d,d}$ and $M_2 = Y_{j'_1,1} \cdots Y_{j'_d,d}$ are equivalent (with respect to $W \subseteq [k]$) if (a) $V(M_1) = V(M_2) = W$; and (b) for each index $q \in [d]$ either j_q, j'_q are both in W or $j_q = j'_q$ (i.e., the same variable $Y_{j_q,q}$ appears in both monomials). Note that this is an equivalence relation and denote $M_1 \equiv M_2$.

Let M_1 be a monomial of $T(M)$ and note the following observations about its structure. **(i)** $V(M_1) \subseteq V(M)$ (any server not in $V(M) = V$, i.e., one that appears more than λ times in M, appears at least the same number of times in M_1, by definition of $T(M)$). **(ii)** $\delta(M_1) \leq \delta(M)$ (any variable that does not contribute to $\delta(M)$ does not contribute to $\delta(M_1)$ either). **(iii)** if $V(M_1) = V(M)$ and $\delta(M_1) = \delta(M)$ then, by definition, $M_1 \equiv M$. **(iv)** if $M_2 \equiv M_1$ (with respect to some W) then M_2 must also be in $T(M)$ (by (i), $W \subseteq V(M)$). It follows that if $M_1 \equiv M_2$ then, at any time during the algorithm, they are either both in Q' or both are not in Q'. This is because it is true at the beginning (all the k^d monomials of the form $Y_{j_1,1} \cdots Y_{j_d,d}$ are in $Q' = Q$) and whenever Q' is modified by subtracting $T(M)$ for some monomial M then if, say, M_1 is in $T(M)$ then so is M_2 and vice versa.

Using the above observations, we now argue the halting of the algorithm. The idea is that, even though new monomials may be added to Q' when subtracting $T(M)$ in Step 3, such monomials M' either have smaller $V(M')$ or smaller $\delta(M')$ and hence we always make progress. This is because in each application of Step 3 we pick M that maximizes $V(M)$ and among those one that maximizes $\delta(M)$. The monomials added to Q', when subtracting $T(M)$ satisfy either (a) $V(M') \subset V(M)$ – in which case it will be dealt in future application of Step 2 if it will still exist; or (b) $V(M') = V(M)$ but $\delta(M') < \delta(M)$ – in which case it will be dealt in future application of Step 3 if it will still exist; or (c) $V(M') = V(M)$ and $\delta(M') = \delta(M)$ – in which case if M is in Q' so is M' and when subtracting $T(M)$ we eliminate both. Once we finish the construction of P_V we do not return to this V anymore.

When the algorithm halts there is no M in Q' for which $|V(M)| \geq k'$. By Claim 3.3 and the choice of d, in such a case for each of these monomials there is at least one $\mathcal{S}_j$ missing at most one variable (from $\{Y_{j,h}\}_{h \in [m]}$). Each such M is now added to a corresponding P_j. Hence, the claim follows. □

Note that, in spite of the recursion, the resulting protocol can still be implemented as a one-round PIR protocol. The indices of the bits that the user needs in the recursive calls are determined by $E(i)$. Thus, in the first round, when the user sends its query for i, it can also send its queries for the indices that it needs from every set V. The properties of this protocol are summarized by the next theorem.

Theorem 3.5 *Suppose there is a PIR protocol $\mathcal{P}$ with communication complexity $\mathrm{C}_{\mathcal{P}}(n,k)$. Let d, λ, k' be positive integers (which may depend on k) such that $k' < k$ and $d \leq (\lambda+1)k - (\lambda-1)k' + (\lambda-2)$. Then there is a PIR protocol $\mathcal{P}'$ with communication complexity*

$$\mathrm{C}_{\mathcal{P}'}(n,k) = O_k\left(n^{1/d} + \sum_{\ell=k'}^{k} \binom{k}{\ell} \mathrm{C}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell)\right). \quad (6)$$

Remark 3.6 For all PIR protocols from the literature (including the current work) $\mathrm{C}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell) = O(\mathrm{C}_{\mathcal{P}}(n^{\lambda k'/d}, k'))$ for $\ell \geq k'$. Thus, the sum in Eq. (6) is dominated by its first term.

Example 3.7 *We demonstrate how to get the first two improved protocols from Figure 1. In the 3-server case, we set $\lambda = k' = 2$ and $d = 7$. By using a 2-server protocol with complexity $O(n^{1/3})$ (see Claim 3.2) the communication complexity is $O(n^{1/7} + (n^{4/7})^{1/3}) = O(n^{4/21})$. In the 4-server case we can rely on the above protocol and use $\lambda = 2, k' = 3$, and $d = 9$ to obtain communication complexity of $O(n^{1/9} + (n^{6/9})^{4/21}) = O(n^{8/63})$.*

3.2. Analysis of the Protocol

The above discussion yields a recursive complexity analysis. Below we get a specific bound by choosing appropriate parameters. This analysis is somewhat crude and is mainly intended for large values of k (which are still viewed as constants). For small values of k, one should be more careful; e.g., the results specified in Figure 1 are derived by choosing optimal values for the parameters.

Lemma 3.8 *For every positive integer i there is a PIR protocol $\mathcal{P}_i$ such that $\mathrm{C}_{\mathcal{P}_i}(n,k) = O_k(n^{2/(ik)})$ for every constant $k \geq (i-1)!$.*

Proof: By induction. The first nontrivial case is $i = 3$, in which the lemma follows from Claim 3.2. For the induction step, suppose that $i \geq 3$ and there exists a PIR protocol $\mathcal{P}_i$ such that $\mathrm{C}_{\mathcal{P}_i}(n,k) = O_k(n^{2/(ik)})$ for every $k \geq (i-1)!$. Using Theorem 3.5 we construct $\mathcal{P}_{i+1}$ such that $\mathrm{C}_{\mathcal{P}_{i+1}}(n,k) = O_k(n^{2/((i+1)k)})$ for every $k \geq i!$. Let $k' = \left\lfloor \frac{k}{i} \right\rfloor \geq (i-1)!$, $\lambda = \left\lceil \frac{i}{2} \right\rceil$ (in particular, $\lambda \geq 2$), and $d = (\lambda+1)k - (\lambda-1)k' \leq (\lambda+1)k - (\lambda-1)k' + (\lambda-2)$, as required to apply Theorem 3.5. Note that $\mathrm{C}_{\mathcal{P}_i}(n^{\lambda\ell/d}, \ell) = O_k(n^{2\lambda/(id)})$ for every $\ell \geq k'$. Thus, to complete the proof it suffices to prove that $2\lambda/(id) \leq 2/((i+1)k)$:

$$\begin{aligned} \frac{\lambda}{id} &\leq \frac{\lambda}{i((\lambda+1)k - (\lambda-1)k/i)} = \frac{\lambda}{k(\lambda i + (i - \lambda + 1))} \\ &\leq \frac{\lambda}{k(\lambda i + \lambda)} = \frac{1}{k(i+1)}, \end{aligned}$$

where the first inequality is by the choice of d and k' and the last inequality is by the choice of λ. □

Corollary 3.9 *There exists a PIR protocol $\mathcal{P}$ such that $\mathrm{C}_{\mathcal{P}}(n,k) = O_k(n^{2\log\log k/(k \log k)})$ for every $k \geq 3$.*

Proof: For $k \geq 3$ the protocol $\mathcal{P}$ executes the protocol $\mathcal{P}_i$ promised by Lemma 3.8, where $i = \lceil \log k / \log\log k \rceil$. Then, $(i-1)! \leq (i-1)^{i-1} \leq \log k^{\log k/\log\log k} = k$, and $\mathrm{C}_{\mathcal{P}}(n,k) = \mathrm{C}_{\mathcal{P}_i}(n,k) = O_k(n^{2\log\log k/(k\log k)})$. □

In the full version of the paper we will show that the above analysis is essentially optimal. This is not to say that there are no other protocols that can do better; it only says that within the freedom that our protocol has in choosing the parameters λ and k', the above choice, that achieves complexity of $n^{O(\log\log k/(k\log k))}$, is essentially the best.

4. PIR Protocols with Short Answers and Locally Decodable Codes

In this section we obtain efficient PIR protocols in which the answer of each server consists of a single bit; we refer to such protocols as "binary protocols." We start by noting that the protocols from the previous section can be transformed in a *generic* way to binary protocols with related complexity. Specifically, given any *linear* k-server PIR protocol in which the total communication with each server is $c(n)$, it is possible to construct a $2k$-server binary protocol with query length $c(n)$ [18].[5] Thus, there is a binary k-server PIR protocol with query length $n^{O(\log\log k/(k\log k))}$. Below is a direct construction which improves the constants in the above exponent. In particular, while the generic transformation improves over the best previously known *binary* protocols only for $k \geq 6$, the following direct construction gives the first improvement when $k = 4$.

Theorem 4.1 *Suppose there is a PIR protocol $\mathcal{P}$ with query length $\mathrm{Q}_{\mathcal{P}}(n,k)$ and answer length $\mathrm{A}_{\mathcal{P}}(n,k)$. Let d, λ, k' be positive integers (which may depend on k) such that $k' < k$ and $d \leq (\lambda+1)k - \lambda k' + (\lambda - 1)$. Then there is a PIR protocol $\mathcal{P}'$ with query length*

$$\mathrm{Q}_{\mathcal{P}'}(n,k) = O_k\left(n^{1/d} + \sum_{\ell=k'}^{k} \mathrm{Q}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell)\right) \tag{7}$$

and answer length $O_k(\sum_{\ell=k'}^{k} \mathrm{A}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell))$. Furthermore, if $\mathcal{P}$ is binary and linear then there is a binary linear $\mathcal{P}'$ with query length as above.

For lack of space, we only sketch the proof. The condition $d \leq (\lambda+1)k - \lambda k' + \lambda - 1$ guarantees that in every monomial M, either $V(M) \geq k'$ or there is a server that knows *all* variables in the monomial. This allows to obtain a stronger variant of Claim 3.4 where the polynomials P_j are of degree 0. Thus, it suffices for each server to send one bit to the user in addition to the answers in the recursive calls. In the linear binary case, it suffices to send to the user the exclusive-or of the additional bit and the answer bits from the recursive calls.

Example 4.2 *We illustrate the use of Theorem 4.1 for obtaining the first two improvements over previous protocols. As a basis we can use the case $k = 3$, for which the best known binary protocol has query length $O(n^{1/2})$ (see Claim 3.1). For $k = 4$ we let $\lambda = 1, k' = 3$, and $d = 5$, and get a 4-server binary protocol with query length $O(n^{1/5} + (n^{3/5})^{1/2}) = O(n^{3/10})$. In the 5-server case we let $\lambda = 1, k' = 4, d = 6$ and get a binary protocol with query length $O(n^{1/5})$.*

Application to locally decodable codes. A binary code $C : \{0,1\}^n \rightarrow \{0,1\}^m$ is said to be (k, δ, ρ)*-locally decodable* if every bit x_i of x can be decoded from $y = C(x)$ with success probability$\geq 1/2 + \rho$ by reading k (randomly chosen) bits of y, even if up to a δ-fraction of y was adversarially corrupted. A *k-query binary locally-decodable code* is a family of $(k, \delta(n), \rho(n))$-locally decodable codes $C_n : \{0,1\}^n \rightarrow \{0,1\}^{m(n)}$ such that $\delta(n), \rho(n)$ are lower bounded by some positive constant, independent of n.

Given a binary k-server PIR protocol with query length $c(n)$, it is possible to construct a k-query binary locally-decodable code of length $O(k2^{c(n)})$ [23]. If the query to each server is uniformly distributed over its domain, as is the case for the protocols we obtain, the encoding of a string $x \in \{0,1\}^n$ can be defined as the concatenation of the servers' answers to all possible queries, i.e., $\mathcal{A}(k, j, x, q)$ for all $j \in [k]$ and all queries q. Thus, we have:

Corollary 4.3 *There is a constant c such that for every k there is a k-query binary locally-decodable code of length $O(2^{n^{c\log\log k/(k\log k)}})$.*

---

[5] It is easy to verify that the protocols constructed in the previous section are in fact linear. The transformation to a binary protocol may proceed as follows. The user generates queries $q_1, \ldots, q_k$ as in the original protocol and sends each q_j to both $\mathcal{S}_j$ and $\mathcal{S}_{k+j}$; each of them generates the corresponding answer a_j but does not send it back. Instead, the user privately retrieves the exclusive-or of the bits that it needs from a_j using the following procedure. The user needs to learn the inner product (in GF(2)) of a_j with some vector b_j it knows. To this end it sends a random vector r_j of length $|a_j|$ to $\mathcal{S}_j$ and $r_{j+k} = r_j - b_j$ to $\mathcal{S}_{j+k}$. Each of the two servers replies with the inner product of a_j and the received vector, allowing the user to recover $\langle b_j, a_j \rangle$ by adding (in GF(2)) the two received bits.

5. An Abstract Framework

In this section we describe an abstract framework which generalizes the specific protocol from Section 3 and captures the scope of the underlying technique. We start by formulating a general linear algebra problem that lies in the core of Claim 3.4.

Fix some field F (where $F = \mathrm{GF}(2)$ by default), and consider the linear space of polynomials over F in the dk variables $Y_{j,h}$, where $j \in [k], h \in [d]$. In fact, we will only be interested in the subspace spanned by the k^d monomials of the form $Y_{j_1,1}Y_{j_2,2} \cdots Y_{j_d,d}$, for $j_1, \ldots, j_d \in [k]$.

We use Z_h as an abbreviation for the sum $\sum_{j=1}^{k} Y_{j,h}$. Following the terminology from [20], a *block* is a polynomial which can be expressed as a product of sums Z_h and variables $Y_{j,h}$. For example, every $T(M)$ from the proof of Claim 3.4 is a block. Note that if b is a block, then its representation as such a product must be unique, and must involve each index $h \in [d]$ exactly once. With each block b, let $\delta(b)$ denote the number of sums Z_h in b, and $V(b)$ denote the set of indices $j \in [k]$ which *do not* occur (in variables $Y_{j,h}$) in the representation of b. For instance, if $d = 5, k = 5$, and $b = Z_1 Y_{5,2} Y_{5,3} Z_4 Y_{2,5}$ then $\delta(b) = 2$ (since Z_1, Z_4 are the sums occurring in b) and $V(b) = \{1, 3, 4\}$. In the context of the PIR application, the block b will be used by the servers in $V(b)$ to construct a polynomial of degree $\delta(b)$ in the variables $Z_1, \ldots, Z_m$.

We now define a key property of sets of blocks, generalizing a corresponding notion from [20].

Definition 5.1 *Let $\mathcal{B}$ be a set of blocks with parameters d, k. We say that $\mathcal{B}$ is* spanning *if the blocks in $\mathcal{B}$, viewed as polynomials in the dk variables $Y_{j,h}$, span the block $Z_1 Z_2 \cdots Z_d$. Denote by $\Delta_{d,k}$ the class of spanning block sets $\mathcal{B}$ with parameters d, k.*

Example 5.2 *Let $d = k = 2$. The block set $\{Y_{1,1}Y_{1,2}, Z_1 Y_{2,2}, Y_{2,1}Y_{1,2}\}$ is spanning. On the other hand, it is easy to verify that block set $\mathcal{B} = \{Y_{1,1}Z_2, Z_1 Y_{2,2}, Y_{2,1}Y_{1,2}\}$ is not spanning, i.e. $\mathcal{B}$ does not span the block $Z_1 Z_2$, although each of the four monomials in $Z_1 Z_2$ is involved in some block in $\mathcal{B}$.*

Example 5.3 *If $d = 2k - 1$, then the set of all blocks b such that $\delta(b) \leq 1$ and $|V(b)| \geq 1$ is spanning. This set of blocks is used in [20] to construct a PIR protocol with communication complexity $O(k^3 n^{1/(2k-1)})$.*

The following set of blocks corresponds to Claim 3.4.

Example 5.4 *Let k, λ, k', d, be as in Claim 3.3. Then the following set $\mathcal{B}$ is spanning. $\mathcal{B}$ includes: (1) every block b such that $\delta(b) \leq 1$ and $|V(b)| \geq 1$; (2) every block b such that $k' \leq |V(b)|$, $\delta(b) \leq \lambda|V(b)|$, and each $j \in [k] \setminus V(b)$ occurs in b more than λ times.*

Generalizing Theorem 3.5, it is possible to use any spanning block set $\mathcal{B}$ for reducing k-server PIR to instances of PIR with a smaller number of servers. This is formalized by the following theorem.

Theorem 5.5 *Suppose there is a PIR protocol $\mathcal{P}$ with communication complexity $\mathrm{C}_{\mathcal{P}}(n, k)$. Then, for any $d = d(k)$ and a spanning block set $\mathcal{B} = \mathcal{B}(d, k) \in \Delta_{d,k}$ there is a PIR protocol $\mathcal{P}'$ with communication complexity*

$$\mathrm{C}_{\mathcal{P}'}(n, k) = O_k \left(n^{1/d} + \sum_{b \in \mathcal{B}} \mathrm{C}_{\mathcal{P}}(n^{\delta(b)/d}, |V(b)|) \right).$$

We note that Theorem 5.5 gives, in a sense, a closed-form expression for the best communication complexity attainable by the current approach. The main difficulty, however, is in finding appropriate choices for the spanning block set $\mathcal{B}$ that optimize the overall complexity. We do not know if, using Theorem 5.5, it is possible to construct a protocol whose complexity is significantly better than the protocol presented in Corollary 3.9. In the full version of the paper we will show that, using Theorem 5.5, one cannot construct a protocol whose complexity is better than $O(n^{1/k^2})$.

Acknowledgments. We thank Enav Weinreb for valuable comments on earlier drafts of this paper. We also thank Don Coppersmith for a helpful discussion.

References

[1] A. Ambainis. Upper bound on the communication complexity of private information retrieval. In *24th ICALP*, *LNCS* 1256, pp. 401–407, 1997.

[2] L. Babai, P. G. Kimmel, and S. V. Lokam. Simultaneous messages vs. communication. In *STACS '95*, *LNCS* 999, pp. 361–372, 1995.

[3] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *STACS '90*, vol. 415 of *LNCS*, pp. 37–48, 1990.

[4] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *J. of Cryptology*, 10(1):17–36, 1997.

[5] A. Beimel and Y. Ishai. Information-theoretic private information retrieval: A unified construction. In *28th ICALP*, vol. 2076 of *LNCS*, pp. 912–926, 2001.

[6] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. In *CRYPTO 2000*, vol. 1880 of *LNCS*, pp. 56–74, 2000.

[7] A. Beimel and Y. Stahl. Robust information-theoretic private information retrieval. *3rd Conf. on Security in Commun. Networks*, 2002.

[8] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT '99*, vol. 1592 of *LNCS*, pp. 402–414, 1999.

[9] R. Canetti, Y. Ishai, R. Kumar, M. K. Reiter, R. Rubinfeld, and R. N. Wright. Selective private function evaluation with applications to private statistics. In *20th PODC*, pp. 293 – 304, 2001.

[10] B. Chor and N. Gilboa. Computationally private information retrieval. In *29th STOC*, pp. 304–313, 1997.

[11] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *J. of the ACM*, 45:965–981, 1998.

[12] A. Deshpande, R. Jain, T Kavita, V. Lokam, and J. Radhakrishnan. Better lower bounds for locally decodable codes. In *16th CCC*, pp. 184–193, 2002.

[13] G. Di-Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for private information retrieval. *J. of Cryptology*, 14(1):37–74, 2001.

[14] G. Di-Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *EUROCRYPT 2000*, vol. 1807 of *LNCS*, pp. 122–138, 2000.

[15] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. In *28th ICALP*, vol. 2076 of *LNCS*, pp. 927–938, 2001.

[16] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In *RANDOM '98*, vol. 1518 of *LNCS*, pp. 200–217, 1998.

[17] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *JCSS*, 60(3):592–629, 2000.

[18] O. Goldreich. Personal communication, 2000.

[19] O. Goldreich, H. Karloff, L. Schulman, and L. Trevisan. Lower bounds for linear locally decodable codes and PIR. In *16th CCC*, pp. 175 – 183, 2002.

[20] Y. Ishai and E. Kushilevitz. Improved upper bounds on information theoretic private information retrieval. *31st STOC*, pp. 79 – 88, 1999.

[21] T. Itoh. Efficient private information retrieval. *IEICE Trans. Fund. of Electronics, Commun. and Comp. Sci.*, E82-A(1):11–20, 1999.

[22] T. Itoh. On lower bounds for the communication complexity of private information retrieval. *IEICE Trans. Fund. of Electronics, Commun. and Comp. Sci.*, E84-A(1):157–164, 2001.

[23] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *32nd STOC*, pp. 80–86, 2000.

[24] A. Kiayias and M. Yung. Secure games with polynomial expressions. In *28th ICALP*, vol. 2076 of *LNCS*, pp. 939–950, 2001.

[25] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *38th FOCS*, pp. 364–373, 1997.

[26] E. Kushilevitz and R. Ostrovsky. One-way trapdoor permutations are sufficient for single-database computationally-private information retrieval. In *EUROCRYPT 2000*, *LNCS* 1807, pp. 104–121, 2000.

[27] E. Mann. Private access to distributed information. Master's thesis, Technion, 1998.

[28] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *33th STOC*, 2001.

[29] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *31st STOC*, pp. 245–254, 1999.

[30] R. Ostrovsky and V. Shoup. Private information storage. In *29th STOC*, pp. 294–303, 1997.

[31] J. F. Raymond. Private information retrieval: Improved upper bound, extension and applications. Master's thesis, McGill University, 2000.

[32] J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *ASIACRYPT '98*, vol. 1514 of *LNCS*, pp. 357–371, 1998.

A. A High-Level View: Replication vs. Degree

In this appendix we try to explain where the improvement over previous protocols comes from. The recursive formulation of our protocol, as well as the various technicalities involved in its implementation and analysis, make it somewhat difficult to trace the actual source for improvement. The following overview of our approach attempts to give a fairly accurate intuition as to *how* and *why* it works, while ignoring some details or difficulties that are dealt with in the technical sections.

A key technique in previous solutions, as well as in ours, is an *arithmetization* of the PIR problem. Using a polynomial representation for the database x, the user's goal is reduced to evaluating a multivariate polynomial $P(\vec{Z})$, known to all k servers, on some point $\vec{z}$ determined by its retrieval index i.[6] This task should be achieved while hiding $\vec{z}$ from each server. There are three parameters associated with the above problem: (1) the *number of variables* in P, denoted by m; (2) the *degree* of P, denoted by d; and (3) the *replication* amount k, i.e., the number of servers to which the polynomial P is known. The first two parameters determine the *description size* of P, which is $\Theta(m^d)$ when d is constant. Using an appropriate polynomial representation, the database size n is roughly equal to the description size of P, i.e. $n = \Theta(m^d)$.

A first observation is that if we manage to reduce the degree of P by a factor of c (without changing the number of variables m by much), the new description size will be roughly the *c-th root* of the original one. The new polynomial can be communicated to the user using only $O(n^{1/c})$ communication bits. We therefore try to reduce the degree of P, possibly updating the evaluation point $\vec{z}$, without revealing $\vec{z}$ to any server. In what follows we first describe the previous approach for achieving this goal, following [2, 5], and then explain where we depart from this approach.

A degree reduction as above is achieved as follows. The user picks random points $\vec{y}_1, \ldots, \vec{y}_k \in F^m$ subject to the condition $\vec{y}_1 + \ldots + \vec{y}_k = \vec{z}$, and sends each server $\mathcal{S}_j$ a query consisting of all k points *except* $\vec{y}_j$. The intuition for this step is that it provides maximal redundancy subject to the requirement of hiding $\vec{z}$. Note that the cost of this step is dominated by $m = O(n^{1/d})$. Thus, for the total communication complexity to be small, the initial representation degree d must be large.

The next step is to express the desired value $P(\vec{z})$ as the value of the polynomial $Q(\vec{Y}_1, \ldots, \vec{Y}_k) \stackrel{\text{def}}{=} P(\vec{Y}_1 + \ldots + \vec{Y}_k)$ at the point $(\vec{y}_1, \ldots, \vec{y}_k)$. The advantage of switching to this new representation is that the value assigned to each of its variables is known to *almost all* servers (in contrast to the original polynomial $P(\vec{Z})$, whose values $\vec{z}$ are completely unknown to the servers and should remain so).

To make use of this advantage, we write Q as a sum of monomials. Every such monomial is a product of d variables. Each variable is missed by exactly one out of the k servers; hence, for the d variables involved in a given monomial there must be *at least one* server which misses *at most* d/k values of these variables. We now assign each monomial to one of the servers corresponding to this monomial, and let each server substitute specific values for all of the variables it knows in each of the monomials assigned to it. After this step, each server holds a polynomial P_j in its *unknown* variables $\vec{Y}_j$, such that the degree of P_j is at most d/k and $\sum_{j=1}^{k} P_j(\vec{y}_j) = P(\vec{z})$. Thus, we can complete the protocol by letting each server j send to the user a *descrip-*

---

[6] Here and in the following all polynomials are assumed to be over a finite field, which is taken to be GF(2) by default.

tion of its lower degree polynomial P_j (e.g., using a list of its coefficients), which allows the user to compute the desired value $P(\vec{z})$.

We now take a more quantitative look at the type of savings obtained by the above degree reduction technique. As discussed above, reducing the representation degree by a factor of k induces a $1/k$-th power reduction in its size. By picking a "high" degree d, the queries sent to each server will be short, and the answers will be of length $O(n^{1/k})$. At a first glance, this seems to be the end of the road. However, a crucial (and easy to overlook) observation is that the above degree analysis involves *integers*. Thus, the reduced degree is actually guaranteed to be bounded by $\lfloor d/k \rfloor$. While such integer truncation operations are typically viewed as a nuisance, in this case they turn out to make make a big difference. In a sense, in an "integer-less world" a PIR protocol with $O(n^{1/k})$ communication is the best we would have.

How far can the advantage of truncation be pushed? Two useful examples are the following. First, assume that $d = k - 1$. In this case, we get the most evident benefit: $\lfloor d/k \rfloor = 0$, instead of $1 - 1/k$ in the fractional case, implying that that each polynomial P_j will have degree-0 (i.e., be a constant) and therefore can be described by one bit. However, a disadvantage of this choice of parameters is that d is rather small, and therefore the length of the queries will be rather large ($O(n^{1/d}) = O(n^{1/(k-1)})$). Still, a useful feature of of the corresponding protocol is that it requires only one answer bit from each server. Indeed, the latter protocol was prior to this work essentially the best protocol of this type. A second useful choice of parameters is $d = 2k - 1$. In this case the answers are longer than before: since $\lfloor d/k \rfloor = 1$ the degree is reduced by a factor of $d = 2k-1$, and consequently the description length of P_j is $O(n^{1/d}) = O(n^{1/(2k-1)})$. However, since the queries now are also shorter, namely of length $O(n^{1/(2k-1)})$, we get a protocol of a smaller total communication complexity. The above protocol was the best known protocol (in terms of the total communication complexity) prior to this work.

In light of the above surprising effect of integer truncation on the complexity of PIR, it is natural to ask whether the savings can be pushed even further. We start with the following observation. The degree reduction process we used may be thought of as a way for *trading replication for degree*: We started with a polynomial P of degree d which is replicated among k servers, and ended up with polynomials P_j of degree $\lfloor d/k \rfloor$, each known to only *one* server. Thus, we have given away all of the original replication, and in return obtained the biggest possible gain in the degree. However, it is not clear a-priori that this greedy approach is optimal. An alternative approach that comes to mind is to apply several *partial* degree reduction steps, hoping to benefit multiple times from the integer truncation effect.

To this end, we generalize the above degree reduction procedure as follows. Suppose that we are willing to reduce the replication from k to k' (rather than 1). Then, we may assign each monomial to some set V of k' servers which *jointly* miss the least number of variables from this monomial. This allows us to write the desired value $P(\vec{z})$ as the sum of values $Q_V(\vec{y})$, where each Q_V is a polynomial known to a set V of at least k' servers. Note that the maximal degree of Q_V increases as k' grows, and in any case is no more than $\lfloor dk'/k \rfloor$.

We return to the previous question: can we gain by reducing the degree (along with the replication) in multiple steps? Intuitively, there is no advantage in applying the above "local" degree reduction process in multiple steps, as the final representation could have been directly attained in one step. The additional key idea that makes such a multi-step process useful is to use additional interaction with the user for adjusting the degree between each two reduction steps. In such a *degree conversion* step, both the number of variables and the degree are changed, but the description size remains the same. For instance, suppose that some set of k' servers holds a degree-3 polynomial Q' in m variables, and the user holds a point $\vec{z}$ whose additive shares $\vec{y}_1, \ldots, \vec{y}_{k'}$ are replicated among the servers as above. Moreover, suppose that it is possible for the servers to locally compute a degree-2 polynomial Q in $O(m^{3/2})$ variables and for the user to locally compute a point $\vec{z}\,'$, such that $Q(\vec{z}) = Q'(\vec{z}\,')$. (Note that such a conversion is presumably plausible, since $(m^{3/2})^2 = m$, and so we have not decreased the description size.) Then, by re-sharing the point $\vec{z}\,'$ among the servers, the user can adjust the degree to 2 without reducing the amount of replication or increasing the description size. Such a degree conversion procedure would allow to obtain additional savings by interleaving reduction steps with degree conversion steps.

We illustrate this by a 4-server example. Suppose that $d = 5$. Dispensing with all the replication in one step (i.e., by letting $k' = 1$), reduces the degree to $\lfloor 5/4 \rfloor = 1$. Instead, we let $k' = 3$. This brings the degree down to $\lfloor 5 \cdot 3/4 \rfloor = 3$, since for any monomial there is a set of 3 servers which jointly miss at most 3 variables from this monomial. Now, we adjust the degree to 2. This increases the number of variables to $O(m^{3/2}) = O((n^{1/5})^{3/2}) = O(n^{3/10})$, and requires the user to send additional queries of comparable size. Finally, we can apply the reduction step again to reduce the replication from 3 to 1. This brings the degree down to 0, and allows the servers to communicate $P(\vec{z})$ to the user by sending a single bit each. Thus, we obtain a protocol with query length $O(n^{3/10})$ and answer length 1 – improving the protocol with query length $O(n^{1/(k-1)})$ described above.

We do not wheter the above degree conversion problem can be solved in general.[7] Instead, we get around this problem by relying on a specific *promise* on the value of the point $\vec{z}$ held by the user. The abstract linear algebra problem that underlies our solution is described in Section 5. The recursive invocations of PIR in our protocol achieve, in effect, the degree-conversions which result in the efficiency improvement.

---

[7] Specifically, it is open if for *every* $d' < d$ it is possibvle to convert m-variavte degre-d polynomials to m'-variavte degre-d' polynomials where $m' = O(m^{d/d'})$.

Breaking the $O(n^{1/(2k-1)})$ Barrier for Information-Theoretic Private Information Retrieval[*]

Amos Beimel[†] Yuval Ishai[‡] Eyal Kushilevitz [§] Jean-François Raymond[¶]

Abstract

Private Information Retrieval (PIR) protocols allow a user to retrieve a data item from a database while hiding the identity of the item being retrieved. Specifically, in information-theoretic, *k-*server *PIR protocols the database is replicated among k servers, and each server learns nothing about the item the user retrieves. The cost of such protocols is measured by the* communication complexity *of retrieving one out of n bits of data. For any fixed k, the complexity of the best protocols prior to our work was $O(n^{\frac{1}{2k-1}})$ (Ambainis, 1997). Since then several methods were developed in an attempt to beat this bound, but all these methods yielded the same asymptotic bound.*

In this work, this barrier is finally broken and the complexity of information-theoretic k-server PIR is improved to $n^{O(\frac{\log\log k}{k\log k})}$. The new PIR protocols can also be used to construct k-query binary locally decodable codes *of length $\exp(n^{O(\frac{\log\log k}{k\log k})})$, compared to $\exp(n^{\frac{1}{k-1}})$ in previous constructions. The improvements presented in this paper apply even for small values of k: the PIR protocols are more efficient than previous ones for every $k \geq 3$, and the locally decodable codes are shorter for every $k \geq 4$.*

1. Introduction

A Private Information Retrieval (PIR) protocol allows a user to retrieve a data item of its choice from a database while preventing the server storing the database from gaining information about the identity of this item. This problem was introduced by Chor, Goldreich, Kushilevitz, and Sudan [11] and since then has attracted a considerable amount of attention (see below). In formalizing the problem, it is convenient to model the database by an n-bit string x where the user, holding some *retrieval index* i, wishes to learn the i-th data bit x_i. A trivial solution to the PIR problem is to send the entire database x to the user. However, while being perfectly private, the *communication complexity* of this solution may be prohibitively large. Indeed, the most significant goal of PIR-related research has been to minimize the communication complexity of PIR protocols. Unfortunately, if the server is not allowed to gain *any* information about the identity of the retrieved bit, then the linear communication complexity of the trivial solution is optimal [11]. To overcome this problem, Chor et al. [11] suggested that the user accesses k replicated copies of the database stored at different servers, requiring that each individual server gets absolutely no information about i. PIR in this setting is referred to as *information-theoretic* PIR.

The best known complexity for information-theoretic PIR protocols prior to the current work is $O(n^{1/(2k-1)})$. This was first obtained for $k = 2$ in [11] and generalized to any fixed value of k by Ambainis [1]. This upper bound remained the best known until this work, in spite of various attempts to improve it [20, 21, 5]. While these attempts resulted in finding new, very different, PIR protocols, they all ended up with the same $O(n^{1/(2k-1)})$ bound. (The constants, which depend on k, were significantly improved; this is in addition to asymptotic improvements for some extensions of the basic problem.) Note that the number of servers, k, is usually considered to be "small" and, in particular, independent of the length of the database, n; for larger values of k, there is a construction ([11] and implicitly in [3, 4]) that gives an $O(\log n)$-server protocol with $O(\log^2 n \log\log n)$ communication bits or alternatively (with different parameters) an $O(\log n/\log\log n)$-server protocol with $\text{poly}(\log n)$ communication.

Other than the interest in PIR protocols for their own sake, they also found various applications (see, e.g., [15, 28, 9]). One particularly interesting application of PIR is for the construction of so-called *locally decodable codes*. A k-query Locally Decodable Code (LDC) allows to encode a database $x \in \{0,1\}^n$ into a string y, such that even if a large fraction of y is adversarially corrupted, each bit of x can still be decoded *with high probability* by probing k, randomly selected, locations in y. (See Section 4 for a more precise definition.) Katz and Trevisan [23] have shown an intimate relation between such codes and information-theoretic PIR. In particular, any information-theoretic PIR protocol can be

---

[*] Some preliminary results of the current work appeared in [31].

[†] CS Dept., Ben-Gurion University. E-mail: beimel@cs.bgu.ac.il.

[‡] CS Dept., Technion. E-mail: yuvali@cs.technion.ac.il. Work done while at Princeton University, AT&T Labs – Research, and DIMACS.

[§] CS Dept., Technion. E-mail: eyalk@cs.technion.ac.il. This research was supported by the fund for the promotion of research at the Technion.

[¶] Accenture, Paris France. E-mail: jean_francois_r@hotmail.com. Work done in part while at McGill University's School of CS. Partly funded by a NSERC PGS-A scholarship.

converted into an LDC of related efficiency. The best previously known upper bound on the length of a k-query *binary* LDC was $m(n) = 2^{O(n^{1/(k-1)})}$. This bound was obtained from PIR protocols with a single answer bit per server.

Our results. We improve over the previous upper bounds for information-theoretic PIR and LDC. Our main contribution is a k-server PIR protocol whose communication complexity is $O(n^{\frac{c \log\log k}{k \log k}})$ for some constant c. (More specifically, our analysis shows that $c = 2$ can be used for every $k \geq 3$.) This protocol can be transformed in a *generic* way [18, 23] into a k-query binary LDC of length $\exp(n^{\frac{c' \log\log k}{k \log k}})$. However, we also provide a direct construction which is significantly better for small values of k. Our protocol is recursive and its analysis is obtained via the solution of a certain recurrence. As mentioned, the most interesting values of k are small ones. Hence, for several such values, we present in Figure 1 an analysis of the communication complexity where the exponent is determined exactly. The results in this figure show that our bounds are better than the previous ones for values which are as small as $k = 3$ for the case of PIR and $k = 4$ for the case of LDC.

	communication of k-server PIR		length of k-query binary LDC	
k	previous	new	previous	new
2	$O(n^{1/3})$	-	$2^{O(n)}$	-
3	$O(n^{1/5})$	$O(n^{1/5.25})$	$2^{O(n^{1/2})}$	-
4	$O(n^{1/7})$	$O(n^{1/7.87})$	$2^{O(n^{1/3})}$	$2^{O(n^{3/10})}$
5	$O(n^{1/9})$	$O(n^{1/10.83})$	$2^{O(n^{1/4})}$	$2^{O(n^{1/5})}$
6	$O(n^{1/11})$	$O(n^{1/13.78})$	$2^{O(n^{1/5})}$	$2^{O(n^{1/7})}$

Figure 1. Upper bounds for small values of k.

Techniques. Our construction borrows some ideas from previous work on PIR. These include the idea of representing the database using polynomials (as in [11, 2] and especially [5]), the notion of "blocks" from [20], and the idea of recursively retrieving bits from the servers' answers (instead of sending the whole answers) as a way to reduce communication. Recursion was used previously in PIR protocols [1, 10, 25]; however, our recursion is somewhat more sophisticated. Assume that we have a PIR protocol $\mathcal{P}$ with the following three properties:

- The queries are short, however, the answers are long.
- The user only needs few bits from each answer.[1]
- There is an overlap between the answers that different servers send to the user. More precisely, each answer consists of several sub-answers and each sub-answer is known to several servers.

This protocol leads to a recursive protocol $\mathcal{P}'$ as follows: The user sends its queries as in $\mathcal{P}$, and each server computes its answer. However, the servers do not send their long answers to the user; instead the user and each subset of servers that hold a common sub-answer execute a PIR protocol in which the user retrieves the bits it needs from this sub-answer. The difficulty of constructing an appropriate protocol $\mathcal{P}$, to be used in the recursion, is in the somewhat contradicting goals of the above description. On one hand, we want the number of sub-answers and their size to be as small as possible. On the other hand, we want the "replication" (i.e., the overlap between sub-answers) to be as large as possible. Organizing the answers appropriately into sub-answers with good parameters according to the paradigm suggested above does not seem to be straightforward. Most of the technical work in this paper shows, in a sense, how to construct a protocol $\mathcal{P}$ with such properties.

Related work. Several extensions of the basic PIR model were studied. These include extensions to t-private protocols, in which the user is protected against collusions of up to t servers [11, 20, 5]; extensions which protect the servers holding the database (in addition to the user), termed symmetric PIR (SPIR) [17, 29]; and other extensions [30, 16, 13, 6, 9, 7]. PIR was also studied in a *computational* setting where privacy should only hold against computationally bounded servers; computational PIR was studied in both the multi-server setting [10] and a single server setting [25, 27, 32, 8, 26, 14, 24]. In contrast to information-theoretic PIR, computational PIR protocols with sublinear communication exist even in the single-server case (under standard cryptographic assumptions).

From a practical point of view, single-server PIR protocols are preferable to multi-server ones for obvious reasons: they avoid the need to maintain replicated copies of the database or to compromise the user's privacy against several colluding servers. Moreover, single-server protocols from the literature obtain better asymptotic communication complexity than information-theoretic protocols with a constant number of servers. However, for typical real-life parameters the known single-server protocols are less efficient than known multi-server (even 2-server) protocols. Furthermore, single-server protocols have some *inherent* limitations which can only be avoided in a multi-server setting. For instance, it is impossible for a (sublinear-communication) single-server PIR protocol to have very short queries (say, $O(\log n)$ bits long) sent from the user to the server, or very short answers (say, one bit long) sent in return. These two extreme types of protocols, which can be realized in the information-theoretic setting, have various applications [13, 6]. Finally, the close relation between information-theoretic PIR and locally decodable codes [23] further motivates the study of PIR in this setting.

No strong general lower bounds on PIR are known. Mann [27] obtained a constant-factor improvement over the trivial $\log_2 n$ bound, for any constant k. In the 2-server case, much stronger lower bounds can be shown under the restriction that the user reconstructs x_i by computing the exclusive-or of a *constant* number of bits sent by the

---

[1] The user cannot reveal to the servers which bits it needs since this information might disclose the index i it is interested in.

servers [19]. Other lower bounds for restricted PIR protocols are given by Itoh [22]. Lower bounds for locally decodable codes appear in [23, 12]. These results still leave an exponential gap between known upper bounds and lower bounds in the general (unrestricted) case.

Organization. In Section 2 we provide some necessary definitions. In Section 3 we describe a concrete PIR protocol with the promised complexity. In Section 4 we describe PIR protocols with short answers and their applications to locally-decodable codes. In Section 5 we describe an abstract framework which generalizes the concrete protocol. Finally, in Appendix A we give a high-level description of our protocol explaining why it saves communication.

2. Preliminaries

We use in our protocols multivariate polynomials. By default, all polynomials are over GF(2). Variables of such polynomials are denoted with capital letters, e.g., Z_h; assignments to these variables are in small letters, e.g., z_h. The term *degree-d* polynomial refers to a polynomial whose *total* degree is *at most* d. For an integer t, $[t]$ denotes the set $\{1, \ldots, t\}$. Finally, $\log r$ should be read as $\log_2 r$.

A k-server PIR protocol involves k servers $\mathcal{S}_1, \ldots, \mathcal{S}_k$, each holding the same n-bit string x (the database), and a user $\mathcal{U}$ who knows n and wants to retrieve some bit x_i, $i \in [n]$, without revealing i. We restrict our attention to *one-round*, 1-private, information-theoretic PIR protocols.

Definition 2.1 (PIR) *A PIR protocol is a triplet of algorithms $\mathcal{P} = (\mathcal{Q}, \mathcal{A}, \mathcal{C})$. At the beginning of the protocol, the user $\mathcal{U}$ invokes $\mathcal{Q}(k, n, i)$ to pick a (randomized) k-tuple of* queries $(q_1, q_2, \ldots, q_k)$, *along with an auxiliary information string* aux. *It sends each server $\mathcal{S}_j$ the query q_j and keeps* aux *for a later use. Each server $\mathcal{S}_j$ responds with an* answer $a_j = \mathcal{A}(k, j, x, q_j)$. *(We can assume without loss of generality that the servers are deterministic; hence, each answer is a* function *of the query and the database.) Finally, $\mathcal{U}$ computes its output by applying the reconstruction algorithm* $\mathcal{C}(k, n, a_1, \ldots, a_k, \text{aux})$. *We view the number of servers k as constant, and require all algorithms to be efficient in the data length n. A protocol $\mathcal{P}$ restricted to a fixed k will be referred to as a k-*server *protocol. A protocol as above should satisfy the following requirements:*

Correctness. *For any $k, n, x \in \{0,1\}^n$ and $i \in [n]$, the user outputs the correct value of x_i with probability 1 (where the probability is over the randomness of $\mathcal{Q}$).*

Privacy. *Each server learns no information about i. Formally, for any $k, n, i_1, i_2 \in [n]$, and server $j \in [k]$, the distributions $\mathcal{Q}_j(k, n, i_1)$ and $\mathcal{Q}_j(k, n, i_2)$ are identical, where $\mathcal{Q}_j$ denotes the j-th output of $\mathcal{Q}$.*

The *communication complexity* of a PIR protocol $\mathcal{P}$, denoted $\mathrm{C}_{\mathcal{P}}(n, k)$, is a function of k and n measuring the total number of bits communicated between the user $\mathcal{U}$ and the k servers maximized over all choices of $x \in \{0,1\}^n$, $i \in [n]$, and random inputs. The *query length* of $\mathcal{P}$, denoted $\mathrm{Q}_{\mathcal{P}}(n, k)$, is the maximal number of bits sent from $\mathcal{U}$ to any single server, and the *answer length*, denoted $\mathrm{A}_{\mathcal{P}}(n, k)$, is the maximal number of answer bits sent by any server.

Finally, we say that a PIR protocol $\mathcal{P}$ is *linear* (over GF(2)) if $\mathcal{U}$ recovers x_i by taking the exclusive-or of some subset of the answer bits determined by aux. All protocols constructed in this work are linear.

3. A Concrete Protocol

We present below a PIR protocol that achieves the desired upper bound. It builds upon several ideas that are borrowed from [20, 5]; however, for self containment, the presentation assumes no knowledge of these works.

The protocol is based on representing the n-bit database x by a multivariate polynomial $P_x(Z_1, \ldots, Z_m)$ over GF(2). The polynomial P_x will be defined in Section 3.1; for the time being we only describe its important features. In this representation we carefully control two parameters: the degree d and the number of variables m which is chosen such that $m = \Theta(n^{1/d})$.[2] The polynomial P_x represents x in the following sense: with every $i \in [n]$ we associate a distinct assignment (also referred to as "encoding") $E(i) \in \{0,1\}^m$; the polynomial P_x satisfies

$$\forall i \in [n], \quad P_x(E(i)) = x_i \tag{1}$$

(we do not care about the value $P_x(\vec{z})$ for assignments $\vec{z}$ which are not of the form $E(i)$, for some i). Each coefficient of P_x is determined by x and hence each server $\mathcal{S}_j$ can compute it. The user $\mathcal{U}$, on the other hand, does not know x. It has an index i, pointing to the bit from x it is interested in, and it can compute $E(i)$. Hence, the PIR problem is reduced to the problem of evaluating $P_x(E(i))$ while keeping $E(i)$ secret from each server. To this end, $\mathcal{U}$ chooses at random $\vec{y}_1, \ldots, \vec{y}_k \in \{0,1\}^m$ subject to the constraint

$$E(i) = \sum_{j=1}^{k} \vec{y}_j \tag{2}$$

and sends to each server $\mathcal{S}_j$ all the $\vec{y}$'s except $\vec{y}_j$. Note that since each $k-1$ of the $\vec{y}$'s are uniformly and independently distributed, a single server can learn no information about i. The user's goal is to evaluate $P_x(\sum_{j=1}^{k} \vec{y}_j) = P_x(E(i)) = x_i$. (Each $\vec{y}_j$ consists of m values $y_{j,h}$, for $j \in [k], h \in [m]$.) Equivalently, we can think of each variable Z_h of P_x as the sum of k variables: $Z_h = \sum_{j=1}^{k} Y_{j,h}$. The value $P_x(E(i))$ is obtained by assigning the value $y_{j,h}$ to each variable $Y_{j,h}$. Let Q_x be the polynomial obtained by viewing P_x as a polynomial in the variables $\{Y_{j,h}\}_{j\in[k]}^{h\in[m]}$. This is a degree-$d$ polynomial in mk variables. Consider

---

[2] To be more precise $m = \Theta(dn^{1/d})$. As we treat d and k as constants, we will ignore constants depending on d and k throughout the paper.

a monomial M of this polynomial; M depends on at most d variables. Since each variable is known to $k-1$ of the servers (i.e., only one server does not know it) then there exists a server that is missing at most $\lfloor d/k \rfloor$ of the variables of M; we assign M to this server (if there is more than one server with this property we pick one arbitrarily).

Suppose, for the moment, that $d = k-1$. In this case $\lfloor d/k \rfloor = 0$; i.e., the server to which M is assigned knows the assignment $y_{j,h}$ for all the variables $Y_{j,h}$ in M and can actually compute the value of M. The PIR protocol therefore consists of $\mathcal{U}$ picking values $\vec{y}_j$ as in (2), sending each $\vec{y}_j$ to all servers except $\mathcal{S}_j$, and each server answering $\mathcal{U}$ with the sum (in GF(2)) of all monomials assigned to it. By the above discussion, the sum of these answers equals x_i. The communication complexity of this protocol is $O(m) = O(n^{1/d}) = O(n^{1/(k-1)})$ bits. More specifically, we have shown:

Claim 3.1 ([13, 20, 5]) *There exists a k-server PIR protocol with query length $O(n^{1/(k-1)})$ and answer length* 1.

Next, consider the case $d = 2k-1$. Again, assign each monomial M to a server that misses $\lfloor d/k \rfloor = 1$ of the variables of M. Each server $\mathcal{S}_j$ can therefore substitute the values for all but (at most) one of the variables of each monomial M assigned to it. After substituting these values, the sum of the monomials assigned to $\mathcal{S}_j$ can be expressed as a degree-1 polynomial $P_j(Y_{j,1}, \ldots, Y_{j,m})$, whose variables $Y_{j,1}, \ldots, Y_{j,m}$ are precisely those whose values are unknown to $\mathcal{S}_j$. Note, however, that if the user could learn all polynomials P_j, then by substituting the correct values $y_{j,h}$ for all their variables and summing up the values of the k polynomials it will get

$$\sum_{j=1}^{k} P_j(y_{j,1}, \ldots, y_{j,m}) = P_x(\sum_{j=1}^{k} \vec{y}_j) = P_x(E(i)) = x_i.$$

The PIR protocol starts as before, but this time $\mathcal{S}_j$ sends the $m+1$ *coefficients* (a single bit each) of the degree-1 polynomial P_j. The communication complexity of this protocol is therefore still $O(m) = O(n^{1/d})$ which, by the choice of d, equals $O(n^{1/(2k-1)})$ bits. To summarize the discussion so far, we have shown how to obtain a PIR protocol with the best known complexity prior to the current work:

Claim 3.2 ([1, 20, 5]) *There exists a k-server PIR protocol with communication complexity $O(n^{1/(2k-1)})$.*

Next, it is useful to note that just further increasing the value of d is of no use. While in such a case each polynomial P_j as above indeed has less variables, it is of a higher degree (i.e., $\lfloor d/k \rfloor$); hence the list of coefficients is no shorter than what we get by choosing $d = 2k-1$, as above. We emphasize that the amount of information that the user needs about each polynomial P_j is very small (i.e., the value $P_j(\vec{y}_j)$); however, it cannot reveal $\vec{y}_j$ to $\mathcal{S}_j$ as this will expose the value $E(i)$ and hence i.

The contribution of this paper starts with the following idea to go around the above difficulty. Suppose that we can choose the parameters in a way that each polynomial which the user wants to evaluate is known to several servers. Rather than asking the servers to send the coefficients, the user can *recursively* retrieve the value of this polynomial by using a PIR protocol among the servers sharing the polynomial. Assume that we can express

$$P_x(\sum_{j=1}^{k} \vec{y}_j) = \sum_{V \subseteq [k]} P_V(z_V), \tag{3}$$

where each polynomial P_V is known to every server in the set V, and z_V is an assignment known to the user. The polynomials P_V may have higher degree (than the degree-1 polynomials that we have in the $d = 2k-1$ case), yet we hope to avoid sending the list of coefficients by the servers and instead let the user get each value $P_V(z_V)$ by applying, recursively, a PIR protocol with the servers of V. Note that the number of servers in each such V is smaller than k, which is a disadvantage compared to the number of servers that we have, say, for retrieving the value $P_x(E(i))$. One may hope, however, that P_V will have a low degree and a small number of variables.

We do not know how to *directly* construct such polynomials (with good parameters); instead, we use a recursive PIR in the following way. Each P_V will be such that, knowing z_V, it suffices to get a small number of its coefficients in order to obtain $P_V(z_V)$; the identity of these coefficients may reveal information about i and hence each coefficient will be retrieved using a PIR protocol. Before showing how to construct the polynomials P_V, we specify their properties that imply the complexity of the overall solution. We use two parameters λ and k'. The parameter k' is a lower bound on the size of the sets V we will use (except for the sets V of size 1 which will also be used). Each polynomial P_V consists of monomials M in which each of the $|V|$ servers misses at most λ of the variables. Therefore, all but at most $\lambda|V|$ variables of M are known to all servers in V and so after substituting the values known to all servers in V the degree of P_V will be at most $\lambda|V|$. The number of variables on which P_V depends is m (as in P_x) and, in fact, the user will seek the value of $P_V(E(i))$. The motivation for doing so has to do with our choice of encoding $E(\cdot)$; in our encoding most bits of $E(i)$ are set to 0 and thus most monomials of $P_V(E(i))$ are set to 0 (and so their coefficients are of no interest). The user therefore needs to retrieve only the coefficients of those monomials where all variables are set to 1. The number of these coefficients is small (at most 2^d). Since P_V has $O(m^{\lambda|V|})$ coefficients, the user can retrieve the value $P_x(E(i))$ using 2^d executions of a $|V|$-server PIR protocol with database of size $O(m^{\lambda|V|}) = O(n^{\lambda|V|/d})$.

Assuming we can indeed find such polynomials P_V with the above properties and that we have a PIR protocol $\mathcal{P}$ with communication complexity $\mathrm{C}_{\mathcal{P}}(n,k)$, we get a protocol $\mathcal{P}'$ with communication complexity

$$\mathrm{C}_{\mathcal{P}'}(n,k) \leq O_k\left(n^{1/d} + \sum_{\ell=k'}^{k} \binom{k}{\ell} \mathrm{C}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell)\right). \tag{4}$$

(The notation O_k indicates that the constant depends on k.) An appropriate choice of parameters will ensure, in particular, that $\lambda k/d < 1$ and so $\mathcal{P}$ is applied to shorter strings.

3.1. Constructing the Polynomials

To complete the description of the protocol, we provide specific implementations for the encoding $E(\cdot)$, the polynomials P_x and P_V, and the values z_V that together satisfy Equation (3). More precisely, we describe an encoding E of length $m = \Theta(n^{1/d})$, polynomials P_x, P_V as above and polynomials P_j of degree 1, such that for every i

$$\begin{aligned} x_i &= P_x(E(i)) \\ &= \sum_{V\subseteq[k],|V|\geq k'} P_V(E(i)) \;+\; \sum_{j=1}^{k} P_j(\vec{y}_j). \end{aligned} \quad (5)$$

(For each P_V we use $z_V = E(i)$ and for each P_j we use $\vec{y}_j$.) Furthermore, each polynomial P_V can be computed from P_x and $\{\vec{y}_j\}_{j\notin V}$ (this holds for $V = \{j\}$ as well).

The construction of E and P_x proceeds as follows. Let $E(1), \ldots, E(n)$ be n distinct binary vectors (strings) of length m and weight d. Such vectors exist if $\binom{m}{d} \geq n$, i.e., $m = \Theta(n^{1/d})$ variables are sufficient.[3] Define

$$P_x(Z_1, \ldots, Z_m) \stackrel{\text{def}}{=} \sum_{i=1}^{n} x_i \prod_{E(i)_\ell = 1} Z_\ell,$$

($E(i)_\ell$ is the ℓth bit of $E(i)$). Since each $E(i)$ is of weight d then the degree of P_x is d. Each assignment $E(i)$ to the variables $Z_1, \ldots, Z_m$ satisfies exactly one monomial in P_x (whose coefficient is x_i); thus, $P_x(E(i)) = x_i$.

Consider the polynomial $Q_x(\{Y_{j,h}\}_{j\in[k]}^{h\in[m]}) \stackrel{\text{def}}{=} P_x(\sum_{j=1}^k Y_{j,1}, \ldots, \sum_{j=1}^k Y_{j,m})$. This is a polynomial with mk variables and degree d. That is, Q_x is obtained from P_x by setting $Z_h = \sum_{j=1}^k Y_{j,h}$. For every monomial M, consider the set $V(M) \subseteq [k]$ that contains all servers that appear at most λ times in M. The first attempt to define P_V is by assigning all monomials with $V(M) = V$ to V and obtaining P_V by substituting $\{\vec{y}_j\}_{j\notin V}$ in these monomials. The resulting polynomial P_V has small degree, namely $\lambda|V|$, and has few variables, namely $m|V|$. However, in this case P_V should be evaluated at the point $\langle \vec{y}_j \rangle_{j\in V}$; this point may be of arbitrary weight and hence we do not know how to apply the recursion.

Before constructing the polynomials P_V, we choose the "correct" value of d, i.e., the maximal value that guarantees that for any monomial M of degree at most d either there is a server $\mathcal{S}_j$ that knows all but at most one variable in the monomial (in this case this monomial contributes to the corresponding P_j), or the set $V(M)$ has size at least k'.

Claim 3.3 *Let $\lambda, k' \leq k$ be parameters, and $d \leq (\lambda+1)k - (\lambda-1)k' + (\lambda-2)$. Furthermore, let M be a monomial of degree at most d in the variables $Y_{j,h}$, where $j \in [k]$ and $h \in [m]$. Then, either there is a server that misses at most one variable, or $|V(M)| \geq k'$.*

Proof: Assume that the claim does not hold. That is, every $\mathcal{S}_j$ misses at least two variables in the monomial M (i.e., at least two of the variables $\{Y_{j,h}\}_{h\in[m]}$ appear in M) and at most $k'-1$ servers miss at most λ variables of M (equivalently at least $k-(k'-1)$ servers miss at least $\lambda+1$ variables). Therefore, the number of variables in the monomial is at least $(k-(k'-1))(\lambda+1) + (k'-1)\cdot 2 \geq d+1$, contradicting the choice of d. ☐

Following is the main technical claim underlying our construction.[4]

Claim 3.4 *Let k, λ, k' and d be as in Claim 3.3, $P_x(Z_1, \ldots, Z_m)$ a polynomial of degree at most d, and $\vec{z} = \sum_{j=1}^k \vec{y}_j$. Then, there are polynomials $P_V(Z_1, \ldots, Z_m)$ for every $V \subseteq [k]$, where $|V| \geq k'$, and polynomials $P_j(Z_1, \ldots, Z_m)$ for $j \in [k]$, such that*

1. *Each polynomial P_V is of degree $\lambda|V|$ and can be computed from P_x and $\{\vec{y}_j\}_{j\notin V}$;*
2. *Each polynomial P_j is of degree 1 and can be computed from P_x and $\{\vec{y}_{j'}\}_{j'\neq j}$;*
3. *$P_x(\vec{z}) = \sum_{V\subseteq[k],|V|\geq k'} P_V(\vec{z}) + \sum_{j\in[k]} P_j(\vec{y}_j)$.*

Proof: It suffices to prove the claim for polynomials that consist of a single monomial (and then summing over all monomials in P_x). Hence, consider, w.l.o.g., $P(Z_1, \ldots, Z_m) = Z_1 Z_2 \cdots Z_d$ (instead of P_x). Let

$$\begin{aligned} Q(\{Y_{j,h}\}_{j\in[k]}^{h\in[m]}) &\stackrel{\text{def}}{=} P(\sum_{j=1}^{k} Y_{j,1}, \ldots, \sum_{j=1}^{k} Y_{j,d}) \\ &= (\sum_{j=1}^{k} Y_{j,1})(\sum_{j=1}^{k} Y_{j,2}) \cdots (\sum_{j=1}^{k} Y_{j,d}). \end{aligned}$$

The polynomial Q has k^d monomials of degree d each. Each monomial M is of the form $Y_{j_1,1} \cdots Y_{j_d,d}$. Denote

$$T(M) \stackrel{\text{def}}{=} \prod_{j_q \in V(M)} Z_q \prod_{j_q \notin V(M)} Y_{j_q,q}.$$

Note that $T(M)$ is expressed in terms of both Z's and Y's and it should be interpreted as follows: (a) when $T(M)$ is part of a polynomial in $\{Y_{j,h}\}$ then it should be interpreted as the polynomial obtained by substituting each Z_q with $\sum_{j=1}^k Y_{j,q}$. (b) when $T(M)$ is part of a polynomial in $\{Z_j\}$ then it should be interpreted as a single monomial of degree $\leq \lambda|V(M)|$ whose coefficient $\prod_{j_q\notin V(M)} Y_{j_q,q}$ is known to all servers in $V(M)$.

---

[3] Alternatively, one can use an encoding with strings of weight *at most* d; however, for small values of d (e.g., constant) this yields only minor efficiency improvements. Another option is to use the encoding of [11, 20] (called **E3** in [5]). This can improve the efficiency of our protocols by a factor of 2^d in some cases, but will further complicate the presentation.

[4] In the full version of this paper we will present an alternative proof of this claim based on the inclusion-exclusion principle.

Below is an algorithm to construct the polynomials P_V as in the claim. The algorithm maintains a polynomial Q' (initially $Q' = Q$) with all the monomials that we need to take care of (monomials may be repeatedly taken out and inserted into Q'). Denote by $\delta(M)$ the number of variables $Y_{j_q,q}$ in M with $j_q \in V(M)$.

1. Set $Q' = Q$ and for all V set $P_V(Z_1, \ldots, Z_m) = 0$.
2. Find a set V such that $V = V(M)$ for some monomial M (currently) in Q', and such that V is of the largest size among all sets $V(M)$ for such M's.
   If $|V| < k'$ then STOP.
3. While there is a monomial M s.t. $V(M) = V$:
   - among these M's pick M that maximizes $\delta(M)$;
   - add $T(M)$ to P_V, set $Q' = Q' - T(M)$.
4. GOTO 2.

To argue the correctness of the algorithm, view any $T(M)$ added to P_V in Step 3 as a sum of $k^{\delta(M)}$ monomials in the $\{Y_{j,h}\}$ variables. Clearly, the P_V's are of the desired degree and their sum evaluated at $\vec{z}$ is $P(\vec{z}) - Q'(\vec{z})$ (for Q' that the algorithm halts with). We need to argue that the algorithm halts.

We say that two monomials $M_1 = Y_{j_1,1} \cdots Y_{j_d,d}$ and $M_2 = Y_{j'_1,1} \cdots Y_{j'_d,d}$ are equivalent (with respect to $W \subseteq [k]$) if (a) $V(M_1) = V(M_2) = W$; and (b) for each index $q \in [d]$ either j_q, j'_q are both in W or $j_q = j'_q$ (i.e., the same variable $Y_{j_q,q}$ appears in both monomials). Note that this is an equivalence relation and denote $M_1 \equiv M_2$.

Let M_1 be a monomial of $T(M)$ and note the following observations about its structure. **(i)** $V(M_1) \subseteq V(M)$ (any server not in $V(M) = V$, i.e., one that appears more than λ times in M, appears at least the same number of times in M_1, by definition of $T(M)$). **(ii)** $\delta(M_1) \leq \delta(M)$ (any variable that does not contribute to $\delta(M)$ does not contribute to $\delta(M_1)$ either). **(iii)** if $V(M_1) = V(M)$ and $\delta(M_1) = \delta(M)$ then, by definition, $M_1 \equiv M$. **(iv)** if $M_2 \equiv M_1$ (with respect to some W) then M_2 must also be in $T(M)$ (by (i), $W \subseteq V(M)$). It follows that if $M_1 \equiv M_2$ then, at any time during the algorithm, they are either both in Q' or both are not in Q'. This is because it is true at the beginning (all the k^d monomials of the form $Y_{j_1,1} \cdots Y_{j_d,d}$ are in $Q' = Q$) and whenever Q' is modified by subtracting $T(M)$ for some monomial M then if, say, M_1 is in $T(M)$ then so is M_2 and vice versa.

Using the above observations, we now argue the halting of the algorithm. The idea is that, even though new monomials may be added to Q' when subtracting $T(M)$ in Step 3, such monomials M' either have smaller $V(M')$ or smaller $\delta(M')$ and hence we always make progress. This is because in each application of Step 3 we pick M that maximizes $V(M)$ and among those one that maximizes $\delta(M)$. The monomials added to Q', when subtracting $T(M)$ satisfy either (a) $V(M') \subset V(M)$ – in which case it will be dealt in future application of Step 2 if it will still exist; or (b) $V(M') = V(M)$ but $\delta(M') < \delta(M)$ – in which case it will be dealt in future application of Step 3 if it will still exist; or (c) $V(M') = V(M)$ and $\delta(M') = \delta(M)$ – in which case if M is in Q' so is M' and when subtracting $T(M)$ we eliminate both. Once we finish the construction of P_V we do not return to this V anymore.

When the algorithm halts there is no M in Q' for which $|V(M)| \geq k'$. By Claim 3.3 and the choice of d, in such a case for each of these monomials there is at least one $\mathcal{S}_j$ missing at most one variable (from $\{Y_{j,h}\}_{h \in [m]}$). Each such M is now added to a corresponding P_j. Hence, the claim follows. □

Note that, in spite of the recursion, the resulting protocol can still be implemented as a one-round PIR protocol. The indices of the bits that the user needs in the recursive calls are determined by $E(i)$. Thus, in the first round, when the user sends its query for i, it can also send its queries for the indices that it needs from every set V. The properties of this protocol are summarized by the next theorem.

Theorem 3.5 *Suppose there is a PIR protocol* $\mathcal{P}$ *with communication complexity* $\mathrm{C}_{\mathcal{P}}(n,k)$*. Let* d, λ, k' *be positive integers (which may depend on* k*) such that* $k' < k$ *and* $d \leq (\lambda+1)k - (\lambda-1)k' + (\lambda-2)$*. Then there is a PIR protocol* $\mathcal{P}'$ *with communication complexity*

$$\mathrm{C}_{\mathcal{P}'}(n,k) = O_k\left(n^{1/d} + \sum_{\ell=k'}^{k} \binom{k}{\ell} \mathrm{C}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell)\right). \quad (6)$$

Remark 3.6 For all PIR protocols from the literature (including the current work) $\mathrm{C}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell) = O(\mathrm{C}_{\mathcal{P}}(n^{\lambda k'/d}, k'))$ for $\ell \geq k'$. Thus, the sum in Eq. (6) is dominated by its first term.

Example 3.7 *We demonstrate how to get the first two improved protocols from Figure 1. In the 3-server case, we set* $\lambda = k' = 2$ *and* $d = 7$*. By using a 2-server protocol with complexity* $O(n^{1/3})$ *(see Claim 3.2) the communication complexity is* $O(n^{1/7} + (n^{4/7})^{1/3}) = O(n^{4/21})$*. In the* 4*-server case we can rely on the above protocol and use* $\lambda = 2, k' = 3$*, and* $d = 9$ *to obtain communication complexity of* $O(n^{1/9} + (n^{6/9})^{4/21}) = O(n^{8/63})$*.*

3.2. Analysis of the Protocol

The above discussion yields a recursive complexity analysis. Below we get a specific bound by choosing appropriate parameters. This analysis is somewhat crude and is mainly intended for large values of k (which are still viewed as constants). For small values of k, one should be more careful; e.g., the results specified in Figure 1 are derived by choosing optimal values for the parameters.

Lemma 3.8 *For every positive integer* i *there is a PIR protocol* $\mathcal{P}_i$ *such that* $\mathrm{C}_{\mathcal{P}_i}(n,k) = O_k(n^{2/(ik)})$ *for every constant* $k \geq (i-1)!$*.*

Proof: By induction. The first nontrivial case is $i = 3$, in which the lemma follows from Claim 3.2. For the induction step, suppose that $i \geq 3$ and there exists a PIR protocol $\mathcal{P}_i$ such that $\mathrm{C}_{\mathcal{P}_i}(n,k) = O_k(n^{2/(ik)})$ for every $k \geq (i-1)!$. Using Theorem 3.5 we construct $\mathcal{P}_{i+1}$ such that $\mathrm{C}_{\mathcal{P}_{i+1}}(n,k) = O_k(n^{2/((i+1)k)})$ for every $k \geq i!$. Let $k' = \lfloor \frac{k}{i} \rfloor \geq (i-1)!$, $\lambda = \lceil \frac{i}{2} \rceil$ (in particular, $\lambda \geq 2$), and $d = (\lambda+1)k - (\lambda-1)k' \leq (\lambda+1)k - (\lambda-1)k' + (\lambda-2)$, as required to apply Theorem 3.5. Note that $\mathrm{C}_{\mathcal{P}_i}(n^{\lambda\ell/d}, \ell) = O_k(n^{2\lambda/(id)})$ for every $\ell \geq k'$. Thus, to complete the proof it suffices to prove that $2\lambda/(id) \leq 2/((i+1)k)$:

$$\begin{aligned} \frac{\lambda}{id} &\leq \frac{\lambda}{i((\lambda+1)k - (\lambda-1)k/i)} = \frac{\lambda}{k(\lambda i + (i - \lambda + 1))} \\ &\leq \frac{\lambda}{k(\lambda i + \lambda)} = \frac{1}{k(i+1)}, \end{aligned}$$

where the first inequality is by the choice of d and k' and the last inequality is by the choice of λ. □

Corollary 3.9 *There exists a PIR protocol $\mathcal{P}$ such that $\mathrm{C}_{\mathcal{P}}(n,k) = O_k(n^{2\log\log k/(k \log k)})$ for every $k \geq 3$.*

Proof: For $k \geq 3$ the protocol $\mathcal{P}$ executes the protocol $\mathcal{P}_i$ promised by Lemma 3.8, where $i = \lceil \log k / \log\log k \rceil$. Then, $(i-1)! \leq (i-1)^{i-1} \leq \log k^{\log k/\log\log k} = k$, and $\mathrm{C}_{\mathcal{P}}(n,k) = \mathrm{C}_{\mathcal{P}_i}(n,k) = O_k(n^{2\log\log k/(k \log k)})$. □

In the full version of the paper we will show that the above analysis is essentially optimal. This is not to say that there are no other protocols that can do better; it only says that within the freedom that our protocol has in choosing the parameters λ and k', the above choice, that achieves complexity of $n^{O(\log\log k/(k \log k))}$, is essentially the best.

4. PIR Protocols with Short Answers and Locally Decodable Codes

In this section we obtain efficient PIR protocols in which the answer of each server consists of a single bit; we refer to such protocols as "binary protocols." We start by noting that the protocols from the previous section can be transformed in a *generic* way to binary protocols with related complexity. Specifically, given any *linear* k-server PIR protocol in which the total communication with each server is $c(n)$, it is possible to construct a $2k$-server binary protocol with query length $c(n)$ [18].[5] Thus, there is a binary k-server PIR protocol with query length $n^{O(\log\log k/(k \log k))}$. Below is a direct construction which improves the constants in the above exponent. In particular, while the generic transformation improves over the best previously known *binary* protocols only for $k \geq 6$, the following direct construction gives the first improvement when $k = 4$.

Theorem 4.1 *Suppose there is a PIR protocol $\mathcal{P}$ with query length $\mathrm{Q}_{\mathcal{P}}(n,k)$ and answer length $\mathrm{A}_{\mathcal{P}}(n,k)$. Let d, λ, k' be positive integers (which may depend on k) such that $k' < k$ and $d \leq (\lambda+1)k - \lambda k' + (\lambda - 1)$. Then there is a PIR protocol $\mathcal{P}'$ with query length*

$$\mathrm{Q}_{\mathcal{P}'}(n,k) = O_k\left(n^{1/d} + \sum_{\ell=k'}^{k} \mathrm{Q}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell)\right) \qquad (7)$$

and answer length $O_k(\sum_{\ell=k'}^{k} \mathrm{A}_{\mathcal{P}}(n^{\lambda\ell/d}, \ell))$. Furthermore, if $\mathcal{P}$ is binary and linear then there is a binary linear $\mathcal{P}'$ with query length as above.

For lack of space, we only sketch the proof. The condition $d \leq (\lambda+1)k - \lambda k' + \lambda - 1$ guarantees that in every monomial M, either $V(M) \geq k'$ or there is a server that knows *all* variables in the monomial. This allows to obtain a stronger variant of Claim 3.4 where the polynomials P_j are of degree 0. Thus, it suffices for each server to send one bit to the user in addition to the answers in the recursive calls. In the linear binary case, it suffices to send to the user the exclusive-or of the additional bit and the answer bits from the recursive calls.

Example 4.2 *We illustrate the use of Theorem 4.1 for obtaining the first two improvements over previous protocols. As a basis we can use the case $k = 3$, for which the best known binary protocol has query length $O(n^{1/2})$ (see Claim 3.1). For $k = 4$ we let $\lambda = 1, k' = 3$, and $d = 5$, and get a 4-server binary protocol with query length $O(n^{1/5} + (n^{3/5})^{1/2}) = O(n^{3/10})$. In the 5-server case we let $\lambda = 1, k' = 4, d = 6$ and get a binary protocol with query length $O(n^{1/5})$.*

Application to locally decodable codes. A binary code $C : \{0,1\}^n \to \{0,1\}^m$ is said to be (k, δ, ρ)*-locally decodable* if every bit x_i of x can be decoded from $y = C(x)$ with success probability$\geq 1/2 + \rho$ by reading k (randomly chosen) bits of y, even if up to a δ-fraction of y was adversarially corrupted. A *k-query binary locally-decodable code* is a family of $(k, \delta(n), \rho(n))$-locally decodable codes $C_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ such that $\delta(n), \rho(n)$ are lower bounded by some positive constant, independent of n.

Given a binary k-server PIR protocol with query length $c(n)$, it is possible to construct a k-query binary locally-decodable code of length $O(k2^{c(n)})$ [23]. If the query to each server is uniformly distributed over its domain, as is the case for the protocols we obtain, the encoding of a string $x \in \{0,1\}^n$ can be defined as the concatenation of the servers' answers to all possible queries, i.e., $\mathcal{A}(k,j,x,q)$ for all $j \in [k]$ and all queries q. Thus, we have:

Corollary 4.3 *There is a constant c such that for every k there is a k-query binary locally-decodable code of length $O(2^{n^{c\log\log k/(k \log k)}})$.*

---

[5] It is easy to verify that the protocols constructed in the previous section are in fact linear. The transformation to a binary protocol may proceed as follows. The user generates queries $q_1, \ldots, q_k$ as in the original protocol and sends each q_j to both $\mathcal{S}_j$ and $\mathcal{S}_{k+j}$; each of them generates the corresponding answer a_j but does not send it back. Instead, the user privately retrieves the exclusive-or of the bits that it needs from a_j using the following procedure. The user needs to learn the inner product (in GF(2)) of a_j with some vector b_j it knows. To this end it sends a random vector r_j of length $|a_j|$ to $\mathcal{S}_j$ and $r_{j+k} = r_j - b_j$ to $\mathcal{S}_{j+k}$. Each of the two servers replies with the inner product of a_j and the received vector, allowing the user to recover $\langle b_j, a_j \rangle$ by adding (in GF(2)) the two received bits.

5. An Abstract Framework

In this section we describe an abstract framework which generalizes the specific protocol from Section 3 and captures the scope of the underlying technique. We start by formulating a general linear algebra problem that lies in the core of Claim 3.4.

Fix some field F (where $F = \mathrm{GF}(2)$ by default), and consider the linear space of polynomials over F in the dk variables $Y_{j,h}$, where $j \in [k], h \in [d]$. In fact, we will only be interested in the subspace spanned by the k^d monomials of the form $Y_{j_1,1}Y_{j_2,2}\cdots Y_{j_d,d}$, for $j_1,\ldots,j_d \in [k]$.

We use Z_h as an abbreviation for the sum $\sum_{j=1}^{k} Y_{j,h}$. Following the terminology from [20], a *block* is a polynomial which can be expressed as a product of sums Z_h and variables $Y_{j,h}$. For example, every $T(M)$ from the proof of Claim 3.4 is a block. Note that if b is a block, then its representation as such a product must be unique, and must involve each index $h \in [d]$ exactly once. With each block b, let $\delta(b)$ denote the number of sums Z_h in b, and $V(b)$ denote the set of indices $j \in [k]$ which *do not* occur (in variables $Y_{j,h}$) in the representation of b. For instance, if $d = 5, k = 5$, and $b = Z_1 Y_{5,2} Y_{5,3} Z_4 Y_{2,5}$ then $\delta(b) = 2$ (since Z_1, Z_4 are the sums occurring in b) and $V(b) = \{1, 3, 4\}$. In the context of the PIR application, the block b will be used by the servers in $V(b)$ to construct a polynomial of degree $\delta(b)$ in the variables $Z_1,\ldots,Z_m$.

We now define a key property of sets of blocks, generalizing a corresponding notion from [20].

Definition 5.1 *Let $\mathcal{B}$ be a set of blocks with parameters d, k. We say that $\mathcal{B}$ is* spanning *if the blocks in $\mathcal{B}$, viewed as polynomials in the dk variables $Y_{j,h}$, span the block $Z_1 Z_2 \cdots Z_d$. Denote by $\Delta_{d,k}$ the class of spanning block sets $\mathcal{B}$ with parameters d, k.*

Example 5.2 *Let $d = k = 2$. The block set $\{Y_{1,1}Y_{1,2}, Z_1 Y_{2,2}, Y_{2,1} Y_{1,2}\}$ is spanning. On the other hand, it is easy to verify that block set $\mathcal{B} = \{Y_{1,1}Z_2, Z_1 Y_{2,2}, Y_{2,1}Y_{1,2}\}$ is not spanning, i.e. $\mathcal{B}$ does not span the block $Z_1 Z_2$, although each of the four monomials in $Z_1 Z_2$ is involved in some block in $\mathcal{B}$.*

Example 5.3 *If $d = 2k - 1$, then the set of all blocks b such that $\delta(b) \leq 1$ and $|V(b)| \geq 1$ is spanning. This set of blocks is used in [20] to construct a PIR protocol with communication complexity $O(k^3 n^{1/(2k-1)})$.*

The following set of blocks corresponds to Claim 3.4.

Example 5.4 *Let k, λ, k', d, be as in Claim 3.3. Then the following set $\mathcal{B}$ is spanning. $\mathcal{B}$ includes: (1) every block b such that $\delta(b) \leq 1$ and $|V(b)| \geq 1$; (2) every block b such that $k' \leq |V(b)|$, $\delta(b) \leq \lambda |V(b)|$, and each $j \in [k] \setminus V(b)$ occurs in b more than λ times.*

Generalizing Theorem 3.5, it is possible to use any spanning block set $\mathcal{B}$ for reducing k-server PIR to instances of PIR with a smaller number of servers. This is formalized by the following theorem.

Theorem 5.5 *Suppose there is a PIR protocol $\mathcal{P}$ with communication complexity $\mathrm{C}_{\mathcal{P}}(n,k)$. Then, for any $d = d(k)$ and a spanning block set $\mathcal{B} = \mathcal{B}(d,k) \in \Delta_{d,k}$ there is a PIR protocol $\mathcal{P}'$ with communication complexity*

$$\mathrm{C}_{\mathcal{P}'}(n,k) = O_k\left(n^{1/d} + \sum_{b \in \mathcal{B}} \mathrm{C}_{\mathcal{P}}(n^{\delta(b)/d}, |V(b)|)\right).$$

We note that Theorem 5.5 gives, in a sense, a closed-form expression for the best communication complexity attainable by the current approach. The main difficulty, however, is in finding appropriate choices for the spanning block set $\mathcal{B}$ that optimize the overall complexity. We do not know if, using Theorem 5.5, it is possible to construct a protocol whose complexity is significantly better than the protocol presented in Corollary 3.9. In the full version of the paper we will show that, using Theorem 5.5, one cannot construct a protocol whose complexity is better than $O(n^{1/k^2})$.

Acknowledgments. We thank Enav Weinreb for valuable comments on earlier drafts of this paper. We also thank Don Coppersmith for a helpful discussion.

References

[1] A. Ambainis. Upper bound on the communication complexity of private information retrieval. In *24th ICALP*, *LNCS* 1256, pp. 401–407, 1997.

[2] L. Babai, P. G. Kimmel, and S. V. Lokam. Simultaneous messages vs. communication. In *STACS '95*, *LNCS* 999, pp. 361–372, 1995.

[3] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *STACS '90*, vol. 415 of *LNCS*, pp. 37–48, 1990.

[4] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *J. of Cryptology*, 10(1):17–36, 1997.

[5] A. Beimel and Y. Ishai. Information-theoretic private information retrieval: A unified construction. In *28th ICALP*, vol. 2076 of *LNCS*, pp. 912–926, 2001.

[6] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. In *CRYPTO 2000*, vol. 1880 of *LNCS*, pp. 56–74, 2000.

[7] A. Beimel and Y. Stahl. Robust information-theoretic private information retrieval. *3rd Conf. on Security in Commun. Networks*, 2002.

[8] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT '99*, vol. 1592 of *LNCS*, pp. 402–414, 1999.

[9] R. Canetti, Y. Ishai, R. Kumar, M. K. Reiter, R. Rubinfeld, and R. N. Wright. Selective private function evaluation with applications to private statistics. In *20th PODC*, pp. 293 – 304, 2001.

[10] B. Chor and N. Gilboa. Computationally private information retrieval. In *29th STOC*, pp. 304–313, 1997.

[11] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *J. of the ACM*, 45:965–981, 1998.

[12] A. Deshpande, R. Jain, T Kavita, V. Lokam, and J. Radhakrishnan. Better lower bounds for locally decodable codes. In *16th CCC*, pp. 184–193, 2002.

[13] G. Di-Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for private information retrieval. *J. of Cryptology*, 14(1):37–74, 2001.

[14] G. Di-Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *EUROCRYPT 2000*, vol. 1807 of *LNCS*, pp. 122–138, 2000.

[15] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. In *28th ICALP*, vol. 2076 of *LNCS*, pp. 927–938, 2001.

[16] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In *RANDOM '98*, vol. 1518 of *LNCS*, pp. 200–217, 1998.

[17] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *JCSS*, 60(3):592–629, 2000.

[18] O. Goldreich. Personal communication, 2000.

[19] O. Goldreich, H. Karloff, L. Schulman, and L. Trevisan. Lower bounds for linear locally decodable codes and PIR. In *16th CCC*, pp. 175 – 183, 2002.

[20] Y. Ishai and E. Kushilevitz. Improved upper bounds on information theoretic private information retrieval. *31st STOC*, pp. 79 – 88, 1999.

[21] T. Itoh. Efficient private information retrieval. *IEICE Trans. Fund. of Electronics, Commun. and Comp. Sci.*, E82-A(1):11–20, 1999.

[22] T. Itoh. On lower bounds for the communication complexity of private information retrieval. *IEICE Trans. Fund. of Electronics, Commun. and Comp. Sci.*, E84-A(1):157–164, 2001.

[23] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *32nd STOC*, pp. 80–86, 2000.

[24] A. Kiayias and M. Yung. Secure games with polynomial expressions. In *28th ICALP*, vol. 2076 of *LNCS*, pp. 939–950, 2001.

[25] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *38th FOCS*, pp. 364–373, 1997.

[26] E. Kushilevitz and R. Ostrovsky. One-way trapdoor permutations are sufficient for single-database computationally-private information retrieval. In *EUROCRYPT 2000*, *LNCS* 1807, pp. 104–121, 2000.

[27] E. Mann. Private access to distributed information. Master's thesis, Technion, 1998.

[28] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *33th STOC*, 2001.

[29] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *31st STOC*, pp. 245–254, 1999.

[30] R. Ostrovsky and V. Shoup. Private information storage. In *29th STOC*, pp. 294–303, 1997.

[31] J. F. Raymond. Private information retrieval: Improved upper bound, extension and applications. Master's thesis, McGill University, 2000.

[32] J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *ASIACRYPT '98*, vol. 1514 of *LNCS*, pp. 357–371, 1998.

A. A High-Level View: Replication vs. Degree

In this appendix we try to explain where the improvement over previous protocols comes from. The recursive formulation of our protocol, as well as the various technicalities involved in its implementation and analysis, make it somewhat difficult to trace the actual source for improvement. The following overview of our approach attempts to give a fairly accurate intuition as to *how* and *why* it works, while ignoring some details or difficulties that are dealt with in the technical sections.

A key technique in previous solutions, as well as in ours, is an *arithmetization* of the PIR problem. Using a polynomial representation for the database x, the user's goal is reduced to evaluating a multivariate polynomial $P(\vec{Z})$, known to all k servers, on some point $\vec{z}$ determined by its retrieval index i.[6] This task should be achieved while hiding $\vec{z}$ from each server. There are three parameters associated with the above problem: (1) the *number of variables* in P, denoted by m; (2) the *degree* of P, denoted by d; and (3) the *replication* amount k, i.e., the number of servers to which the polynomial P is known. The first two parameters determine the *description size* of P, which is $\Theta(m^d)$ when d is constant. Using an appropriate polynomial representation, the database size n is roughly equal to the description size of P, i.e. $n = \Theta(m^d)$.

A first observation is that if we manage to reduce the degree of P by a factor of c (without changing the number of variables m by much), the new description size will be roughly the *c-th root* of the original one. The new polynomial can be communicated to the user using only $O(n^{1/c})$ communication bits. We therefore try to reduce the degree of P, possibly updating the evaluation point $\vec{z}$, without revealing $\vec{z}$ to any server. In what follows we first describe the previous approach for achieving this goal, following [2, 5], and then explain where we depart from this approach.

A degree reduction as above is achieved as follows. The user picks random points $\vec{y}_1, \ldots, \vec{y}_k \in F^m$ subject to the condition $\vec{y}_1 + \ldots + \vec{y}_k = \vec{z}$, and sends each server $\mathcal{S}_j$ a query consisting of all k points *except* $\vec{y}_j$. The intuition for this step is that it provides maximal redundancy subject to the requirement of hiding $\vec{z}$. Note that the cost of this step is dominated by $m = O(n^{1/d})$. Thus, for the total communication complexity to be small, the initial representation degree d must be large.

The next step is to express the desired value $P(\vec{z})$ as the value of the polynomial $Q(\vec{Y}_1, \ldots, \vec{Y}_k) \overset{\text{def}}{=} P(\vec{Y}_1 + \ldots + \vec{Y}_k)$ at the point $(\vec{y}_1, \ldots, \vec{y}_k)$. The advantage of switching to this new representation is that the value assigned to each of its variables is known to *almost all* servers (in contrast to the original polynomial $P(\vec{Z})$, whose values $\vec{z}$ are completely unknown to the servers and should remain so).

To make use of this advantage, we write Q as a sum of monomials. Every such monomial is a product of d variables. Each variable is missed by exactly one out of the k servers; hence, for the d variables involved in a given monomial there must be *at least one* server which misses *at most* d/k values of these variables. We now assign each monomial to one of the servers corresponding to this monomial, and let each server substitute specific values for all of the variables it knows in each of the monomials assigned to it. After this step, each server holds a polynomial P_j in its *unknown* variables $\vec{Y}_j$, such that the degree of P_j is at most d/k and $\sum_{j=1}^{k} P_j(\vec{y}_j) = P(\vec{z})$. Thus, we can complete the protocol by letting each server j send to the user a *descrip-*

---

[6] Here and in the following all polynomials are assumed to be over a finite field, which is taken to be GF(2) by default.

tion of its lower degree polynomial P_j (e.g., using a list of its coefficients), which allows the user to compute the desired value $P(\vec{z})$.

We now take a more quantitative look at the type of savings obtained by the above degree reduction technique. As discussed above, reducing the representation degree by a factor of k induces a $1/k$-th power reduction in its size. By picking a "high" degree d, the queries sent to each server will be short, and the answers will be of length $O(n^{1/k})$. At a first glance, this seems to be the end of the road. However, a crucial (and easy to overlook) observation is that the above degree analysis involves *integers*. Thus, the reduced degree is actually guaranteed to be bounded by $\lfloor d/k \rfloor$. While such integer truncation operations are typically viewed as a nuisance, in this case they turn out to make make a big difference. In a sense, in an "integer-less world" a PIR protocol with $O(n^{1/k})$ communication is the best we would have.

How far can the advantage of truncation be pushed? Two useful examples are the following. First, assume that $d = k - 1$. In this case, we get the most evident benefit: $\lfloor d/k \rfloor = 0$, instead of $1 - 1/k$ in the fractional case, implying that that each polynomial P_j will have degree-0 (i.e., be a constant) and therefore can be described by one bit. However, a disadvantage of this choice of parameters is that d is rather small, and therefore the length of the queries will be rather large ($O(n^{1/d}) = O(n^{1/(k-1)})$). Still, a useful feature of of the corresponding protocol is that it requires only one answer bit from each server. Indeed, the latter protocol was prior to this work essentially the best protocol of this type. A second useful choice of parameters is $d = 2k - 1$. In this case the answers are longer than before: since $\lfloor d/k \rfloor = 1$ the degree is reduced by a factor of $d = 2k - 1$, and consequently the description length of P_j is $O(n^{1/d}) = O(n^{1/(2k-1)})$. However, since the queries now are also shorter, namely of length $O(n^{1/(2k-1)})$, we get a protocol of a smaller total communication complexity. The above protocol was the best known protocol (in terms of the total communication complexity) prior to this work.

In light of the above surprising effect of integer truncation on the complexity of PIR, it is natural to ask whether the savings can be pushed even further. We start with the following observation. The degree reduction process we used may be thought of as a way for *trading replication for degree*: We started with a polynomial P of degree d which is replicated among k servers, and ended up with polynomials P_j of degree $\lfloor d/k \rfloor$, each known to only *one* server. Thus, we have given away all of the original replication, and in return obtained the biggest possible gain in the degree. However, it is not clear a-priori that this greedy approach is optimal. An alternative approach that comes to mind is to apply several *partial* degree reduction steps, hoping to benefit multiple times from the integer truncation effect.

To this end, we generalize the above degree reduction procedure as follows. Suppose that we are willing to reduce the replication from k to k' (rather than 1). Then, we may assign each monomial to some set V of k' servers which *jointly* miss the least number of variables from this monomial. This allows us to write the desired value $P(\vec{z})$ as the sum of values $Q_V(\vec{y})$, where each Q_V is a polynomial known to a set V of at least k' servers. Note that the maximal degree of Q_V increases as k' grows, and in any case is no more than $\lfloor dk'/k \rfloor$.

We return to the previous question: can we gain by reducing the degree (along with the replication) in multiple steps? Intuitively, there is no advantage in applying the above "local" degree reduction process in multiple steps, as the final representation could have been directly attained in one step. The additional key idea that makes such a multi-step process useful is to use additional interaction with the user for adjusting the degree between each two reduction steps. In such a *degree conversion* step, both the number of variables and the degree are changed, but the description size remains the same. For instance, suppose that some set of k' servers holds a degree-3 polynomial Q' in m variables, and the user holds a point $\vec{z}$ whose additive shares $\vec{y}_1, \ldots, \vec{y}_{k'}$ are replicated among the servers as above. Moreover, suppose that it is possible for the servers to locally compute a degree-2 polynomial Q in $O(m^{3/2})$ variables and for the user to locally compute a point $\vec{z}\,'$, such that $Q(\vec{z}) = Q'(\vec{z}\,')$. (Note that such a conversion is presumably plausible, since $(m^{3/2})^2 = m$, and so we have not decreased the description size.) Then, by re-sharing the point $\vec{z}\,'$ among the servers, the user can adjust the degree to 2 without reducing the amount of replication or increasing the description size. Such a degree conversion procedure would allow to obtain additional savings by interleaving reduction steps with degree conversion steps.

We illustrate this by a 4-server example. Suppose that $d = 5$. Dispensing with all the replication in one step (i.e., by letting $k' = 1$), reduces the degree to $\lfloor 5/4 \rfloor = 1$. Instead, we let $k' = 3$. This brings the degree down to $\lfloor 5 \cdot 3/4 \rfloor = 3$, since for any monomial there is a set of 3 servers which jointly miss at most 3 variables from this monomial. Now, we adjust the degree to 2. This increases the number of variables to $O(m^{3/2}) = O((n^{1/5})^{3/2}) = O(n^{3/10})$, and requires the user to send additional queries of comparable size. Finally, we can apply the reduction step again to reduce the replication from 3 to 1. This brings the degree down to 0, and allows the servers to communicate $P(\vec{z})$ to the user by sending a single bit each. Thus, we obtain a protocol with query length $O(n^{3/10})$ and answer length 1 – improving the protocol with query length $O(n^{1/(k-1)})$ described above.

We do not wheter the above degree conversion problem can be solved in general.[7] Instead, we get around this problem by relying on a specific *promise* on the value of the point $\vec{z}$ held by the user. The abstract linear algebra problem that underlies our solution is described in Section 5. The recursive invocations of PIR in our protocol achieve, in effect, the degree-conversions which result in the efficiency improvement.

---

[7] Speficically, it is open if for *every* $d' < d$ it is possibvle to convert m-variavte degre-d polynomials to m'-variavte degre-d' polynomials where $m' = O(m^{d/d'})$.