Distributed Point Functions and Their Applications*

Niv Gilboa[1] and Yuval Ishai[2,**]

[1] Dept. of Communication Systems Eng., Ben-Gurion University, Beer-Sheva, Israel
gilboan@bgu.ac.il
[2] Dept. of Computer Science, Technion, Haifa, Israel
yuvali@cs.technion.ac.il

Abstract. For $x, y \in \{0,1\}^*$, the point function $P_{x,y}$ is defined by $P_{x,y}(x) = y$ and $P_{x,y}(x') = 0^{|y|}$ for all $x' \neq x$. We introduce the notion of a *distributed point function* (DPF), which is a keyed function family F_k with the following property. Given x, y specifying a point function, one can efficiently generate a key pair (k_0, k_1) such that: (1) $F_{k_0} \oplus F_{k_1} = P_{x,y}$, and (2) each of k_0 and k_1 hides x and y. Our main result is an efficient construction of a DPF under the (minimal) assumption that a one-way function exists.

Distributed point functions have applications to private information retrieval (PIR) and related problems, as well as to worst-case to average-case reductions. Concretely, assuming the existence of a strong one-way function, we obtain the following applications.

- **Polylogarithmic 2-server binary PIR.** We present the first 2-server computational PIR protocol in which the length of each query is polylogarithmic in the database size n and the answers consist of a single bit each. This improves over the $2^{O(\sqrt{\log n})}$ query length of the protocol of Chor and Gilboa (STOC '97). Similarly, we get a polylogarithmic "PIR writing" scheme, allowing secure non-interactive updates of a database shared between two servers. Assuming just a standard one-way function, we get the first 2-server private keyword search protocol in which the query length is polynomial in the keyword size, the answers consist of a single bit, and there is no error probability. In all these protocols, the computational cost on the server side is comparable to applying a symmetric encryption scheme to the entire database.
- **Worst-case to average-case reductions.** We present the first worst-case to average-case reductions for PSPACE and EXPTIME complete languages that require only a constant number of oracle queries. These reductions complement a recent negative result of Watson (TOTC '12).

Keywords: Distributed point function, PIR, secure keyword search, worst-case to average-case reductions.

* Research received funding from the European Union's Tenth Framework Programme (FP10/2010-2016) under grant agreement no. 259426 ERC-CaC.
** Supported in part by ISF grant 1361/10 and BSF grant 2012378.

P.Q. Nguyen and E. Oswald (Eds.): EUROCRYPT 2014, LNCS 8441, pp. 640–658, 2014.

1 Introduction

For $x, y \in \{0,1\}^*$, the point function $P_{x,y}$ is defined by $P_{x,y}(x) = y$ and $P_{x,y}(x') = 0^{|y|}$ for all $x' \neq x$. Motivated by the goal of improving the efficiency of private information retrieval (PIR) [8,24] and related cryptographic primitives, we introduce and study the notion of a *distributed point function* (DPF). Informally speaking, a DPF is a representation of a point function $P_{x,y}$ by two keys k_0 and k_1. Each key individually hides x, y, but there is an efficient algorithm $Eval$ such that $Eval(k_0, x') \oplus Eval(k_1, x') = P_{x,y}(x')$ for every x'. Letting F_k denote the function $Eval(k, \cdot)$, the functions F_{k_0} and F_{k_1} can be viewed as an additive secret sharing of $P_{x,y}$.

A simple implementation of a DPF is to let k_0 specify the entire truth-table of a *random* function $F_{k_0} : \{0,1\}^{|x|} \to \{0,1\}^{|y|}$ and k_1 specify the truth-table of $F_{k_1} = F_{k_0} \oplus P_{x,y}$. Since each of k_0 and k_1 is random, this solution is perfectly secure. The problem with this solution is that the size of each key is exponential in the input size. Our main goal is to obtain a DPF with polynomial key size.

To demonstrate the usefulness of this new primitive, consider the goal of obtaining a 2-server secure keyword search protocol with low communication complexity. In such a protocol, two servers hold a large database $D = \{w_1, \ldots, w_n\}$, where each w_i is an ℓ-bit keyword, and a user wishes to find whether $w \in D$ while hiding w from each server. Given a DPF scheme, the user creates keys (k_0, k_1) for the point function $P_{w,1}$ and sends one key to each server. Given a key k_b, $b \in \{0,1\}$, server b returns the answer bit $a_b = \bigoplus_{j=1}^{n} F_{k_b}(w_j)$. The user then computes $a_0 \oplus a_1$, which is equal to 1 if and only if $w \in D$. Essentially the same solution applies to 2-server PIR, where D is an n-bit database and the user wants to privately retrieve the i-th bit D_i.

The connection to PIR can be used to translate linear lower bounds on the communication complexity of 2-server PIR protocols with short answers [8,32] into an exponential lower bound of $2^{\Omega(|x|)}$ on the DPF key size if we require that each key hide x with information-theoretic security. However, this lower bound does not hold in the context of computational PIR (CPIR), where the security requirement is relaxed to computational security [7].

Based on the above discussion, we define a DPF to be a pair of PPT algorithms $(Gen, Eval)$, such that Gen receives x and y as input and creates the keys (k_0, k_1). In particular, the efficiency of Gen forces the key size to be polynomial in $|x| + |y|$. Each key individually must give no information in the computational sense on x, y. However, as described previously, $Eval(k_0, x') \oplus Eval(k_0, x') = P_{x,y}(x')$ for every x'.

Our Contribution. Our main result is establishing the feasibility of a DPF under the (minimal) assumption that a one-way function exists. Our construction uses a recursion that compresses the keys k_0 and k_1. The base scheme is the simple solution described above with each key of length $2^{|x|}\,|y|$. The recursion runs for $\lceil \log |x| \rceil$ steps. In each step of the recursion, the key is compressed to almost a square root of its former size. Random portions of the key are replaced with seeds for a pseudo-random generator (PRG) that can be expanded to longer

strings that are pseudo-random. Carefully correlating the seeds in the two keys ensures that running *Eval* on the two compressed keys and x' and then taking the XOR of the results still gives $P_{x,y}(x')$ as it does with the uncompressed keys.

Our construction looks quite attractive from a concrete efficiency point of view and may well give rise to the most practical solutions to date to PIR and related problems. The key size in our DPF is roughly $8\kappa \cdot |x|^{\log 3}$, where κ is the seed length of the underlying PRG and log is a base 2 logarithm. (This analytical bound is somewhat pessimistic; we present optimized key sizes for typical lengths of x in Table 1.)

Using the transformation described above, we get a 2-server CPIR protocol with query size equal to the DPF key size and a single bit answer from each server. The single bit answer feature is appealing in situations where the same user queries are used many times, say when the database is rapidly updated but the user's interests remain the same. We refer to a protocol that has this feature as *binary* CPIR.

Our protocol improves over the first CPIR protocol from [7], which implicitly relies on a DPF of super-polynomial complexity $|x|^{O(\sqrt{\log |x|})}$, and gives rise to the first polylogarithmic communication 2-server CPIR protocol based on (exponentially hard)[1] one-way functions, and the first *binary* polylogarithmic 2-server PIR protocol under any standard assumption.

In terms of computational cost on the server side, which typically forms the practical efficiency bottleneck in PIR, the computation of each server on a database of size n roughly corresponds to producing n pseudorandom bits.[2] (The computation on the client's side is negligible.) This is faster by orders of magnitude than the $\Omega(n)$ public-key operations required by known single-server CPIR protocols (cf. [24,29,10,25,17]).

Additional Applications. The above applications to 2-server PIR also apply to the related problem of private information storage [26] (aka "PIR writing"), where a client wants to non-interactively update entry i in a database D which is additively secret-shared between two servers without revealing i to each server. We get the first polylogarithmic solution to this problem (assuming exponentially strong OWFs). We note that single-server CPIR protocols or even stronger primitives such as fully homomorphic encryption [16] do not apply in this setting.

[1] In this work we say that a one-way function is *exponentially hard* if it is hard to invert by circuits of size 2^{n^c}, for some $c > 0$. The proof of Theorem 5 shows that such a one-way function is necessary for the existence of binary 2-server CPIR protocols in which the query length is polylogarithmic in the database size n and security holds against poly(n)-time distinguishers. Alternatively, using the two-parameter definition of CPIR from [25] that limits the distinguisher to run in poly(κ) time (independently of n), we get binary 2-server PIR with query length $\kappa \cdot$polylog(n) assuming the existence of a standard one-way function.

[2] A naive usage of a DPF requires a separate DPF evaluation for each nonzero entry of the database. In the full version we describe a method for amortizing this cost.

As the previous example demonstrates, DPF can be used to get qualitatively improvements over previous protocols for private keyword search [9,15,27]. Recall that in the two-server variant of this problem, two servers hold a set of words $\{w_1, \ldots, w_n\}$ and a user wishes to find out whether a word w is part of the set, while hiding w from each server. Previous solutions to this problem either make intensive use of public-key cryptography or alternatively involve data structures that have overhead in communication, round complexity, storage complexity, update cost, and error probability. Our protocol avoids all these disadvantages using only symmetric cryptography. It involves a single communication round, requires no data structures, has no error probability, and is the first private keyword search protocol we are aware of (under any standard assumption) that only requires one-bit answers. In more concrete terms, the query length of our protocol is $O(\kappa \cdot |w|^{\log 3})$ and the dominant computational cost involves a small number of PRG invocations (roughly corresponding to the DPF key size) for each keyword in the database. It can easily support database updates and be used in the streaming model of [27]. It can also be extended to support private keyword search with payloads, where the answer size of each server is equal to the size of the payload.

Finally, we present an application of DPFs to complexity theory. Assuming the existence of an exponentially hard one-way function, we get the first worst-case to average case reduction for PSPACE and EXPTIME languages which only makes $O(1)$ oracle calls. Concretely, we show a language L in PSPACE (or in EXPTIME) such that if an algorithm A decides L correctly on all but a δ fraction of the instances, then every language L' in PSPACE (or EXPTIME) admits a polynomial time oracle algorithm R such that R^A decides L' with good probability on *every* instance. The new feature of our reduction is that R makes only two calls to its oracle A. The best previous reductions required $\Omega(n/\log n)$ calls for inputs of length n [1,2]. A different version of the reduction applies to the case where A decides L correctly on all but a δ fraction of the instances, and on the other instances returns "don't know". In this case, R^A correctly decides any instance in L', with probability 1, while the expected number of calls to A is $O(1)$.

Alternatives and Related Work. In the information-theoretic setting for PIR, the best known 2-server protocol requires $O(n^{1/3})$ bits of communication [8]. However, in the case of *binary* 2-server PIR, the query size must be linear in n [8,32,3]. Much better protocols are known if there are 3 or more servers and security should only hold against a single server. The best known 3-server protocols [33,14,4] have queries of size $2^{O(\sqrt{\log n \cdot \log \log n})}$ and single-bit answers. Note that unlike our 2-server protocols, this communication complexity is super-polynomial in the bit-length of the user's input i. Polylogarithmic information-theoretic PIR protocols are known to exist only with $\Omega(\log n/\log \log n)$ servers [8]. A general technique from [5] can be used to convert a k-server binary PIR protocol with security against a single server into a k^t-server binary PIR protocol with security against t servers and comparable communication complexity. This technique can be applied to our 2-server protocol to yield t-private 2^t-server CPIR protocols with polylogarithmic communication.

While in this work we mainly focus on 2-server CPIR protocols, a better studied model for CPIR is the single-server model, introduced in [24]. Single-server protocols with polylogarithmic communication are known to exist under standard cryptographic assumptions [10,25,17,6].

Our two-server CPIR protocol has three main advantages over its single-server counterparts. First, as discussed above, our protocol is significantly more efficient in computation. Second, our protocol has single-bit answers, instead of answers that are at least the size of a security parameter (typically the ciphertext size in an underlying public-key encryption scheme). The third advantage is that our protocol relies on a much weaker cryptographic assumption, namely the existence of one-way functions. In contrast, single-server PIR protocols imply oblivious transfer [13], which in turn implies public-key encryption. We get even more significant advantages for the problems of private keyword search or PIR writing, where standard CPIR does not apply and alternative solutions have additional costs.

Organization. In Section 2 we present definitions and notation. Section 3 describes a construction for a distributed point function. Three applications of a distributed point function are presented in Section 4, a CPIR scheme in subsection 4.1, a scheme to privately retrieve information by keywords in subsection 4.2 and a worst case to average case reduction for EXPTIME and PSPACE languages in subsection 4.3. Finally, a proof that the existence of a DPF implies the existence of a one-way function appears in Section 6.

2 Definitions and Notation

Notation 1. *For $x, y \in \{0,1\}^*$, the* point function $P_{x,y} : \{0,1\}^{|x|} \to \{0,1\}^{|y|}$ *is defined by $P_{x,y}(x) = y$ and $P_{x,y}(x') = 0^{|y|}$ for all $x' \neq x$.*

Definition 1. *A* distributed point function *is a pair of PPT algorithms $DPF = (Gen, Eval)$ with the following syntax:*

- *$Gen(x, y)$, where $x, y \in \{0,1\}^*$, outputs a pair of keys (k_0, k_1). When y is omitted it is understood to be the single bit 1.*
- *$Eval(k, x', m)$, where $k, x' \in \{0,1\}^*$ and $m \in \mathbb{N}$, outputs $y' \in \{0,1\}^*$.*

DPF must satisfy the following correctness and secrecy requirements.

Correctness: *For all $x, x', y \in \{0,1\}^*$ such that $|x| = |x'|$*

$$Pr[(k_0, k_1) \leftarrow Gen(x, y) : Eval(k_0, x', |y|) \oplus Eval(k_1, x', |y|) = P_{x,y}(x')] = 1.$$

Secrecy: *For $x, y \in \{0,1\}^*$ and $b \in \{0,1\}$, let $D_{b,x,y}$ denote the probability distribution of k_b induced by $(k_0, k_1) \leftarrow Gen(x, y)$. There exists a PPT algorithm Sim such that the following distribution ensembles are computationally indistinguishable:*

1. $\{Sim(b, |x|, |y|)\}_{b \in \{0,1\}, x, y \in \{0,1\}^*}$
2. $\{D_{b,x,y}\}_{b \in \{0,1\}, x, y \in \{0,1\}^*}$

The above definition captures the intuitive security requirement that k_b reveals nothing except $b, |x|$, and $|y|$. We will also be interested in exponentially strong DPFs, which satisfy the stronger requirement that for some constant $c > 0$, the above two ensembles are $(2^{(|x|+|y|)^c}, 2^{-(|x|+|y|)^c})$-computationally indistinguishable.

Notation 2. *Let $\oplus$ denote bitwise exclusive-or and let $||$ denote concatenation of strings. We use $+$ to denote addition over a field or a vector space, as implied by the context.*

Notation 3. *Let $\mathbb{F}_{2^q}$ denote the finite field with 2^q elements and let $\mathbb{F}^n$ denote the vector space of dimension n over a field $\mathbb{F}$. The i-th unit vector of length n over $\mathbb{F}$ is denoted by e_i. The j-th element in a vector $v \in \mathbb{F}^n$ is denoted by $v[j]$.*

We sometimes view the input and output of *Gen* and *Eval* as elements of a finite field with an appropriate number of elements instead of as binary strings. The correctness requirement can be restated as $\Pr[(k_0, k_1) \leftarrow Gen(x, y) : Eval(k_0, x', |y|) + Eval(k_1, x', |y|) = P_{x,y}(x')] = 1$, with addition over $\mathbb{F}_{2^{|y|}}$.

3 Distributed Point Function

3.1 Initial Scheme

If we dispense with the requirement that *Gen* and *Eval* run in polynomial time then we can construct a fairly simple scheme for a DPF as follows. $Gen(x, y)$ outputs two keys k_0, k_1 such that $k_0, k_1 \in (\mathbb{F}_{2^{|y|}})^{2^{|x|}}$. Each key is regarded as a vector of length $2^{|x|}$ over the field $\mathbb{F}_{2^{|y|}}$ and is chosen randomly with the constraint that $k_0[x] + k_1[x] = y$, while $k_0[x'] + k_1[x'] = 0$, for all $x', x' \neq x$. $Eval(k, x', |y'|)$ returns $k[x']$ for every x'. Clearly, this scheme has both the correctness and secrecy properties required in a DPF.

The scheme we propose recursively compresses the keys k_0, k_1. Starting with the scheme above, which we denote $\mathrm{DPF}_0 = (Gen_0, Eval_0)$, each recursion step compresses the length of the keys to slightly more than a square root of their length in the previous step. Repeating the process $\log |x|$ times results in polynomial length.

As an initial attempt towards constructing the next step, $\mathrm{DPF}_1 = (Gen_1, Eval_1)$, we consider a simpler scheme $\mathrm{DPF}_1^* = (Gen_1^*, Eval_1^*)$. The $2^{|x|}$ possible inputs x to the point function $P_{x,y}$ are arranged in a table with 2^m rows and 2^μ columns for some m, μ such that $m + \mu = |x|$. Each input x' is viewed as a pair $x' = (i', j')$, which represents the location of x' in the table.

Let $G : \{0,1\}^\kappa \longrightarrow \{0,1\}^{2^\mu |y|}$ be a pseudo-random generator. Let the representation of x as a pair be $x = (i, j)$. Gen_1^* first chooses uniformly at random and independently $2^m + 1$ seeds of length κ each, $s_1, \ldots, s_{i-1}, s_i^0, s_i^1, s_{i+1}, \ldots, s_{2^m}$. The output of Gen_1^* is a pair (k_0, k_1) defined by $k_0 = s_1, \ldots, s_{i-1}, s_i^0, s_{i+1}, \ldots, s_{2^m}$ and $k_1 = s_1, \ldots, s_{i-1}, s_i^1, s_{i+1}, \ldots, s_{2^m}$. Given input k, $x' = (i', j')$ and $|y'|$, the algorithm $Eval_1^*(k, x', |y'|)$ uses G to obtain $2^\mu \cdot |y'|$ bits by computing $G(k[i'])$.

This expanded string is viewed as a vector of length 2^μ over $\mathbb{F}_{2^{|y'|}}$. $Eval_1^*(k, x', |y'|)$ returns the j'-th entry of this vector, $G(k[i'])[j']$.

While DPF_1^* seems promising in terms of key length it is only partially correct. For each $x' = (i', j')$ such that $i' \neq i$, we have that:

$$Eval_1^*(k_0, x', |y'|) \oplus Eval_1^*(k_1, x', |y'|) = G(s_{i'})[j'] + G(s_{i'})[j'] = 0.$$

However, if $i' = i$, we have that

$$Eval_1^*(k_0, x', |y'|) \oplus Eval_1^*(k_1, x', |y'|) = G(s_i^0)[j'] + G(s_i^1)[j'],$$

and this value is wrong with overwhelming probability.

One possible approach to correct this deficiency of DPF_1^* is as follows. Associate each element in the vector space $\mathbb{F}_{2^{|y|}}^{2^\mu}$ with an element in the field $\mathbf{F}_{2^{|y|2^\mu}}$ in the natural way. Compute the element $CW \leftarrow (G(s_i^0) + G(s_i^1))^{-1} \cdot (ye_j)$ over $\mathbf{F}_{2^{|y|2^\mu}}$. Modify Gen_1^* by concatenating CW to both k_0 and k_1. Modify $Eval_1^*$ by computing $G(k[i']) \cdot CW$ over $\mathbf{F}_{2^{|y|2^\mu}}$, regarding the result as a vector over $\mathbb{F}_{2^{|y'|}}$ and returning the j'-th entry of this vector.

The actual approach we use in the next two algorithms, Gen_1 and $Eval_1$ is slightly more complex, involving two correction elements CW_0 and CW_1 instead of just one. This approach allows a recursion, further compressing the key size, which does not seem to be possible using a single CW.

In both Algorithm 1 implementing Gen_1 and Algorithm 2 we assume that $|y| \leq \kappa + 1$.

Algorithm 1. $Gen_1(x, y)$

Let $G : \{0,1\}^\kappa \longrightarrow \{0,1\}^{\kappa 2^{|x|/2}}$ be a pseudo-random generator.
if $(|y| \cdot 2^{|x|} \leq \kappa + 1)$ **then**
    Return $Gen_0(x, y)$.
Let $m \leftarrow \lceil \log((\frac{|y| \cdot 2^{|x|}}{\kappa+1})^{1/2}) \rceil$ where κ is the length of the seeds.
$\mu \leftarrow \lceil \log((\frac{2^{|x|} \cdot (\kappa+1)}{|y|})^{1/2}) \rceil$.
Choose $2^m + 1$ seeds $s_1, \ldots, s_i^0, s_i^1, \ldots, s_{2^m}$ randomly and independently from $\{0,1\}^\kappa$.
Choose 2^m random bits $t_1, \ldots, t_{2^m}$.
Let $t_i^0 \leftarrow t_i$ and $t_i^1 \leftarrow t_i \oplus 1$.
Choose two random vectors $r_0, r_1 \in \mathbb{F}_{2^{|y|}}^{2^\mu}$ such that $r_0 + r_1 = y \cdot e_j$.
Let $CW_b \leftarrow G(s_i^b) + r_b$, for $b = 0, 1$, with addition in $\mathbb{F}_{2^{|y|}}^{2^\mu}$.
Let $k_b \leftarrow s_1 || t_1, \ldots, s_i^b || t_i^b, \ldots, s_{2^m} || t_{2^m}, CW_0, CW_1$, for $b = 0, 1$.
Return (k_0, k_1).

We argue that DPF_1 is correct by looking at the following cases. If $(|y| \cdot 2^{|x|} \leq \kappa + 1)$ then Gen_1 executes Gen_0 and $Eval_1$ executes $Eval_0$ with correct results. Otherwise, if $i' \neq i$ then

Algorithm 2. $Eval_1(k, x', |y'|)$

1: Let $G : \{0,1\}^\kappa \longrightarrow \{0,1\}^{2^{x/2}\kappa}$ be a pseudo-random generator.
2: **if** $(|y'| \cdot 2^{|x'|} \leq \kappa + 1)$ **then**
3: Return $Eval_0(k, x', |y'|)$.
4: Let $m \leftarrow \lceil \log((\frac{|y'| \cdot 2^{|x'|}}{\kappa+1})^{1/2}) \rceil$ where κ is the length of the seeds.
5: $\mu \leftarrow \lceil \log((\frac{2^{|x'|} \cdot (\kappa+1)}{|y'|})^{1/2}) \rceil$.
6: Parse k as $k = s_1 || t_1, \ldots, s_{2^m} || t_{2^m}, CW_0, CW_1$.
7: Let the location of x' in the $2^m \times 2^\mu$ table be $x' = (i', j')$.
8: Let $v \leftarrow G(s_{i'}) + CW_{t_{i'}}$, with addition in $\mathbb{F}_{2^{|y|}}^{2^\mu}$.
9: Return $v[j']$.

$$Eval_1^*(k_0, x', |y'|) \oplus Eval_1^*(k_1, x', |y'|) = (G(s_{i'}) + CW_{t'_i})[j'] + (G(s_{i'}) + CW_{t'_i})[j'] = 0.$$

If $i' = i$ then

$$Eval_1^*(k_0, x', |y'|) \oplus Eval_1^*(k_1, x', |y'|) = (G(s_i^0) + CW_{t_i})[j'] + (G(s_i^1) + CW_{t_i \oplus 1})[j'] = r_0[j'] + r_1[j'].$$

By the choice of r_0 and r_1, if $j' \neq j$ then $Eval_1^*(k_0, x', |y'|) \oplus Eval_1^*(k_0, x', |y'|) = 0$, while if $j' = j$, i.e $x' = x$ then $Eval_1^*(k_0, x', |y'|) \oplus Eval_1^*(k_0, x', |y'|) = y$.

Intuitively speaking, DPF_1 is secret, because k_0 and k_1 are each pseudo-random. Note that the only parts of a key k_b which are not completely independent of the rest are s_i^b, CW_0 and CW_1. Together, these elements satisfy $CW_0 + CW_1 + G(s_i^b) + G(s_i^{1 \oplus b}) = y \cdot e_i$. However, since k_b does not include the seed $s_i^{1 \oplus b}$, the three elements s_i^b, CW_0 and CW_1 are polynomially indistinguishable from a random string.

If $|x|$ and $|y|$ are so small that $|y| \cdot 2^{|x|} \leq (\kappa + 1)$ then Gen_1 has similar length output to Gen_0. Otherwise, the keys in DPF_1 are significantly smaller than the keys of DPF_0. Specifically, the total length of the seeds and additional bits (t_i) is $2^m(\kappa + 1)$. The total length of the correction words is $2\,|y| \cdot 2^\mu$. The total length of a key is at most $6((\kappa + 1)\,|y| \cdot 2^{|x|})^{1/2}$, which is only about $6\kappa^{1/2}$ larger than a square root of the key size of DPF_0.

The computational complexity of Gen_1 and $Eval_1$ is proportional to the length of the keys and therefore slightly more than a square root of the complexity of the matching algorithms Gen_0 and $Eval_0$.

3.2 Full Scheme

While DPF_1 is a major improvement over DPF_0, the running time of Gen_1 (and that of $Eval_1$) is still exponential in $|x|$. Improving the scheme by repeating the compression step recursively requires the following two observations.

Gen_1 chooses r_0, r_1 randomly in line 9 so that $r_0, r_1 \in (\mathbb{F}_{2^{|y|}})^{2^\mu}$ and $r_0 + r_1 = y \cdot e_j$. Therefore, the pair (r_0, r_1) is distributed identically to the output of $Gen_0(j, y)$.

In addition, the keys k_0 and k_1 that Gen_1 creates have a list of seeds for G and associated bits. Specifically, k_b includes $\sigma_b \stackrel{\triangle}{=} s_1||t_1, \ldots, s_i^b||t_i^b, \ldots, s_{2^m}||t_{2^m}$ for $b = 0, 1$. Regarding σ_0 and σ_1 as vectors in $(\mathbb{F}_{2^{\kappa+1}})^{2^m}$ we have that $\sigma_0 + \sigma_1 = (s_i^0||t_i^0 + s_i^1||t_i^1) \cdot e_i$. If $s_i \stackrel{\triangle}{=} s_i^0 \oplus s_i^1$ then $\sigma_0 + \sigma_1 = (s_i||1) \cdot e_i$. Therefore, the pair (σ_0, σ_1) is distributed identically to the output of $Gen_0(i, s_i||1)$.

The conclusion is that Gen_1 can be implemented by two calls to Gen_0. Similarly, we can define a pair of algorithms $Gen_\ell(x, y) = Gen(\ell, x, y)$ and $Eval_\ell(x, y) = Eval(\ell, x, y)$ by using recursive calls to $Gen_{\ell-1}(x, y)$ and $Eval_{\ell-1}(x, y)$. The scheme DPF_ℓ is defined as the pair of algorithms $Gen_{\ell-1}$ and $Eval_{\ell-1}$. Gen_ℓ is described in Algorithm 3 and $Eval_\ell$ is described in Algorithm 4. Setting $\ell = 1$ yields identical Gen_1 and $Eval_1$ algorithms to those defined in the previous sub-section.

In both the following algorithms, we assume that $|y| \leq \kappa + 1$.

Algorithm 3. $Gen_\ell(x, y)$

Let $G : \{0,1\}^\kappa \longrightarrow \{0,1\}^\alpha$ be a PRG (for α that will be determined in step 11).
if $(\ell = 0)$ or $(|y| \cdot 2^{|x|} \leq \kappa + 1)$ **then**
    Choose two random vectors $k_0, k_1 \in (\mathbb{F}_{2|y|})^{2^{|x|}}$, such that $k_0 + k_1 = y \cdot e_x$.
    Return (k_0, k_1).
Let $m \leftarrow \lceil \log((\frac{|y| \cdot 2^{|x|}}{\kappa+1})^{1/2}) \rceil$.
Let $\mu \leftarrow |x| - m$.
Regard x as a pair $x = (i, j)$, $i \in \{0,1\}^m, j \in \{0,1\}^\mu$.
Choose a random κ-bit string s_i and let $t_i = 1$ be a bit.
Recursively compute $(\sigma_0, \sigma_1) \leftarrow Gen_{\ell-1}(i, s_i||t_i)$.
Let $s_i^0||t_i^0 \leftarrow Eval_{\ell-1}(\sigma_0, i, \kappa + 1)$ and let $s_i^1||t_i^1 \leftarrow Eval_{\ell-1}(\sigma_1, i, \kappa + 1)$.
Recursively compute $(r_0, r_1) \leftarrow Gen_{\ell-1}(j, y)$. Let $\alpha \leftarrow |r_0| (= |r_1|)$.
Let $CW_{t_i^b} \leftarrow G(s_i^b) + r_b$, for $b = 0, 1$, with addition in $\mathbb{F}_2^\alpha$.
Let $k_b \leftarrow \sigma_b||CW_0||CW_1$, for $b = 0, 1$.
Return (k_0, k_1).

3.3 Analysis

We proceed to prove that $\mathrm{DPF}_\ell = (Gen_\ell, Eval_\ell)$ is a distributed point function and analyze the complexity of Gen_ℓ and $Eval_\ell$. This analysis will determine the computational complexity of Gen_ℓ and $Eval_\ell$ and the size of the output of Gen_ℓ.

The reason to generalize DPF_1 to DPF_ℓ is to improve performance and specifically to obtain two algorithms Gen_ℓ and $Eval_\ell$ that run in polynomial time. Since Gen_ℓ outputs two keys (k_0, k_1), the length of the keys is a lower bound on its computational complexity. In the next proposition we provide an upper bound on the length of the output of Gen_ℓ.

Proposition 1. *For every $\kappa, \ell \in \mathbb{N}$ and every $x, y \in \{0,1\}^*$, such that $|y| \leq \kappa + 1$, if (k_0, k_1) is a possible output of $Gen_\ell(x, y)$ then the length of k_0 and k_1 is at most $3^\ell (2^{|x|})^{\frac{1}{2^\ell}} \cdot (4(\kappa + 1))^{\frac{2^\ell - 1}{2^\ell}} |y|^{\frac{1}{2^\ell}}$.*

Algorithm 4. $Eval_\ell(k, x', |y'|)$

1: Let $G : \{0,1\}^\kappa \longrightarrow \{0,1\}^\alpha$ be a PRG (for α that will be determined in step 7).
2: **if** $(\ell = 0)$ or $(|y'| \cdot 2^{|x'|} \leq \kappa + 1)$ **then**
3: Return $k[x']$.
4: Let $m \leftarrow \lceil \log((\frac{|y'| \cdot 2^{|x'|}}{\kappa+1})^{1/2}) \rceil$.
5: Let $\mu \leftarrow |x| - m$.
6: Regard x' as a pair $x' = (i', j')$, $i' \in \{0,1\}^m, j' \in \{0,1\}^\mu$.
7: Parse k as $k = \sigma||CW_0||CW_1$. Let $\alpha \leftarrow |CW_0|\,(= |CW_1|)$.
8: Let $s_{i'}||t_{i'} \leftarrow Eval_{\ell-1}(\sigma, i', \kappa + 1)$.
9: Let $v \leftarrow G(s_{i'}) + CW_{t_{i'}}$, with addition in $\mathbb{F}_2^\alpha$.
10: Let $y' \leftarrow Eval_{\ell-1}(v, j', |y'|)$.
11: Return y'.

Proof. We prove the proposition by induction on ℓ. For $\ell = 0$, each key is chosen from $(\mathbb{F}_{2^{|y|}})^{2^{|x|}}$ and is therefore of length $2^{|x|} \cdot |y|$ which is exactly what is obtained by setting $\ell = 0$ in $3^\ell (|y|\, 2^{|x|})^{\frac{1}{2^\ell}} \cdot (4(\kappa+1))^{\frac{2^\ell - 1}{2^\ell}}$.

For the induction step, let $\ell \geq 1$ and assume that the proposition is correct for $\ell - 1$. If $|y| \cdot 2^{|x|} \leq \kappa + 1$ then Gen_ℓ runs the same algorithm as Gen_0 and the key size that Gen_ℓ outputs is $|y| \cdot 2^{|x|}$. Since $|y| \cdot 2^{|x|} \leq \kappa + 1$, we deduce that $(|y| \cdot 2^{|x|})^{\frac{2^\ell - 1}{2^\ell}} \leq (4(\kappa+1))^{\frac{2^\ell - 1}{2^\ell}}$ and therefore,

$$2^{|x|} \cdot |y| \leq (2^{|x|})^{1/2^\ell} \cdot (4(\kappa+1))^{\frac{2^\ell - 1}{2^\ell}} |y|^{1/2^\ell}.$$

If $|y| \cdot 2^{|x|} > \kappa + 1$ then Gen_ℓ outputs two keys k_0, k_1 such that $k_b = \sigma_b||CW_0||CW_1$ for $b = 0, 1$. $Gen_\ell(x, y)$ computes σ_b by $(\sigma_0, \sigma_1) \leftarrow Gen_{\ell-1}(i, s_i||t_i)$. Since $i \in \{0,1\}^m$, the length of i is $\lceil \log(\frac{|y| \cdot 2^{|x|}}{\kappa+1})^{1/2} \rceil$. The length of $s_i||t_i$ is $\kappa + 1$. Therefore, by the induction hypothesis the length of σ_b is

$$\begin{aligned} 3^{\ell-1} \cdot (2^{\lceil \log(\frac{|y| \cdot 2^{|x|}}{\kappa+1})^{1/2} \rceil})^{\frac{1}{2^{\ell-1}}} (4(\kappa+1))^{\frac{2^{\ell-1}-1}{2^{\ell-1}}} (\kappa+1)^{\frac{1}{2^{\ell-1}}} \leq \\ 3^{\ell-1} (2(\frac{|y| \cdot 2^{|x|}}{\kappa+1})^{1/2})^{\frac{1}{2^{\ell-1}}} 4^{\frac{2^{\ell-1}-1}{2^{\ell-1}}} (\kappa+1) = \\ 3^{\ell-1} (|y| \cdot 2^{|x|})^{\frac{1}{2^\ell}} (4(\kappa+1))^{\frac{2^\ell - 1}{2^\ell}} \end{aligned}$$

CW_0 and CW_1 are the same length as r_0 and r_1. $Gen_\ell(x, y)$ computes r_0 and r_1 by $(r_0, r_1) \leftarrow Gen_{\ell-1}(j, y)$, where $j \in \{0,1\}^\mu$. The value of μ is set to $\mu \leftarrow \lceil \log(\frac{2^{|x|} \cdot (\kappa+1)}{|y|})^{1/2} \rceil$ and by the induction hypothesis, the length of each of

r_0 and r_1 is at most:

$$3^{\ell-1} \cdot (2^{\lceil \log(\frac{2^{|x|}\cdot(\kappa+1)}{|y|})^{1/2} \rceil})^{\frac{1}{2^{\ell-1}}} (4(\kappa+1))^{\frac{2^{\ell-1}-1}{2^{\ell-1}}} |y|^{\frac{1}{2^{\ell-1}}} \leq$$
$$3^{\ell-1}((\frac{4(\kappa+1)\cdot 2^{|x|}}{|y|})^{1/2})^{\frac{1}{2^{\ell-1}}} 4^{\frac{2^{\ell-1}-1}{2^{\ell-1}}} |y|^{\frac{1}{2^{\ell-1}}} =$$
$$3^{\ell-1}(|y|\cdot 2^{|x|})^{\frac{1}{2^\ell}} (4(\kappa+1))^{\frac{2^\ell-1}{2^\ell}}$$

The length of k_b is the sum of the lengths of σ_b, CW_0 and CW_1, which is:

$$3 \cdot 3^{\ell-1}(|y|\cdot 2^{|x|})^{\frac{1}{2^\ell}} (4(\kappa+1))^{\frac{2^\ell-1}{2^\ell}} = 3^\ell (|y|\cdot 2^{|x|})^{\frac{1}{2^\ell}} (4(\kappa+1))^{\frac{2^\ell-1}{2^\ell}}.$$

□

Table 1 shows the length of a key k_b (either k_0 or k_1) for $|y| = 1$ and for some values of $|x|$ that are typical in applications of a distributed point function. The length of a key can be compared to the domain size, which is $2^{|x|}$. The leftmost column of the table shows the value of $|x|$. The next two columns show the depth of the recursion (ℓ) and the key size for after an actual execution of the algorithm that minimized the key size. The last two columns show the depth of the recursion and the key size as predicted by Proposition 1 for $\ell = \lceil \log |x| \rceil$.

Table 1. Key length in bytes for some values of $|x|$

$\|x\|$	ℓ (exact)	$\|k_b\|$ (exact)	ℓ (Prop. 1)	$\|k_b\|$ (Prop. 1)
20	2	1298	3	4513
40	4	5000	4	20003
80	5	18906	5	72941
160	6	61943	6	241256

Corollary 1. *In the special case of $y = 1$, by setting $\ell = \lceil \log |x| \rceil$ each key that Gen_ℓ outputs is of length at most $8(\kappa+1)\,|x|^{\log 3}$ bits.*

Proposition 2. *[Correctness] The scheme $DPF_\ell = (Gen_\ell, Eval_\ell)$ is* correct *as defined for a distributed point function, for every $\ell = 0, 1, \ldots$.*

The correctness claim for $\mathrm{DPF}_\ell = (Gen_\ell, Eval_\ell)$ is proved using induction similarly to the correctness proof of $\mathrm{DPF}_1 = (Gen_1, Eval_1)$.

The next step we take is analysis of the computational complexity of $Eval_\ell$ and Gen_ℓ. That complexity depends on the computational complexity of the PRG G.

Notation 4. *Since G is a PRG, it stretches a κ-bit seed s to an n-bit string $G(s)$ in polynomial time in n. Let $\gamma \geq 1$ be a constant such that computing $G(s)$ with additional $O(n)$ work can be done in time at most n^γ. We need γ as a*

bound on the work that Gen_ℓ performs aside from its recursive calls. Therefore, it can be defined exactly such that n^γ is a bound on the work that $Gen_\ell(x, y)$ does in lines $1 - 10, 12$ and $14 - 16$, where n is the length of the output, i.e. $n = 2 \cdot 3^\ell (2^{|x|})^{\frac{1}{2^\ell}} \cdot (4(\kappa+1))^{\frac{2^\ell - 1}{2^\ell}} |y|$.

Proposition 3. *If $|y| \leq \kappa + 1$ then the computational complexity of $Eval_\ell(x, y)$ is at most $3^{\ell+1}[(2^{|x|})^{\frac{1}{2^\ell}} \cdot (4(\kappa+1))^{\frac{2^\ell-1}{2^\ell}} |y|^{\frac{1}{2^\ell}}]^\gamma$ for any $x \in \{0,1\}^*$ and $\ell \in \mathbb{N}$.*

Proof Sketch. We prove the claim by induction on ℓ. For the base case, $\ell = 0$, the claim is obvious. For the induction step, the work that $Eval_\ell$ does can be divided into three parts. The first part is made up of the recursive call in line 8 to $Eval_{\ell-1}(\sigma, i', \kappa + 1)$, the second includes the recursive call in line 10 to $Eval_{\ell-1}(v, j', |y'|)$ and the third part of the algorithm is made up of all the operations apart from the two recursive calls. Most of the work in the third part is done in line 9, which includes an expansion of a seed by the PRG G.

By induction and the definition of γ, the computational complexity for each of the three parts is at most

$$3^\ell [(2^{|x|})^{\frac{1}{2^\ell}} \cdot (4(\kappa+1))^{\frac{2^\ell-1}{2^\ell}} |y|^{\frac{1}{2^\ell}}]^\gamma.$$

□

Proposition 4. *If $|y| \leq \kappa + 1$ then the computational complexity of $Gen_\ell(x, y)$ is at most $8 \cdot 3^{\ell+2}[(2^{|x|})^{\frac{1}{2^\ell}} \cdot (\kappa+1)]^\gamma$ any $x \in \{0,1\}^*$ and $\ell \in \mathbb{N}$.*

The proof is omitted and will appear in a full version of the paper.

In the next part of the analysis we show that DPF_ℓ is secret. We do so by proving that each key k_b is pseudo-random and can therefore be simulated.

Notation 5. *Let $L(Gen_\ell(x, y))$ denote the length of k_0 and k_1 induced by $(k_0, k_1) \leftarrow Gen_\ell(x, y)$. Let $T(Gen_\ell(x, y))$ denote the computational complexity of $Gen_\ell(x, y)$.*

Notation 6. *Let R denote the distribution induced on r_b and S denote the distribution induced on σ_b by the coin tosses of $Gen_\ell(x, y)$. Let U_n denote the uniform distribution on strings of length n and let $G(U_\kappa)$ denote the distribution induced by choosing a random string of length κ and extending it by G to $|r_b|$ bits.*

Notation 7. *Denote by K the distribution of the key k_b. Let $S||S'$ denote the distribution of $\sigma_b||s_i^{1-b}$. Let $CW^{(1)} \leftarrow G(s)+r_{1-b}$, where s is a uniformly random seed of length κ. Let $k_b^{(1)}$ be identical to k_b, except for replacing $CW_{t_i^{1-b}}$ by $CW^{(1)}$ and let $K^{(1)}$ denote the distribution induced by $k_b^{(1)}$. Let $CW^{(2)} \leftarrow u^{(2)} + r_{1-b}$, where $u^{(2)}$ is a uniformly random string of length $|r_{1-b}|$. Let $k_b^{(2)}$ be identical to $k_b^{(1)}$, except for replacing $CW^{(1)}$ by $CW^{(2)}$ and let $K^{(2)}$ denote the distribution induced by $k_b^{(2)}$. Let $CW^{(3)} \leftarrow G(s_i^b) + u^{(3)}$, where $u^{(3)}$ is a uniformly random string of length $|r_b|$. Let $k_b^{(3)}$ be identical to $k_b^{(2)}$, except for replacing $CW_{t_i^b}$ by $CW^{(3)}$ and let $K^{(3)}$ denote the distribution induced by $k_b^{(3)}$.*

Proposition 5. *Let $\ell \in \mathbb{N}$, let $x \in \{0,1\}^*$ and let $G : \{0,1\}^\kappa \longrightarrow \{0,1\}^\alpha$ be a $T(\kappa), \varepsilon(\kappa)$-pseudo-random generator, for $\alpha = 4(\kappa+1) \cdot 3^\ell (2^{|x|})^{\frac{1}{2^\ell}}$. Then, for $b = 0,1$, the distribution K on the outputs k_b of $Gen_\ell(x,y)$ is $(T(\kappa) - T(Gen_\ell(x,y)), \frac{1}{2}(3^\ell - 1)\varepsilon(\kappa))$-computationally indistinguishable from $U_{L(Gen_\ell(x,y))}$.*

Proof Sketch: We use induction on ℓ to prove the statement. In the base of the induction, $\ell = 0$, the key k_b is distributed uniformly and therefore the claim is obvious. In the induction step, we use a hybrid argument on the ensembles induced from the distributions $K, K^{(1)}, K^{(2)}, K^{(3)}$ and $U_{|k_b|}$ when x ranges over $\{0,1\}^*$.

Proposition 6. *Let $x, y \in \{0,1\}^*$, let $\kappa = \max\{|x|, |y|\}$, let $\ell = \lceil \log |x| \rceil$ and let $G : \{0,1\}^\kappa \longrightarrow \{0,1\}^\alpha$ be a $T(\kappa), \varepsilon(\kappa)$-pseudo-random generator, for $\alpha = 4(\kappa+1) \cdot 3^\ell (2^{|x|})^{\frac{1}{2^\ell}}$.*

1. *If G is a pseudo-random generator, i.e. $T(\kappa) \geq q(\kappa)$ and $\varepsilon(\kappa) \leq 1/q'(\kappa)$ for any two polynomials $q(\cdot), q'(\cdot)$ then $DPF_\ell = (Gen_\ell(x,y), Eval_\ell(x,y))$ satisfies the* secrecy *requirement for a distributed point function.*
2. *If G is an exponentially strong PRG G, i.e. $T(\kappa) = 2^{\kappa^c}$ and $\varepsilon(\kappa) = 2^{-\kappa^c}$ for some constant $c > 0$ then $DPF_\ell = (Gen_\ell(x,y), Eval_\ell(x,y))$ satisfies the* exponential secrecy *requirement for a distributed point function.*

Proof Sketch. For any polynomial $T'(\kappa)$ we have that $T(\kappa) = T'(\kappa) + T(Gen_\ell(x,y))$ is also a polynomial in κ. If G is a PRG, its output is $(T(\kappa), \varepsilon(\kappa))$-computationally indistinguishable from U_n for a negligible function $\varepsilon(\kappa)$. Setting $\varepsilon'(\kappa) = \frac{1}{2}(3^\ell - 1)\varepsilon(\kappa) \leq \frac{1}{2}(3\,|x|^{\log 3} - 1)\varepsilon(\kappa)$, we have by Proposition 5 that k_b, the output of $Gen_\ell(x,y)$, is $(T'(\kappa), \varepsilon'(\kappa))$-computationally indistinguishable from the uniform distribution, which satisfies the secrecy property. The exponential secrecy property is proved using a similar argument. □

Theorem 1. *The existence of a one-way function implies the existence of a DPF. If the one-way function is exponentially strong, so is the DPF.*

Proof. We show that the construction of $\mathrm{DPF}_{\lceil \log |x| \rceil}$, is a DPF assuming the existence of a one-way function. By Proposition 2, DPF_ℓ is *correct* for any $\ell = 0, 1, \ldots$.

A series of works, beginning with [22] and currently culminating in [31] establish that the existence of one-way functions implies the existence of pseudo-random generators and furthermore that the existence of an exponentially hard one-way function implies the existence of an exponentially strong PRG.

Proposition 6 proves that given a security parameter $\kappa = \max\{|x|, |y|\}$, the scheme $\mathrm{DPF}_{\lceil \log |x| \rceil}$ satisfies the secrecy requirement if G is a PRG and satisfies the exponential secrecy requirement if G is an exponentially strong PRG. Therefore, the existence of one-way functions implies that $\mathrm{DPF}_{\lceil \log |x| \rceil}$ satisfies the secrecy requirement and the existence of an exponentially strong one-way function implies that $\mathrm{DPF}_{\lceil \log |x| \rceil}$ satisfies the exponential secrecy requirement.

Propositions 4 and 3 prove that Gen_ℓ and $Eval_\ell$ are polynomial time algorithms for $\ell = \lceil \log |x| \rceil$. These results together satisfy Definition 1.

The converse of the first statement of Theorem 1, that DPF implies a one-way function is also true, see Theorem 5.

4 Applications

4.1 Computationally Private Information Retrieval

In the problem of private information retrieval (PIR) [8] a user wishes to retrieve the i-th bit of an n-bit string $z = (z_1, \ldots, z_n)$. This string, called the database, is held by several different non-colluding servers. The goal of the user is to obtain the bit z_i without revealing any information on i to any individual server. The index i can be hidden in an information-theoretical or computational sense, in which case the scheme is called a CPIR scheme. A formal definition of a two-server CPIR is as follows.

Definition 2. *A two-server CPIR protocol involves two servers S_0 and S_1, each holding the same n-bit database z, and a user. The protocol $\mathcal{P} = (Dom_Q, Dom_A, Q, A, M)$ consists of a query domain Dom_Q, an answer domain Dom_A, and three polynomial-time algorithms: a probabilistic query algorithm Q, an answering algorithm A and a reconstruction algorithm M. To retrieve z_i, the i-th bit of z, the user computes two queries $Q(n, i, r) = (q_0, q_1) \in (Dom_Q)^2$ using random coin tosses r. For each $b, b \in \{0, 1\}$, the server S_b receives q_b and computes an answer $a_b = A(b, z, q_b) \in Dom_A$. The user receives a_0 and a_1 and recovers z_i by applying the reconstruction algorithm $M(i, r, a_0, a_1)$. A two-server CPIR protocol must satisfy the following requirements:*

Correctness: *For every $n, n \in \mathbb{N}$, every $z \in \{0,1\}^n$ and every $i \in \{1, \ldots, n\}$ and given a random string r*

$$Pr[(q_0, q_1) \leftarrow Q(n, i, r) : M(i, r, A(0, z, q_0), A(1, z, q_1)) = z_i] = 1.$$

Secrecy: *Let $D_{b,\lceil \log n \rceil, i}$, $b \in \{0,1\}$, $n \in \mathbb{N}$ and $i \in \{1, \ldots, n\}$, denote the probability distribution on q_b induced by $(q_0, q_1) \leftarrow Q(n, i, r)$. There exists a PPT algorithm Sim such that the following distribution ensembles are computationally indistinguishable:*

1. $\{Sim(b, \lceil \log n \rceil)\}_{b \in \{0,1\}, n \in \mathbb{N}}$
2. $\{D_{b,\lceil \log n \rceil, i}\}_{b \in \{0,1\}, n \in \mathbb{N}, i \in \{1, \ldots, n\}}$.

The main measure of the efficiency of a PIR scheme is its communication complexity, which is the maximum number of bits exchanged between the user and servers over the choices of z, i and r. The query complexity is $\log |Dom_Q|$ and the answer complexity is $\log |Dom_A|$.

In the following theorem we show how to turn a DPF scheme into a two-server CPIR scheme.

Theorem 2. *Let $DPF = (Gen, Eval)$ be a distributed point function and let $m(b, |x|, |y|) = \max\{|k_b| : b \in \{0,1\}, (k_0, k_1) \leftarrow Gen(x, y)\}$. There exists a CPIR scheme that for every n has query complexity $m(b, \log n, 1)$, answer complexity 1 and thus total communication complexity $2m(b, \log n, 1) + 2$.*

The full proof is omitted, but it relies on the procedure outlined in the introduction that involves computing $Eval(q_b, j, 1)$ separately for every $j = 1, \ldots, n$. The computational complexity of such a procedure is n times the computation required for a single invocation of $Eval$. Specifically, about $n \cdot |k_b|$ pseudo-random bits need to be computed. However, there is a more efficient alternative that for each node in the PRF tree rooted by k_b computes all of the node's children instead of a single one. This alternative results in computing less than $n + 2n^{1/2}$ pseudo-random bits.

By using the construction of $\mathrm{DPF}_{\lceil \log \log n \rceil}$ from Section 3, we get the following:

Corollary 2. *The existence of a one-way function f implies the existence of a two-server CPIR scheme with query complexity $O(\kappa(\log n)^{\log 3})$ and answer complexity 1. The term $\kappa = \kappa(n)$ is the length of a seed for a PRG $G : \{0,1\}^{\kappa(n)} \to \{0,1\}^n$ that is implied by f being a one-way function.*

4.2 Private Information Retrieval by Keywords

In the problem of Private Information Retrieval by Keywords [9,15] several servers hold a copy of the same set of n words, $w_1, \ldots, w_n$, of the same length ν. A user holds a word w and wishes to find out whether $w \in \{w_1, \ldots, w_n\}$ without providing information to any individual server on w.

Theorem 3. *The existence of a one-way function f implies the existence of a two-server scheme for private information retrieval by keywords with query length $O(\kappa \nu^{\log 3})$ and answer length 1. The term $\kappa = \kappa(n)$ is the length of a seed for a PRG $G : \{0,1\}^{\kappa(n)} \to \{0,1\}^n$ that is implied by f being a one-way function.*

Proof. The user generates two queries (q_0, q_1) by running $(q_0, q_1) \leftarrow Gen_{\lceil \log n \rceil}(w)$. Upon receiving a query q_b, the server S_b returns $a_b = \sum_{j=1}^{n} Eval(q_b, w_j) \bmod 2$ as its answer. The reconstruction algorithm M returns $a_0 + a_1 \bmod 2$. The correctness, secrecy and query length are proved in the same way as in Theorem 2.

The above protocol can be efficiently extended to the case where each keyword w_i has an associated nonzero payload p_i of length γ and the user should output p_i if w_i is in the database and output 0 otherwise. This is done by having each server respond with $a_b = \sum_{j=1}^{n} p_j \cdot Eval(q_b, w_j)$ where addition is in $\mathbb{F}_2^{\gamma}$. In the full version we will describe the application to PIR writing.

4.3 Worst-Case to Average-Case Reduction

A worst-case to average-case reduction transforms any average-case algorithm for one language L_2 into an algorithm that with good probability works on all inputs

for another language L_1. We show how our scheme for a DPF translates into a worst-case to average-case reduction for languages in PSPACE and EXPTIME. Our result improves on known worst-case to average-case reductions for PSPACE and EXPTIME by requiring our worst-case algorithm to make only a constant number of queries to an average-case algorithm.

An interesting way to view our result is as a transformation of a heuristic algorithm to a worst-case algorithm. Given a heuristic algorithm A that correctly decides a certain language in PSPACE or EXPTIME on a $1-\delta$ fraction of the inputs we show that for *any* language L in PSPACE or EXPTIME there exists an algorithm that decides L with good probability on *any* input and makes only two calls to the heuristic algorithm.

We complement a negative result from Theorem 3 of Watson's paper [30]. That theorem rules out a similar result where A is unbounded, thus applying to a reduction of "type 1" in his classification. We show that a reduction of "type 2" *is* possible (under standard assumptions), thus his negative result for type 1 reductions is tight.

We assume the existence of an exponentially hard one-way function. Given such a function there exists an exponentially strong PRG G [31].

Theorem 4. *Let $\delta < 1/2$ and assume the existence of an exponentially hard one-way function. Then,*

1. *There exist a language L_2 in EXPTIME (PSPACE) and a constant $c > 0$ such that if there is an algorithm A, which for any κ runs in time at most 2^{κ^c} and correctly decides a $1-\delta$ fraction of the instances of $L_2 \cap \{0,1\}^\kappa$ then for any language L in EXPTIME (or PSPACE) there exists a probabilistic, polynomial time oracle algorithm R such that R^A decides any instance in $L \cap \{0,1\}^\kappa$ with error at most $2\delta + 2^{-\kappa^c}$ and with only two queries to A.*
2. *There exist a language L_2 in EXPTIME (PSPACE) and a constant $c > 0$ such that if there is an algorithm A running in time at most 2^{κ^c} that correctly decides a $1-\delta$ fraction of the instances of $L_2 \cap \{0,1\}^\kappa$ and returns $\perp$ on any other instance then for any language L in EXPTIME (or PSPACE) there exists a probabilistic, polynomial time oracle algorithm R such that R^A decides any instance in $L \cap \{0,1\}^\kappa$ with only $1/(1-2\delta-2^{-\kappa^c})$ expected queries to A.*

The proof is omitted and will appear in the full version of the paper

5 Efficient Implementation DPF with Large Output

Algorithms 3 and 4 assume that $|y| \leq \kappa+1$. Given x and y, seeds must be chosen to be of length κ such that $|y| \leq \kappa+1$. For concrete applications in which the length κ is given, a more efficient implementation is possible.

Let $DPF = (Gen, Eval)$ be a distributed point function for any $x \in \{0,1\}^*$ and any $y, |y| \leq \kappa+1$. Define $DPF' = (Gen', Eval')$ on any $x, y \in \{0,1\}^*$ as follows. Let $G : \{0,1\}^\kappa \to \{0,1\}^{|y|}$ be a PRG. $Gen'(x,y)$ checks if y is the all-zero

string. If it is, then $Gen'(x, y)$ executes $(k_0, k_1) \leftarrow Gen(x, 0^{\kappa+1})$, chooses a random element $r \in \mathbb{F}_{2^{|y|}}$ and outputs a key pair $(k_0^{'} = k_0||r, k_1^{'} = k_1||r)$. If y is not zero then $Gen'(x, y)$ chooses a random seed $s \in \{0,1\}^{\kappa}$ and executes $(k_0, k_1) \leftarrow Gen(x, s)$. It then computes $r \leftarrow y \cdot (G(Eval(k_0, x, |y|)) + G(Eval(k_1, x, |y|)))^{-1}$ over $\mathbb{F}_{2^{|y|}}$ and outputs a key pair $(k_0^{'} = k_0||r, k_1^{'} = k_1||r)$.

$Eval'(k||r, x, |y'|)$ returns as its output $r \cdot G(Eval(k, x, |y'|))$ with computation over $\mathbb{F}_{2^{|y|}}$. The correctness is easy to verify and the secrecy of the DPF' follows from the secrecy of DPF and from G being a PRG.

6 DPF Implies OWF

In this section we show that the existence of a distributed point function implies the existence of a one-way function. As Theorem 2 proves, the existence of a DPF implies the existence of a binary two-server CPIR scheme with query length $o(n)$. We rely on the following lemma.

Lemma 1. *[19] Suppose there exist a pair of distribution ensembles $\{X_n\}_{n\in\mathbb{N}}$, $\{Y_n\}_{n\in\mathbb{N}}$ and a polynomial $p(\cdot)$ such that X_n and Y_n can be sampled in time polynomial in n; The statistical distance between X_n and Y_n is greater than $1/p(n)$ for all n, and X_n and Y_n are computationally indistinguishable. Then a one-way function exists.*

Theorem 5. *Suppose that there is a two-server CPIR protocol with query length $o(n)$ and binary answers. Then a one-way function exists.*

Proof. Denote the two servers by S_0, S_1 and define the following distribution ensembles:

- $X_n = (i, b, Q_b(n, i))$ where $i \in \{1, \ldots, n\}$ is a random index, $b \in \{0, 1\}$ is a random server index, and $Q_b(n, i)$ is a random PIR query to server S_b on a database of size n and index i.
- $Y_n = (i, b, Q_b(n, i'))$ where $i' \in \{1, \ldots, n\}$ is a random index picked independently of i.

The secrecy property of the PIR protocol implies that X_n and Y_n are indistinguishable. They are also efficiently samplable, since the query generation in the PIR protocol is efficient. It remains to show that they are statistically far.

Suppose towards contradiction that X_n, Y_n are $(1/n^2)$-close for infinitely many n. It follows that, for infinitely many n, there are no i, j, b such that $Q_b(n, i)$ and $Q_b(n, i')$ are more than 0.1-far. Thus, Q defines for infinitely many n an information-theoretic 2-server PIR protocol with statistical privacy error $\epsilon \leq 0.1$, sublinear-size queries, and binary answers. Such a protocol is known not to exist (see [32], Thm 8).

References

1. Beaver, D., Feigenbaum, J.: Hiding instances in multioracle queries. In: Choffrut, C., Lengauer, T. (eds.) STACS 1990. LNCS, vol. 415, pp. 37–48. Springer, Heidelberg (1990)

2. Babai, L., Fortnow, L., Nisan, N., Wigderson, A.: BPP has subexponential time simulations unless EXPTIME has publishable proofs. In: Proc. of the Sixth Annual Structure in Complexity Theory Conference, pp. 213–219 (1991)
3. Beigel, R., Fortnow, L., Gasarch, W.I.: A tight lower bound for restricted pir protocols. Computational Complexity 15(1), 82–91 (2006)
4. Beimel, A., Ishai, Y., Kushilevitz, E., Orlov, I.: Share Conversion and Private Information Retrieval. In: IEEE Conference on Computational Complexity 2012, pp. 258–268 (2012)
5. Barkol, O., Ishai, Y., Weinreb, E.: On Locally Decodable Codes, Self-Correctable Codes, and t-Private PIR. Algorithmica 58(4), 831–859 (2010)
6. Brakerski, Z., Vaikuntanathan, V.: Efficient Fully Homomorphic Encryption from (Standard) LWE. In: Proceedings of FOCS 2011, pp. 97–106 (2011)
7. Chor, B., Gilboa, N.: Computationally Private Information Retrieval. In: Proc. of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC 1997), pp. 304–313 (1997)
8. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. Journal of the ACM (JACM) 45(6), 965–981 (1998)
9. Chor, B., Gilboa, N., Naor, M.: Private information retrieval by keywords, TR CS0917, Dept. of Computer Science, Technion (1997)
10. Cachin, C., Micali, S., Stadler, M.: Computationally Private Information Retrieval with Polylogarithmic Communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999)
11. Desmedt, Y.: Society and Group Oriented Cryptography: A New Concept. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 120–127. Springer, Heidelberg (1988)
12. Desmedt, Y., Frankel, Y.: Threshold Cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg (1990)
13. Di Crescenzo, G., Malkin, T., Ostrovsky, R.: Single Database Private Information Retrieval Implies Oblivious Transfer. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 122–138. Springer, Heidelberg (2000)
14. Efremenko, K.: 3-query locally decodable codes of subexponential length. In: Proc. of the 41st Annual ACM Symposium on Theory of Computing (STOC 2009), pp. 39–44 (2009)
15. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (2005)
16. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proc. of the 41st Annual ACM Symposium on Theory of Computing (STOC 2009), pp. 169–178 (2009)
17. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer, Heidelberg (2005)
18. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM (JACM) 33(4), 792–807 (1986)
19. Goldreich, O.: A Note on Computational Indistinguishability. Inf. Process. Lett. 34(6), 277–281 (1990)
20. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press (2000)

21. Haitner, I., Harnik, D., Reingold, O.: Efficient pseudorandom generators from exponentially hard one-way functions. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006 Part II. LNCS, vol. 4052, pp. 228–239. Springer, Heidelberg (2006)
22. Hastad, J., Impagliazzo, R., Levin, L., Luby, M.: A Pseudorandom Generator from any One-way Function. SIAM J. Comput. 28(4), 1364–1396 (1999)
23. Holenstein, T.: Pseudorandom Generators from One-Way Functions: A Simple Construction for Any Hardness. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 443–461. Springer, Heidelberg (2006)
24. Kushilevitz, E., Ostrovsky, R.: Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In: Proc. of the 38th Symposium on Foundations of Computer Science (FOCS 1997), pp. 364–373 (1997)
25. Lipmaa, H.: An Oblivious Transfer Protocol with Log-Squared Communication. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 314–328. Springer, Heidelberg (2005)
26. Ostrovsky, R., Shoup, V.: Private information storage. In: In Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 294–303. ACM (1997)
27. Ostrovsky, R., Skeith III, W.E.: Private Searching on Streaming Data. J. Cryptology 20(4), 397–430 (2007); Shoup, V.: Private information storage. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 294–303. ACM (1997)
28. Shamir, A.: How to Share a Secret. CACM 22(11), 612–613 (1979)
29. Stern, J.P.: A New Efficient All-Or-Nothing Disclosure of Secrets Protocol. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 357–371. Springer, Heidelberg (1998)
30. Watson, T.: Relativized Worlds without Worst-Case to Average-Case Reductions for NP. Journal of ACM Transactions on Computation Theory (TOCT) 4(3), Article No. 8 (September 2012)
31. Vadhan, S., Zheng, C.: Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In: Proc. of the 44th Annual ACM Symposium on the Theory of Computing (STOC 2012), pp. 817–836 (2012)
32. Wehner, S., de Wolf, R.: Improved Lower Bounds for Locally Decodable Codes and Private Information Retrieval. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1424–1436. Springer, Heidelberg (2005)
33. Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. In: Proc. of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007), pp. 266–274 (2007)