Cryptography Meets Algorithms (15893) Lecture Notes

Lecture 6: 2-Server PIR from Distributed Point Functions

Scribe: Trevor Leong

February 20, 2024

In earlier lectures, we learned an information-theoretic 2-server PIR scheme by Dvir and Gopi [DG16], which achieves $n^{O(\sqrt{\log\log n/\log n})}$ communication per query. In this lecture, we will show how to get a 2-server PIR with logarithmic communication relying only on a pseudorandom generator (PRG). This scheme is also interesting from a practical perspective since in practice, we can use AES to realize the PRG, and modern CPUs have hardware acceleration for evaluating AES.

Recall that from our undergraduate cryptography course, we know that the existence of a PRG is equivalent to the existence of a one-way function (OWF) [HILL99]. Also, from an earlier lecture, we learned that any 1-server classical PIR scheme with non-trivial bandwidth implies Oblivious Transfer which cannot be constructed in a blackbox manner from OWF [IR89]. Therefore, the scheme we will talk about today is in the 2-server setting.

1 Preliminary: Pseudorandom Generator

We will rely on a pseudorandom generator (PRG), which takes in a short random seed and expands the seed to a longer pseudorandom string.

Definition 1 (PRG)**.** *Let $\ell(\cdot)$ be a polynomial and let $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a deterministic polynomial-time algorithm. G is a* PRG *if it has the following properties:*

- ***Expansion:*** *$\forall n, \ell(n) > n$.*
- ***Pseudorandomness:*** *for any probabilistic polynomial-time distinguisher D, there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that*

$$\left| \Pr_{r \xleftarrow{\$} \{0,1\}^{\ell(n)}}[D(r) = 1] - \Pr_{s \xleftarrow{\$} \{0,1\}^{n}}[D(G(s)) = 1] \right| \leq \mathsf{negl}(n)$$

2 Distributed Point Function

Definition 2 (Point function)**.** *A point function parametrized by some point $x \in \{0,1\}^\ell$ is a function that evaluates to 1 at x, and evaluates to 0 everywhere else. We will henceforth use the notation $P_x\{0,1\}^\ell \to \{0,1\}$ to denote a point function. By definition, $P_x(x) = 1$ and $P_x(x') = 0$ for $x' \neq x$.*

Boyle, Gilboa and Ishai [BGI16] introduced the concept of a distributed point function. A distributed point function is a functional secret-sharing of a point function. In this lecture, we will specifically focus on a *2-way* secret sharing of a point function. Essentially, given

some point function P_{x^*}, one can "secretly share" the function to two keys k_L, k_R. Then, for party $t \in \{L, R\}$ which receives the key k_t, it can evaluate the function on any point x and get a share of the outcome denoted $\mathsf{Eval}(k_t, x)$. It is guaranteed that in every point x, $\mathsf{Eval}(k_L, x) \oplus \mathsf{Eval}(k_R, x) = P_{x^*}(x)$. In other words, it is possible to combine the two outcome-shares to reconstruct the evaluation of the point function at any point. Finally, security of the DPF requires that each party $t \in \{L, R\}$ does not learn the "special point" (i.e., x^*) given its individual key k_t.

Definition 3 (2-share DPF)**.** *A DPF is a pair of possibly randomized algorithms (*Gen,Eval*) with the following syntax:*

- $\mathsf{Gen}(1^\lambda, x^*)$*: Outputs a pair of keys* k_L, k_R*.*
- $\mathsf{Eval}(1^\lambda, k, x)$*: Outputs the evaluation outcome* $y \in \{0, 1\}$*.*

Correctness. *Correctness requires that for any* λ*, any* ℓ*, any* $x^*, x \in \{0,1\}^\ell$*,*

$$\Pr\left[k_L, k_R \leftarrow \mathsf{Gen}(1^\lambda, x^*) : \mathsf{Eval}(k_L, x) \oplus \mathsf{Eval}(k_R, x) = P_{x^*}(x)\right] = 1$$

Security. *Security requires that there exists a probabilistic polynomial-time simulator* Sim*, such that for any* $\ell = \ell(\lambda)$ *that is a polynomial function in* λ*, and any* $x^* \in \{0,1\}^{\ell(\lambda)}$*, the following experiments are computationally indistinguishable for both* $t = L$ *and* $t = R$*:*

- $\mathsf{Real}(1^\lambda, x^*)$*:* $k_L, k_R \leftarrow \mathsf{Gen}(1^\lambda, x^*)$ *and output* k_t*;*
- $\mathsf{Ideal}(1^\lambda)$*: Output* $\mathsf{Sim}(1^\lambda, \ell)$*.*

Intuitively, security requires that the any individual key k_L or k_R can be simulated without knowledge of the special point x^*.

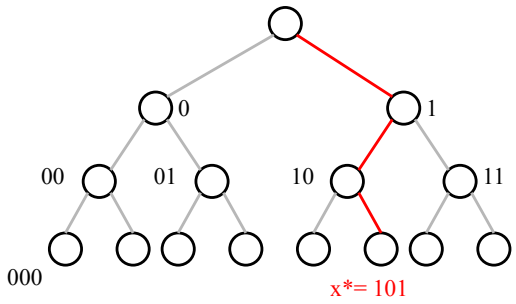
3 DPF $\Longrightarrow$ 2-Server PIR

Given a DPF scheme henceforth denoted $(\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$, we can construct a 2-server PIR scheme as follows. Henceforth we use $\mathsf{DB} \in \{0,1\}^n$ to denote the database.

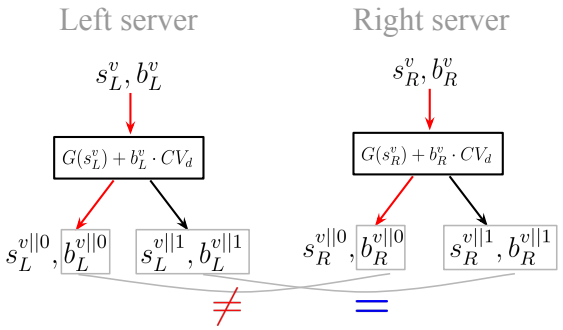
1. Given the query i, the client computes $(k_L, k_R) \leftarrow \mathsf{DPF.Gen}(1^\lambda, i)$
2. The client sends k_L to the left server and sends k_R to the right server.
3. Each server $t \in \{L, R\}$ receives k_t, and replies $y_t := \bigoplus_{j \in [n]} \mathsf{DPF.Eval}(k_t, j) \cdot \mathsf{DB}[j]$;
4. the client receives y_L and y_R from the two servers respectively, and outputs $y_L \oplus y_R$.

Correctness. It is not hard to check that the answer output by the client is correct by the DPF's correctness:

$$\begin{aligned}
y_0 \oplus y_1 &= \left(\bigoplus_j \mathsf{DB}[j] \cdot \mathsf{DPF.Eval}(k_0, j)\right) \oplus \left(\bigoplus_j \mathsf{DB}[j] \cdot \mathsf{DPF.Eval}(k_1, j)\right) \\
&= \left(\bigoplus_j \mathsf{DB}[j] \cdot (\mathsf{DPF.Eval}(k_0, j) \oplus \mathsf{DPF.Eval}(k_1, j))\right) \\
&= \left(\bigoplus_j \mathsf{DB}[j] \cdot P_i(j)\right) \\
&= \mathsf{DB}[i].
\end{aligned}$$



(a) The binary tree structure used in DPF. Each node has a unique name. The path from the root to the leaf node x^* is the "special path".



(b) Expansion during the evaluation algorithm. The key generation computes the correction vector CV in a way that guarantees the following: for any u not on the special path, $(s_L^u, b_L^u) = (s_R^u, b_R^u)$; and for any u on the special path, $b_L^u \neq b_R^u$ and moreover, the pair (s_L^u, s_R^u) is indistinguishable from two independent random strings.

Figure 1: Illustration of the DPF construction.

Security. Security of the PIR follows directly from the security of the DPF.

4 DPF Construction

We describe the elegant DPF construction of Boyle, Gilboa, and Ishai [BGI16].

We now show how to construct an efficient DPF based on a pseudorandom generator G:

$$G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}.$$

The algorithm is based on a binary tree expansion idea.

Binary tree structure. Suppose we want to evaluate the DPF at n points denoted $0, 1, \ldots, n-1$, and we assume that n is a power of 2.

Imagine that there is a binary tree as depicted in Figure 1a. Each node in the tree has a name: the root's name is empty, and the two children of a node v are named $v||0$ and $v||1$, respectively. Henceforth, we say that the root is at *depth* 0, the leaves are at depth $\log n$, and so on. Each leaf node corresponds to a point in $\{0, 1, \ldots, n-1\}$. In particular, it helps to express each point in a binary representation. For a point function P_{x^*}, the path from the root to the leaf node x^* is called the special path highlighted in red in the figure.

Key structure and evaluation algorithm. The DPF's Gen algorithm outputs two keys $k_L = ((s_L, b_L), \mathsf{CV})$ and $k_R = ((s_R, b_R), \mathsf{CV})$, where

- s_L and s_R are both λ-bit PRG seeds;
- $b_L, b_R \in \{0,1\}$ are flags indicating whether correction is necessary during the key expansion (see use of the correction vector later in the algorithm). It is guaranteed that $b_L \neq b_R$; and
- $\mathsf{CV} = (\mathsf{CV}_1, \ldots, \mathsf{CV}_{\log n})$ is a correction vector.

We will focus on the left server's perspective for describing the evaluation algorithm. The right server's algorithm is the same except that its input is k_R instead of k_L. Imagine that initially, the root of the tree is associated with the pair (s_L, b_L) which comes from k_L. Starting

from the root, we will perform a key expansion to compute a pair (s_L^v, b_L^v) for every node v in the tree. Each coordinate of the correction vector $\mathsf{CV}_1, \ldots, \mathsf{CV}_{\log n}$ will be consumed at each different level of the tree during the key expansion process.

More specifically, suppose some node v has the pair (s_L^v, b_L^v), we can compute the corresponding values at its two children $v||0$ and $v||1$ as follows where d denotes the depth of v's children:

$$(s_L^{v||0}, b_L^{v||0}), (s_L^{v||1}, b_L^{v||1}) \leftarrow G(s_L^v) \oplus \begin{cases} \mathbf{0} & \text{if } b_L^v = 0; \\ \mathsf{CV}_d & \text{if } b_L^v = 1. \end{cases}$$

Notice that the correction component CV_d is only applied if $b_L^v = 1$.

At the end of the expansion process, for each leaf node x, let b_L^x and b_R^x be the two bits associated with the leaf x output by the left and right servers, respectively. The outcome of the DPF at point x is then $b_L^x \oplus b_R^x$.

Key generation algorithm. The key generation algorithm samples random $s_L \xleftarrow{\$} \{0,1\}^\lambda$ and $s_R \xleftarrow{\$} \{0,1\}^\lambda$ at the root nodes for the left and right servers, respectively. It chooses a random bit $b_L \xleftarrow{\$} \{0,1\}$ and sets $b_R = b_L \oplus 1$. Then, it will choose the correction vector $\mathsf{CV}_1, \ldots, \mathsf{CV}_{\log n}$ in a way such that the following **invariants** are guaranteed for each tree node v:

- For every tree node v that is not on the special path leading to x^*, it must be that $(s_L^v, b_L^v) = (s_R^v, b_R^v)$.
- For every tree node v that is on the special path leading to x^*, it must be that $b_L^v \neq b_R^v$, and moreover, the pair (s_L^v, s_R^v) is indistinguishable from two independent λ-bit random strings.

Note that for some node v, if $(s_L^v, b_L^v) = (s_R^v, b_R^v)$ is already guaranteed, then for any node u that is in the subtree of v, $(s_L^u, b_L^u) = (s_R^u, b_R^u)$ is automatically guaranteed because the left and right servers will behave identically when the apply the same expansion algorithm to compute all values in the subtree of v. Therefore, in the key generation algorithm, we can compute the correction vector CV using only the special path, to maintain the aforementioned invariants.

The detailed key generation algorithm is specified in Figure 2. It is not hard to see that the key size is $\lambda + 1 + \log n \cdot (2\lambda + 2) = O_\lambda(\log n)$.

Analysis. The correctness can be verified by inductively checking that the aforementioned invariants hold at every level.

Security can also be proven inductively. We prove left-server security below since right-server security is symmetric. Henceforth, we use $x^*[:d]$ to denote the first d bits of the binary representation of x^*. Due to the security of the pseudorandom generator G, it suffices to prove that if $G(s)$ outputs a truly random string given a random seed s, then $(s_L, b_L, \mathsf{CV}_1, \ldots, \mathsf{CV}_d)$ is a uniform random string.

- *Base case.* For the base case, observe that the terms (s_L, b_L) are random and independent of s_R.
- *Induction step.* Now, suppose that the joint distribution of $(s_L, b_L, \mathsf{CV}_1, \ldots, \mathsf{CV}_{d-1})$ and $s_R^{x^*[:d-1]}$ are random. We want to prove that the joint distribution of $(s_L, b_L, \mathsf{CV}_1, \ldots, \mathsf{CV}_d)$ and $s_R^{x^*[:d]}$ are random. Without loss of generality, assume that $x^*[d] = 0$ since the other direction is symmetric. Let $v = x^*[:d-1]$. By our induction hypothesis, given

Generation Algorithm: $\mathsf{Gen}(1^\lambda, x^*)$

Initialization:

- Sample s_L, s_R as two λ-bit random strings.
- Sample a random bit b_L. Let $b_R = b_L \oplus 1$.
- Let $\{x^*[1], \ldots, x^*[\log n]\}$ be x^*'s binary representation.

Constructing correction vectors:

Initialize v to be an empty string.

For $d \in \{1, \ldots, \log n\}$:

– Sample a random string $r \xleftarrow{\$} \{0,1\}^\lambda$.

– Solve for $\mathsf{CV}_d \in \{0,1\}^{2\lambda+2}$ such that the following constraints are satisfied:

(1) Let $(s_L^{v||0}, b_L^{v||0}), (s_L^{v||1}, b_L^{v||1}) \leftarrow G(s_L^v) \oplus (b_L^v \cdot \mathsf{CV}_d)$; *// Expansion on the LHS*

(2) Let $(s_R^{v||0}, b_R^{v||0}), (s_R^{v||1}, b_R^{v||1}) \leftarrow G(s_R^v) \oplus (b_R^v \cdot \mathsf{CV}_d)$; *// Expansion on the RHS*

(3a) If $x^*[d] = 0$: add the following constraint:

$$(s_L^{v||0}, b_L^{v||0}, s_L^{v||1}, b_L^{v||1}) \oplus (s_R^{v||0}, b_R^{v||0}, s_R^{v||1}, b_R^{v||1}) = (r, 1, 0^\lambda, 0).$$

(3b) If $x^*[d] = 1$: add the following constraint:

$$(s_L^{v||0}, b_L^{v||0}, s_L^{v||1}, b_L^{v||1}) \oplus (s_R^{v||0}, b_R^{v||0}, s_R^{v||1}, b_R^{v||1}) = (0^\lambda, 0, r, 1).$$

– Let $v \leftarrow v||x^*[d]$.

Output: Output the following k_L, k_R:

$$k_L = (s_L, b_L, \mathsf{CV}_1, \ldots, \mathsf{CV}_{\log n})$$
$$k_R = (s_R, b_R, \mathsf{CV}_1, \ldots, \mathsf{CV}_{\log n})$$

Figure 2: DPF key generation algorithm.

$(s_L, b_L, \mathsf{CV}_1, \ldots, \mathsf{CV}_{d-1})$, s_R^v is random, and given our assumption on G, $G(s_R^v)$ is also random. By construction, we have

$$s_R^{v||0} = G(s_R^v)[:\lambda] \oplus b_R^v \cdot \mathsf{CV}_d[:\lambda]$$
$$CV_d = G(s_L^v) \oplus G(s_R^v) \oplus (r, 1, 0, 0)$$

We can conclude that the pair $(s_R^{v||0}, \mathsf{CV}_d)$ is random given $(s_L, b_L, \mathsf{CV}_1, \ldots, \mathsf{CV}_{d-1})$. In particular, the randomness of $s_R^{v||0}$ relies on the first λ bits of $G(s_R^v)$ being random, and the randomness of the first λ bits of CV_d relies on the random string $r \in \{0,1\}^\lambda$, the randomness of the remaining $\lambda + 2$ bits of CV_d relies on the randomness of the last $\lambda + 2$ bits of $G(s_R^v)$.

References

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server pir with subpolynomial communication. *Journal of the ACM (JACM)*, 63(4):1–15, 2016.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[IR89] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 44–61, New York, NY, USA, 1989. Association for Computing Machinery.