Private Information Retrieval and Distributed Point Functions

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2025

The techniques we have so far seen in this class are based on XOR/additive secret shares of values. Based on this paradigm, we have constructed secure protocols for Boolean circuits, and now even for RAM programs. Our costs in all such programs run quasi-linearly[1] in the cost of the target program. We've been able to do achieve such execution securely in the presence of a semi-honest adversary, and even in the presence of a malicious adversary who corrupts any subset of parties. Again, all of this was essentially built on the notion of secret-shared data.

Today, we will introduce an additional technique for sharing information that is useful in the context of secure computation, and also broadly in cryptographic applications in general. This technique allows parties to secret share *functions* rather than bits, and hence it is appropriately named *function secret sharing* (FSS). The motivation for introducing FSS is that it can improve the efficieny of many cryptographic tasks.

Today, we will discuss what FSS is, focusing on a *particular* kind of function secret sharing called a *distributed point function* (DPF). To motivate DPFs, we will explore a famous problem called Private Information Retrieval (PIR).

A Motivating Task: Two-Server PIR

Private Information Retrieval is a problem motivated by the desire for a *client* to securely download a piece of information from some database. In particular, suppose there are m non-colluding servers $S_0, ..., S_{m-1}$, each of whom holds some "database", which for today will just be a length-n bitstring $x \in \{0,1\}^n$. A client holds some index α, and she would like to download the database entry x_α while keeping α hidden. Formally, we want semi-honest simulation security against each server. Two notes on this model:

- We do not need to consider subsets of servers, since we assume they do not collude.
- The database x is *not* considered private, so we do not need security against the client. The only goal is to securely deliver x_α.

Moreover, we would like the client to download the database item x_α as efficiently as possible.

Remark 1 (Comparison with ORAM)**.** *PIR and ORAM at the surface appear similar, but at closer inspection the problems are quite different:*

- *ORAM considers an* amortized *notion of efficiency. Namely, the client submits many read/write queries to x and we only insist that this is efficient in the aggregate. In PIR,* each *query should be efficient.*
- *In ORAM, there is a single client whose database x is private. In PIR, the database x is public, and it is instructive to think of there being* many *clients, each of whom wishes to securely access the same x.*
- *Following the above discussion, in PIR the server cost should not scale significantly with the number of clients. Essentially, the server algorithm should be essentially stateless, keeping only the database x in memory. Note that in ORAM this is* not *the case, as if there are c clients, then the server would have to store separately the encrypted memory for* each *client.*

[1] Here, "quasi" in general hides polynomial factors of the security parameter, polylogarithmic factors of the program cost, and scaling in number of parties.

PIR is challenging. Indeed, suppose there is only one server, i.e. $m = 1$. Here, is is challenging to see how the client can obtain x_α without leaking α, unless[2] the client simply downloads *everything* from the database. Indeed, this strategy—where the client downloads everything—is called *trivial* PIR, and it is straightforwardly PIR, but inefficient.

However, once more servers are involved, it becomes more plausible that the client might be able to download data more efficiently. Indeed, there exist very beautiful protocols that two-server PIR at sublinear communication cost (e.g. the client uploads/downloads $O(n^{1/3})$ bits), even without using cryptography at all [CGKS95]. Today we will see how one can get an even better two-server PIR protocol, costing only $\tilde{O}(\lg n)$ bits, by using a DPF.

Secret-Shared One Hot Vectors. So, let us restate our setting. We have two servers, S_0 and S_1, each of whom hold a copy of some length-n public string $x \in \{0,1\}^n$. The client holds an index $\alpha \in [n]$, and she wishes to obtain x_α without leaking α.

To show a two-server protocol, it will be convenient to introduce the notion of a *one-hot encoding*. A one-hot encoding of α is string that is all zeros, except at the α-th position, where it is 1.

Definition 1. *Let $\alpha \in [n]$ be an index. The* one-hot encoding *of α is a string $\texttt{hot}(\alpha) \in \{0,1\}^n$, defined as follows:*

$$\texttt{hot}(\alpha)_i = \begin{cases} 1 & \text{if } \alpha = i \\ 0 & \text{otherwise} \end{cases}$$

Now, suppose the client constructs an XOR secret share $[\texttt{hot}(\alpha)]$, and then sends one share to each server. Each server locally computes the *inner product* of this vector with x. That is, the servers jointly compute the following:

$$\begin{aligned} \bigoplus_i (x_i \cdot [\texttt{hot}(\alpha)]_i) &= \left[\bigoplus_i x_i \cdot \texttt{hot}(\alpha)_i \right] \\ &= \left[\bigoplus_i x_i \cdot \begin{cases} 1 & \text{if } \alpha = i \\ 0 & \text{otherwise} \end{cases} \right] = [x_\alpha] \end{aligned}$$

From here, the servers can send their two bits to the client, who locally reconstructs x_α.

While this protocol works, so far, it has not improved cost: The total communication cost is still $O(n)$, since the client sends two shares of length n. However, this is still interesting in comparison to the trivial protocol, since the client now *downloads* only $O(1)$ bits.

Remark 2. *As an aside in our path to DPFs, the above protocol can be adjusted to achieve $O(\sqrt{n})$ communication. The idea here is a kind of "load balancing" between the cost of uploading data and the cost of downloading data. In particular, we can have the servers split x into $\sqrt{n}$* chunks*, each of size $O(\sqrt{n})$. The client now downloads the entire chunk corresponding to their desired bit α. Since the database now only has $\sqrt{n}$ items, the one-hot shares are also of size $\sqrt{n}$. The databases must now send back shares of the entire chosen chunk, but this is also only $O(\sqrt{n})$ bits.*

DPFs

One way to understand what a DPF is to reconsider the above protocol. The client wishes to convey to the server secret shares of some long string $\texttt{hot}(\alpha)$. A key point is this: While the string $\texttt{hot}(\alpha)$ is long, it has almost no information in it! Indeed, the entire string is described by only a $\lg n$-bit number α. This is somewhat maddening—surely the client doesn't need to send $O(n)$ bits to convey such a small amount of information!

Indeed, the client does not (assuming that pseudorandom generators exist). A DPF is essentially a way to very significantly compress the secret-sharing of the vector $\texttt{hot}(\alpha)$. Formally, we can define a DPF as follows:

[2] In fact, there are ways to achieve efficient single-server PIR from strong encryption schemes, e.g. from fully homorphic encryption (FHE). We will not look at such schemes today.

Definition 2 (Two Party DPF)**.** *A DPF is a pair of algorithms* `KeyGen`, `Eval`*:*

- `KeyGen` *takes as input a point* $\alpha \in [n]$ *and a value* $\beta \in \{0,1\}^w$*. It outputs two keys* k_0 *and* k_1*.*
- `Eval` *takes as input a key* k *and an index* i*. It outputs a* w*-bit string.*

The DPF is ***correct*** *if the following holds for all* α*,* i*, and* β*:*

$$\Pr\left[\texttt{Eval}(k_0, i) \oplus \texttt{Eval}(k_1, i) = \begin{cases}\beta & \text{if } i = \alpha \\ 0 & \text{otherwise}\end{cases} \quad \text{where} \quad (k_0, k_1) \leftarrow \texttt{KeyGen}(\alpha, \beta)\right] = 1$$

In terms of security, we consider a standard simulation-based notion, where each key k_i is indistinguishable from the output of some simulator that gets only the parameters of the DPF (i.e., w and n) [BGI16].

Although we will not see the construction until next time, a two-party DPF can be constructed assuming only that PRGs exist, and the resulting scheme will have keys of length only $O(\lg n\lambda + w)$ bits.

In other words, the DPF is a cryptographic representation of a secret-shared one-hot vector. In particular, one can interpret DPF keys as a kind of secret sharing of the following *point function*:

$$\texttt{point}_{\alpha,\beta}(i) = \begin{cases}\beta & \text{if } i = \alpha \\ 0 & \text{otherwise}\end{cases}$$

Note that point functions and one-hot encodings are subtely different. In short, a point function might have an efficient representation, whereas a one-hot encoding is long. Put another way, the scaled one-hot encoding $\beta \cdot \texttt{hot}(\alpha)$ is equal to the *truth table* for the funciton $\texttt{point}_{\alpha,\beta}$.

Given a DPF, here is a much more communication-efficient two-server PIR protocol:

- The client uses $k_0, k_1 \leftarrow \texttt{KeyGen}(i, 1)$ to share a point function that encodes its desired index i.
- The client sends k_0 to S_0 and k_1 to S_1.
- The two servers S_b compute the following:

$$\left(\bigoplus_i x_i \cdot \texttt{Eval}(k_0, i), \bigoplus_i x_i \cdot \texttt{Eval}(k_1, i)\right) = [x_\alpha]$$

  The above equality follows correctness of the DPF and the same reasoning as applied earlier.

- The servers send their shares back to the client, who locally reconstructs the output.

Because efficient DPFs exist, this protocol consumes only $O(\lg n\lambda)$ bits of communication!

Next Time

We have now introduced a new problem—PIR—and explored its connections to point functions and DPFs. Next time, we will demonstrate how to construct a DPF from a PRG, and we will discuss connections between the technique and MPC.

References

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995.