Privacy Enhancing Technologies FS2025
Lecture 13-14 – Private Information Retrieval

Florian Tramèr

AGENDA

1. PIR
2. 2-server PIR
3. Single-server PIR
4. Saving server computation

Recap so far

In the last lecture, we saw one example of a succinct primitive that can be made imminently practical: Succinct Zero-knowledge proofs. In the following two lectures, we will cover other types of primitives where a form of succinctness is desired—PIR and ORAM. Existing implementations of these primitives range from "near-practical" (i.e., prototypes exist and might get deployed if we can make them a bit more efficient), to "research products" (i.e., the protocol can be implemented in principle but the overhead is too large).

Here's a good time to note that generally when deploying privacy-preserving technology, your competition is a *non-private* primitive that has been optimized to death for speed. That is, in many situations, privacy has been sacrificed over time to make way for utility. Winning privacy back without sacrificing (too much) speed is hard.

1 Private Information Retrieval (PIR)

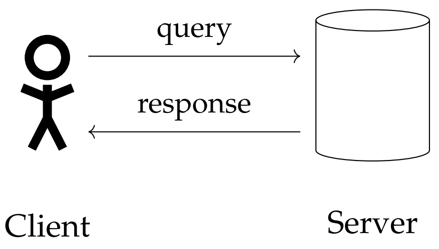
We'll now talk about another amazing result in cryptography which, similarly to zero-knowledge, may seem impossible at first glance.

One of my friends—Henry Corrigan-Gibbs—likes to refer to PIR as an "almost-perfect research idea", in reference to Dan Spielman's desiderata for perfect research:

1. It has a beautiful theory.
2. It works in practice.
3. It solves a problem people care about.

It's rare to have ideas that meet all three criteria. Many of the crypto things we cover in this class hit 2.5/3, missing out (for now) on the practical aspect (as we'll see, PIR falls in this category). Other things we'll cover (e.g., machine learning, differential privacy) maybe don't quite hit all three yet either. If you're going to do research later on, it's nice to try and find problems that hit all three (to some degree).

The goal of Private Information Retrieval is as simple as it is ambitious: let's build a private Google:



This general search-response paradigm appears in many applications on the Internet, e.g., search engines, DNS lookups, WebMD queries, searching the news, or social media, etc.

In all these scenarios, the client's query can leak sensitive information (their search history, medical symptoms, where they live, their political views, etc.)

Today, the server learns all this information from the client's query. And of course, they'll make use of this information to somehow make more money, e.g, via targeted advertisement or by selling it to third-parties...

This brings us to the fundamental question that PIR addresses:

Can you query a database without leaking any information about the client's query?

Trivial PIR. There is a trivial answer to the above question (note that we do not ask for privacy for the server): the client could just download *the entire database* from the server and do the search locally.

This is of course entirely impractical for most applications. So what's missing to make PIR interesting is a notion of succinctness.

Sublinear database search. So let's ask for something stronger. The client should be able to query the database without leaking their query, and with communication complexity that scales *sub-linearly* with the size of the database.

Now this sounds a lot less trivial! Unfortunately, unconditionally this is impossible... [CKGS98].

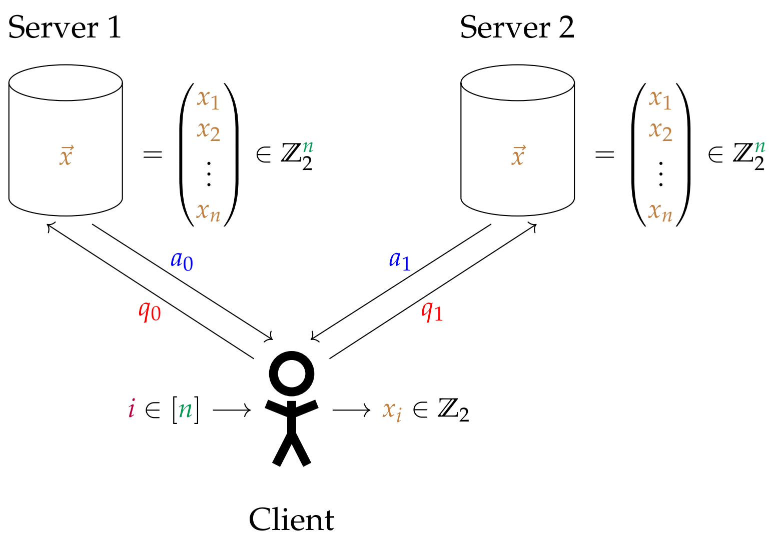
But when has that stopped anyone (especially in cryptography)? We'll just have to make some assumptions:

1. We can achieve PIR by assuming that two or more *non-colluding* servers host copies of the database.
2. We assume public-key cryptography exists.

We'll see an example of each in this lecture.

2 2-Server PIR

Before we formalize PIR, let's see what a two (or more) server PIR protocol looks like.


Server 1
Server 2
$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{Z}_2^n$
$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{Z}_2^n$
a_0
q_0
a_1
q_1
$i \in [n] \longrightarrow$
$\longrightarrow x_i \in \mathbb{Z}_2$
Client

The security we aim for here is against *non-colluding* servers. That is, even if all servers are malicious, as long as they don't share information with each other, the protocol is secure.

Here's how we formally define PIR in this setting:

Definition 1 (2-Server PIR)**.** A 2-Server PIR protocol consists of three algorithms:

- $q_0, q_1 \leftarrow \texttt{Query}(i)$
- $a_b \leftarrow \texttt{Answer}(\vec{x}, q_b)$
- $x_i \leftarrow \texttt{Reconstruct}(a_0, a_1)$

The scheme should satisfy the following properties:

- **Correctness:** $\forall n \in \mathbb{N}, i \in [n], \vec{x} \in \mathbb{Z}_2^n$:

$$\Pr\left[\texttt{Reconstruct}(a_0, a_1) = x_i \ : \ \begin{array}{r} q_0, q_1 \leftarrow \texttt{Query}(i) \\ a_0 \leftarrow \texttt{Answer}(\vec{x}, q_0) \\ a_1 \leftarrow \texttt{Answer}(\vec{x}, q_1) \end{array}\right] = 1 \, .$$

- **Security:** $\forall n \in \mathbb{N}, i, i' \in [n], \beta \in \{0,1\}$:

$$\{q_\beta \ : \ q_0, q_1 \leftarrow \texttt{Query}(i)\} \overset{c}{\approx} \{q_\beta \ : \ q_0, q_1 \leftarrow \texttt{Query}(i')\}$$

- **Sublinear communication:** $\forall n \in \mathbb{N}, i \in [n], \vec{x} \in \mathbb{Z}_2^n$:

$$|\texttt{Query}(i)| + |\texttt{Answer}(\vec{x}, q_0)| + |\texttt{Answer}(\vec{x}, q_1)| = o(n) \, .$$

2.1 Building Intuition: Hiding the Query Index with Secret Sharing

Let's start with a strawman scheme to get some intuition. This scheme will provide us with privacy, but won't save us (much) in terms of communication.

Suppose the client wants to know the i-th bit of the database $\vec{x}$. This can be represented by the computation:

$$\vec{x} \cdot e_i = \vec{x} \cdot [\underbrace{0, 0, \dots, 0}_{i-1}, 1, \underbrace{0, \dots, 0}_{n-i}]$$

Now, let the client produce an *additive* secret sharing of $e_i = q_0 \oplus q_1$, where $q_0 \xleftarrow{\$} \mathbb{Z}_2^n$. The client can send q_0 to the first server, and q_1 to the second. This leaks no information about i to the servers. Both servers can compute $\vec{x} \cdot q_0$ and $\vec{x} \cdot q_1$ locally and send the result to the client. The client can then reconstruct $\vec{x} \cdot e_i$ by XORing the two results.

What have we gained? Recall that in the trivial PIR scheme, the client has to download the entire database of n bits. Here, the client has to *upload* $2n$ bits, but only download 2 bits. So we've pushed all the download cost into an upload cost. That seems unhelpful.

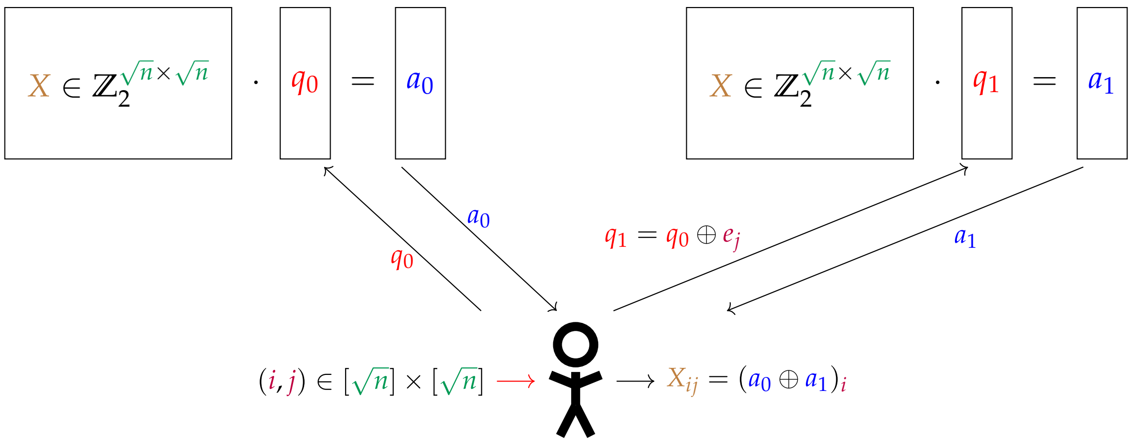
Actually, this scheme is (somewhat) useful if the database were to store records of more than 1 bit each, e.g., if each row is a 100-bit vector. Then, the trivial PIR would require downloading $100n$ bits, while this scheme requires uploading $2n$ bits and downloading 200 bits. Let's see how we can improve upon this.

2.2 Load-balancing to Achieve $O(\sqrt{n})$ Communication.

We'll now build on this idea to get a very simple 2-server PIR protocol with $O(\sqrt{n})$ communication and *information-theoretic* security (under the non-collusion assumption). This scheme is due to [CKGS98].

The idea is that we'll take our database of n bits and turn it into a smaller database with more bits per row! Specifically, we'll view the database as a matrix X of $\sqrt{n} \times \sqrt{n}$ bits. If the client wants the element at position (i, j), they'll do a PIR protocol to reconstruct the entire j-th column, and then use the i-th bit of the reconstructed column as the answer. Using our previous strawman scheme, this takes $O(\sqrt{n})$ communication.

$$\begin{aligned}
\texttt{Query}((i,j)) : \quad & q_0 \xleftarrow{\$} \mathbb{Z}_2^{\sqrt{n}} \\
& q_1 = q_0 \oplus e_j \\
\texttt{Answer}(X, q_b) : \quad & a_b \leftarrow X \cdot q_b \in \mathbb{Z}_2^{\sqrt{n}} \\
\texttt{Reconstruct}(a_0, a_1) : \quad & x_i \leftarrow (a_0 \oplus a_1)_i
\end{aligned}$$



$X \in \mathbb{Z}_2^{\sqrt{n}\times\sqrt{n}}$
$\cdot$
q_0
$=$
a_0
$X \in \mathbb{Z}_2^{\sqrt{n}\times\sqrt{n}}$
$\cdot$
q_1
$=$
a_1
a_0
q_0
$q_1 = q_0 \oplus e_j$
a_1
$(i,j) \in [\sqrt{n}] \times [\sqrt{n}] \longrightarrow$
$\longrightarrow X_{ij} = (a_0 \oplus a_1)_i$

Lemma 1. The above scheme is a 2-server PIR protocol with $O(\sqrt{n})$ communication and information-theoretic security.

Proof.

- *Correctness*:

$$\begin{aligned}(a_0 \oplus a_1)_i &= (X \cdot q_0 \oplus X \cdot q_1)_i \\ &= (X \cdot q_0 \oplus X \cdot (q_0 \oplus e_j))_i \\ &= (X \cdot q_0 \oplus X \cdot q_0 \oplus X \cdot e_j)_i \\ &= (X \cdot e_j)_i = X_{ij} \,.\end{aligned}$$

- *Security*: q_0 is uniformly random in $\mathbb{Z}_2^{\sqrt{n}}$, and so is q_1 (this is an *additive* secret sharing).
- *Communication*:
  - *upload:* $|q_0| + |q_1| = 2\sqrt{n}$ bits
  - *download:* $|a_0| + |a_1| = 2\sqrt{n}$ bits
  - *total:* $O(\sqrt{n})$ bits

□

3 Single-server PIR

The previous schemes worked because we could "encrypt" the client's query by additively secret-sharing it between the two servers. If we have a single server, we'll now just use regular encryption! But we'll need an encryption scheme with a special property: *linearly homomorphic encryption*.

$$\texttt{Enc}(k, m_1) + \texttt{Enc}(k, m_2) = \texttt{Enc}(k, m_1 + m_2)$$

Remember that we saw an example of a homomorphic *commitment* scheme in the first lecture. Constructing linearly-homomorphic encryption relies on similar ideas. Such schemes are quite common. We can build them from common assumptions, such as Diffie-Hellman, quadratic residuosity, Learning with Errors, etc.

3.1 Back to Our Strawman Scheme

We can directly plug in linearly-homomorphic encryption into our strawman PIR protocol, in place of additive secret-sharing. That is, the client computes the query vector

$$q = \texttt{Enc}(k, e_j) := [\texttt{Enc}(k,0), \ldots, \texttt{Enc}(k,1), \ldots, \texttt{Enc}(k,0)] \,,$$

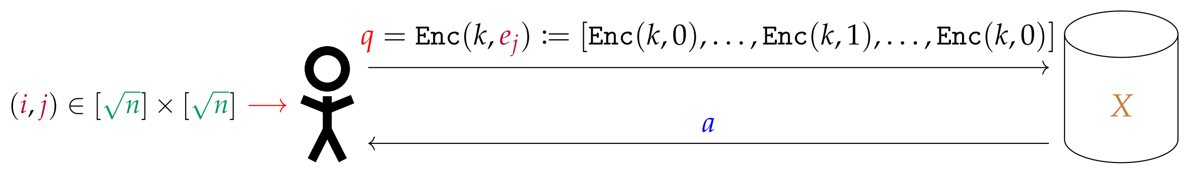
where we just encrypt the basis vector component-wise. Then, the server can compute

$$\vec{x} \cdot q = \vec{x} \cdot \texttt{Enc}(k, e_j) = \texttt{Enc}(k,0) \cdot x_1 + \cdots + \texttt{Enc}(k,1) \cdot x_j + \cdots + \texttt{Enc}(k,0) \cdot x_n = \texttt{Enc}(k, x_j)$$

and send the result to the client. The client can then decrypt the result to get x_j.

3.2 Load Balancing

Let's now apply load-balancing again. Our protocol now looks like this:



$q = \texttt{Enc}(k, e_j) := [\texttt{Enc}(k,0), \ldots, \texttt{Enc}(k,1), \ldots, \texttt{Enc}(k,0)]$
$(i, j) \in [\sqrt{n}] \times [\sqrt{n}] \longrightarrow$
a
X

$$\begin{aligned}(\texttt{Dec}(k, a))_i &= \texttt{Dec}(k, \texttt{Enc}(k, X_j)_i \\ &= X_{ij}\end{aligned}$$

$$\begin{aligned}a = X \cdot q &= X \cdot \texttt{Enc}(k, e_j) \\ &= \texttt{Enc}(k, X \cdot e_j) \\ &= \texttt{Enc}(k, X_j)\end{aligned}$$

requires only
adding ciphertexts

The total communication is $|q| + |a| = \sqrt{n} + \sqrt{n}$ ciphertexts, or $O(\lambda\sqrt{n})$ bits.

3.3 Recursive PIR

3.3.1 Recursive PIR

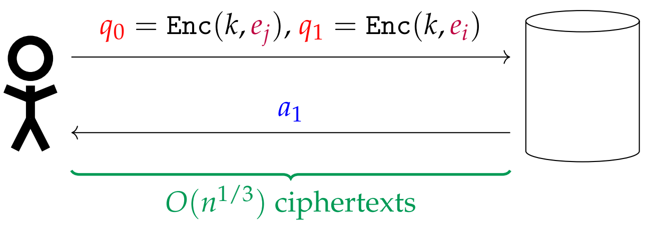
3.1.1.1 Recursive PIR This is one of the coolest ideas to make single-server PIR more efficient. Notice that in our $\sqrt{n}$ scheme above, the client always receives $\sqrt{n}$ elements from the server, but they are only interested in *one* of them (i.e., the i-th one).

Essentially, what we're doing here is a trivial PIR over $\sqrt{n}$ elements, by downloading a whole column of X. So why don't we re-do PIR here too? Let's see how to use this idea with one level of recursion to get a $O(n^{1/3})$ scheme.

We're going to view our database as a matrix of $n^{2/3}$ rows of $n^{1/3}$ bits each.

$$X = \left.\left[\overbrace{\qquad\qquad}^{n^{1/3}}\right]\right\} n^{2/3}$$

The first layer of PIR selects one column of the database. This column contains $n^{2/3}$ ciphertexts. The server can now view these ciphertexts as another database matrix of dimension $n^{1/3} \times n^{1/3}$. And then we do a second layer of PIR to select one column in this matrix.



$$n^{2/3}\left\{\left[\overbrace{\ X\ }^{n^{1/3}}\right] \cdot [q_0] = [a_0]\right\} n^{2/3} \text{ ctxts}$$

express as a matrix of ctxts

$$n^{1/3}\left\{\left[\overbrace{\ a_0\ }^{n^{1/3}}\right] \cdot [q_1] = [a_1]\right\} n^{1/3} \text{ ctxts}$$

But now, we're still sending back $O(n^{1/3})$ ciphertexts even though the client just needs one. So we could recurse again! But there's a catch: we need to ensure that ciphertexts don't "blow up" in size. Indeed, note that

$$a_0 = X \cdot \texttt{Enc}(k, e_j) = \texttt{Enc}(k, X \cdot e_j) = \texttt{Enc}(k, X_j)$$

and

$$a_1 = a_0 \cdot \texttt{Enc}(k, e_i) = \texttt{Enc}(k, a_0 \cdot e_i) = \texttt{Enc}(k, \texttt{Enc}(k, \dots)) \, .$$

That is, the final ciphertexts in a_1 are encryptions of other ciphertexts. If the encryption scheme has high *expansion* (i.e., ciphertexts are much larger than plaintexts), then every layer of recursion will blow up the size of the ciphertexts. We thus need a linearly homomorphic encryption scheme that has low expansion (e.g., [DJ01]).

Then, applying this idea recursively, we get a PIR protocol with communication complexity $O(\text{polylog} n)$.

Q: Does this recursive PIR idea work for 2-server PIR? Why or why not?

4 Computational Complexity

So far, we have ignored the computational complexity of our PIR schemes. But this is where the bottleneck tends to lie in practice today.

Most notably, in all the schemes we discussed, the server's work is **linear** in the size of the database. That is, for every query from the client, the server has to scan the entire database and cannot make use of any efficient search data structure.

In some sense, this is inherent: if the server doesn't read some bit x_i in the database in response to a query, then we learn that the query wasn't for x_i. There are a few ways to get around this limitation, to aim to make PIR practical for real-world usage:

- *Batching:* If we have multiple queries to answer simultaneously, there are clever hash-based schemes that partition a DB into multiple chunks so that we can answer all queries in one go. These schemes end up answering k queries on a DB of size n with server work $O((\lambda \log k) n)$ [IKOS04].
- *Pre-processing the DB:* The server could do the linear work upfront, to pre-compute some data structure that would allow for faster responses later [BIM00]. This has been a very active area of research in the past few years.

  [CGK20] give a 2-server scheme with an offline linear-time pre-processing phase (independent of any queries), that later allows the server to answer online user queries with *sublinear* communication and computation complexity $O(\sqrt{n} \cdot \text{polylog}\, n)$. A nice property of this scheme is that the server still stores the database in its original form.

  A beautiful recent work of Lin et al. [LMW23] (STOC'23 best paper) designs a single-server PIR scheme where the server first encodes the database into some special data structure in near-linear time, so that it can later answer clients' online queries with complexity only $O(\text{polylog}\, n)$! This is essentially the best we could hope for, so there's a lot of excitement that further empirical improvements to this (theoretical) work could result in practical single-server PIR applications.

References

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20*, pages 55–73. Springer, 2000.

[CGK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 44–75. Springer, 2020.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001 Cheju Island, Korea, February 13–15, 2001 Proceedings 4*, pages 119–136. Springer, 2001.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 262–271, 2004.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 595–608, 2023.