PIR Extensions

Notes by Henry Corrigan-Gibbs

MIT - 6.5610
Lecture 6 (February 21, 2024)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Logistics

- Post project ideas on Piazza by Friday

Outline

- Review: Private information retrieval
- Review: Square-root PIR scheme.
- Handling longer database records.
- **Stretch break**
- Reducing communication: The full K-O scheme

Review: Private information retrieval

Recall the problem of private information retrieval that Alexandra defined in the last lecture:

- A server that holds an N-bit database, $D \in \{0,1\}^N$,
- a user that holds an index $i \in \{1,\ldots,N\}$, and
- the user's goal is to learn the i-th bit of the database, D_i, without revealing i to the server.

Formally, a *private information retrieval* (PIR) is a triple of randomized algorithms $(\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ that satisfy the following three properties:

1. **Correctness**: for any index $i \in \{1,\ldots,N\}$ and any database $D \in \{0,1\}^N$,

$$\Pr\left[\mathsf{Reconstruct}(\mathsf{st},\mathsf{ans},i) = D_i : \begin{array}{c}(\mathsf{st},\mathsf{qu}) \leftarrow \mathsf{Query}(1^\lambda, i) \\ \mathsf{ans} \leftarrow \mathsf{Answer}(D,\mathsf{qu})\end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

   That is, the user will correctly recover D_i with high probability.

2. **Security**: for any two indices $i, j \in \{1,\ldots,N\}$,

$$\left\{\mathsf{qu} : (\_,\mathsf{qu}) \leftarrow \mathsf{Query}(1^\lambda, i)\right\} \approx \left\{\mathsf{qu} : (\_,\mathsf{qu}) \leftarrow \mathsf{Query}(1^\lambda, j)\right\}.$$

   That is, the server's view looks computationally indistinguishable whether the user is making a query for i or making a query for j. This is very similar to the semantic-security definition of an encryption scheme.

   As we discussed, CPA security is a stronger definition of secret-key encryption since it allows the adversary to encryptions of many messages of its choice under the same key. Why is this one-time definition good enough?

3. **Succinctness**: The total bit-length of the outputs of $\mathsf{Query}(1^\lambda, \cdot)$ and $\mathsf{Answer}(\cdot,\cdot)$ is less than N. (Without succinctness, the client might as well download the entire database.)

A classic PIR scheme: Square-root PIR

We will start by discussing a classic PIR scheme due to Kushilevitz and Ostrovsky (1997). It uses a symmetric-key additively homomorphic encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with keyspace $\mathcal{K}$ and message space $\mathbb{Z}_2$.

The server will represent its N-bit database as a matrix, $\mathbf{D} \in \mathbb{Z}_2^{\sqrt{N}\times\sqrt{N}}$. The user will represent its query as an index $(i,j) \in [\sqrt{N}] \times [\sqrt{N}]$ into this matrix.

The protocol proceeds as follows:

- $\mathsf{Query}\left(1^\lambda, (i,j)\right)$:
  - Build the unit vector $\mathbf{u}_j \in \mathbb{Z}_2^{\sqrt{N}}$ that consists of all "0"s except for a single "1" at position j.

  - Sample a fresh encryption key $k \xleftarrow{R} \mathcal{K}$.
  - Compute the query vector $\mathsf{qu} \leftarrow \mathsf{Enc}(k, \mathbf{u}_j)$. (*Note:* Here we encrypt each element of the query vector component-wise.)
  - Set $\mathsf{st} \leftarrow k$ and output $(\mathsf{st}, \mathsf{qu})$.

- $\mathsf{Answer}(\mathbf{D}, \mathsf{qu})$:
  - Output the answer vector $\mathsf{ans} \leftarrow \mathbf{D} \cdot \mathsf{qu}$. (Here we use the fact that computing a matrix-vector product with a plaintext matrix and a component-wise encrypted vector just requires addition of encrypted values.)

- $\mathsf{Reconstruct}(\mathsf{st}, \mathsf{ans}, (i, j))$:
  - Recover the encryption key $k \leftarrow \mathsf{st}$.
  - Decrypt the answer vector as $\mathbf{v} \leftarrow \mathsf{Dec}(k, \mathsf{ans})$, where $\mathbf{v} \in \mathbb{Z}_2^{\sqrt{N}}$.
  - Output the i-th entry of $\mathbf{v}$.

This scheme indeed meets all the requirements of a PIR protocol:

1. **Correctness**: If the underlying encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is linearly homomorphic, we have that:

$$\begin{aligned} \mathsf{ans} &= \mathbf{D} \cdot \mathsf{Enc}(k, \mathbf{u}_j) \\ &= \mathsf{Enc}(k, \underbrace{\mathbf{D} \cdot \mathbf{u}_j}_{\text{mod } 2}). \end{aligned}$$

   By construction, the vector $\mathbf{D} \cdot \mathbf{u}_j$ here corresponds exactly to the j-th column of database matrix $\mathbf{D}$. So, decrypting the answer vector ans and outputting its i-th entry will exactly recover the element at position (i, j) in the database $\mathbf{D}$.

2. **Security**: PIR security follows from the CPA-security of the underlying encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$.

3. **Succinctness**: The output of Query here consists of $\sqrt{N}$ ciphertexts (one per element in the encrypted vector). Similarly, the output of Answer consists of $\sqrt{N}$ ciphertexts. So, the total communication between the user and the server here is much smaller than N.

Trading upload for download

Before we leave this PIR scheme, I wanted to mention one other interesting property of it: We can generalize the scheme to have the database be a matrix of dimension $\alpha \times \frac{N}{\alpha}$. Then client uploads an

N/α-element row vector (of ciphertexts) and receives in response a α-element column vector (of ciphertexts).

This observation lets us trade off upload for download: If, for whatever reason, we would rather increase the size of the client upload at the cost of download, the same PIR scheme with different parameters can have upload $N^{2/3}$ and download $N^{1/3}$ ciphertexts (for $\alpha = N^{1/3}$). Or upload $N/100$ and download 100 ciphertexts, etc.

This will be useful in a moment.

Note: All of the extensions we will discuss today make only *black-box* use of the underlying PIR scheme. That means that they will work with whatever crazy PIR scheme you come up with from whichever computational assumption (factoring, discrete-log, etc.).

Longer database records

We described the PIR scheme with respect to a database of N one-bit records. A more realistic setting involves a database of N records, each of ℓ bits. How do we construct a PIR scheme in this case?

Observe that the square-root PIR scheme we have already seen allows the client to fetch an entire *column* of the database in one go. So if $\ell < \sqrt{N}$ that scheme unmodified will do the trick.

For other PIR schemes that have other communication costs, there is another more general trick that we can use.

The most naïve approach for handling databases with ℓ-bit records would be to jam all of the records into a single database of $N\ell$ bits. Then, the client makes ℓ queries to this database to fetch bit of its desired record, one at a time.

Say that the original PIR scheme has upload cost $U(N)$ bits and download cost $D(N)$ bits. Let's write $U(N, \ell)$ and $D(N, \ell)$ to denote the upload/download cost of a PIR scheme with N ℓ-bit records. Then this new scheme has:

- Upload: $U(N, \ell) = \ell \cdot U(N)$
- Download $D(N, \ell) = \ell \cdot D(N)$

We can do better! Imagine "slicing" the database into ℓ databases, each N bits long. The first database D_1 contains the *first* bit of all N database records. The second database D_2 contains the *second* bit of all N database records. And so on...

To make a query for the ith record, the client queries the ith bit of the first database D_i using the PIR scheme for an N-bit database.

The server then answer's the client's query *with respect to each of the ℓ databases*. So the client makes one query and receives ℓ responses.

The communication cost is now:

- Upload: $U(N, \ell) = U(N)$.
- Download: $D(N, \ell) = \ell \cdot D(N)$.

This saves a factor of ℓ on the upload cost. In many applications $\ell \gg 1024$, so this savings is non-trivial.

Rebalancing to handle longer records.

We can combine our trick of trading upload for download in the square-root PIR scheme with this trick of handling longer records.

In particular, for every α, we have a PIR scheme for ℓ-bit records with:

- Upload: $U(N, \ell) = N/\alpha$.
- Download: $D(N, \ell) = \ell \cdot \alpha$.

To minimize communication, we can choose the parameter α to balance the upload and download:

$$\begin{aligned} N/\alpha &= \ell \cdot \alpha \\ N &= \ell \cdot \alpha^2 \\ N/\ell &= \alpha^2 \\ \sqrt{N/\ell} &= \alpha \end{aligned}$$

- Upload: $U(N, \ell) = \sqrt{\ell N}$.
- Download: $D(N, \ell) = \sqrt{\ell N}$.

What it means in practice. Say that your database consists of 16 million articles ($N = 2^{24}$), each of 2KB$= 2^{11}$ in length. If you use certain encryption schemes (e.g., Paillier) you can jam 256 bytes of data into 512 bytes of ciphertext, so the effective length of each database record is $\ell = 4$ ciphertexts worth.

The upload and download cost of this PIR scheme will be $\sqrt{\ell N} = \sqrt{2^2 \cdot 2^{24}} = 2^{13}$ ciphertexts in either direction. This is roughly 4MB—not bad for a database of 32 GB in size!

Things I will not talk about

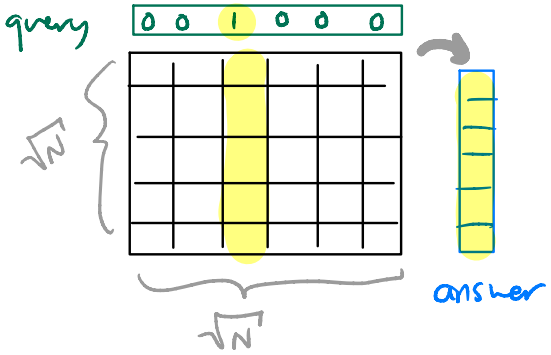
There are two neat tricks that I want to mention but not discuss in depth:

- **Batch PIR**: It is possible to fetch a batch of B records using PIR at almost the server-side cost of fetching a single record. (The naïve strategy of running the PIR scheme B times gives server running time NB. This approach gets time $O(N \log N)$, essentially independent of the batch size B.
- **PIR by keywords**: You can build a PIR scheme that supports key-value lookups from any standard PIR scheme. The basic idea is that and data structure you can implement in RAM, you can implement in PIR; just replace each RAM lookup with a PIR query.

PIR with smaller communication

The PIR scheme we have seen so far has a communication cost of roughly $\sqrt{N}$ bits. One surprise, again due to Kushilevitz and Ostrovsky, is that we can drive down the communication cost of PIR to something much smaller—even subpolynomial in N in some cases.

Figure 1: The square-root PIR scheme.



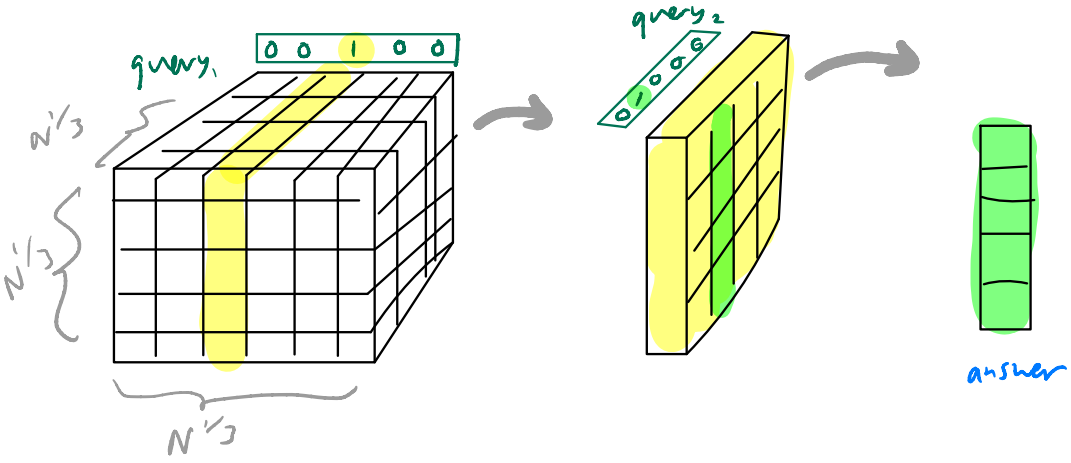


Figure 2: The cube-root PIR scheme.

Intuition. One way to think about the KO PIR scheme is as a generalization of the square-root scheme we saw so far.

In the square-root scheme, we represent database as a two-dimensional array (i.e., a matrix). The client sends up a vector of size roughly equal to the length *one side* of this array. The client receives a vector of length roughly equal to the other side of this array.

But why stop at two dimensions? We can go to three!

We can represent the database as a three-dimensional array (i.e., a cube/tensor) where each side has length $N^{1/3}$. The client sends up *two vectors* of ciphertexts, where the length of each is the length of one side of the cube. The client

Background. First, let's recall what we know about PIR so far:

- For ℓ-bit records:

$$U(N, \ell) = U(N), \text{ and} \tag{1}$$
$$D(N, \ell) = \ell \cdot D(N). \tag{2}$$

- From our square-root PIR scheme rebalanced to make the upload very large and download very small, we have a PIR scheme that satisfies:

$$U(N) = N \cdot |\mathsf{ct}|, \text{ and}$$
$$D(N) = |\mathsf{ct}|,$$

  where $|\mathsf{ct}|$ denotes the size of a ciphertext for our additively homomorphic encryption scheme.

- **Main scheme.** Putting the PIR ideas from the prior two bullets together, we get a PIR scheme with upload

$$U(N, \ell) = N \cdot |\mathsf{ct}|, \text{ and} \tag{3}$$
$$D(N, \ell) = \ell \cdot |\mathsf{ct}|, \tag{4}$$

- **Trivial scheme.** There's also the trivial PIR scheme in which the client just downloads the entire database. This has:

$$U(N) = 0, \text{ and}$$
$$D(N) = N.$$

Now, the very neat idea of Kushilevitz and Ostrovsky is to show that you can construct a PIR scheme on an N-record database by invoking a PIR scheme on an $N/2$-record database. With a careful recursive construction, they show that this can reduce the communication cost by a *lot*:

Theorem 1 (KO97)**.** *Assume there exists a linearly homomorphic encryption scheme. Then for every constant $\epsilon > 0$, there is a PIR scheme that has communication cost $O(N^\epsilon \cdot \mathrm{poly}(\lambda))$, on database size N and security parameter λ.*

Usually I just try to describe the high-level idea of the KO construction without going into details. This year, I am going to attempt to give you the details. My hope is to convince you that what you already know about PIR, plus some cleverness, is enough to construct a state-of-the-art PIR scheme.

There are other state-of-the-art PIR schemes that use fully homomorphic encryption (next week). But if you want to build PIR from additively/linearly homomorphic encryption, this K-O scheme is the best we have.

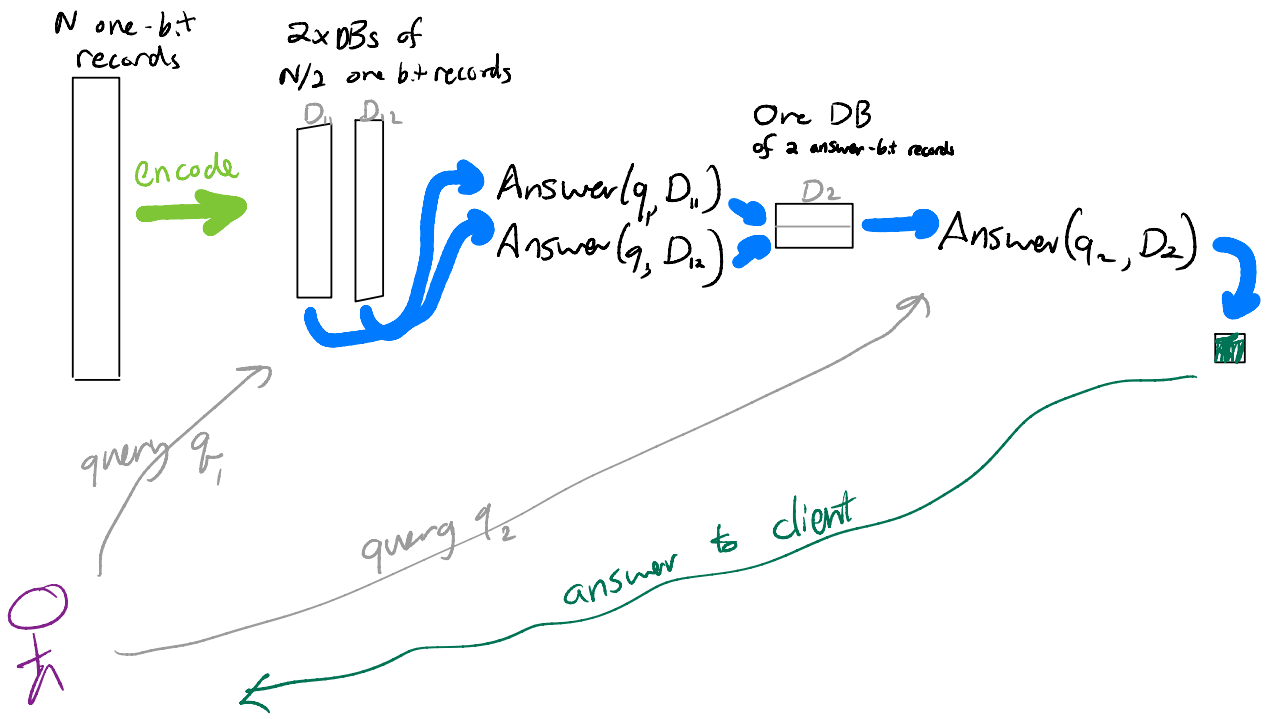


Figure 3: My poor drawing of one step of the K-O recursive PIR scheme with $\alpha = 2$.

Construction. I will describe the construction in words and pictures and then go through the analysis with a bit more precision.

The way the construction works, on an N-bit database, is as follows:

1. The client and server view the N-bit database as two $N/2$-record databases.
2. The client sends *one* query to the server for its desired index into the $N/2$-bit database.
3. The server answers the client's query with respect to *each* database. Now the server has two PIR answers to send to the client. The server does NOT send these to the client.
4. Instead, the client and server view these two answers as a new two-record database.
5. The client uses the **main PIR scheme** (Eqs. (3) and (4)) to fetch *one* of these two records. This requires the client to send a second PIR query to the server.

Notice that the client sends *two* PIR queries to the server: one for the $N/2$-bit and one for a two-record database.

A generalization, which turns out to be important, is to divide the database into α chunks, each of size N/α. The basic scheme takes $\alpha = 2$, but we may want to go farther.

Analysis. Now let us figure out what the communication cost of this scheme looks like when we take α to be arbitrary. First, the upload. There is one query to a database of size N/α, where each record has ℓ bits. There is a second query to a database of α records, where each record is the *answer* to a PIR query to the (N/α)-sized database. Putting this into symbols, we have:

$$\begin{aligned} U(N,\ell) &= U\left(\frac{N}{\alpha},\ell\right) + U(\alpha, \underbrace{D\left(\frac{N}{\alpha},\ell\right)}_{\text{size of PIR answers}}). \\ &= U\left(\frac{N}{\alpha}\right) + \alpha\,|\mathsf{ct}|\,, \end{aligned}$$

where the second line comes from applying Eq. (1) to both the recursive $\frac{N}{\alpha}$-record scheme and using the "main" PIR scheme (Eqs. (3) and (4)) to the α-record PIR schemes.

And now, the download. The client downloads one PIR answer from a database of α records. Each entry in this database is itself the answer to a PIR query to a database of N/α records. Mathematically, we have:

$$\begin{aligned} D(N,\ell) &= D\Bigg(\alpha, \underbrace{D\left(\frac{N}{\alpha},\ell\right)}_{\text{size of PIR answers}}\Bigg) \\ &= |\mathsf{ct}| \cdot D\left(\frac{N}{\alpha},\ell\right). \end{aligned}$$

Here, the second line comes from plugging in the main PIR scheme for the two-record PIR scheme.

If our original database has $\ell = 1$-bit records, we now have:

$$D(N) = |\mathsf{ct}| \cdot D\left(\frac{N}{\alpha}\right).$$

So now we have pretty clean recursive expressions for both the upload and download costs.

DO YOU WANT MORE RECURSION??? Yes. The answer is always yes. Except when it's no. But it's usually yes.

More recursion. If we continue this recursive process for k steps, what we get is:

$$U(N) = U\left(\frac{N}{\alpha^k}\right) + \alpha k \cdot |\mathsf{ct}|$$
$$D(N) = |\mathsf{ct}|^k \cdot D\left(\frac{N}{\alpha^k}\right).$$

Now if we plug in the trivial PIR scheme (download the entire database) at the bottom of the recrusion (i.e., for $N/2^k$-size database), we get:

$$U(N) = \alpha k \cdot |\mathsf{ct}|$$
$$D(N) = |\mathsf{ct}|^k \left(\frac{N}{\alpha^k}\right).$$

Recognizing an old friend. If we take $\alpha = \sqrt{N}$ and $k = 1$, we actually get back exactly the square-root PIR scheme! There, we represented the database as a $\sqrt{N}$ by $\sqrt{N}$ matrix.

As k increases, you can think of this scheme as representing the database as an object of higher and higher dimensions. So for $k = 2$, we treat the database as a cube with side length $N^{1/3}$, for $k = 3$ we have some sort of higher-dimensional cube thing, and so on. The parameter α is essentially telling us what the side length of the cube/hypercube is.

The client uploads one vector of ciphertexts to the server for each *side* of the cube.

Setting the parameters. We can plug in whatever values for k and α we want and see what happens to the upload and download. For instance, take $\alpha = N^{1/100}$ and $k = 100$.

We then have

$$U(N) = 100 \cdot N^{1/100} |\mathsf{ct}|$$
$$D(N) = |\mathsf{ct}|^{100}.$$

In a theoretical sense, this gives a pretty good PIR scheme! The upload is roughly $N^{1/100}$—much much much smaller than the square-root $N^{1/2}$ scheme we saw before.

The download is $|\mathsf{ct}|^{100}$ which is some polynomial that is *independent of the database size*. So the total communication is $N^{1/100} \cdot \mathrm{poly}(\lambda)$.

The problem, of course, is that the $|\mathsf{ct}|^{100}$ download cost is galactic in practice—it's polynomial in the security parameter, but it's an absurd polynomial.

When people implement this scheme, they typically take something like $\alpha = N^{1/3}$ and $k = 2$, to get a scheme that's just slightly better in communication than the square-root scheme.

There are lots of fancy optimizations to this basic scheme that I will not have time to discuss. But just know that you can do a little better than what I have shown here.

We have many constructions of better-than-KO PIR schemes from other assumptions. As far as I know, KO is the best scheme that makes black-box use of linearly homomorphic encryption.

Question. *Is there a PIR scheme that makes black-box use of a linearly homomorphic encryption scheme and that has communication* $\text{polylog}(N)$ *on database size N?*

As we will see next week, lattice-based crypto can give us schemes with extremely small communication: something like $\log(N) + \text{poly}(\lambda)$. Since it takes $\log N$ bits to specify which bit of the database you're interested in, this cost is just about as low as you can hope to go.

References