1 Lecture 15: Two-Party PIR

1.1 Multi-Server PIR

Consider a scheme where we have two servers that do not communicate, each with a copies of the same database. A user wants an entry $x_i i$ from the database, and sends a query to each server, and gets a response back from each server. We should have the following guarantees:

1. (CORRECTNESS) The user learns the entry x_i.
2. (PRIVACY) Neither server learns anything about i.
3. (COMMUNICATION) All communications are significantly less than n (less than linear-order communication complexity)

As it turns out, for databases that do not communicate at all, we can construct this with NO cryptographic assumptions! We organize the database into a 2d $\sqrt{n} \times \sqrt{n}$ array of entries. Then, from one database, we request a bitwise XOR of some number of columns, including the one with x_i. And, from the other database, we request the same thing, except we omit the column containing x_i. In the end, we can XOR the two results from the databases, and we should end up with the full column containing x_i, from which we will learn x_i, as desired.

The communication is $O(\sqrt{n})$, and privacy is guaranteed, since neither database can possibly figure out which column contains the desired result unless they talk to each other (since the column containing the entry may or may not be included in the request).

1.1.1 Improving Precision

Now, what we proposed above gives the user a full row from the database, but what if we only want the single desired entry?

For this, it will be easier to consider a case where we have 4 databases, no two of which have pairwise communicate. Then, the user will generate two length-$\sqrt{n}$ strings, one of which corresponds to the rows of the database, and the other corresponding to the columns. We will call them r_x and r_y respectively. For each string, the user will create a variation r_x^i (r_y^i) where the bit corresponding to the row (or column) containing x_i is inverted. Then, to each of the databases, the user will send one of r_x, r_x^i and one of r_y, r_y^i, sending a different combination to each DB.

Each database will respond with an XOR of the intersections of the different included columns and rows, and the user can XOR these responses to get only x_i.

1.1.2 Improving Communication Complexity

As with previous single-database PIR, we want to decrease this from $n^{1/2}$ to $n^{1/3}$. We do this by considering the databases to be 3-dimensional, rather than 2-dimensional, and along the lines of our improved-precision solution, we now need 8 databases. To

each database, we send (r_x, r_y, r_z), or some inversion, and we will end up with $O(n^{1/3})$ communication complexity.

In order to do this with two databases, we have each database simulate 4 databases. For instance, one database will get (r_x, r_y, r_z), and will send back the following responses:

1. Response for (r_x, r_y, r_z).
2. All $n^{1/3}$ responses for (r_x^i, r_y, r_z), for all $0 \leq i < \sqrt[3]{n}$
3. All $n^{1/3}$ responses for $(r_x r_y^i, r_z)$, for all $0 \leq i < \sqrt[3]{n}$
4. All $n^{1/3}$ responses for (r_x, r_y, r_z^i), for all $0 \leq i < \sqrt[3]{n}$

The second database would get the version of the triple where all 3 are inverted, and would do the same as the first database, since $(r_x^i)^i = r_x$. Then, the user can pick out the correct values to figure out the value of the desired entry.

1.2 Connection to Locally Decodable Codes (LDC)

The idea of LDC's is to encode a message in such a way that if some bits are corrupted, we can still recover the lost information.

Suppose that we have a PIR scheme which allows the user to send queries of $O(\log(n))$, and each database responds with a single bit, which allows the user to reconstruct the single desired bit x_i. Then, the user has $n = 2^{\log(n)}$ possible questions that it can ask (since it may invert random bits here and there), and each question has the same single-bit answer from each database. So, we can concatenate all of the possible answers together into a string of length $2n = 2^{\log(n)+1}$, which will be our code-word for our LDC.

The idea here is that even if a fraction of each database is corrupted, which causes some of the code-word to become corrupted, the user really only needs one bit from each half of the code-word to properly reconstruct their desired bit. So the probability that the desired bit can be recovered remains high.

More precisely, this is because as far as the user and the desired bit is concerned,