CMSC 858K — Advanced Topics in Cryptography April 6, 2004

Lecture 18

Lecturer: William Gasarch *Scribe(s): Avinash J. Dalal, Julie Staub*

1 Summary — What is Private Information Retrieval?

In this set of lecture notes we begin a brief discussion of **Private Information Retrieval** (PIR). We begin by defining what PIR is about. We then present two PIR protocols, and give simple proofs that these two protocols are indeed correct. Finally, we discuss some generalizations of these basic schemes.

Let us begin by introducing the problem. Suppose we have a database $DB = (x_1, \ldots, x_n)$, viewed as an n-bit string (i.e., the database has n entries, with each entry being a single bit). Suppose further that a user wants to learn the i^{th} bit of the database *without leaking to the database administrator*[1] *any information about* i (in an information-theoretic sense).

Informally, this is known as the PIR problem. Note that there does exist a solution: simply send to the user the contents of the entire database. The user can discard all the entries other than the one it is interested in, and the database clearly learns no information about which entry this is. The communication complexity of this solution, however, is $O(n)$. The PIR problem asks: can we do better?

It is actually not too difficult to show that it is impossible to do better in the above scenario if we require information-theoretic privacy for the user (i.e., even an all-powerful database learns nothing about the bit i the user is interested in) [1]. Thus, we must relax the scenario a bit to allow for more efficient solutions. One possibility that we explore here is to assume that there are multiple (identical) *copies* of the database, and the administrators of these databases cannot communicate with each other. We see that this allows for much more efficient solutions (in terms of their communication complexity).

2 A First Approach

We discuss here an approach yielding schemes with communication complexity $\sqrt[d]{n}$ when there are 2^d copies of the databases. The scheme will be illustrated first for the cases of $d = 2$ and $d = 3$ and we then discuss its extension to the general case.

A 4-Database PIR Protocol. Suppose we have 4 identical copies of the data in databases DB_1, DB_2, DB_3, DB_4. We show a PIR protocol with $O(\sqrt{n})$ communication complexity in this setting. View the data (which is an n-bit string) as a $\sqrt{n} \times \sqrt{n}$ array $\{x_{i,j}\}_{1 \le i,j \le \sqrt{n}}$ (we assume for simplicity that n is a square). When the user wants to learn bit x_{i^*,j^*} at position (i^*, j^*) he proceeds as follows:

1. Choose two random strings S, T each of length $\sqrt{n}$, and view these as subsets of $\{1, \ldots, \sqrt{n}\}$ in the natural way.

[1] From now on, "the database" will refer to the data itself as well as to the "administrator" of this data.

2. Let $S' = S \oplus \{i^*\}$ and $T' = T \oplus \{j^*\}$. Here, we let

$$S \oplus \{s\} \stackrel{\text{def}}{=} \begin{cases} S \cup \{s\} & s \notin S \\ S \setminus \{s\} & s \in S \end{cases}.$$

   We now have four subsets (equivalently, $\sqrt{n}$-bit strings) S, T, S', T'.

3. Send:

   - (S, T) to DB_1;
   - (S, T') to DB_2;
   - (S', T) to DB_3; and
   - (S', T') to DB_4.

4. Database k receives a pair of subsets $(\hat{S}_k, \hat{T}_k)$, and sends to the user the single bit:

$$X_k \stackrel{\text{def}}{=} \bigoplus_{i \in \hat{S}_k, j \in \hat{T}_k} x_{i,j}.$$

   Here, of course, $\oplus$ represents the bit-wise xor operation.

5. Upon receiving the responses from the databases, the user computes

$$x_{i^*,j^*} = X_1 \oplus X_2 \oplus X_3 \oplus X_4.$$

We first show that the above protocol is *correct*; i.e., that the user correctly recovers the bit of interest. To see this, note that:

$$X_1 \oplus X_2 \oplus X_3 \oplus X_4 = \bigoplus_{i \in S, j \in T} x_{i,j} \oplus \bigoplus_{i \in S', j \in T} x_{i,j} \oplus \bigoplus_{i \in S, j \in T'} x_{i,j} \oplus \bigoplus_{i \in S', j \in T'} x_{i,j}.$$

Now, we claim that for each $(i, j) \neq (i^*, j*)$, the value $x_{i,j}$ appears an even number of times in the above sum; in particular:

- If $i \neq i^*$ and $j \neq j^*$ then (i, j) is in either zero or all of the sets $S \times T$, $S \times T'$, $S' \times T$, $S' \times T'$.
- If $i = i^*$ but $j \neq j^*$ (or vice versa) then (i, j) is in either zero or exactly two of the sets $S \times T$, $S \times T'$, $S' \times T$, $S' \times T'$.

Thus, all of these contributions cancel out. On the other hand, the value x_{i^*,j^*} appears an odd number of times in the above sum, and hence $X_1 \oplus X_2 \oplus X_3 \oplus X_4 = x_{i^*,j^*}$, as desired.

We next argue that the above protocol is *private*; this is straightforward since each database simply receives a pair of uniformly-distributed $\sqrt{n}$-bit strings. (It is easy to see that S, T are uniformly distributed. For the case of, e.g., S', note that $S' \oplus \{i^*\}$ is uniformly distributed since S is uniformly distributed; the set S is acting as a "one-time pad" of sorts.)

An 8-Database PIR Protocol. As Dr. Gasarch would ask: "Can we do better?". In particular, what if we had more copies of the database? In fact, we can generalize the above protocol to one with communication complexity $O(\sqrt[3]{n})$ when there are 8 databases $DB_1, \ldots, DB_8$ available. Here, we view the data as a $\sqrt[3]{n} \times \sqrt[3]{n} \times \sqrt[3]{n}$ cube $\{x_{i,j,k}\}_{1 \leq i,j,k \leq \sqrt[3]{n}}$. When the user wants to learn the bit x_{i^*,j^*,k^*} at position (i^*, j^*, k^*) he proceeds as follows:

1. Choose three random strings R, S, T each of length $\sqrt[3]{n}$, and view these as subsets of $\{1, \ldots, \sqrt[3]{n}\}$ in the natural way.

2. Let $R' = R \oplus \{i^*\}$, $S' = S \oplus \{j^*\}$, and $T' = T \oplus \{k^*\}$, where the notation is as in the previous section. We now have eight subsets (equivalently, $\sqrt[3]{n}$-bit strings) R, R', S, T, S', T'.

3. Send:

$$\begin{array}{ll} (R,S,T) \text{ to } DB_1 & (R',S,T) \text{ to } DB_5 \\ (R,S,T') \text{ to } DB_2 & (R',S,T') \text{ to } DB_6 \\ (R,S',T) \text{ to } DB_3 & (R',S',T) \text{ to } DB_7 \\ (R,S',T') \text{ to } DB_4 & (R',S',T') \text{ to } DB_8 \end{array}$$

4. Database w receives subsets $(\hat{R}_w, \hat{S}_w, \hat{T}_w)$, and sends to the user the single bit:

$$X_w \stackrel{\text{def}}{=} \bigoplus_{i \in \hat{R}_w, j \in \hat{S}_w, k \in \hat{T}_w} x_{i,j,k}.$$

5. Upon receiving the responses from the databases, the user xor's them all together to compute the desired bit.

Proofs of correctness and privacy are exactly as before, and are therefore omitted.

2.1 Extending the Scheme

It should be clear that the above approach generalizes to give a 2^d-database scheme with communication complexity $\sqrt[d]{n}$, for any integer $d \geq 1$. The database is viewed as a d-dimensional hypercube $(\sqrt[d]{n})^d$, and then the approach above is applied. For details, see [1].

3 Improving the Previous Approach

Here, we show how the last scheme can be improved: we show a scheme with the same communication complexity but using only *two* databases. The basic intuition leading to this improvement is to balance the communication between the user and the databases. In the previous scheme, the user sends $O(\sqrt[3]{n})$ bits to each database, while each database responds with a single bit. By shifting more communication onto the databases, we are able to reduce the number of databases needed.

Recall in the previous scheme (of Section 2), the user chooses initial random sets R, S, T and then sends a triple of sets (based on these sets as well as the index of interest) to each of the eight databases. We show how the information sent by these eight databases can in fact be sent by two databases... but still without leaking any information to either of these databases about the index the user is interested in.

Let R, S, T, R', S', T' be as in Section 2. The user will send R, S, T to the first database, and R', S', T' to the second. (Note that these leak no information to either database about the index the user is interested in.) We would like the databases to now "simulate" the actions of the eight databases in the original scheme. Clearly, it is easy for the first database

to simulate DB_1 from before and equally easy for the second database to simulate DB_8. What about the other databases? Consider how the first database might simulate the actions of DB_2. To do this, it seemingly needs to know T'; on the other hand, if it knew T' then (using T) it would be able to determine k^* and this would leak information about the user's query. Instead, what it will do is *try all possible values for* T', and simulate DB_2 for each possibility. Since T and T' differ in only one position, there are only $\sqrt[3]{n}$ possibilities for T'. So, simulating the response of DB_2 for each of these possibilities only requires an additional $\sqrt[3]{n}$ bits (one bit per possibility).

In exactly this way, the first database can simulate the responses of DB_3 and DB_4. The total communication it sends to the user is therefore $3\sqrt[3]{n}+1$ bits.

The second database simulates DB_5, DB_6, and DB_7 in a similar manner (and acts exactly as DB_8 would). So it also sends $3\sqrt[3]{n}+1$ bits to the user. At this point, it is clear that the user can recover the desired answer (it just picks out the appropriate bits from the two responses, and then xor's them together as in the previous scheme), and the total communication complexity is $O(\sqrt[3]{n})$ bits.

3.1 Extending this Approach for More Databases

Let us try to generalize the approach of the previous section. Abstractly, we can view the eight triples of sets that the user sends to the eight databases in the original scheme as binary strings of length three: the tuple (R, S, T) corresponds to $(0, 0, 0)$, (R, S, T') corresponds to $(0, 0, 1)$, etc. In the improved scheme, we send the "sets" $(0, 0, 0)$ to the first database and let it simulate $(0, 0, 1)$, $(0, 1, 0)$, and $(1, 0, 0)$. We also send the "sets" $(1, 1, 1)$ to the second database and let it simulate $(1, 1, 0)$, $(1, 0, 1)$, and $(0, 1, 1)$. The key point is that each database simulates queries of Hamming distance 1 from the query it receives. This is what allows the total communication complexity to remain $O(\sqrt[3]{n})$.

Consider the 16-database, $O(\sqrt[4]{n})$-bit PIR protocol which is the extension of the schemes in Section 2. Here, the user sends "queries" in the form of 4-bit strings (i.e., the "query" $(0, 0, 0, 0)$ represents four random subsets (R, S, T, W) of $\{1, \ldots, \sqrt[4]{n}\}$). If we try to apply the improvement above, we see that we cannot do it using only two databases: if we send $(0, 0, 0, 0)$ to the first database and ask it to simulate query $(1, 1, 0, 0)$, for example, then it will have to try $\sqrt[4]{n} \times \sqrt[4]{n}$ different possibilities (namely, $\sqrt[4]{n}$ possibilities for each of R' and S') and the communication complexity will be too high. A little thought shows that in order to implement this improvement we need *four* databases: the user will send $(0, 0, 0, 0)$ to the first, $(1, 1, 0, 0)$ to the second, $(0, 0, 1, 1)$ to the third, and $(1, 1, 1, 1)$ to the fourth and now every 4-bit string in within Hamming distance 1 of at least one of these representatives.

For further information, see [1] or the extensive PIR references available at [2].

References

[1] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private Information Retrieval. *Journal of the ACM* 45(6): 965–981, 1998.

[2] W. Gasarch. http://www.cs.umd.edu/~gasarch/pir