Stanford University
Topics in Cryptography (CS 355) **Lecture #16**
Instructor: Lior Rotem **May 25, 2023**

Disclaimer *These lecture notes are aimed to serve as a supplementary resource, and are not written as a replacement for attending the in-person lecture. Some material might appear here in a more sketched form than in the lecture, and vice versa. These notes probably contain typos and might even contain errors. If you find any, please let me know.*

Outline *In this lecture, we talk about private information retrieval (PIR) schemes. We define what they are and focus on two settings: the two-server information-theoretic setting and the single-server computational setting. In both settings, we define a notion of security and see a construction satisfying it.*

1 General Purpose MPC and its Limitations

In the last few lectures, we saw how to compute general functions "securely"; namely, secure multiparty computation (MPC) and differential privacy (DP). In MPC, parties learn nothing but the output of the computed function. This is a very strong notion that allows to *a lot*. In particular, many cryptographic tasks can be rendered as an MPC task, and are thus solvable by general-purpose MPC protocols.

This is not without cost. One of the prices we pay for solving tasks with the big hammer of general purpose MPC is high communication complexity. The protocols that we saw require communication complexity that is at least proportional to the size of the ciruict computing the function. This can be prohibitive, as we will see in a minute. In the remainder of the course, we will see what can be done about that. Today we will focus on Private Information Retrieval (PIR).

Example: Private browsing. Consider the task of retrieving a webpage from a server. Abstractly, we can think of this process as the following simplified scenario: Our browser issues an index query to a server with access to the entire internet represented in some database form. The server then replies with the relevant information. But the query itself can reveal sensitive information (e.g., personal, medical, etc.). One solution is to use MPC. If we denote the database by DB and the query by i, then the client and the server can just run an MPC protocol for the function $f(\mathsf{DB}, i) = (\mathsf{DB}[i], \perp)$, where the first output is the client's and the second output is the server's. But the circuit computing this function is of size at least DB, which is massive! What can we do?

2 Private Information Retrieval

A private information retrieval (PIR) allows us to do exactly what is described above: A client holding some index i can learn the i-th entry of a server-stored database DB, without leaking anything about the index i to the server. But wait, Isn't that the definition of OT (oblivious transfer) we saw a couple of lectures ago? The answer is no. OT is a stronger primitive, that

also requires that the client learns nothing about DB other than $\mathsf{DB}[i]$. In PIR, we have no such requirements, which may lead to more efficient constructions.

A trivial solution and inherent limitations. Since there is no notion of "sender privacy" in PIR, a trivial solution is just to have the server send the entire database DB to the client. This is perfectly secure since the server obviously learns nothing about i. But this is also utterly inefficient. If DB is large (as in the private browsing example), this solution quickly becomes infeasible. So we are interested in a solution in which the total communication between the server and the client is *sublinear* in the size of DB. It is not hard to see that this is impossible to obtain if we do not put any computational restrictions on the server. Intuitively, if the communication is less than $|\mathsf{DB}|$, then it must "lose information" on at least one entry j of DB. Since the communication must encode $\mathsf{DB}[i]$, an unbounded server can compute j and deduce that $i \neq j$.

There are two ways to circumvent this inherent impossibility:

1. Replacing the server with two or more non-colluding servers. This is the original approach taken by Chor, Goldreich, Kushilevitz, and Sudan [CGKS95].
2. Assuming that the server is computationally bounded. This was initiated by Kushilevitz and Ostrovsky in [KO97].

We will see both approaches in this lecture. We will focus on the basic solutions proposed in the 1990s, but improvements are known, and we will briefly review the state of the art.

3 Two-Server PIR

In a two-server PIR scheme, we assume that there are two non-colluding servers S_0 and S_1, each of which holds a copy of the same DB. On input index i, the client interacts with each of the servers, and at the end of the interaction, the client should learn $\mathsf{DB}[i]$. However, none of the servers should learn anything about i. We will focus here on the case where each entry of the database is a bit, but this can be generalized.

Formally, a two-server PIR scheme is a triple of algorithms:

- $\mathit{Query}(N, i) \to (q_0, q_1)$. Here, N is the size of DB, i is the index that the client wants to learn, and q_b is the query to server b.
- $\mathit{Answer}(\mathsf{DB}, q) \to a$. As before, DB is the database. q is a query received from the client, and a is the server's response. Note that there is no difference here between the response generation of S_0 and that of S_1. This will be the case in the scheme that we will see, and is often the case, but one can also think of schemes in which both servers operate differently.
- $\mathit{Reconstruct}(a_0, a_1) \to b$. Here a_0, a_1 are the responses of the servers, and b is the reconstructed entry.

Correctness requires that the reconstructed bit should match the desired entry of the table. More

formally, for any $N \in \mathbb{N}$, any $i \in [N]$, and any $\mathsf{DB} \in \{0,1\}^N$, it holds that

$$\Pr\left[b = \mathsf{DB}[i] \ : \ \begin{array}{c}(q_0, q_1) \xleftarrow{\$} \mathit{Query}(N, i) \\ a_0 \leftarrow \mathit{Answer}(\mathsf{DB}, q_0),\ a_1 \leftarrow \mathit{Answer}(\mathsf{DB}, q_1) \\ b \leftarrow \mathit{Reconstruct}(a_0, a_1)\end{array}\right] = 1.$$

As for security, we ask that no server, working alone, can deduce anything about i. This is formalized as follows: For any DB and $i \in [|\mathsf{DB}|]$, let $Q_0(\mathsf{DB}, i)$ denote the random variable corresponding to q_0 on client input i and server input DB. Define $Q_1(\mathsf{DB}, i)$ analogously. Then, we require that for any $\mathsf{DB} \in \{0,1\}^*$ and for every pair (i, i'), it holds that

$$Q_0(\mathsf{DB}, i) \equiv Q_0(\mathsf{DB}, i')$$

and

$$Q_1(\mathsf{DB}, i) \equiv Q_1(\mathsf{DB}, i').$$

In words, this means that the distribution of the query to S_0 is the same for every possible client query, and the same holds for S_1.

Note that the definition assumes that the servers do not collude since it looks at $Q_0(\mathsf{DB}, i)$ and $Q_1(\mathsf{DB}, i)$ separately. If we look at the joint distribution $(Q_0(\mathsf{DB}, i), Q_1(\mathsf{DB}, i))$, it can leak a lot of information about i; in fact, it must, if we want to get sublinear communication.

An inefficient warm-up. As a warm-up, consider the following construction:

- $\mathit{Query}(N, i)$: Sample a random $q_0 \xleftarrow{\$} \{0,1\}^N$ and set $q_1 \leftarrow q_0 \oplus e_i$.
- $\mathit{Answer}(\mathsf{DB}, q)$: Output the inner product $a \leftarrow \langle \mathsf{DB}, q \rangle$.
- $\mathit{Reconstruct}(a_0, a_1)$: Output $b \leftarrow a_0 \oplus a_1$.

Correctness follows since

$$a_0 \oplus a_1 = \langle \mathsf{DB}, q_0 \rangle \oplus \langle \mathsf{DB}, q_0 \oplus e_i \rangle = \langle \mathsf{DB}, q_0 \oplus q_0 \oplus \mathsf{DB}[i] \rangle = \langle \mathsf{DB}, e_i \rangle = \mathsf{DB}[i].$$

Security follows from the fact that for any $i \in [N]$, both q_0 and q_1 are uniformly random N-bit strings.

The problem with this construction is that the communication is still linear in N. We just switched the direction in which the majority of the communication flows: from the client to the server instead of from the server to the client. Still, we have two extremes. Perhaps we can interpolate between them?

The CGKS construction. The CGKS construction relies on the idea above, but balances between the length of the queries and the answers. Assume for simplicity that N is a perfect square and let $n = \sqrt{N}$ (otherwise, we can pad DB to the closest perfect square). The idea is to view DB as an $n \times n$ matrix A. Then, q_0, q_1 can be of length n instead of N, and the response is of length n as well. In more detail, we now view the client's input as a pair of indices $(i, j) \in [n] \times [n]$. The scheme is then defined by:

- *Query*$(N, (i, j))$: Sample a random $q_0 \xleftarrow{\$} \{0,1\}^n$ and set $q_1 \leftarrow q_0 \oplus e_j$.
- *Answer*(DB, q): Parse DB as an $n \times n$ matrix A, and output the product $a \leftarrow A \cdot q$.
- *Reconstruct*(a_0, a_1): Compute $u \leftarrow a_0 \oplus a_1$ and output $u[i]$.

Correctness follows from the fact that

$$u = a_0 \oplus a_1 = A \cdot q_0 \oplus A \cdot q_1 = A \cdot (q_0 \oplus q_1) = A \cdot e_j = a_j,$$

where a_j is the jth column of A. Hence $a_j[i] = A_{i,j} = \mathsf{DB}[i, j]$.

Security follows as before: both q_0 and q_1 are uniformly random vectors for every choice of (i, j). As for communication complexity: Both the queries and the answers are in $\{0,1\}^n$, and so the total communication is $O(n) = O(\sqrt{N})$ which is indeed sublinear.

PIR and locally-decodable codes. The problem of multi-server information-theoretically secure PIR is very closely related to the problem of designing *locally-decodable codes* (LDCs). These are a special type of error-correcting codes. To decode 1 bit of the original message, one does not have to read the entire codeword. Say that we an LDC that maps messages of length N to codewords of length L over some alphabet $[C]$, and to decode a single bit from the message, it is sufficient to read k symbols from the codeword. If the marginal distribution over the location of each symbol does not depend on the index of the bit to be decoded, then this immediately gives us a k-server PIR scheme. The communication complexity per server is $\log L + C$. The above schemes can be seen as obtained from the Hadamard code. A more efficient 2-server scheme can be obtained by relying on more advanced codes. The best known 2-server scheme is due to Dvir and Gopi (2015), and achieves communication complexity of $O(N^{\sqrt{\log\log N/\log N}})$ (sublinear in N).

4 Computational Single-Server PIR

The above scheme can be seen as using "linearly homomorphic" secret sharing: The queries are additive secret shares q_0, q_1 of e_j. Each server applies a linear function A to the share q it receives, and then summing the responses $a_0 \oplus a_1$ gives $A \cdot (q_0 \oplus q_1)$ by the linearly homomorphic properties of the secret sharing scheme used.

The idea of [KO97] is to "compile" this into a single-server scheme by using linearly homomorphic encryption. Say that we have a symmetric encryption scheme for which there is an operation $+$ such that $Enc(k, m_0) + Enc(k, m_1) = Enc(k, m_0 \oplus m_1)$. For example, this can be achieved using ElGamal encryption if the number of additions in not too large.

A first construction. Using an encryption scheme as above, one can easily compiler the CGKS scheme into a computational single server scheme.

- *Query*$(N, (i, j))$: Sample a key k for the above encryption scheme and let q be the vector of encryptions of e_j's coordinates. That is, $q \xleftarrow{\$} (Enc(k, 0), \ldots, Enc(k, 1), \ldots, Enc(k, 0))$. Record k for reconstruction.
- *Answer*(DB, q): Parse DB as an $n \times n$ matrix A, and output the product $a \leftarrow A \cdot q$.

- *Reconstruct*(k, a): Compute u as the vector of decryptions of the entries of a: $u \leftarrow (Dec(k, a_1), \ldots, Dec(k, a_n))$. Output $u[i]$.

Correctness follows from the linear homomorphic property of the encryption scheme. Namely,

$$a = A \cdot q = (\sum_t A_{1,t} \cdot q_t, \ldots, \sum_t A_{n,t} \cdot q_t) = (Enc(k, A_{1,j}), \ldots, Enc(k, A_{n,j})).$$

Security follows from the semantic security of the encryption scheme. Namely, an encryption of e_j (by encrypting each entry separately) is computationally indistinguishable from an encryption of $e_{j'}$ for every $j, j' \in [n]$.

If the ciphertext size of the encryption scheme is $O(\lambda)$, then the overall communication is $O(\lambda n) = O(\lambda\sqrt{N})$.

Improving the communication complexity. The above protocol allows the client to learn the entire jth column of the database. But the client is only interested in $\mathsf{DB}[i, j]$. For correctness, the server only needs to send to ith ciphertext in a to the client, but we cannot to that because it would break security. The rough idea is thus to treat the server's response a as a new, **smaller**, database DB', and apply PIR again, so that the client only learns a_i, whose decryption is $\mathsf{DB}[i, j]$. In more detail, assume $N = n^3$. We now view DB as an $n \times n^2$ matrix A. Assume for simplicity that ciphertexts are 1-bit long (this cannot be the case, but we will revisit this assumption later).

The scheme is obtained by computing two queries q_0 and q_1. q_0 is an encryption of e_{j_0} as before, where j_0 is the column of A on which the desired bit lies. The server can compute $A \cdot q_0$ to get an encryption of the j_0th column of A. But now, this is a vector of length n^2. So we can view it as an $n \times n$ matrix B. Let j_1 be the column of B on which the desired bit lies. So in addition to q_0, the client also computes q_1 as an encryption of e_{j_1} and sends it to the server, who replies with $a_1 \leftarrow B \cdot q_1$. Overall, the query is composed of q_0 and q_1 and the response is a_1. These are all vectors of length $n = N^{1/3}$.

Sticking with the recursion. The above scheme lets the client learn $n = N^{1/3}$ entries of the database. This is still more than what we set out to do. We can continue with the recursion further; for a parameter ℓ, we can actually get a scheme with communication complexity $\tilde{O}(\ell \cdot N^{1/\ell})$.

However, using a deeper recursion, a technical subtlety arises. We (falsely) assumed that a ciphertext is 1 bit long. This is not the case. To see why this is a problem, consider the $O(N^{1/3})$-communication PIR scheme above. a_0 is a vector of n^2 bit-encryptions. If each encryption is λ bits long, a_0 is of length $\lambda^2 n^2$. This means that q_1 needs to be of length λn. If we again encrypt bit by bit, we need to encrypt λn bits, which means that a_1 will be of length $\lambda^2 n$. So the communication complexity explodes very quickly. To handle this, we need an encryption scheme that: (1) is linearly holomorphic; (2) supports a larger message space; and (3) has a good rate, in the sense that an encryption of a message m is only very slightly longer than m. One such encryption scheme is Damgård–Jurik scheme.

State of the art. By now, we know how to achieve single-server computationally-private PIR schemes with communication which is polylogarithmic in $|\mathsf{DB}|$. A link to a survey on PIR schemes can be found on the course's website.