Lecture 21: Cryptographic Techniques: Homomorphic Encryption, Private Information Retrieval.

Lecturer: Vinod Vaikuntanathan **Date:** April 23, 2026
Scribes: Albert Wang, Atul Nadig

1. Private Information Retrieval

Definition 1.1. Suppose you have a server with a *database* $\mathcal{D}$ (for our purposes, an array of bits), and a client with an index i who wants to know the value of $\mathcal{D}[i]$ without revealing i to the server.

Ideally, we'd want our protocol to have *correctness*, *privacy*, and be *efficient*.

> **Correctness**: The client always learns the correct value of $\mathcal{D}[i]$.
>
> **Privacy**: The server must be unable to distinguish between queries for any two indices i and j (given arbitrary or bounded time and computation).
>
> **Efficiency**: The number of bits sent in the communication should be $o(n)$ (where n is the number of bits in the database). Ideally, $\text{polylog}(n)$.

A trivial solution that attains correctness and privacy is to send the entire array $\mathcal{D}$ to be the client. However, it evidently doesn't meet our requirements that the scheme should be efficient.

It turns out that if we require perfect or statistical indistinguishability in our definition of privacy, then it is simply not possible to create such a scheme. To see this, note that the responses from the server to the client constitute essentially a compression of the entire database (since the queries are identical for all indices). But since the response can be at most $n - 1$ bits long, there must be a collision in its value for two different databases — which is a contradiction.

This argument rules out privacy against arbitrary computation. However, it no longer holds when we bound ourselves to polynomial time distinguishers, since it is difficult to find collisions.

Before we move on to cryptographic schemes, we'll construct as a motivating example a protocol that achieves perfect privacy against distinguishers with arbitrary computation in a slightly different setup.

Suppose we have two separate servers S_1, S_2 both with a copy of the database $\mathcal{D}$ but are unable to communicate. Take i and its one-hot encoding $u_i \in \mathbb{F}_2^n$. Select a random vector $r \in \mathbb{F}_2^n$; request $\langle \mathcal{D}, r \rangle$ from S_1 and $\langle \mathcal{D}, r \oplus u_i \rangle$ from S_2. The client can now compute

$$\langle \mathcal{D}, r \rangle \oplus \langle \mathcal{D}, r \oplus u_i \rangle = \langle \mathcal{D}, u_i \rangle = \mathcal{D}[i].$$

It is clear that this scheme is both correct and also information-theoretically private, as both the distributions of r and $r \oplus u_i$ are uniform.

This is certainly an improvement from the above case. Although the overall communication is $O(n)$, the server only sends $O(1)$ bits in response to $O(n)$-size queries. It is possible to improve the total communication complexity using a *meet in the middle* trick:

Index the database as a $\sqrt{n} \times \sqrt{n}$ table, indexed by (i, j) with $i, j \in \{1, 2, \ldots, \sqrt{n}\}$. Now, we can send the same queries as before, except we are instead looking for the entire column of the database that our index k lies in — suppose it is c_k. Then, we repeat the above argument with a one-hot encoding for c_k — say u_k and require the servers to return $\mathcal{D} \cdot r$ and $\mathcal{D} \cdot (r \oplus u_k)$, which are now $\sqrt{n}$ bits long, and we can recover the value of $\mathcal{D} \cdot u_k$ and from this, the value of $D[k]$.

Theorem 1.1 (Dvir-Gopi, 2014)**.** *It is possible to solve the* 2*-server case in* $O(2^{\sqrt{\log n}})$ *bits. [1]*

With this, we will construct a cryptographic solution to the one-server case, assuming *additively homomorphic encryption.*

1.1. Homomorphic encryption

Definition 1.2 (Homomorphic Encryption)**.** Let there be algorithms Gen, Enc, Dec for an encryption protocol, with an additional algorithm Eval as follows:

- Gen is an output-only randomized algorithm that outputs a secret key sk (and optionally, a public key).
- $\text{Enc}(\text{sk}, m)$ is a randomized algorithm that encrypts a message m with secret key sk.
- $\text{Dec}(\text{sk}, c)$ is a randomized algorithm that decrypts a code c encrypted with secret key sk and returns the original message m.
- $\text{Eval}(+, c_1, c_2)$ inputs two encrypted messages c_1, c_2 and outputs a new encrypted message c^* such that
$$\text{Dec}(\text{sk}, c^*) = \text{Dec}(\text{sk}, c_1) + \text{Dec}(\text{sk}, c_2).$$
Notice that this algorithm is not given the secret key, though it does have access to the public key. [1]

At first glance, it is surprising that such a scheme exists. However, it turns out that under certain hardness assumptions, we can construct such a scheme.

Definition 1.3 (Quadratic reciprocity problem)**.** Consider the following problem: given a modulo N and a residue a, determine whether or not a is a *quadratic residue* modulo N, i.e. if there exists x with $x^2 \equiv a \pmod{N}$.

This problem is believed to be computationally difficult in the case of arbitrary N. However, if N is a prime, it is possible to compute it efficiently using *quadratic reciprocity.*

[1] In general, we could require homomorphic encryption with arbitrary operations in place of +; it turns out such *fully homomorphic* schemes exist.

Let the public key be a product of primes $N = pq$, and let the secret key be the two primes (p, q) themselves. Let y be some non-quadratic residue modulo n.

We now encode messages $m \in \{0, 1\}$ as follows; let $r \in \mathbb{Z}_N$ uniform and take $\text{Enc}(0) = r^2 \pmod N$, $\text{Enc}(1) = r^2 \cdot y \pmod N$. To construct $\text{Dec}(\text{sk}, m)$, it suffices to reduce $\text{Enc}(\text{sk}, m)$ modulo p, q and decide if they are quadratic residues modulo p, q.

It is then possible to compute Eval by simply multiplying the two encrypted messages together, since

$$\text{Eval}(\text{Enc}(\text{sk}, m_1), \text{Enc}(\text{sk}, m_2)) = r_1^2 \cdot r_2^2 \cdot y^{m_1+m_2} \sim r^2 \cdot y^{m_1+m_2}.$$

With a homomorphic encryption scheme constructed, we can now combine the 2-server solution with an additive homomorphic encryption scheme to obtain a cryptographic solution to the 1-server case. First, we construct a similar scheme as before that achieves $O(n)$ client upload and $O(1)$ server download. The client will one-hot encode the indices $u_1, \ldots, u_n$ and send the values of $\text{Enc}(\text{sk}, u_i)$ to the server. Using Eval, the server can now compute $\text{Enc}(\text{sk}, \langle \mathcal{D}, u_i \rangle)$ by using Eval to add together all $\text{Enc}(\text{sk}, u_i)$ where $\mathcal{D}[i] = 1$. This can then be optimized with the meet-in-the-middle approach from before to achieve a cryptographically secure 1-server solution with just $O(\sqrt{n})$ commmunication.

However, this algorithm is wasteful, in a sense. Once the client receives the $O(\sqrt{n})$ length ciphertext, they simply throw away all but one entry. But really, we can think of this as another instance of the Private Information Retrieval problem, we are trying to choose one of these $\sqrt{n}$ bits but cannot let the server know which one it is. This suggests that we might employ a recursive approach to this problem.

In the first stage, we can instead split as a $N^\varepsilon \times N^{1-\varepsilon}$ matrix, and have u_k be a N^ε long vector. The recursion turns out to be

$$(N, N^\varepsilon) \to (N^{1-\varepsilon} \cdot \lambda, N^\varepsilon) \to \cdots \to$$

This process stops when we have $N_i^{1-\varepsilon} \cdot \lambda < N_i$, or when $N_i > \lambda^{1/\varepsilon}$. It turns out that it is possible to use fully homomorphic encryption to go even further, all the way to a communication complexity of $\text{polylog}(n)$. Here is the idea for it:

We can think of this entire problem as computing a function of the database, such that

$$f_{\mathcal{D}}(x) = \sum_{i \in \{1,2,\cdots,N\}} \delta_i(x) \cdot \mathcal{D}[i]$$

where $\delta_i(x) = 1 \iff i = x$.

If we treat i and x as binary strings $i = \overline{i_1 i_2 \ldots i_{\log N}}$, $x = \overline{x_1 x_2 \ldots x_{\log N}}$ of length $\log N$ with bits $i_b, x_b \in \{0, 1\}$, we may write $\delta_i(x)$ as a polynomial

$$\delta_i(x) = (-1)^\bullet \cdot \prod_{b=1}^{\log N} (x_b - (1 - i_b))$$

for a suitable choice of sign. From here the target $f_{\mathcal{D}}(x)$ is simply a polynomial function of bits $x_1, \ldots, x_{\log N}$. Since we assume a fully homomorphic encryption scheme, the server may compute the desired encrypted result and send it back to the client, achieving $O(\text{polylog}\, n)$ communication.

Remark 1.2. A similar problem, *keyword lookup*, where the server has a set of objects from some universe and the client wishes to determine whether their object x is in the collection, can be resolved by simply constructing a hash table and using the scheme we constructed above.

One drawback of the scheme we have created in this lecture is that we require the server to spend considerable (poly(n) per query) compute to respond. Without preprocessing, it is actually forced to have the server compute be $O(n)$, since otherwise by examining which part of the array is not touched by a query, it is possible to learn information about the query index i. However, with poly(n) preprocessing it actually becomes possible to achieve poly$(\log n)$ query complexity. [2]

References

[1] Zeev Dvir and Sivakanth Gopi. 2-server pir with sub-polynomial communication, 2014. URL https://arxiv.org/abs/1407.6692.

[2] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. Cryptology ePrint Archive, Paper 2022/1703, 2022. URL https://eprint.iacr.org/2022/1703.