Linearly Homomorphic Encryption and Private Information Retrieval

Notes by Alexandra Henzinger

MIT - 6.5610
Lecture 5 (February 20, 2024)

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

Logistics

- Pset 2 out today (due 3/1)

Outline

- Review: secret-key encryption from LWE
- Linearly homomorphic encryption
- Private information retrieval (PIR)
  - **Stretch break**
  - Square-root PIR scheme
  - Fast PIR from LWE

In the last lecture, we defined the learning-with-errors assumption (LWE) and saw how to use it to construct a secret-key encryption scheme. In this lecture, we will discuss a useful property of the LWE-based encryption scheme—namely, that it is *linearly homomorphic*—and see how to use it to build schemes for *private information retrieval*.

Review: secret-key encryption from LWE

In the last lecture, Yael showed how to build secret-key encryption from LWE [6]. We will begin by reviewing this construction, but slightly generalizing the notation: namely, we will encrypt *vectors* of messages, and our ciphertexts will be *matrices*.

As we saw last time, given parameters (m, n, q, χ), the LWE assumption states that the following distributions are indistinguishable

(to any computationally-bounded algorithm):

$$\left\{ \begin{bmatrix} \\ \mathbf{A} \\ \\ \end{bmatrix}, \begin{bmatrix} \\ \mathbf{A} \\ \\ \end{bmatrix} \times \begin{bmatrix} \\ \mathbf{s} \\ \\ \end{bmatrix} + \begin{bmatrix} \\ \mathbf{e} \\ \\ \end{bmatrix} \right\} \approx \left\{ \begin{bmatrix} \\ \mathbf{A} \\ \\ \end{bmatrix}, \begin{bmatrix} \\ \mathbf{u} \\ \\ \end{bmatrix} \right\},$$

where $\mathbf{A}$ is a random matrix in $\mathbb{Z}_q^{m \times n}$, $\mathbf{s}$ is a random vector in $\mathbb{Z}_q^n$, $\mathbf{e}$ is a random vector sampled from the (small) error distribution χ^m, and $\mathbf{u}$ is a random vector in $\mathbb{Z}_q^m$.

Then, to encrypt any message vector $\mathbf{v} \in \mathbb{Z}_2^m$, we can proceed as follows:

- We sample the secret key to be a random vector $\mathbf{s} \in \mathbb{Z}_q^n$.
- $\mathsf{Enc}\left(\mathbf{v} \in \mathbb{Z}_2^m, \mathbf{s} \in \mathbb{Z}_q^n\right) \to \mathsf{ct} \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$:
  - Sample a random matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$.
  - Sample a random error vector $\mathbf{e} \in \mathbb{Z}_q^m$, where each entry is sampled from the error distribution χ.
  - Output $(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \lfloor q/2 \rfloor \cdot \mathbf{v})$.
- $\mathsf{Dec}\left((\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m, \mathbf{s} \in \mathbb{Z}_q^n\right) \to \mathbf{v} \in \mathbb{Z}_2^m$:
  - Compute $\mathbf{c} = \mathbf{b} - \mathbf{A} \cdot \mathbf{s} \in \mathbb{Z}_q^m$.
  - Round each entry of $\mathbf{c}$ to the nearest multiple of $\lfloor q/2 \rfloor$, divide by $\lfloor q/2 \rfloor$, and output the result.

Effectively, the Enc function here is "padding" the message vector with the result of $(\mathbf{A} \cdot \mathbf{s} + \mathbf{e})$, which the LWE assumption tells us looks uniformly random to any computationally-bounded algorithm. Then, the Dec function is subtracting out a large portion of the pad (namely, the $\mathbf{A} \cdot \mathbf{s}$ component), and finally removing the remaining "small" error with the following check: for each location in the resulting vector,

- if it is "close" to $\lfloor q/2 \rfloor$, output that the message was "1".
- if it is "close" to 0, output that the message was "0".

Decryption here is going to recover the correct message, as long as the error sampled from the distribution χ is small enough. For example, if we take χ to be the distribution that outputs a uniformly random value in the range $[-B, \ldots, B]$, then decryption will always succeed as long as $B < q/4$.

Linearly homomorphic encryption

As Henry mentioned in the first class, one of the goals of cryptography is to compute on encrypted data. It turns out that our encryption

scheme from LWE allows for a limited version of this: we can compute *linear* functions (i.e., additions) directly on encrypted data.

Specifically, an encryption scheme Enc is *linearly homomorphic* if, for any secret key k and any two messages m_0, m_1, it holds that:

$$\mathsf{Enc}(k, m_0) + \mathsf{Enc}(k, m_1) = \mathsf{Enc}(k, m_0 + m_1).$$

The secret-key encryption scheme from LWE meets this property. That is, for any secret key $\mathbf{s} \in \mathbb{Z}_q^n$ and any two message vectors $\mathbf{v}_0, \mathbf{v}_1 \in \mathbb{Z}_2^m$, we have:

$$\begin{aligned}
&\mathsf{Enc}(\mathbf{v}_0, \mathbf{s}) + \mathsf{Enc}(\mathbf{v}_1, \mathbf{s}) \\
&= (\mathbf{A}_0,\ \mathbf{A}_0 \cdot \mathbf{s} + \mathbf{e}_0 + \lfloor q/2 \rfloor \cdot \mathbf{v}_0) + (\mathbf{A}_1,\ \mathbf{A}_1 \cdot \mathbf{s} + \mathbf{e}_1 + \lfloor q/2 \rfloor \cdot \mathbf{v}_1) \\
&= (\mathbf{A}_0 + \mathbf{A}_1,\ \mathbf{A}_0 \cdot \mathbf{s} + \mathbf{e}_0 + \lfloor q/2 \rfloor \cdot \mathbf{v}_0 + \mathbf{A}_1 \cdot \mathbf{s} + \mathbf{e}_1 + \lfloor q/2 \rfloor \cdot \mathbf{v}_1) \\
&= (\mathbf{A}_0 + \mathbf{A}_1,\ (\mathbf{A}_0 + \mathbf{A}_1) \cdot \mathbf{s} + (\mathbf{e}_0 + \mathbf{e}_1) + \lfloor q/2 \rfloor \cdot (\mathbf{v}_0 + \mathbf{v}_1)) \\
&\approx \left(\underbrace{\mathbf{A}_0 + \mathbf{A}_1}_{\mathbf{A}_{\text{new}}},\ \underbrace{(\mathbf{A}_0 + \mathbf{A}_1)}_{\mathbf{A}_{\text{new}}} \cdot \mathbf{s} + \underbrace{(\mathbf{e}_0 + \mathbf{e}_1)}_{\mathbf{e}_{\text{new}}} + \lfloor q/2 \rfloor \cdot \underbrace{(\mathbf{v}_0 \oplus \mathbf{v}_1)}_{\mathbf{v}_{\text{new}}} \right)
\end{aligned}$$

In other words, the output of $\mathsf{Enc}(\mathbf{v}_0, \mathbf{s}) + \mathsf{Enc}(\mathbf{v}_1, \mathbf{s})$ corresponds to a fresh ciphertext encrypting the new message $(\mathbf{v}_0 \oplus \mathbf{v}_1)$. There are only two slight caveats here:

1. the new error component is $(\mathbf{e}_0 + \mathbf{e}_1)$, i.e., it corresponds to the sum of *two* samples from the error distribution χ^m.
2. if q is odd, it only approximately holds that

   $$\lfloor q/2 \rfloor \cdot (\mathbf{v}_0 + \mathbf{v}_1) \approx \lfloor q/2 \rfloor \cdot (\mathbf{v}_0 \oplus \mathbf{v}_1) \mod q,$$

   since $2 \cdot \lfloor q/2 \rfloor \in \{-2, -1, 0\} \mod q$.

These two differences only affect the *lowest*-order bits of our ciphertexts. As a result, we can handle both of these issues by simply setting our parameters such that decryption succeeds with high probability, even with a slightly larger error distribution. For example, decryption will always succeed if we set $2B + 2 < q/4$. More broadly, we can set our parameters suitably to allow for any (polynomial) number of homomorphic additions to be performed on these LWE ciphertexts.

This *linear homomorphism* turns out to be extremely useful in building cryptosystems that perform some (restricted) computations on encrypted data—for example, aggregating encrypted votes. In the remainder of this lecture, we will see how to use it to build private information retrieval.

Private information retrieval

Today, when we query a remote database, the server that hosts the database learns our query. In many cases, this leaks personal information about us: for example, WebMD learns our medical conditions, AirBnB learns our travel plans, and Google learns our search queries. In an ideal world, we would like to query remote databases *privately*—that is, without revealing what we are searching for. We can achieve this type of functionality using cryptography, and in particular linearly homomorphic encryption.

Specifically, we will restrict our attention to the following simplified problem statement. Say we have:

- a server that holds an N-bit database, $D \in \mathbb{Z}_2^N$,
- a user that holds an index $i \in \{0, 1, \ldots, N-1\}$, and
- our user's goal is to learn the i-th bit of the database, D_i, without revealing i to the server.

To solve this problem, we define a *private information retrieval* (PIR) scheme to be a triple of randomized algorithms $(\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ that meet the following three properties [2]:

1. **Correctness**: for any index $i \in \{0, 1, \ldots, N-1\}$, any database $D \in \mathbb{Z}_2^N$, and any security parameter n,

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

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

2. **Security**: for any two indices $i, j \in \{0, 1, \ldots, N-1\}$,

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

   That is, the server's view looks computationally indistinguishable whether the user is making a query for i or making a query for j.

3. **Succinctness**: the total bit-length of the client's query (output by $\mathsf{Query}(1^n, \cdot)$) and the server's answer (output by $\mathsf{Answer}(D, \cdot)$) is less than N. This ensures that the total number of bits communicated between the user and the server is smaller than N.

   This requirement is needed to rule out trivial PIR schemes where the user just downloads the whole database from the server.

Many beautiful lines of work have showed how to construct PIR with communication that scales only polylogarithmically with the database size, N, from an array of cryptographic assumptions. In this class, we will see how to construct non-trivial PIR with communication complexity $O(\sqrt{N})$ from any linearly homomorphic encryption scheme. Then, we will discuss how to make this PIR scheme concretely efficient using the LWE assumption.

Why care? At first glance, it may seem like the PIR problem statement is very contrived. However, it turns out that, if we can build efficient protocols for this problem, we can then directly apply them to solve many more natural and more complex tasks: for example, private reads from a database with arbitrary-length records, and private lookups to a key-value store by key (rather than by index).

Server work in PIR. While modern PIR has reasonably small communication, a major bottleneck is the server-side computation required to answer PIR queries. In both of the PIR schemes we will see, the server's work is *linear* in the database size N—essentially, the server needs to touch every bit in the database to answer a query. This turns out to be inherent [1]: if the server didn't touch any one location in the database, it would learn that the user was not reading that entry. An exciting line of work on PIR has aimed to find ways to circumvent this lower bound: e.g., by preprocessing the computation, or by amortizing it across many queries or many clients.

A classic PIR scheme: Square-root PIR

We will start by discussing a classic PIR scheme due to Kushilevitz and Ostrovsky [4]. In this scheme, the key idea is that our server will represent its N-bit database as a matrix, $\mathbf{D} \in \mathbb{Z}_2^{\sqrt{N} \times \sqrt{N}}$. Now, assume that the user wants to read the database bit at location $(i, j) \in [\sqrt{N}] \times [\sqrt{N}]$. Given a linearly-homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, the protocol proceeds as follows:

- $\mathsf{Query}\,(1^n,\ (i,j))$:
  - 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 secret key $\mathsf{sk} \leftarrow \mathsf{Gen}(1^n)$.
  - Output the query vector $\mathsf{qu} \leftarrow \mathsf{Enc}(\mathsf{sk}, \mathbf{u}_j)$, along with the client-held state $\mathsf{st} \leftarrow (\mathsf{sk}, i)$.
- $\mathsf{Answer}(\mathbf{D},\ \mathsf{qu})$:
  - Output the answer vector $\mathsf{ans} \leftarrow \mathbf{D} \cdot \mathsf{qu}$.
- $\mathsf{Reconstruct}(\mathsf{ans},\ \mathsf{st})$:
  - Parse the client-held state st as (sk, i).
  - Decrypt the answer vector as $\mathbf{v} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ans})$, 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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is linearly homomorphic, we have that:

$$\begin{aligned}\mathsf{ans} &= \mathbf{D} \cdot \mathsf{Query}\,(1^n,\,(i,j)) \\ &= \mathbf{D} \cdot \mathsf{Enc}(\mathsf{sk}, \mathbf{u}_j) \\ &= \mathsf{Enc}(\mathsf{sk}, \underbrace{\mathbf{D} \cdot \mathbf{u}_j}_{\text{mod } 2}).\end{aligned}$$

   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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

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.

Fast PIR from LWE

To make this square-root PIR scheme practical, we need to ensure that both its communication cost and the server-side computation cost are reasonable. One approach to doing so is to instantiate it with lattice-based cryptography, and specifically with the secret-key encryption scheme from LWE that we saw at the start of this lecture.

When we do so naïvely, we arrive at the following PIR protocol between a server that holds a database matrix $\mathbf{D} \in \mathbb{Z}_2^{\sqrt{N} \times \sqrt{N}}$ and a user that wants to read the bit at location $(i, j) \in [\sqrt{N}] \times [\sqrt{N}]$:

Here, we will rely on the LWE assumption with parameter $m \geq \sqrt{N}$.

- $\mathsf{Query}\,(1^n,\,(i,j)) \to (\mathsf{qu} \in \mathbb{Z}_q^{\sqrt{N} \times n} \times \mathbb{Z}_q^{\sqrt{N}}, \mathsf{st} \in \mathbb{Z}_q^n \times [\sqrt{N}])$:
  - 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 random matrix $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{\sqrt{N} \times n}$.
  - Sample a random secret key vector $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$.
  - Sample a random error vector $\mathbf{e} \xleftarrow{R} \chi^{\sqrt{N}}$.
  - Output the query $\mathsf{qu} \leftarrow (\mathbf{A},\, \mathbf{A} \cdot \mathbf{a} + \mathbf{e} + \lfloor q/2 \rfloor \cdot \mathbf{u}_j)$, along with the client-held state $\mathsf{st} \leftarrow (\mathbf{s}, i)$.
- $\mathsf{Answer}(\mathbf{D},\, \mathsf{qu}) \to \mathsf{ans} \in \mathbb{Z}_q^{\sqrt{N} \times n} \times \mathbb{Z}_q^{\sqrt{N}}$:
  - Parse the query qu as $(\mathbf{A}, \mathbf{b})$.
  - Output the answer $\mathsf{ans} \leftarrow (\mathbf{D} \cdot \mathbf{A}, \mathbf{D} \cdot \mathbf{b})$.

- $\mathsf{Reconstruct}(\mathsf{ans}, \mathsf{st}) \to b \in \{0,1\}$:
  - Parse the client-held state st as $(\mathbf{s}, i)$.
  - Parse the answer ans as $(\mathbf{H}, \mathbf{c})$.
  - Compute $\mathbf{v} = \mathbf{c} - \mathbf{H} \cdot \mathbf{s} \in \mathbb{Z}_q^{\sqrt{N}}$.
  - Round the i-th entry of $\mathbf{v}$ to the nearest multiple of $\lfloor q/2 \rfloor$, divide it by $\lfloor q/2 \rfloor$, and output the result.

This PIR scheme works (in the sense that it is correct, secure, and succinct), but it is far from concretely efficient because of the overhead of shipping around and computing on the large $\mathbf{A}$-matrices, whose size scales with the lattice dimension n. Fortunately though, we can do much better by taking advantage of three simple yet effective optimizations [3]:

In practice, we need to take the lattice dimension $n \approx 1024$ and the modulus $q \approx 2^{16}$, so the total communication to make a private query to an N-bit database would be roughly $10^4 \cdot \sqrt{N}$ bits.

1. LWE is still secure, even if the same $\mathbf{A}$-matrix is reused in polynomially many problem instances, assuming we use a random and independently generated secret vector $\mathbf{s}$ and error vector $\mathbf{e}$ each time [5]. As a result, we can reuse the same $\mathbf{A}$-matrix in our PIR scheme to build polynomially many PIR queries—even by *different* users.

   In particular, we will now think of the $\mathbf{A}$-matrix as a public parameter of the PIR scheme, known to all of the users and to the server. (In practice, $\mathbf{A}$ may be generated using a PRG applied to short, public seed.) With this optimization, we save a large amount of communication, because the $\mathbf{A}$-matrix no longer needs to be sent from the users to the server.

   Intuitively speaking, this is the case because the $\mathbf{A}$-matrix is revealed in the clear, so the server could "simulate" arbitrarily many LWE samples with the same $\mathbf{A}$ (but different $\mathbf{s}$ and $\mathbf{e}$) in its head. This statement can be formalized via a polynomial-time reduction.

2. Since the $\mathbf{A}$-matrix is known in advance, the server can now compute the result of $\mathbf{H} \leftarrow \mathbf{D} \cdot \mathbf{A}$ once, ahead of time, and store it. This saves a great deal of computation because the server can now simply send back the precomputed $\mathbf{H}$ to answer each query, rather than computing it each time.

3. The users can also prefetch the value of $\mathbf{H} \leftarrow \mathbf{D} \cdot \mathbf{A}$ once, ahead of time. One way to think of $\mathbf{H}$ here is as a "hint" about the database contents. This greatly shrinks the overall scheme's per-query download. In particular, both the upload and the per-query download are now *independent* of the lattice dimension, n.

Taken together, these optimizations result in a PIR scheme that is (reasonably) efficient. When $q = 2^{16}$, to make a PIR query, the user uploads $\log q \sqrt{N} \approx 16\sqrt{N}$ bits to the server. To answer the query, the server performs one 16-bit addition and one 16-bit multiplication per bit of the database (to compute $\mathbf{D} \cdot \mathbf{b}$). Finally, the server sends back $\log q \sqrt{N} \approx 16\sqrt{N}$ bits to the user.

To see an example of (a slight variant of) this scheme in action, check out: `https://playground.blyss.dev/passwords`.

References

[1] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *J. Cryptol.*, 2004.

[2] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995.

[3] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, August 2023. USENIX Association.

[4] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, 1997.

[5] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

[6] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 2009.