Fully Homomorphic Encryption (part I)

Notes by Alexandra Henzinger

MIT - 6.5610
Lecture 7 (February 24, 2024)

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

Outline

In this class (and the next class), we will cover one of the most exciting and surprising advances in cryptography in the last twenty years: *fully homomorphic encryption*—that is, encryption schemes that let us evaluate *arbitrary* functions directly on encrypted data, without ever decrypting it.

Fully homomorphic encryption (FHE) is an extremely powerful tool: from it, we can construct many of the primitives we have seen so far (private information retrieval, public-key encryption, etc), as well as a huge array of new tools and applications. In particular, we can securely and privately *outsource* the computation of any function to an untrusted server by encrypting our inputs, sending them to the server, having the server homomorphically evaluate its chosen function directly on the encrypted inputs, and then letting the server send us back the encrypted result.

History of FHE. While the idea of FHE was first suggested in 1978 by Rivest, Adleman, and Dertouzos [6], it took until 2009 for the first secure (i.e., unbroken) construction to be proposed in a paper by Craig Gentry—who, at the time, was a PhD student at Stanford [3]. Since then, there have been many improvements to known FHE schemes. In 2011, Brakerski and Vaikuntanathan [1] showed how to construct FHE directly from the learning-with-errors (LWE) assumption, which we covered last week [5]. In 2013, Gentry, Sahai, and Waters gave an elegant and conceptually simple construction of FHE (sometimes referred to as the "GSW scheme") [4], which also relies only on LWE. We will see this GSW construction in the next two lectures.

In particular, we will cover:

- FHE definition (today)
- **Stretch break**

- Constructing FHE from LWE:
  - Step one (today): *levelled FHE* that can support bounded-depth computations
    - Background and intuition
    - The GSW construction
  - Step two (next Monday): *bootstrapping* FHE to support arbitrary-depth computations
- FHE applications (next Monday)
- Open questions in FHE (next Monday)

FHE definition and syntax

We will start by formally defining the syntax of a FHE scheme. In particular, we will work with a FHE scheme where:

- the message space is bits (i.e., $\mathcal{M} = \mathbb{Z}_2$), and
- the model of computation is Boolean circuits (i.e., circuits composed of additions and multiplications gates over $\mathbb{Z}_2$—or, equivalently, XOR and AND gates).

Since additions and multiplications over $\mathbb{Z}_2$ are "mod 2"-complete, this is an extremely general model of computation that captures the programs we can write on computers today.

The syntax that we will see here is for a *secret-key* FHE scheme, though public-key variants exist as well. In fact, the distinction is not particularly important because sufficiently powerful secret-key FHE implies public-key FHE.

We will define a FHE scheme with respect to a function class $\mathcal{C}$, which contains all of the circuits that our scheme can homomorphically evaluate. Given such a function class $\mathcal{C}$, we say that a FHE scheme is a tuple of four efficient (i.e., PPT) randomized algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ that look as follows:

Here, the differences to standard secret-key encryption, as we saw in lecture 4, are highlighted in blue.

- $\mathsf{KeyGen}(1^n) \to (\mathsf{sk}, \mathsf{ek})$.

  Given as input the security parameter, $n \in \mathbb{N}$, the KeyGen algorithm outputs:

  - a decryption key sk (kept secret by the user), and
  - an evaluation key ek (which we think of as public, because it is revealed to any party that will perform homomorphic computations on ciphertexts encrypted with sk).

- $\mathsf{Enc}(\mathsf{sk}, \mu) \to \mathsf{ct}$.

  As usual, the Enc algorithm takes as input a secret key sk and a message bit μ and outputs a ciphertext ct encrypting μ.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mu$.

As usual, the Dec algorithm takes as input a secret key sk and a ciphertext ct and outputs the message bit μ encrypted by ct.

- $\mathsf{Eval}(\mathsf{ek}, F, \mathsf{ct}_1, \dots, \mathsf{ct}_\ell) \to \tilde{\mathsf{ct}}$.

  The Eval algorithm is given as input:

  - an evaluation key ek,
  - a Boolean circuit $F : \mathbb{Z}_2^\ell \to \mathbb{Z}_2$ that belongs the supported function class $\mathcal{C}$, and
  - ℓ ciphertexts representing the inputs to F.

  This notation means that the circuit F maps ℓ input bits to a single output bit.

  Then, Eval produces a ciphertext that encrypts the circuit output (i.e., the evaluation of F on each of the encrypted input bits). Here, we call the output $\tilde{\mathsf{ct}}$ the "evaluated" ciphertext.

We require the four algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ to satisfy the following three properties:

1. **Correctness**: for every security parameter $n \in \mathbb{N}$, for every circuit $F \in \mathcal{C}$, and for any inputs $\mu_1, \dots, \mu_\ell \in \mathbb{Z}_2$ to F, it holds that:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \tilde{\mathsf{ct}}) = F(\mu_1, \dots, \mu_\ell) : \begin{array}{r} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^n) \\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_i) \text{ for all } i \in [\ell] \\ \tilde{\mathsf{ct}} \leftarrow \mathsf{Eval}(\mathsf{ek}, F, \mathsf{ct}_1, \dots, \mathsf{ct}_\ell) \end{array}\right] \geq 1 - \mathrm{negl}(n).$$

   That is, encrypting inputs to F with Enc, then calling Eval, and finally decrypting with Dec correctly produces the evaluation of circuit F on the given inputs (with high probability).

2. **CPA-security** (as defined in Lecture 2): For every PPT adversary $\mathcal{A}$ there exists a negligible function $\mu : \mathbb{N} \to \mathbb{N}$ such that for every security parameter $n \in \mathbb{N}$, $\mathcal{A}$ wins in the following game with probability at most $\frac{1}{2} + \mu(n)$:

   - The challenger samples $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^n)$ and sends $(1^n, \mathsf{ek})$ to the adversary.
   - $\mathcal{A}$ chooses $m_i \in \{0, 1\}$ and receives $c_i \leftarrow \mathsf{Enc}(\mathsf{sk}, m_i)$.
     This step can be repeated polynomially many times.
   - The challenger chooses a random bit $b \leftarrow \{0, 1\}$ and sends the ciphertext $c \leftarrow \mathsf{Enc}(\mathsf{sk}, b)$ to the adversary.
   - The adversary $\mathcal{A}$ upon receiving c outputs a bit b' and it wins if and only if $b' = b$.

3. **Compactness**: for every security parameter $n \in \mathbb{N}$, for every circuit $F \in \mathcal{C}$, and for any inputs $\mu_1, \dots, \mu_\ell \in \mathbb{Z}_2$ to F, let

$$\begin{array}{r} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^n) \\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_i) \text{ for all } i \in [\ell] \\ \tilde{\mathsf{ct}} \leftarrow \mathsf{Eval}(\mathsf{ek}, F, \mathsf{ct}_1, \dots, \mathsf{ct}_\ell) \end{array}$$

Then, it must hold that the *bit-length* of both the "fresh" ciphertexts ct_i and the "evaluated" ciphertext $\tilde{\mathsf{ct}}$ depends only on the security parameter n, and is independent of the size $|F|$ of the evaluated circuit or the number of inputs ℓ.

Since Enc and Eval must be PPT algorithms, the bit-length of fresh ciphertexts and of evaluated ciphertexts here will be $O(\text{poly}(n))$.

There also exist weaker but still interesting definitions of compactness, which require that the size of the evaluated ciphertext be *sublinear* in the size of the evaluated circuit, $|F|$.

Why do we need all three properties? A meaningful FHE scheme must simultaneously satisfy all three properties described above—otherwise, trivial and uninteresting solutions exist:

1. Without correctness, we could just use an encryption scheme that is not homomorphic.
2. Without security, each ciphertext could just consist of the message that it is encrypting in the clear.
3. Without compactness, the Eval algorithm could just output the concatenation of the circuit F and each of the input ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$. Then, Dec could just decrypt each of the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$ and directly evaluate the circuit F on the resulting values.

With respect to what function class can (and will) we build FHE? We defined an FHE scheme with respect to a *function class* that governs what type of circuits the scheme can homomorphically evaluate. Clearly, the more expressive this function class is, the more powerful our FHE scheme will be. Various flavors of homomorphic encryption exist for various function classes.

In particular, any Boolean circuit can be written using only addition and multiplication gates over $\mathbb{Z}_2$. We will differentiate between circuits based on the number of addition/multiplication gates that they contain, as well as the layout of these gates.

- **Circuits with only addition gates**: a circuit that consists of only additions over $\mathbb{Z}_2$ can be evaluated using a *linearly homomorphic* encryption scheme. As we saw in lecture 5, we can build linearly homomorphic encryption from LWE.

This is known as the Regev encryption scheme, since it was introduced by Regev in [5].

- **Circuits with only multiplication gates**: similarly, a circuit that consists of only multiplications over $\mathbb{Z}_2$ can be evaluated using a *multiplicatively homomorphic* encryption scheme. Standard encryption schemes such as El Gamal [2] and RSA [7] are multiplicatively homomorphic, but not over $\mathbb{Z}_2$, rather over Z_p^* where p is a large prime for El Gamal, and over Z_n^* where n is a product of two large primes for RSA.

We do not cover these schemes in this course since they are not post-quantum secure.

In the next two lectures, we will show how to build FHE for very general function classes:

- **Arbitrary circuits of a bounded depth**: today, we will see how to build an FHE scheme that supports bounded-depth circuits, also called a *levelled* FHE scheme.

  Roughly, the reason why our scheme will only support bounded-depth computations is because each homomorphic evaluation of an addition or multiplication gate will incur some error growth. Once this error grows too large, the ciphertexts will be garbled and can no longer be decrypted. So, to avoid this, the number of addition and multiplication gates that we can evaluate will be bounded.

- **Arbitrary circuits**: in the next lecture, we will see how to boost a levelled FHE scheme to one that can support arbitrary circuits, under some additional assumptions. The key idea here will be a procedure to "refresh" ciphertexts to reset their error to a small level. This beautiful technique will let us homorphically evaluate any Boolean circuit under encryption, with an overhead that is only polynomial in the security parameter n.

To build these FHE schemes, we will see how to homomorphically evaluate addition gates and multiplication gates on ciphertexts. Composing these two types of gates will then give us the power to evaluate general Boolean circuits.

Building "levelled" FHE: Background and intuition

We will start by providing some background and intuition for Gentry, Sahai, and Water's construction of levelled FHE.

At a high level, the GSW scheme relies on the observation that the eigenvectors of matrices are preserved across addition and multiplication. Specifically, for any dimension $\ell \in \mathbb{N}$, let $\mathbf{C}_1, \mathbf{C}_2 \in \mathbb{Z}_q^{\ell \times \ell}$ denote two matrices. Let vector $\mathbf{v} \in \mathbb{Z}_q^\ell$ be an eigenvector of matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ with eigenvalues $\lambda_1 \in \mathbb{Z}_q$ and $\lambda_2 \in \mathbb{Z}_q$, respectively. This means that the following relations hold:

$$\mathbf{C}_1 \cdot \mathbf{v} = \lambda_1 \cdot \mathbf{v} \in Z_q^\ell$$
$$\mathbf{C}_2 \cdot \mathbf{v} = \lambda_2 \cdot \mathbf{v} \in \mathbb{Z}_q^\ell$$

Then, we get the following two properties:

- $(\mathbf{C}_1 + \mathbf{C}_2) \cdot \mathbf{v} = \mathbf{C}_1 \cdot \mathbf{v} + \mathbf{C}_2 \cdot \mathbf{v} = (\lambda_1 + \lambda_2) \cdot \mathbf{v}$

  That is, the vector $\mathbf{v}$ remains an eigenvector of the summed matrix $(\mathbf{C}_1 + \mathbf{C}_2)$, with associated eigenvalue $\lambda_1 + \lambda_2 \in \mathbb{Z}_q$.

- $(\mathbf{C}_1 \cdot \mathbf{C}_2) \cdot \mathbf{v} = \mathbf{C}_1 \cdot (\mathbf{C}_2 \cdot \mathbf{v}) = \mathbf{C}_1 \cdot \mathbf{v} \cdot \lambda_2 = \mathbf{v} \cdot \lambda_1 \cdot \lambda_2$

That is, the vector $\mathbf{v}$ remains an eigenvector of the product matrix $(\mathbf{C}_1 \cdot \mathbf{C}_2)$, with associated eigenvalue $\lambda_1 \cdot \lambda_2 \in \mathbb{Z}_q$.

Intuition. To build FHE, we will take advantage of this behavior of eigenvectors as follows: in our scheme, the secret key will be a vector with entries in $\mathbb{Z}_q$. Then, roughly speaking, our ciphertexts will be matrices in $\mathbb{Z}_q^{\ell \times \ell}$ whose *eigenvector* will be the secret-key vector and whose associated *eigenvalue* will be the message being encrypted. With this setup, we see that:

- To decrypt a ciphertext matrix, it suffices to multiply the matrix by the secret-key vector. This produces the underlying message, scaled by the secret-key vector.
- To evaluate an addition gate on two ciphertexts, it suffices to add up the two matrices. This produces a ciphertext that encrypts the *sum* of the underlying messages.
- To evaluate a multiplication gate on two ciphertexts, it suffices to multiply the two matrices. This produces a ciphertext that encrypts the *product* of the underlying messages.

Unfortunately, we cannot exactly instantiate this template to build a secure encryption scheme—after all, efficient algorithms exist to find the eigenvectors of matrices. However, using the learning-with-errors (LWE) assumption, we can construct an *approximate* version of this scheme. Specifically, in our FHE scheme, the secret-key vector $\mathbf{s}$ will be a *noisy* eigenvector of each ciphertext matrix $\mathbf{C} \in Z_q^{\ell \times \ell}$ such that the following equation holds:

$$\mathbf{C} \cdot \mathbf{s} = \mathbf{s} \cdot \mu + \mathbf{e}, \tag{1}$$

where $\mu \in Z_2$ is the message being encrypted and $\mathbf{e} \in \mathbb{Z}_q^{\ell}$ is a small error vector. We will carefully set up our encryption scheme so that this invariant (i.e., equation (1)) holds after encryption, and is preserved after computing (a bounded number of) homomorphic addition and multiplication gates.

Building "levelled" FHE: the GSW construction

Now that we have built up some intuition, we will dive into the GSW construction. Here, we will work with the LWE assumption with parameters (m, n, q, χ) with sufficiently large m.

Specifically, we need $m \geq (n+1) \cdot \log q \cdot c$, where c is a bound on the number of messages we will encrypt with any one secret key.

Given these LWE parameters, we will let $\ell = (n+1) \cdot \log q$. Then, the construction works as follows:

- $\mathsf{KeyGen}(1^n) \to \mathbf{s} \in \mathbb{Z}_q^{n+1}$.
  - Sample a random vector $\mathbf{s}' \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$.

For now, we will ignore the evaluation key ek. We will need this evaluation key in the next lecture, to boost our scheme to handle arbitrary-depth encryption circuits.

  - Output the vector $\mathbf{s} = \begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix} \in \mathbb{Z}_q^{(n+1)}$ as the secret key.
- $\mathsf{Enc}(\mathbf{s} \in \mathbb{Z}_q^{(n+1)}, \mu \in \mathbb{Z}_2) \to \mathbf{C} \in \mathbb{Z}_q^{\ell \times (n+1)}$.
  - Sample a random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{\ell \times n}$.
  - Sample an error vector $\mathbf{e} \xleftarrow{R} \chi^{\ell}$.
  - Build the matrix $\mathbf{B} = \begin{pmatrix} \mathbf{A} & || & \mathbf{A}\mathbf{s}' + \mathbf{e} \end{pmatrix} \in \mathbb{Z}_q^{\ell \times (n+1)}$. That is, $\mathbf{B}$ here is the $\mathbf{A}$-matrix with the vector computed as $(\mathbf{A}\mathbf{s}' + \mathbf{e})$ appended as another column.
  - Let $\mathbf{G} \in \mathbb{Z}_q^{\ell \times (n+1)}$ be some fixed, public *error-correcting matrix* that we will define later.
  - Output the matrix $\mathbf{C} = \mathbf{B} + \mu \cdot \mathbf{G}$ as the ciphertext.

As we will see, we will need to carefully craft this matrix $\mathbf{G}$ to allow us to support homomorphic multiplications.

- $\mathsf{Dec}(\mathbf{s} \in \mathbb{Z}_q^{(n+1)}, \mathbf{C} \in \mathbb{Z}_q^{\ell \times (n+1)}) \to \mu \in \mathbb{Z}_2$.
  - Compute the vector $\mathbf{v} = \mathbf{C} \cdot \mathbf{s}$.
  - Output "0" if the magnitude of each entry of $\mathbf{v}$ is small (i.e., less than $q/4$). Otherwise, output "1."

Before we describe how to perform homomorphic additions and multiplications, we will first argue that this scheme is (1) secure, and (2) correct, in the sense that calling Dec on a fresh ciphertext output by Enc produces the correct result.

Security. The LWE assumption with parameters (m, n, q, χ) tells us that

$$(\mathbf{A}, \mathbf{A}\mathbf{s}' + \mathbf{e}) \approx_C (\mathbf{A}, \mathbf{u}),$$

where $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{m \times n}$, $\mathbf{s}' \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \xleftarrow{R} \chi^m$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m$. So, under the LWE assumption, the matrix $\mathbf{B} \in \mathbb{Z}_q^{\ell \times (n+1)}$ generated as part of the Enc algorithm must computationally indistinguishable from a random matrix $\mathbf{U} \xleftarrow{R} \mathbb{Z}_q^{\ell \times (n+1)}$ (as long as $m \gg \ell$). As a result, the ciphertext $\mathbf{C} = \mathbf{B} + \mu \cdot \mathbf{G}$ (for some $\mu \in \mathbb{Z}_2$) must also be computationally indistinguishable from random. This implies CPA security.

Correctness of *Enc* **and** *Dec***.** For any security parameter $n \in \mathbb{N}$, let $\mathbf{s}$ denote the secret-key vector output by running $\mathsf{KeyGen}(1^n)$. Then, for any message $\mu \in \mathbb{Z}_2$, we observe that:

$$\mathbf{C} \leftarrow \mathsf{Enc}(\mathbf{s}, \mu) = \begin{pmatrix} \mathbf{A} & || & \mathbf{A}\mathbf{s}' + \mathbf{e} \end{pmatrix} + \mu \cdot \mathbf{G}.$$

To decrypt this ciphertext $\mathbf{C}$, the decryption algorithm Dec computes the product with the secret-key vector $\mathbf{s}$, and checks whether each entry of the resulting vector is large or not. Here, we observe that:

$$\begin{aligned}
\mathbf{C} \cdot \mathbf{s} &= \Big(\Big(\mathbf{A} \quad || \quad \mathbf{A}\mathbf{s}' + \mathbf{e}\Big) + \mu \cdot \mathbf{G}\Big) \cdot \mathbf{s} \\
&= \Big(\Big(\mathbf{A} \quad || \quad \mathbf{A}\mathbf{s}' + \mathbf{e}\Big) + \mu \cdot \mathbf{G}\Big) \cdot \begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix} \\
&= \Big(\mathbf{A} \quad || \quad \mathbf{A}\mathbf{s}' + \mathbf{e}\Big) \cdot \begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix} + \mu \cdot \mathbf{G} \cdot \begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix} \\
&= -\mathbf{A}\mathbf{s}' + \mathbf{A}\mathbf{s}' + \mathbf{e} + \mu \cdot \mathbf{G} \cdot \begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix} \\
&= \mu \cdot \mathbf{G}\mathbf{s} + \mathbf{e}. \qquad (2)
\end{aligned}$$

We refer to equation (2) as the "decryption invariant." We have just shown that this decryption invariant holds after encryption with Enc. Our goal will be to make sure that this decryption invariant is preserved across homomorphic operations.

Indeed, if this decryption invariant holds (with a small enough error vector $\mathbf{e}$, and with a non-zero matrix $\mathbf{G}$), then decryption will succeed in recovering the underlying message μ: if $\mu = 0$, the product of $\mathbf{C} \cdot \mathbf{s}$ exactly recovers the error vector $\mathbf{e}$, whose entries we know will have low norm (i.e., they are close to 0). On the other hand, if $\mu = 1$, the product of $\mathbf{C} \cdot \mathbf{s}$ recovers the vector $\mathbf{G}\mathbf{s} + \mathbf{e}$. Since $\mathbf{s}$ here contains a uniformly random vector in $\mathbb{Z}_q^n$, and since the matrix $\mathbf{G}$ is non-zero, we expect at least one entry of $\mathbf{G}\mathbf{s} + \mathbf{e}$ to have large norm (i.e., to be close to $q/2$), whenever q is sufficiently large. As a result, checking whether the entries of the resulting vector are "big" or "small" lets us recover the encrypted message μ.

Finally, we will now discuss how to homomorphically evaluate bounded-depth circuits.

Homomorphic addition. To compute an addition gate on two ciphertexts, it suffices to add their corresponding matrices. That is:

$\mathsf{Eval}(``+", \mathbf{C}_1, \mathbf{C}_2) \to \mathbf{C}$.

- Output $\mathbf{C} = \mathbf{C}_1 + \mathbf{C}_2$

This works, because it (roughly) preserves the decryption invariant in equation (2). Namely, when we call Dec on the output of $\mathsf{Eval}(``+", \mathbf{C}_1, \mathbf{C}_2)$, where matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ respectively encrypt the messages μ_1 and μ_2 and satisfy the invariant in (2), we get that:

$$\begin{aligned}
\mathsf{Eval}(``+", \mathbf{C}_1, \mathbf{C}_2) \cdot \mathbf{s} &= (\mathbf{C}_1 + \mathbf{C}_2) \cdot \mathbf{s} \\
&= \mathbf{C}_1\mathbf{s} + \mathbf{C}_2\mathbf{s} \\
&= (\mu_1 \cdot \mathbf{G}\mathbf{s} + \mathbf{e}_1) + (\mu_2 \cdot \mathbf{G}\mathbf{s} + \mathbf{e}_2) \\
&= \underbrace{(\mu_1 + \mu_2)}_{\text{new message}} \cdot \mathbf{G}\mathbf{s} + \underbrace{(\mathbf{e}_1 + \mathbf{e}_2)}_{\text{new error}}.
\end{aligned}$$

It is worth noting that there is some slight error growth here: the error in the resulting ciphertext may double in magnitude (exactly as in the linearly homomorphic encryption scheme that we saw in lecture 5). However, since this error growth is relatively small, it is manageable as long as we set our LWE parameters to be large enough.

Addition as presented here is over $\mathbb{Z}_q$ (rather than mod 2). This is still sufficient to implement arbitrary Boolean circuits.

Homomorphic multiplication. To compute a multiplication gate on two ciphertexts, we will need to do something slightly more complicated than just taking the product of their corresponding matrices. Specifically, we need to carefully pick the error-correcting matrix $\mathbf{G}$ to support homomorphic multiplications. We define an associated function h that has two properties (we will be slightly sloppy with the notation here):

1. given as input an arbitrary matrix $\mathbf{C}$, the function h outputs a matrix that is $\log q \times$ wider than $\mathbf{C}$, but whose entries are all *small* (namely, they are values in $\{0, 1\}$), and
2. the function h is the "inverse" operation to the matrix $\mathbf{G}$, in the sense that for any matrix $\mathbf{C}$, it holds that

$$h(\mathbf{C}) \cdot \mathbf{G} = \mathbf{C}.$$

Then, we can homomorphically evaluate multiplications as follows:

$\mathsf{Eval}(\text{"}\times\text{"}, \mathbf{C}_1, \mathbf{C}_2) \to \mathbf{C}$.

- Output $\mathbf{C} = h(\mathbf{C}_1) \cdot \mathbf{C}_2$

Intuitively speaking, this works because:

1. Approximate eigenvectors are preserved across multiplication, as long as the matrix that we multiply by has *small* entries (to prevent large error growth).
2. Applying the h-transform to the ciphertext matrix $\mathbf{C}_1$ ensures that we are multiplying by a matrix with small entries.

More formally, when we call Dec on the output of $\mathsf{Eval}(\text{" } \times \text{ "}, \mathbf{C}_1, \mathbf{C}_2)$, where matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ respectively encrypt the messages μ_1 and

μ_2 and satisfy the invariant in (2), we get that:

$$\begin{aligned}
\mathsf{Eval}(\text{“}\times\text{”}, \mathbf{C}_1, \mathbf{C}_2)\cdot \mathbf{s} &= (h(\mathbf{C}_1)\cdot \mathbf{C}_2)\cdot \mathbf{s}\\
&= h(\mathbf{C}_1)\cdot(\mathbf{C}_2\cdot \mathbf{s})\\
&= h(\mathbf{C}_1)\cdot(\mu_2\cdot \mathbf{Gs}+\mathbf{e}_2)\\
&= \mu_2\cdot h(\mathbf{C}_1)\cdot \mathbf{Gs} + h(\mathbf{C}_1)\cdot \mathbf{e}_2\\
&= \mu_2\cdot \mathbf{C}_1\mathbf{s} + h(\mathbf{C}_1)\cdot \mathbf{e}_2\\
&= \mu_2\cdot(\mu_1\cdot \mathbf{Gs}+\mathbf{e}_1) + h(\mathbf{C}_1)\cdot \mathbf{e}_2\\
&= \underbrace{(\mu_1\cdot\mu_2)}_{\text{new message}}\cdot \mathbf{Gs} + \underbrace{(\mu_2\mathbf{e}_1 + h(\mathbf{C}_1)\cdot \mathbf{e}_2)}_{\text{new error}}
\end{aligned}$$

Here, since μ_2 is a bit and the entries of the matrix $h(\mathbf{C}_1)$ are also bits, we know that the error in the resulting ciphertext must be small in magnitude. Namely, since $\mathbf{C}_1$ has dimensions ℓ-by-$(n+1)$, we know that $h(\mathbf{C}_1)$ will have dimensions ℓ-by-$(n+1)\cdot\log q$, and so the error in the output ciphertexts will be at most a factor of $((n+1)\cdot\log q+1)$ larger than the error in either of the input ciphertexts.

What error-correcting matrix **G** *should we use?* Gentry, Sahai, and Waters gave a clean and simple way to instantiate the error-correcting matrix **G** and its associated “inverse” operation h. In particular, they take the function h to be binary decomposition. The binary decomposition of a matrix $\mathbf{C}\in\mathbb{Z}_q^{\ell\times(n+1)}$ produces a matrix $\hat{\mathbf{C}}\in\mathbb{Z}_2^{\ell\times(n+1)\cdot\log q}$—which is a slightly wider matrix that still contains the same information, but whose entries are now just bits!

Then, the corresponding error-correcting matrix **G** is the binary “recomposition” matrix. Concretely, **G** is a matrix in $\mathbb{Z}_q^{(n+1)\cdot\log q\times(n+1)}$ that looks as follows (where all of the unmarked entries are zeros):

$$\mathbf{G} = \begin{bmatrix}
1 & & & \\
2 & & & \\
\vdots & & & \\
2^{\log q-1} & & & \\
 & 1 & & \\
 & 2 & & \\
 & \vdots & & \\
 & 2^{\log q-1} & & \\
 & & \cdots & \\
 & & & 1\\
 & & & 2\\
 & & & \vdots\\
 & & & 2^{\log q-1}
\end{bmatrix}$$

What type of circuits can we evaluate? All in all, the encryption scheme we just saw has the following error growth: after each add gate, the error doubles; after each multiplication gate, the error is multiplied by $\approx n \log q$, on LWE security parameter n and LWE modulus q. So, given error that falls in the initial range $[-B, \ldots, B]$, we can still decrypt after evaluating any Boolean circuit of depth up to d, as long as $q \gg (n \log q)^d \cdot 2B$. Equivalently, for large enough q, the depth we can support is roughly $d \approx n^{0.99}$.

In the next lecture, we will see an absolutely beautiful idea to boost this encryption scheme to compute arbitrary-depth circuits!

References

[1] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[2] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[3] Craig Gentry. A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University, 2009.

[4] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92. Springer, 2013.

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

[6] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[7] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.