Lecture 15: Fully Homomorphic Encryption

Notes by Yael Kalai (inspired by notes by Alexandra Henzinger)

MIT - 6.5620
Lecture 15 (October 27, 2025)

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

Recap

Last lecture we talked about Lattice-based cryptography.

- We introduced the **Learning with Errors** (LWE) assumption, which is a family of assumptions $\mathsf{LWE}_{q,n,m,\chi}$, that assert that

$$(\mathbf{A}, \mathbf{As} + \mathbf{e}) \approx (\mathbf{A}, \mathbf{u})$$

  where $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^m$, and where $n = \lambda$ is the security parameter and the rest of the parameters are set as a function of n.

- We showed Regev's public-key encryption scheme [6], where
  - $\mathsf{Gen}(1^\lambda)$ outputs $\mathsf{pk} = (\mathbf{A}, \mathbf{As} + \mathbf{e})$ and $\mathsf{sk} = \mathbf{s}$, where the public key $\mathbf{B} := (\mathbf{A}, \mathbf{As} + \mathbf{e}) \in \mathbb{Z}_q^{m \times (n+1)}$ (obtained by adding the column $\mathbf{As} + \mathbf{e}$ to $\mathbf{A}$) and the secret key $\mathbf{s} \in \mathbb{Z}_q^n$ are generated by sampling

$$\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \chi^m.$$

  - $\mathsf{Enc}(\mathsf{pk}, b)$ outputs $\mathbf{rB} + m \cdot \lfloor q/2 \rfloor \cdot (0, \ldots, 0, 1)$, where $\mathbf{r} \leftarrow \{0,1\}^m$.
  - $\mathsf{Dec}(\mathsf{sk}, \mathbf{c})$ output 0 if and only if $|\mathbf{c} \cdot (-\mathbf{s}, 1)| \leq q/4$.

*Today: Fully Homomorphic Encryption (*FHE*)*

This week we will cover one of the most exciting and surprising advances in cryptography in the last 20 years: *fully homomorphic encryption*! This is an encryption scheme that allows us to evaluate *arbitrary* functions on the encrypted data, without ever decrypting!

We will present a formal definition, give motivation, and present a construction, along with a security proof.

History of FHE

The idea of FHE was first suggested in 1978 by Rivest, Adleman, and Dertouzos [7], though it took until 2009 for the first 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 [6]. 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.

Homomorphic encryption schemes

Last lecture we saw that Regev's encryption scheme (described above) satisfies the property that given $\mathsf{Enc}(\mathsf{pk}, b_1), \mathsf{Enc}(\mathsf{pk}, b_2)$ it is easy to compute $\mathsf{Enc}(\mathsf{pk}, b_1 \oplus b_2)$,[1] by simply outputting

$$\begin{aligned}
&\mathsf{Enc}(\mathsf{pk}, b_1) + \mathsf{Enc}(\mathsf{pk}, b_2) = \\
&\mathbf{r}_1\mathbf{B} + b_1\lfloor q/2 \rfloor \cdot (0, \ldots, 0, 1) + \mathbf{r}_2\mathbf{B} + b_2\lfloor q/2 \rfloor \cdot (0, \ldots, 0, 1) = \\
&(\mathbf{r}_1 + \mathbf{r}_2)\mathbf{B} + (b_1 + b_2) \cdot \lfloor q/2 \rfloor \cdot (0, \ldots, 0, 1) \approx \\
&(\mathbf{r}_1 + \mathbf{r}_2)\mathbf{B} + (b_1 \oplus b_2) \cdot \lfloor q/2 \rfloor \cdot (0, \ldots, 0, 1) = \\
&\mathsf{Enc}(\mathsf{pk}, b_1 \oplus b_2).
\end{aligned}$$

[1] This property is what allowed the secret-key version to be converted to a public-key encryption scheme.

We call such a scheme homomorphic w.r.t. the $\oplus$ operation.
As we will see in the HW, the Goldwasser-Micali encryption scheme [5] that we saw last week is also homomorphic w.r.t. the $\oplus$ operation. We also mention that the El-Gamal encryption scheme [2] is homomorphic w.r.t. multiplication in the group. Recall that

$$\mathsf{Enc}(g^s, m) = g^r, g^{r \cdot s} \cdot m.$$

So, multiplying the ciphertexts coordinate-wise gives an encryption of the product of the messages, where the product is group multiplication. The RSA trapdoor permutation,

$$f_N(x) = x^e \bmod N,$$

is also homomorphic w.r.t. multiplication in $\mathbb{Z}_N^*$; specifically,

$$f_N(x_1 \cdot x_2) = f_N(x_1) \cdot f_N(x_2).$$

Remark. Homomorphism can be seen both as a feature and as a bug. It may be dangerous to give the adversary the ability to generate ciphertexts related to our honest ciphertexts, as the adversary could

use this ability for malice. This is what motivates the definition of CCA-secure encryption. However, as we will see today, a homomorphic encryption scheme is also extremely useful.

This week, we will see how to construct a *fully homomorphic* encryption scheme, which is homomorphic w.r.t. both $\oplus$ and multiplication (modulo 2).

FHE *Definition and Syntax*

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

- The message space is bits (i.e., $\mathcal{M} = \{0,1\}$).
- The model of computation is Boolean circuits (i.e., circuits composed of additions and multiplications gates mod 2, or equivalently, XOR and AND gates).

We note that additions and multiplications mod 2 is complete in the sense that every efficiently computable function can be represented via a circuit with polynomially many addition and multiplication gates mod 2.

In what follows, we define an FHE scheme w.r.t. a circuit class $\mathcal{C}$, which contains all the circuits that the FHE scheme can homomorphically evaluate.

Definition 1. A homomorphic encryption scheme w.r.t. a class of Boolean circuits $\mathcal{C} = \{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$, where each $C \in \mathcal{C}_\ell$ is a circuit $C : \{0,1\}^\ell \to \{0,1\}$, consists of four efficient algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with the following syntax:

- $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has the same syntax as a public-key encryption scheme. Namely:
  - $\mathsf{Gen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$**:** Gen takes as input the security parameter (in unary) and outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$.
  - $\mathsf{Enc}(\mathsf{pk}, \mu) \to \mathsf{ct}$**:** Enc takes as input a public-key pk and a message $\mu \in \{0,1\}$,[2] and outputs a ciphertext ct.
  - $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mu$**:** Dec takes as input a secret-key sk and a ciphertext ct and outputs a message $\mu \in \{0,1\}$.
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \to \tilde{\mathsf{ct}}$**:** Eval takes as input a public-key pk, a Boolean circuit $C \in \mathcal{C}_\ell$, where $C : \{0,1\}^\ell \to \{0,1\}$, and ℓ ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$ and outputs an evaluated ciphertext $\tilde{\mathsf{ct}}$.

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

[2] We denote the message by μ since m will be used to denote the number of rows in the matrix A corresponding to the LWE assumption.

1. **Correctness:** For every polynomial $\ell : \mathbb{N} \to \mathbb{N}$ there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$ the following holds for $\ell = \ell(\lambda)$: For any $C \in \mathcal{C}_\ell$ and any inputs $\mu_1, \ldots, \mu_\ell \in \{0,1\}$,

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \tilde{\mathsf{ct}}) = C(\mu_1, \ldots, \mu_\ell) : \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, \mu_i) \text{ for all } i \in [\ell] \\ \tilde{\mathsf{ct}} \leftarrow \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)\end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

2. **Security:** $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure encryption scheme w.r.t. the message space $\mathcal{M} = \{0,1\}$.

3. **Compactness:** For every polynomial $\ell = \ell(\lambda)$ there exists a negligible function negl such that for every security parameter λ, every $C \in \mathcal{C}_\ell$, and every $\mu_1, \ldots, \mu_\ell \in \{0,1\}$,

$$\Pr[|\tilde{\mathsf{ct}}| = |\mathsf{ct}_1| = \ldots = |\mathsf{ct}_\ell|] = 1 - \mathrm{negl}(\lambda)$$

where

$$\tilde{\mathsf{ct}} = \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$$

and where the probability is over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, and over $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, \mu_i)$ for every $i \in [\ell]$.

Remark. Without the compactness requirement constructing a fully homomorphic encryption can be done trivially, as follows:

$$\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) := (C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell).$$

where decryption of a ciphertext of the form $(C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ is done by decrypting each ct_i and applying C to all decrypted messages.

Therefore, some notion of compactness is necessary for this definition to be interesting. One could imagine various stronger notions of compactness. For example, one could demand that the evaluated ciphertext $\tilde{\mathsf{ct}}$ "has the same form as a "fresh" ciphertext.

Remark. Note that while we defined the Eval function to take as input circuit C that only output a single bit, we can easily extend Eval to take as input a circuit C that output many bits $C : \{0,1\}^\ell \to \{0,1\}^k$, as follows: Let $C = (C_1, \ldots, C_k)$ where $C_i : \{0,1\}^\ell \to \{0,1\}$ outputs the i'th bit of C, and define

$$\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) := (\mathsf{Eval}(\mathsf{pk}, C_1, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell), \ldots, \mathsf{Eval}(\mathsf{pk}, C_k, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$$

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 $\mathbb{Z}_2$ (i.e., where addition and multiplication is mod 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. So far in the course, we have seen encryption schemes that let us compute either additions or multiplications on ciphertexts—but *not* both!

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

- **Arbitrary circuits of a bounded depth**: We will first 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 (in the underlying LWE assumption). Once this error grows too large, the ciphertexts will no longer be decryptable. So, to avoid this, the number of addition and multiplication gates that we can evaluate will be bounded.

- **Arbitrary circuits**: We will then see how to boost a levelled FHE scheme to one that can support arbitrary circuits, under some additional assumptions. This is done using a beatiful boosting technique due to Gentry [3]. The key idea is a procedure to "refresh" ciphertexts to reset their error to a small level. This beautiful technique will let us homomorphically evaluate *any* Boolean circuit under encryption, with an overhead that is only polynomial in the security parameter λ.

Motivation: Delegation

Suppose we wish to put all our secret data on the cloud, but we are concerned about the privacy of our data. By now we are experts on encryption, so we can simply encrypt our data and upload it to the cloud encrypted. But now suppose we want the cloud to do computations on our private data. For example, suppose I want to fetch a certain email of picture, and I want the cloud to send me only that email or picture.

If we use a CCA-secure encryption then the problem is that this is "too secure," as it will not allow the cloud to perform this computation. A fully-homomorphic encryption scheme is precisely what we

need!

Constructing "Levelled" FHE: Background and Intuition

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 $n \in \mathbb{N}$, let $\mathbf{C}_1, \mathbf{C}_2 \in \mathbb{Z}_q^{n \times n}$ denote two matrices. Let vector $\mathbf{v} \in \mathbb{Z}_q^n$ 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:

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

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 in $\mathbb{Z}_q^n$. Then, roughly speaking, our ciphertexts will be matrices in $\mathbb{Z}_q^{n \times n}$ whose *eigenvector* will be the secret-key vector and whose associated *eigenvalue* will be the message being encrypted. With this setup:

- 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 $\mathbf{C} \in Z_q^{n \times n}$ such that the following equation holds:

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

where $\mu \in \{0,1\}$ is the message being encrypted and $\mathbf{e} \in \mathbb{Z}_q^n$ 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.

The construction is based on the LWE assumption, and is extremely similar to Regev's encryption. Recall that a Regev encryption of a message $\mu \in \{0,1\}$ is one LWE vector $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e)$ masked with $\mu \lfloor q/2 \rfloor (0, \ldots, 0, 1)$, where $\lfloor q/2 \rfloor (0, \ldots, 0, 1)$ can be thought of as a fixed public "gadget" vector.

A GSW encryption of a message $\mu \in \{0,1\}$ consists of many LWE vectors $(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$ masked with $\mu \mathbf{G}$ where $\mathbf{G}$ is a cleverly chosen "gadget" matrix.

The GSW construction is associated with parameters q, n, m, χ, where n is the security parameter, $q, m \in \mathbb{N}$ are functions of n, and χ is an error distribution which is assumed to output elements in $\mathbb{Z}_q$ of bounded size (say, elements in $[-\sigma, \sigma]$).

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

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$:
  - Sample a random vector $\mathbf{s} \leftarrow \mathbb{Z}_q^n$.
  - Sample a random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$
  - Sample a random error vector $\mathbf{e} \leftarrow \chi^m$.
  - Set $\mathbf{B} = (\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e}) \in \mathbb{Z}_1^{m \times (n+1)}$.
  - Set $\mathbf{t} = \begin{pmatrix} -\mathbf{s} \\ 1 \end{pmatrix} \in \mathbb{Z}_q^{(n+1)}$ as the secret key.
  - Output $(\mathsf{pk}, \mathsf{sk}) = (\mathbf{B}, \mathbf{t})$.

  Note that $\mathbf{B}\mathbf{t} = \mathbf{e} \approx \mathbf{0}$, thus $\mathbf{t}$ is an approximate eigenvector of $\mathbf{B}$ with the eigenvector 0.
- $\mathsf{Enc}(\mathbf{B}, \mu) \to \mathbf{C} \in \mathbb{Z}_q^{N \times (n+1)}$.
  - Sample a random matrix $\mathbf{R} \leftarrow \{0,1\}^{N \times m}$.

  - Output $\mathbf{C} = \mathbf{RB} + \mu\mathbf{G}$ as the ciphertext, where $\mathbf{G} \in \mathbb{Z}_q^{N \times (n+1)}$ is some fixed "gadget" matrix, which is a public matrix that we will define later.

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

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

Properties of this construction. Before we describe how to perform homomorphic additions and multiplications, we first argue that this scheme is CPA-secure, and is correct in the sense that calling Dec on a *fresh* ciphertext output by Enc produces the correct result.

Correctness: To argue that correctness holds note that

$$\mathbf{C} \cdot \mathbf{t} = (\mathbf{RB} + \mu\mathbf{G}) \cdot \mathbf{t} = \mathbf{Re} + \mu\mathbf{Gt} = \mu\mathbf{Gt} + \mathbf{e}' \tag{2}$$

where $\mathbf{e}'$ is an error vector. We refer to equation (2) as the *"decryption invariant."* We will make sure that this decryption invariant is preserved across homomorphic operations.

We have just shown that this decryption invariant holds after encryption with Enc, where the error vector $\mathbf{e}' = \mathbf{Re}$, and thus has all its coordinates in $[-m\sigma, m\sigma]$.

Note that if this decryption invariant holds, with a small enough error vector $\mathbf{e}'$, then decryption will succeed in recovering the underlying message μ with probability $1 - \text{negl}$, if (for example) $\mathbf{G}$ has full rank (which it does).

- If $\mu = 0$, the product of $\mathbf{C} \cdot \mathbf{t}$ exactly recovers the error vector $\mathbf{e}'$, and we set $m\sigma < q/4$, so that the ciphertext will be correctly decrypted to 0.
- If $\mu = 1$, the product of $\mathbf{C} \cdot \mathbf{t}$ recovers the vector $\mathbf{Gt} + \mathbf{e}'$. Since $\mathbf{t}$ here contains a uniformly random vector in $\mathbb{Z}_q^n$, and since we choose the matrix $\mathbf{G}$ to be full rank, with overwhelming probability at least one entry of $\mathbf{Gt} + \mathbf{e}'$ has large norm (i.e., in $[\frac{q}{4}, \frac{3q}{4}]$).

Therefore, checking whether the entries of the resulting vector are "big" or "small" lets us recover the encrypted message μ.

CPA-*security:* The security proof is identical to the security proof of Regev's security proof. Specifically, suppose that there exists a poly-size $\mathcal{A}$ that wins in the CPA security proof with probability $1/2 + \epsilon$, where $\epsilon = \epsilon(\lambda)$ is a non-negligible function.

By the LWE assumption, even if we replace the public $\mathbf{B}$ with a uniformly random $\mathbf{B} \leftarrow \mathbb{Z}_q^{m\times(n+1)}$, it should still be the case that $\mathcal{A}(\mathbf{B})$ wins in the CPA game with probability $1/2 + \epsilon - \text{negl}$.

We next claim that if $\mathbf{B}$ is uniformly distributed in $\mathbb{Z}_q^{m\times(n+1)}$ then even an all-powerful $\mathcal{A}$ has only a negligible advantage in winning in the CPA game. Specifically, we argue that if $\mathbf{B} \leftarrow \mathbb{Z}_q^{m\times(n+1)}$ is uniformly random then

$$(\mathbf{B}, \mathbf{RB}) \equiv (\mathbf{B}, \mathbf{U})$$

This follows immediately from the Leftover Hash Lemma, as long as $m >> n \log q$.

Lemma 2 (Leftover Hash Lemma (special case):)**.** *Fix any $\epsilon \in (0,1)$, then for $\mathbf{A} \leftarrow \mathbb{Z}_q^{m\times n}$, $\mathbf{r} \leftarrow \{0,1\}^m$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$, it holds that*

$$(\mathbf{A}, \mathbf{r}\cdot\mathbf{A}) \overset{\epsilon}{\equiv} (\mathbf{A}, \mathbf{u})$$

if $m > n\log q + 2\cdot\log(1/\epsilon)$.

Homomorphic Operations

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

$\mathsf{Eval}(\text{“}+\text{”}, \mathbf{C}_1, \mathbf{C}_2) \to \mathbf{C}$.

- 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 $\mathsf{Eval}(\text{“}+\text{”}, \mathbf{C}_1, \mathbf{C}_2)$, 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:

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

Remark. Note that addition as presented here is mod q (rather than mod 2). However, using addition mod q and multiplication mod 2, one can easily compute addition mod 2, by

$$(b_1 + b_2) \bmod 2 = ((b_1 + b_2) \bmod q) - ((b_1 \cdot b_2) \bmod 2) \bmod q.$$

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 Regev's ciphertext). However, since this error growth is relatively small, it is manageable as long as we set our LWE parameters correctly; in particular, set q to be significantly larger than n and m. It will be useful to think of $q = n^{\omega(1)}$, i.e., being super-polynomial in n.

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 (their product is not even syntactically defined!). Specifically, we will need to carefully pick the error-correcting matrix $\mathbf{G}$ to support homomorphic multiplications.

Defining the matrix $\mathbf{G}$. To define $\mathbf{G}$, we first define a function h which is the "bit-decomposition" function, that converts any element in $x \in \mathbb{Z}_q$ into a vector in $\mathbf{b} \in \{0,1\}^{\lceil \log q \rceil}$ which is the bit decomposition of x; i.e.,

$$x = \sum_{i=0}^{\lceil \log q \rceil - 1} b_i 2^i.$$

We also define h as a function on matrices (and in particular on ciphertexts), in which case it takes any input $\mathbf{C} \in \mathbb{Z}_q^{m \times (n+1)}$ and outputs a matrix $h(\mathbf{C}) \in \{0,1\}^{m \times (n+1)\lceil \log q \rceil}$, which is $\lceil \log q \rceil$ wider, and each entry $x \in \mathbb{Z}_q$ in the matrix $\mathbf{C}$ is replaced with its binary decomposition $h(x)$.

The matrix $\mathbf{G} \in \mathbb{Z}_q^{N \times (n+1)}$, where $N = (n+1)\lceil \log q \rceil$, is the "bit recomposition" matrix, which does the "inverse" operation to h, so that for any matrix $\mathbf{C}$,

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

Concretely, $\mathbf{G}$ is a matrix in $\mathbb{Z}_q^{N \times (n+1)}$ that looks as follows (where all of the unmarked entries are zeros):

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

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:

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

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 N-by-$(n+1)$, we know that $h(\mathbf{C}_1)$ will have dimensions N-by-N (since recall that $N = (n+1) \cdot \lceil \log q \rceil$), and so the error in the output ciphertexts will be at most a factor of N larger than the error in either of the input ciphertexts.

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 $[-\sigma, \sigma]$, we can still decrypt after evaluating any Boolean circuit of depth up to d, as long as $q \gg (n \log q)^d \cdot 2\sigma$. 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. Efficient fully homomorphic encryption from (standard) lwe. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 97–106. IEEE, 2011.

[2] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology — CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, CA, USA, 1984. Springer.

[3] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM, 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*, pages 75–92. Springer, 2013.

[5] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377. ACM, 1982.

[6] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[7] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, editors, *Foundations of Secure Computation*, pages 165–179. Academic Press.