Lecture 16: Fully Homomorphic Encryption (Cont.)

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

MIT - 6.5620
Lecture 16 (October 29, 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 defined the notion of a homomorphic encryption scheme w.r.t. a circuit class $\mathcal{C} = \{_\ell\}_{\ell \in \mathbb{N}}$.
- We presented the GSW construction [2].

Recap of the GSW Construction

The GSW construction is extremely similar to the Regev construction. Similarly to the Regev construction, it 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 $\mathbf{B} = (\mathbf{A}, \mathbf{As} + \mathbf{e}) \in \mathbb{Z}_q^{m \times (n+1)}$.
  - 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 $\mathbf{Bt} = \mathbf{e} \approx \mathbf{0}$, 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."

Today

- Show how to homomorphically compute on GSW ciphertexts.
- Argue that this scheme only supports bounded-depth circuits, also called a *levelled* FHE scheme.

  Roughly, the reason why the GSW scheme will only supports 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.
- Finally, we will see how to bootstrap a levelled FHE scheme to one that is fully-homomorphic, i.e., can support arbitrary circuits, under some additional assumptions. This is done using a beatiful boosting technique due to Gentry [1]. 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.

Properties of the GSW 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

$$\mathbf{C} \cdot \mathbf{t} = (\mathbf{RB} + \mu\mathbf{G}) \cdot \mathbf{t} = \mathbf{Re} + \mu\mathbf{Gt} = \mu\mathbf{Gt} + \mathbf{e}' \tag{1}$$

where $\mathbf{e}'$ is an error vector. We refer to equation (1) 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 1 (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*

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

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 (1). 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 (1), we get that:

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

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 (1), 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 $d \approx n^{0.99}$.

Bootstrapping GSW to support arbitrary-depth computations

In his original paper, constructing FHE, Gentry [1] introduced a brilliant technique to *refresh* ciphertexts, which lets us go from a ciphertext encrypting a message μ with a lot of noise (as a result of performing homomorphic computations) to a separate ciphertext encrypting μ with only a little noise (namely, the baseline level of noise needed for security). This technique is referred to as “bootstrapping.”

Bootstrapping intuition. At a high level, bootstrapping is based on the following intuition: if the server was given the secret-key $\mathbf{t}$, then it could lower the noise level in a ciphertext $\mathbf{C}$ by computing $\mathsf{Dec}(\mathbf{t}, \mathbf{C})$ to recover the underlying message μ, and then computing a fresh ciphertext $\mathbf{C}'$ that encrypts μ with the baseline level of noise by running $\mathsf{Enc}(\mathbf{B}, \mu)$. Then, the server could continue performing homomorphic operations on the ciphertext $\mathbf{C}'$. Clearly though, this would

be insecure: we cannot let the server learn the secret-key vector $\mathbf{t}$, otherwise it could decrypt all of the ciphertexts (exactly as we've suggested) and this would completely break security.

However, we can play a trick that is very similar to this approach: namely, we can have the user send the server the *encryption* of its secret-key vector $\mathbf{t}$. Then, the server can *homomorphically*—i.e., under encryption—evaluate the Dec algorithm on a ciphertext encrypting μ, as long as Dec has sufficiently low circuit complexity. The output of this computation will be an encryption of the same message μ, but with *low* error. We will now discuss this bootstrapping technique, as well as the assumption that is needed to make it work (circular security), in more detail.

Bootstrapping construction. To perform bootstrapping, we will think of the decryption algorithm, Dec, as a Boolean circuit, $\mathcal{C}_{\mathsf{Dec}}$. This circuit takes as input two parameters: the secret-key vector $\mathbf{t} \in \mathbb{Z}_q^{n+1}$ and a ciphertext matrix $\mathbf{C} \in \mathbb{Z}_q^{N\times(n+1)}$. Then, the circuit outputs the bit $\mu \in \{0,1\}$ encrypted by the ciphertext $\mathbf{C}$.

Here, we are using the fact that we can write any polynomial-time computation, such as Dec, as a Boolean circuit that is also requires polynomial time to evaluate.

Now, in the setting we are working in, the server knows the full ciphertext $\mathbf{C}$ that it wants to bootstrap. So, we can think of this ciphertext $\mathbf{C}$ as *hard-coded* into the circuit $\mathcal{C}_{\mathsf{Dec}}$ that we are trying to evaluate. That is, we will denote by $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$ the circuit $\mathcal{C}_{\mathsf{Dec}}$ in which the input ciphertext $\mathbf{C}$ has been fixed. In other words, the modified circuit $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$ now takes as input *only* the secret-key vector $\mathbf{t} \in \mathbb{Z}_q^{n+1}$, and it spits out the message bit $\mu \in \{0,1\}$ encrypted by the fixed ciphertext $\mathbf{C}$.

This new circuit $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$ is exactly what we will homomorphically evaluate. Namely, the user will include as part of its publice-key a public evaluation key, denoted by ek, which will include the *encryption* of each bit of its secret-key vector, $\mathbf{t} \in \mathbb{Z}_q^{n+1}$. We can write this evaluation key as:

$$\mathsf{ek} = (\mathsf{ct}_{\mathbf{t}_1}, \ldots, \mathsf{ct}_{\mathbf{t}_n}) = (\mathsf{Enc}(\mathbf{t}, \mathbf{t}_1), \ldots, \mathsf{Enc}(\mathbf{t}, \mathbf{t}_n)) .$$

We are being slightly sloppy with notation here and directly encrypting $\mathbb{Z}_q$ values. In reality, the evaluation key here consists of the encryption of each bit in the bit-decomposition of each entry of $\mathbf{t}$.

Then, given a ciphertext $\mathbf{C}$ that has the maximal amount of allowable noise in it (such that it is still decryptable), the server will:

1. build the Boolean circuit $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$, which correspond to the decryption circuit with the ciphertext $\mathbf{C}$ hard-coded in it, and
2. homomorphically evaluate this decryption circuit to get the resulting ciphertext $\mathbf{C}'$:

$$\mathbf{C}' \leftarrow \mathsf{Eval}\left(\mathcal{C}_{\mathsf{Dec},\mathbf{C}}, \mathsf{ct}_{\mathbf{t}_1}, \ldots, \mathsf{ct}_{\mathbf{t}_n}\right) .$$

Here, we observe two crucial properties:

- We designed the circuit $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$ to output the message bit $\mu \in \{0,1\}$ encrypted in the ciphertext $\mathbf{C}$. So, the new ciphertext $\mathbf{C}'$, which is the result of homomorphically evaluating $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$, will then be an encryption of μ.
- The amount of noise contained in the new ciphertext $\mathbf{C}'$ depends *only* on
  1. the noise contained in the encryptions $\mathsf{ct}_{\mathbf{t}_1}, \ldots, \mathsf{ct}_{\mathbf{t}_n}$ given as part of the evaluation key ek, and
  2. the depth of the decryption circuit $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$.

  So, since the encryptions given in the evaluation key ek are fresh (in that they have the minimum amount of required noise), the output ciphertext $\mathbf{C}'$ can also have low noise!

As a result, the ciphertext $\mathbf{C}'$ is an encryption of the same message as the ciphertext $\mathbf{C}$, but $\mathbf{C}'$ is guaranteed to have low noise! This is exactly what we set out to construct. However, there are two important caveats that we need to verify to make this technique work:

1. Decryption circuit complexity. For bootstrapping to be useful, the underlying levelled FHE scheme must be powerful enough to homomorphically evaluate (a) the decryption circuit $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$ and (b) at least one extra addition or multiplication gate. In other words, we need the decryption circuit to be "shallow" enough to fit into the function class supported by our levelled FHE scheme. Concretely, if the decryption circuit $\mathcal{C}_{\mathsf{Dec},\mathbf{C}}$ has depth d, we need the levelled FHE scheme to support computations of depth at least $d+1$.

When this is the case, we can homomorphically evaluate any arbitrary-depth boolean circuit as follows:

1. *Homomorphic addition:* we replace each "ADD" gate by a homomorphic addition (i.e., $\mathbf{C} \leftarrow \mathsf{Eval}(\text{“}+\text{”}, \mathbf{C}_1, \mathbf{C}_2)$ in the notation of last lecture), followed by a homomorphic decryption (i.e., $\mathbf{C}' \leftarrow \mathsf{Eval}(\mathcal{C}_{\mathsf{Dec},\mathbf{C}}, \mathsf{ek})$).

   At the end of this procedure, we have a ciphertext $\mathbf{C}'$ with noise in the range $[-\ell^d \cdot B_0, \cdots, \ell^d \cdot B_0]$, where d is the depth of the decryption circuit.

2. *Homomorphic multiplication:* we replace each "MUL" gate by a homomorphic multiplication (i.e., $\mathbf{C} \leftarrow \mathsf{Eval}(\text{“}\times\text{”}, \mathbf{C}_1, \mathbf{C}_2)$), followed by a homomorphic decryption (i.e., $\mathbf{C}' \leftarrow \mathsf{Eval}(\mathcal{C}_{\mathsf{Dec},\mathbf{C}}, \mathsf{ek})$).

   At the end of this procedure, we again have a ciphertext $\mathbf{C}'$ with noise in the range $[-N^d \cdot \sigma, N^d \cdot \sigma]$, where d is the depth of the decryption circuit.

By alternating computing on and refreshing the ciphertexts in this way, the noise in our ciphertexts will never grow too large. No matter the degree of our computation, we will always be able to decrypt.

In the case of GSW, the decryption circuit has depth $O(\log n)$. (This is depth stems from the comparison operations required to check whether each value is "big" or "small".) As we saw, the levelled version of GSW is powerful enough to support computations up to depth $n^{0.99}$. So we can indeed bootstrap!

2. Circular security. To preserve security, the server cannot learn any information about the secret key. Intuitively, the way we are achieving this is by *encrypting* the secret-key vector, under itself. Proving that is secure requires the additional assumption that GSW encryption is *circular secure*—that is, that publishing the ciphertexts that encrypt each bit of the secret key under itself, i.e.,

$$\mathsf{ct}_{\mathbf{t}_i} = \mathsf{Enc}(\mathbf{t}, \mathbf{t}_i) \text{ for } i \in [n],$$

hides the secret key.

In general, we do not know how to show that GSW is circular secure from just the LWE assumption. Building FHE without the circular security assumption is still an open question and an active area of research!

In fact, there exist semantically-secure encryption schemes that are provably *not* circular secure.

Applications of FHE

Now that we have constructed FHE, we will discuss three particular applications that it lets us construct:

There are many, many more applications of (full) FHE. Coming up with a new application might be a fun avenue for a class project!

1. *Private delegation.* In private delegation, a user wants to outsource the computation of some function to a powerful but untrusted server, without revealing its input to the computation.

   Some examples of this are: a user may want to query a LLM on a sensitive prompt, without revealing her prompt. Or, a user may want to outsource the storage of her emails to a cloud server and to keep the contents of her emails hidden, while retaining the ability to search over them.

2. *Secure collaboration.* In secure collaboration, multiple users want to jointly evaluate a function on their hidden inputs, while revealing nothing but the output of the function.

   Some examples of this are: hospitals may want to collaborate to train machine learning models, while keeping sensitive patient data hidden. Or, banks may want to run an auction without revealing (or learning) each bidder's bid and each seller's price.

3. *Private database lookups.* Private database lookups are a generalization of private information retrieval (which we covered in lectures 5 and 6). In this setting, a user wants to make arbitrary queries to a remote database, while hiding her queries. With FHE, the server hosting the database can answer these queries under encryption.

   These private database queries could take many forms: for example, a user may want to make general SQL queries. Or, a user may want to make a private query to a web search engine (e.g., Google), without revealing her query string.

Open questions in FHE

The GSW scheme that we saw in these last two lectures is an incredibly powerful tool that can unlock an array of cryptographic applications. However, building and improving on known FHE constructions is an active and exciting area of research in which many interesting questions remain unanswered. Two such open questions are:

1. *Can we build* FHE *from assumptions other than lattices?*

   In cryptography, it is always good to construct any given primitive from a variety of assumptions—this gives us confidence that, even if any one assumption turns out to be broken, our primitive still exists. In the case of FHE, we (roughly) only know how to build it from lattice-based assumptions (e.g., LWE, ring-LWE). Building FHE from a number-theoretic assumption—for example, the hardness of factoring (e.g., RSA) or the hardness of discrete log (e.g., DDH)—would be a big research result!

2. *Can we make* FHE *concretely efficient and practical?*

   While the construction of FHE that we covered today is possible in theory, it is still very far from concretely efficient in practice. In large part, this is due to the computational costs of

   - representing computations as Boolean circuits, and
   - incurring $O(\text{poly}(n))$ (or even just $O(\text{polylog}(n))$) overhead per gate.

   The state-of-the-art in research today is that evaluating functions with low multiplicative degree (e.g., degree 2 or 3) and high additive degree can be concretely efficient. However, once we try to evaluate circuits with high multiplicative degree, the LWE parameters that we must use become large and computing on ciphertexts (and, in particular, bootstrapping them) becomes very expensive.

   While FHE isn't used much in practice yet, building truly practical FHE would have an enormous impact on the world!

References

[1] 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.

[2] 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.