Fully Homomorphic Encryption

We will let λ be a security parameter throughout this lecture. That is, all honest parties run in time a fixed polynomial in λ, and adversaries can run in time an arbitrary polynomial in λ.

In a fully homomorphic (private or public-key) encryption, anyone can take a set of encrypted messages $\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_k)$ and produce an encryption of *any polynomial-time computable function* of them, that is, $\mathsf{Enc}(f(x_1, \ldots, x_k))$ where f is any function with a $\mathsf{poly}(\lambda)$-size circuit. By a result of Rothblum, any private-key (even additively) homomorphic encryption scheme can be converted to a public-key homomorphic scheme, so we will focus our attention on private-key schemes henceforth.

The formal definition of the functionality of fully homomorphic encryption follows.

- $\mathsf{KeyGen}(1^\lambda)$: produces a secret key sk, possibly together with a public evaluation key ek.
- $\mathsf{Enc}(\mathsf{sk}, \mu)$, where $\mu \in \{0,1\}$: produces a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}, c)$: outputs μ.
  // So far, everything is exactly as in a regular secret-key encryption scheme.
- $\mathsf{Eval}(\mathsf{ek}, f, c_1, \ldots, c_k)$ takes as input a $\mathsf{poly}(\lambda)$-size circuit that computes a function $f : \{0,1\}^k \to \{0,1\}$, as well as k ciphertexts $c_1, \ldots, c_k$, and outputs a ciphertext c_f.

Correctness says that

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{ek}, f, \mathsf{Enc}(\mathsf{sk}, \mu_1), \ldots, \mathsf{Enc}(\mathsf{sk}, \mu_k))) = f(\mu_1, \ldots, \mu_k)$$

for all $f, \mu_1, \ldots, \mu_k$ with probability 1 over the sk, ek and the randomness of all the algorithms.

Security is just semantic (IND-CPA) security, that is the encryptions of any two sequences of messages $(\mu_i)_{i \in \mathsf{poly}(\lambda)}$ and $(\mu'_i)_{i \in \mathsf{poly}(\lambda)}$ are computationally indistinguishable. (the fact that the encryption scheme is homomorphic is a functionality requirement, and does not change the notion of security.)

$$\left(\mathsf{Enc}(sk, \mu_1), \ldots, \mathsf{Enc}(\mathsf{sk}, \mu_{p(\lambda)})\right) \approx_c \left(\mathsf{Enc}(sk, \mu'_1), \ldots, \mathsf{Enc}(\mathsf{sk}, \mu''_{p(\lambda)})\right)$$

A final and important property is compactness, that is, $|c_f| = \mathsf{poly}(\lambda)$, independent of the circuit size of f. (Weaker compactness conditions are possible, and indeed, we will see one as we go along.)

1 The Gadget Matrix

First define the gadget vector (or the bit-decomposition gadget vector) $\mathbf{g}$:

$$\mathbf{g}^T := \begin{bmatrix} 1 & 2 & 4 & \ldots & 2^{\lceil \log q \rceil - 1} \end{bmatrix} \in \mathbb{Z}_q^{1 \times \lceil \log q \rceil}$$

Note that given any $v \in \mathbb{Z}_q$, it is easy to produce a vector $\mathbf{x} \in \mathbb{Z}_q^{\lceil \log q \rceil \times 1}$ such that (a) the entries of $\mathbf{x}$ are small, in fact they are binary; and (b) $\mathbf{g}^T \mathbf{x} = v \bmod q$. $\mathbf{x}$ simply contains the bits of v from the least to the most significant bit.

Extending this to n dimensions, we define the *gadget matrix* $\mathbf{G}$:

$$\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^T \in \mathbb{Z}_q^{n \times n\lceil \log q \rceil}$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix. In other words, $\mathbf{G}$ is the block diagonal $n \times (n\lceil \log q \rceil)$ matrix with $\mathbf{g}^T$ in each of its diagonal blocks.

Similar to the above, it is easy, given a vector $\mathbf{v} \in \mathbb{Z}_q^n$, to produce a bit vector $\mathbf{x}$ such that $\mathbf{G}\mathbf{x} = \mathbf{v} \bmod q$. We will let $\mathbf{x}$ be denoted as $\mathbf{G}^{-}(\mathbf{v})$.

2 The GSW Scheme

The first candidate FHE scheme was due to Gentry in 2009. The first LWE-based FHE Scheme was due to Brakerski and Vaikuntanathan in 2011. We will present a different FHE scheme due to Gentry, Sahai and Waters (2013) which is both simple and quite flexible.

- KeyGen: the secret key is a vector $\mathbf{s} = \begin{bmatrix} \mathbf{s}' \\ -1 \end{bmatrix}$ where $\mathbf{s}' \in \mathbb{Z}_q^n$.
- Enc: output $\mathbf{A} + \mu\mathbf{G}$ where $\mathbf{A}$ is a random matrix such that

  $$\mathbf{s}^T\mathbf{A} \approx \mathbf{0} \pmod{q}$$

  and $\mathbf{G}$ is a fixed matrix chosen cleverly; we will see how in a bit.

  Here is one way to do it: choose a random matrix $\mathbf{A}'$ and let

  $$\mathbf{A} := \begin{bmatrix} \mathbf{A}' \\ (\mathbf{s}')^T\mathbf{A}' + \mathbf{e}' \end{bmatrix}$$

- Dec: Let $\mathbf{C}$ be a ciphertext and $\mathbf{c}$ be its last column. Output

  $$\mathsf{Round}_{q/2}(\mathbf{s}^T\mathbf{c})$$

  where $\mathsf{Round}_{q/2}$ outputs 1 on input x where $|x - q/2| \le q/4$ and 0 otherwise.
- Eval: we will show how to ADD (over the integers) and MULT (mod 2) the encrypted bits which will suffice to compute all Boolean functions.

3 How to Add and Multiply (without errors)

Let's start with a variant of the scheme where the ciphertext is

$$\mathbf{C} = \mathbf{A} + \mu\mathbf{I}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{s}^T\mathbf{A} = \mathbf{0}$ (as opposed to $\mathbf{s}^T\mathbf{A} \approx \mathbf{0}$.)

Now,

$$\mathbf{s}^T\mathbf{C} = \mu\mathbf{s}^T$$

- ADD$(\mathbf{C}_1, \mathbf{C}_2)$ outputs $\mathbf{C}_1 + \mathbf{C}_2$. This is an encryption of $\mu_1 + \mu_2$ since

  $$\mathbf{s}^T(\mathbf{C}_1 + \mathbf{C}_2) = (\mu_1 + \mu_2)\mathbf{s}^T$$

  Eigenvalues add.
- MULT$(\mathbf{C}_1, \mathbf{C}_2)$ outputs $\mathbf{C}_1\mathbf{C}_2$. This is an encryption of $\mu_1\mu_2$ since

  $$\mathbf{s}^T(\mathbf{C}_1\mathbf{C}_2) = \mu_1\mathbf{s}^T\mathbf{C}_2 = \mu_1\mu_2\mathbf{s}^T$$

  Eigenvalues multiply.

We need one ingredient now to turn this into a *real* FHE scheme, namely the gadget matrix.

4 How to Add and Multiply (with errors)

We have to be careful to multiply approximate equations by small numbers. Once we make adjustments to this effect, we get the GSW scheme. The ciphertext is

$$\mathbf{C} = \mathbf{A} + \mu \mathbf{G}$$

where $\mathbf{s}^T \mathbf{A} \approx \mathbf{0}$. Think of $\mathbf{G}$ as an error correcting artifact for the message μ.

Now,

$$\mathbf{s}^T \mathbf{C} \approx \mu \mathbf{s}^T \mathbf{G}$$

which is the approximate eigenvalue equation.

- ADD$(\mathbf{C}_1, \mathbf{C}_2)$ outputs $\mathbf{C}_1 + \mathbf{C}_2$. This is an encryption of $\mu_1 + \mu_2$ since

  $$\mathbf{s}^T(\mathbf{C}_1 + \mathbf{C}_2) \approx (\mu_1 + \mu_2)\mathbf{s}^T \mathbf{G}$$

  Approximate eigenvalues add (if you don't do it too many times.)

- MULT$(\mathbf{C}_1, \mathbf{C}_2)$ outputs $\mathbf{C}_1 \mathbf{G}^-(\mathbf{C}_2)$. This is an encryption of $\mu_1 \mu_2$ since

  $$\mathbf{s}^T(\mathbf{C}_1 \mathbf{G}^-(\mathbf{C}_2)) = (\mathbf{s}^T \mathbf{C}_1)\mathbf{G}^-(\mathbf{C}_2) \approx (\mu_1 \mathbf{s}^T \mathbf{G})\mathbf{G}^-(\mathbf{C}_2) = \mu_1(\mathbf{s}^T \mathbf{C}_2) \approx \mu_1 \mu_2 \mathbf{s}^T \mathbf{G}$$

  where the first $\approx$ is because $\mathbf{G}^-(\mathbf{C}_2)$ is small and the second $\approx$ because μ_1 is small.

  Approximate eigenvalues multiply if you only multiply by small numbers/matrices.

Put together, it is not hard to check that you can evaluate depth-d circuits of NAND gates with error growth $m^{O(d)}$. (You can do better for log-depth circuits by converting them to branching programs; see Brakerski-Vaikuntanathan 2014.)

5 Bootstrapping to an FHE

With this, we get a *leveled FHE* scheme. That is, we can set parameters (in particular $q = m^{\Omega(d)}$) such that the scheme is capable of evaluating depth-d circuits. The parameters of the scheme will grow polynomially with the *depth* of the circuits it computes.

What if we want to set parameters such that the scheme can evaluate circuits of *any polynomial depth*? That would be an FHE scheme for real. Essentially the only way we know to construct an FHE scheme at this point is using Gentry's bootstrapping technique which we describe below. (although, when we study obfuscation later on in the course, we will see an alternative.) Doing so involves making an additional assumption on the *circular security* of the GSW encryption scheme. Whether the circular security of the GSW scheme follows from LWE is an open question.

5.1 The Idea

Assume that you are the homomorphic evaluator and in the course of homomorphic evaluation, you get two (GSW) ciphertexts $\mathbf{C}$ and $\mathbf{C}'$ which are (a) *decryptable* to μ and μ' respectively, in the sense that their decryption noise has ℓ_∞ norm less than $q/4$; but (b) *not computable*, in the sense that they will become undecryptable after another homomorphic evaluation, say of a NAND. What should you do with these ciphertexts?

Here is an idea: If you had the secret key, you could decrypt $\mathbf{C}$ and $\mathbf{C}'$, re-encrypt them with *fresh small* noise and proceed with the computation. In fact, you could do this after every gate. But this is clearly silly. If you had the secret key, why bother with encrypted computation in the first place?

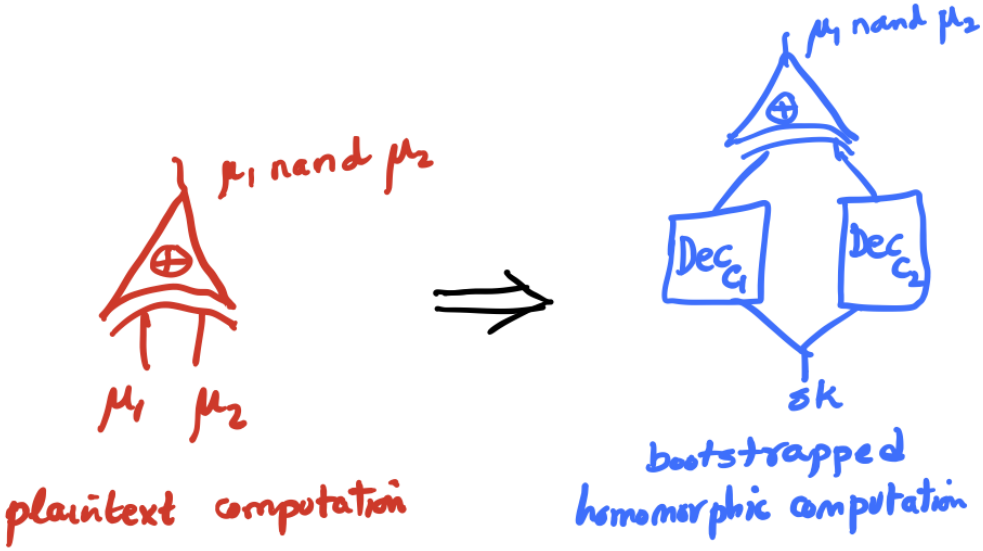
Here is a better idea: assume that you have *a ciphertext* $\tilde{\mathbf{C}}$ *of the FHE secret key encrypted under the secret key* itself (a so-called "circular encryption"). Then, you could homomorphically evaluate the following circuit on input $\tilde{\mathbf{C}}$:

$$\mathsf{BootNAND}_{\mathbf{C},\mathbf{C}'}(\mathsf{sk}) = \mathsf{Dec}_{\mathsf{sk}}(\mathbf{C}) \ \ \mathsf{NAND} \ \ \mathsf{Dec}_{\mathsf{sk}}(\mathbf{C}')$$

What you get out is an encryption of μ NAND μ'. How did this happen (and what *did* happen?) First of all, note that $\widetilde{\mathbf{C}}$ is a *fresh* encryption of sk. Secondly, assume that the BootNAND circuit (which is predominantly the decryption circuit) has small depth, small enough that the homomorphic evaluation can handle it. The output of the circuit on input sk is indeed μ NAND μ'; therefore, putting together this discussion, the output of the homomorphic evaluation of the circuit is *an encryption of* μ NAND μ' under sk.

Once we can implement BootNAND, this is how we evaluate every NAND gate. You get as input two ciphertexts $\mathbf{C}$ and $\mathbf{C}'$. You **do not** homomorphically evaluate on them, as then you will get garbage. Instead, use them to construct the circuit $\mathsf{BootNAND}_{\mathbf{C},\mathbf{C}'}$. and homomorphically evaluate it on an encryption $\widetilde{\mathbf{C}}$ of the secret key sk that you are given as an additional *evaluation key*.

Voila! This gives us a fully homomorphic encryption scheme.



5.2 Circular Security

Is it OK to publish a circular encryption? Does the IND-CPA security of the scheme hold when the adversary additionally gets such an encryption? First of all, the IND-CPA security of the underlying encryption

scheme (GSW in this case) alone does not tell us anything about what happens in this scenario. Indeed, you can construct an IND-CPA secure encryption scheme whose security *breaks completely* given such a circular encryption. (I will leave it as an exercise.)

Secondly, and quite frustratingly, we do know how to show that the Regev encryption scheme is circular-secure assuming LWE, but showing that the GSW scheme is circular-secure is one of my favorite open problems in lattice-based cryptography. For some modest progress on this question, see my paper with Daniele Micciancio.