CS 294. Encrypted Computation from Lattices

In this lecture, we will explore various facets of encrypted computation which, generally speaking, refers to the set of cryptographic tasks where you *encrypt* computational objects – for example, a program or a circuit and/or its input – in a way that anyone holding these encrypted objects can perform meaningful manipulations on them. Examples include (fully) homomorphic encryption, (various flavors of) attribute-based encryption, (fully) homomorphic signatures, constrained pseudorandom functions, functional encryption and indistinguishability obfuscation.

We will see constructions of all but the last two in this lecture. Indeed, we will present a single lattice tool, the *key lattice equation*, that will give us *all* these constructions.

1 Fully Homomorphic Encryption

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.
- $\mathsf{Enc}(sk, \mu)$, where $\mu \in \{0, 1\}$: produces a ciphertext c.
- $\mathsf{Dec}(sk, c)$: 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

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

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.)

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

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 in the sequel.)

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

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

  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: exercise.
- 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.

4 How to Add and Multiply (without 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. 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.

The only way we know to construct an FHE scheme at this point is using Gentry's bootstrapping technique which we describe below. Doing so involves making an additional assumption on the *circular security* of the GSW encryption scheme, which we don't know how to reduce to LWE.

5.1 The Idea

Assume that you are the homomorphic evaluator and in the course of homomorphic evaluation, you get two ciphertexts C and 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 C and 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?

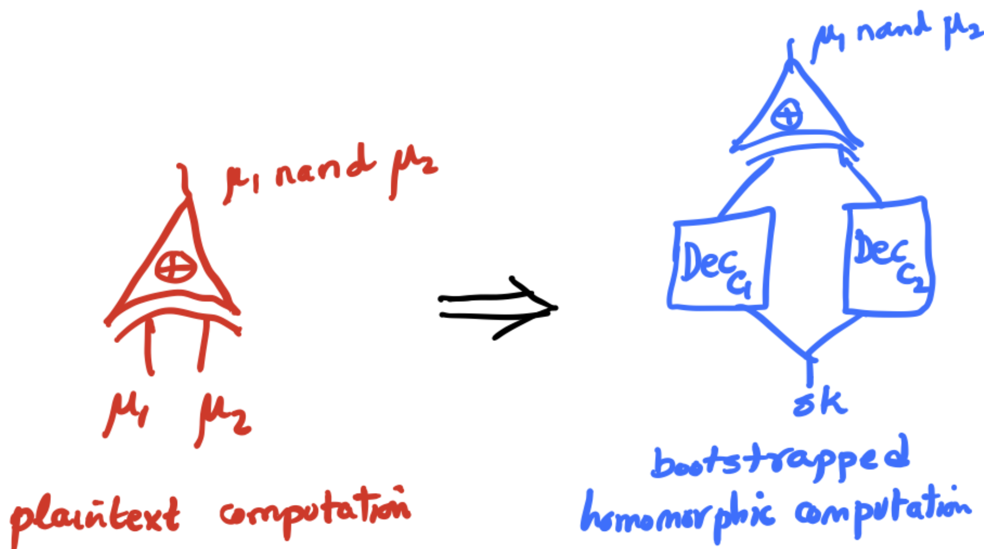
Here is a better idea: assume that you have *a ciphertext $\tilde{C}$ of the FHE secret key encrypted under the secret key* itself (a so-called "circular encryption"). Then, you could homomorphically evaluate the following circuit on input $\tilde{C}$:

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

What you get out is an encryption of μ NAND μ'. How did this happen (and what *did* happen?) First of all, note that $\widetilde{C}$ 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 C and C'. You **do not** homomorphically evaluate on them, as then you will get garbage. Instead, use them to construct the circuit $\mathsf{BootNAND}_{C,C'}$. and homomorphically evaluate it on an encryption $\widetilde{C}$ 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.

6 The Key Equation

Let us abstract out the mathematics behind GSW into a *key lattice equation* which will guide us through constructing the rest of the primitives in this lecture.

Recall the approximate eigenvector relation:

$$\mathbf{s}^T \mathbf{A}_i \approx \mu_i \mathbf{s}^T \mathbf{G}$$

and rewrite it as

$$\mathbf{s}^T (\mathbf{A}_i - \mu_i \mathbf{G}) \approx \mathbf{0} \tag{1}$$

Let $\mathbf{A}_f$ be the homomorphically evaluated ciphertext for a function f. We know that

$$\mathbf{s}^T \mathbf{A}_f \approx f(\vec{\mu}) \mathbf{s}^T \mathbf{G}$$

or

$$\mathbf{s}^T (\mathbf{A}_f - f(\vec{\mu}) \mathbf{G}) \approx \mathbf{0} \tag{2}$$

We will generalize this to arbitrary matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ – not necessarily ones that share the same eigenvector.

First, we know that $\mathbf{A}_f$ is a function of $\mathbf{A}_1, \dots, \mathbf{A}_\ell$ and f (*but not* $\mu_1, \dots, \mu_\ell$). Henceforth, when we say $\mathbf{A}_f$, we will mean a matrix obtained by the GSW homomorphic evaluation procedure. (That is, homomorphic addition of two matrices is matrix addition; homomorphic multiplication is matrix multiplication after bit-decomposing the second matrix).

Second, and very crucially, we can show that for any sequence of matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$,

$$[\mathbf{A}_1 - \mu_1 \mathbf{G} || \ldots || \mathbf{A}_\ell - \mu_\ell \mathbf{G}] \; \mathbf{H}_{f,\vec{\mu}} = \mathbf{A}_f - f(\vec{\mu})\mathbf{G}$$

where $\mathbf{H}_{f,\vec{\mu}}$ is a matrix with small coefficients. We call this the key lattice equation.

To see this for addition, notice that

$$[\mathbf{A}_1 - \mu_1 \mathbf{G} || \mathbf{A}_2 - \mu_2 \mathbf{G}] \; \underbrace{\begin{bmatrix} \mathbf{I} \\ \mathbf{I} \end{bmatrix}}_{\mathbf{H}_{+,\mu_1,\mu_2}} = \mathbf{A}_1 + \mathbf{A}_2 - (\mu_1 + \mu_2)\mathbf{G} = \mathbf{A}_+ - (\mu_1 + \mu_2)\mathbf{G}$$

and for multipication,

$$[\mathbf{A}_1 - \mu_1 \mathbf{G} || \mathbf{A}_2 - \mu_2 \mathbf{G}] \; \underbrace{\begin{bmatrix} \mathbf{G}^-(\mathbf{A}_2) \\ \mu_1 \mathbf{I} \end{bmatrix}}_{\mathbf{H}_{\times,\mu_1,\mu_2}} = \mathbf{A}_1 \mathbf{G}^-(\mathbf{A}_2) - \mu_1 \mu_2 \mathbf{G} = \mathbf{A}_\times - \mu_1 \mu_2 \mathbf{G}$$

By composition, we get that

$$[\mathbf{A}_1 - \mu_1 \mathbf{G} || \mathbf{A}_2 - \mu_2 \mathbf{G} || \ldots || \mathbf{A}_\ell - \mu_\ell \mathbf{G}] \; \mathbf{H}_{f,\vec{\mu}} = \mathbf{A}_f - f(\vec{\mu})\mathbf{G}$$

where $\mathbf{H}_{f,\mu}$ is a matrix with small entries (roughly proportional to $m^{O(d)}$ where d is the circuit depth of f).

An Advanced Note: Given arbitrary matrices $\mathbf{A}_i$ and $\mathbf{A}_f$, there exists such a small matrix $\mathbf{H}$; but if $\mathbf{A}_f$ is arbitrary, it is hard to find.

Let's re-derive FHE from the key equation:

- The ciphertexts are the matrices $\mathbf{A}_i$ and we picked them such that

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

- Homomorphic evaluation is computing $\mathbf{A}_f$ starting from $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$.
- Correctness of homomorphic eval follows from the key equation: We know that

$$\mathbf{s}^T[\mathbf{A}_1 - \mu_1\mathbf{G}||\ldots||\mathbf{A}_\ell - \mu_\ell\mathbf{G}] \approx \mathbf{0}$$

  by the equation above that characterizes ciphertexts. Therefore, by the key equation,

$$\mathbf{s}^T[\mathbf{A}_f - f(\vec{\mu})\mathbf{G}] = \mathbf{s}^T[\mathbf{A}_1 - \mu_1\mathbf{G}||\ldots||\mathbf{A}_\ell - \mu_\ell\mathbf{G}]\ \mathbf{H}_{f,\vec{\mu}} \approx \mathbf{0}$$

  as well meaning that $\mathbf{A}_f$ is an encryption of $f(\vec{\mu})$. Note that no one needs to know or compute the matrix $\mathbf{H}$; it only appears in the analysis.

7 Fully Homomorphic Signatures

We will use the key equation quickly in succession to derive three applications. The first is fully homomorphic signatures (FHS). Here is a first take in defining what one might want from an FHS scheme: a way to take a bunch of messages $\mu_1, \ldots, \mu_\ell$ together with their signatures $\sigma_1, \ldots, \sigma_\ell$ that verify under a public key PK and compute a signature σ_f of the message $f(\mu_1, \ldots, \mu_\ell)$ that verifies under PK (for any function f).

However, this is meaningless. You could produce signatures for constant functions $f_\alpha(x) = \alpha$ and thereby forge the signature on *any message* whatsoever.

Rather, what we need from an FHS is that it produces a signature σ_f that *binds* the output of a computation $f(\vec{\mu})$ with the computation itself f. Here is the definition:

1. $PK, f \to PK_f$.
2. $(\mu_1, \sigma_1), \ldots, (\mu_\ell, \sigma_\ell) \to (f(\vec{\mu}), \sigma_f)$.
   // Both the operations above are as expensive as computing f.
3. $\mathsf{Verify}(PK_f, f(\vec{\mu}), \sigma_f) = 1$.
4. For any f and any $y \neq f(\vec{\mu})$, no PPT adversary can produce a (fake) signature σ' such that $\mathsf{Verify}(PK_f, y, \sigma') = 1$ (except with negligible probability.)

Why is this useful? An application is (online-offline) verifiable delegation of computation.

Here is a basic construction using the key equation.

- PK is $\mathbf{B}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell$. (This scheme can sign ℓ bits). SK is trapdoor of $\mathbf{B}$.
- Signature σ_i for a message μ_i is a short $\mathbf{R}_i$ such that $\mathbf{B}\mathbf{R}_i = \mathbf{A}_i - \mu_i \mathbf{G}$.
  // (Can you see how the signing algorithm works?)
- PK_f is $\mathbf{A}_f$.
- To homomorphically compute on the signatures, start from the key equation:

$$[\mathbf{A}_1 - \mu_1 \mathbf{G} || \ldots || \mathbf{A}_\ell - \mu_\ell \mathbf{G}] \; \mathbf{H}_{f,\vec{\mu}} = \mathbf{A}_f - f(\vec{\mu})\mathbf{G}$$

  Notice that the way we constructed signatures,

$$\mathbf{B} \; \underbrace{[\mathbf{R}_1 || \ldots || \mathbf{R}_\ell] \; \mathbf{H}_{f,\vec{\mu}}}_{\sigma_f} = \underbrace{\mathbf{A}_f}_{PK_f} - f(\vec{\mu})\mathbf{G}$$

  σ_f is thus the homomorphic signature of $f(\vec{\mu})$ under PK_f.
- Why can't an adversary cheat? Suppose an adversary produces a signature σ' that verifies for the message $y \neq f(\vec{\mu})$ w.r.t. PK_f. So,

$$\mathbf{B}\sigma' = \mathbf{A}_f - y\mathbf{G}$$

  Subtracting the last two equations, we get

$$\mathbf{B}(\sigma' - \sigma_f) = (f(\vec{\mu}) - y)\mathbf{G}$$

  So, $\sigma' - \sigma_f$ is an inhomogenous trapdoor for $\mathbf{B}$, constructing which breaks SIS.

8 Attribute-based Encryption

Attribute-based encryption (ABE) generalizes IBE in the following way.

- Setup produces MPK, MSK.
- Enc uses MPK to encrypt a message m relative to attributes $(\mu_1, \ldots, \mu_\ell) \in \{0,1\}^\ell$.

  (In an IBE scheme, $\vec{\mu} = ID$.)

- KeyGen uses MSK to generate a secret key SK_f for a given Boolean function $f : \{0,1\}^\ell \to \{0,1\}$.

  (IBE is the same as ABE where f is restricted to be the point (delta) function $f_{ID'}(ID) = 1$ iff $ID = ID'$.)

- Dec gets $\vec{\mu}$ (attributes are in the clear) and uses SK_f to decrypt a ciphertext C if $f(\vec{\mu}) = 1$ (true). If $f(\vec{\mu}) = 0$, Dec simply outputs $\perp$.

Here is an ABE scheme (called the BGG+ scheme) using the key equation. It's best to view this as a generalization of the Agrawal-Boneh-Boyen IBE scheme.

- KeyGen outputs matrices $\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell$ and a vector $\mathbf{v}$ and these form the MPK. The MSK is the trapdoor for $\mathbf{A}$.

- Enc computes

$$\mathbf{s}^T[\mathbf{A}||\mathbf{A}_1 - \mu_1\mathbf{G}||\ldots||\mathbf{A}_\ell - \mu_\ell\mathbf{G}]$$

  (plus error, of course, and we will consider that understood.) Finally, the message is encrypted as $\mathbf{s}^T\mathbf{v} + e + m\lfloor q/2 \rfloor$.

- Let's see how Dec might work. You (and in fact anyone) can compute

$$\mathbf{s}^T[\mathbf{A}||\mathbf{A}_1 - \mu_1\mathbf{G}||\ldots||\mathbf{A}_\ell - \mu_\ell\mathbf{G}]\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_{f,\vec{\mu}} \end{bmatrix} = \mathbf{s}^T[\mathbf{A}||\mathbf{A}_f - f(\vec{\mu})\mathbf{G}]$$

  using the key equation.

  If you had a short $\mathbf{r}$ that maps $[\mathbf{A}||\mathbf{A}_f - \mathbf{G}]$ to $\mathbf{v}$, that is

$$[\mathbf{A}||\mathbf{A}_f - \mathbf{G}]\ \mathbf{r} = \mathbf{v}$$

  you can decrypt and find m. (Can you fill in the blanks?)

Two notes:

- The security definition mirrors IBE exactly, and the security proof of this scheme mirrors that of the ABB IBE scheme that we did in the last lecture. I will leave it to you as an exercise. The reference is the work of [BGG+14].
- One might wonder if the attributes $\vec{\mu}$ need to be revealed. The answer is "NO", in fact one can construct an attribute-hiding ABE scheme (also called a predicate encryption scheme). There are two flavors of security of such a scheme, the weaker one can be realized using LWE [GVW15] and the stronger one implies indistinguishability obfuscation, a *very* powerful cryptographic primitive which we don't know how to construct from LWE yet. More in the next lecture.

9 Constrained PRF

A constrained PRF is a special type of PRF where the owner of the PRF key K can construct a special key K_f which enables anyone to compute

$$\forall x \text{ s.t. } f(x) = 0 : \quad \mathsf{PRF}(K, x)$$

(here, arbitrarily and for convention, we will set 0 to mean `true`.) The PRF values at all other values should remain hidden given K_f, the *constrained key.*

We will only consider single-key CPRFs here, that is the adversary gets to see the constrained key for a single function f of her choice. Constructing many-key CPRFs from LWE is another one of my favorite open problems!

More generally, the adversary can get a single constrained key together with oracle access to the PRF (as usual). Her job is to compute $\mathsf{PRF}(K, x^*)$ for some x^* where (a) $f(x^*) = 1$ (false) and (b) she did not make an oracle query on x^*.

Here is a construction of a constrained PRF using the key equation. See [BV15] for details and extensions.

- The scheme has public parameters $\mathbf{B}_0, \mathbf{B}_1$ and $\mathbf{A}_1, \ldots, \mathbf{A}_k$ where k is an upper bound on the description length of any function f that will be constrained. The PRF key is $\mathbf{s}$.
- To define the PRF, consider the universal function $\mathcal{U}$:

$$\forall f \text{ where} |f| \leq k, x \in \{0,1\}^\ell : \quad \mathcal{U}(f, x) = f(x)$$

  The key equation applied to the universal circuit $\mathcal{U}$ now tells us that

$$\begin{aligned} & [\mathbf{A}_1 - f_1\mathbf{G}|| \ldots ||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_{x_1} - x_1\mathbf{G}|| \ldots ||\mathbf{B}_{x_\ell} - x_\ell\mathbf{G}] \ \mathbf{H}_{\mathcal{U},f,x} \\ & = \mathbf{A}_{\mathcal{U},x} \ - \ \mathcal{U}(f,x)\mathbf{G} \\ & = \mathbf{A}_{\mathcal{U},x} - f(x)\mathbf{G} \end{aligned}$$

  Here, $\mathbf{A}_{\mathcal{U},x}$ (which we will simply denote as $\mathbf{A}_x$) is a result of the GSW homomorphic evaluation on the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_k, \mathbf{B}_{x_1}, \ldots, \mathbf{B}_{x_\ell}$.

- The PRF is defined to be $\lfloor \mathbf{s}^T \mathbf{A}_x \rceil$ on every input x.

  Note that the PRF has to be defined *independent of which function the key will later be constrained with.* Indeed, this definition of the PRF does not depend on f at all.

Here is a construction of a constrained PRF using the key equation. See [BV15] for details and extensions.

- The scheme has public parameters $\mathbf{B}_0, \mathbf{B}_1$ and $\mathbf{A}_1, \ldots, \mathbf{A}_k$ where k is an upper bound on the description length of any function f that will be constrained. The PRF key is $\mathbf{s}$.
- To define the PRF, consider the universal function $\mathcal{U}$:

$$\forall f \text{ where} |f| \leq k, x \in \{0,1\}^\ell : \quad \mathcal{U}(f, x) = f(x)$$

The key equation applied to the universal circuit $\mathcal{U}$ now tells us that

$$\begin{aligned}[\mathbf{A}_1 - f_1\mathbf{G}|| \ldots ||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_{x_1} - x_1\mathbf{G}|| \ldots ||\mathbf{B}_{x_\ell} - x_\ell\mathbf{G}] \ \mathbf{H}_{\mathcal{U},f,x} \\ = \mathbf{A}_{\mathcal{U},x} \ - \ \mathcal{U}(f,x)\mathbf{G} \\ = \mathbf{A}_{\mathcal{U},x} - f(x)\mathbf{G}\end{aligned}$$

Here, $\mathbf{A}_{\mathcal{U},x}$ (which we will simply denote as $\mathbf{A}_x$) is a result of the GSW homomorphic evaluation on the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_k, \mathbf{B}_{x_1}, \ldots, \mathbf{B}_{x_\ell}$.
- The PRF is defined to be $\lfloor \mathbf{s}^T \mathbf{A}_x \rceil$ on every input x.
- The constrained key for a function f is

$$\mathbf{s}^T[\mathbf{A}_1 - f_1\mathbf{G}|| \ldots ||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_0||\mathbf{B}_1 - \mathbf{G}]$$

On input x, the constrained eval proceeds as follows. First you can get

$$[\mathbf{A}_1 - f_1\mathbf{G}|| \ldots ||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_{x_1} - x_1\mathbf{G}|| \ldots ||\mathbf{B}_{x_\ell} - x_\ell\mathbf{G}]$$

for any x of your choice. Second, using the key equation, multiplying this on the right by $\mathbf{H}_{\mathcal{U},f,x}$:

$$\mathbf{s}^T[\mathbf{A}_x - f(x)\mathbf{G}]$$

from which one computes $\lfloor \mathbf{s}^T \mathbf{A}_x \rceil := \mathsf{PRF}(K, x)$ if $f(x) = 0$ (true).

Here is a construction of a constrained PRF using the key equation. See [BV15] for details and extensions.

- The scheme has public parameters $\mathbf{B}_0, \mathbf{B}_1$ and $\mathbf{A}_1, \ldots, \mathbf{A}_k$ where k is an upper bound on the description length of any function f that will be constrained. The PRF key is $\mathbf{s}$.
- To define the PRF, consider the universal function $\mathcal{U}$:

$$\forall f \text{ where} |f| \leq k, x \in \{0,1\}^\ell : \quad \mathcal{U}(f, x) = f(x)$$

The key equation applied to the universal circuit $\mathcal{U}$ now tells us that

$$\begin{aligned}
[\mathbf{A}_1 - f_1\mathbf{G}|| \ldots ||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_{x_1} - x_1\mathbf{G}|| \ldots ||\mathbf{B}_{x_\ell} - x_\ell\mathbf{G}]\ \mathbf{H}_{\mathcal{U},f,x} \\
= \mathbf{A}_{\mathcal{U},x} \ - \ \mathcal{U}(f,x)\mathbf{G} \\
= \mathbf{A}_{\mathcal{U},x} - f(x)\mathbf{G}
\end{aligned}$$

Here, $\mathbf{A}_{\mathcal{U},x}$ (which we will simply denote as $\mathbf{A}_x$) is a result of the GSW homomorphic evaluation on the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_k, \mathbf{B}_{x_1}, \ldots, \mathbf{B}_{x_\ell}$.

- The PRF is defined to be $\lfloor \mathbf{s}^T \mathbf{A}_x \rceil$ on every input x.
- The constrained key for a function f is

$$\mathbf{s}^T[\mathbf{A}_1 - f_1\mathbf{G}|| \ldots ||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_0||\mathbf{B}_1 - \mathbf{G}]$$

On input x, you can compute

$$\mathbf{s}^T[\mathbf{A}_x - f(x)\mathbf{G}]$$

- For security: suppose an adversary managed to compute

$$\mathbf{s}^T\mathbf{A}_x + \mathbf{e}$$

for some x where $f(x) = 1$. We can ourselves compute

$$\mathbf{s}^T(\mathbf{A}_x - \mathbf{G}) + \mathbf{e}'$$

using constrained evaluation. Put together, these reveal $\mathbf{s}^T\mathbf{G}$ plus error, and therefore $\mathbf{s}$, breaking LWE.

References

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer, 2014.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2015.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, 2015.