**CS 7880: Special Topics in Cryptography** 9/17/20

# Lecture 2: FHE From Gound Up

*Lecturer: Daniel Wichs* *Scribe: Alan Turing*

The notes describe an elegant way of constructing FHE by starting with an extremely simple cryptosystem and adding functionality one small piece at a time. This exposition was suggested by Daniele Micciancio at his Eurocrypt 2019 invited talk.

## 1 Basic Symmetric Encryption Scheme from LWE

- $\mathsf{Enc}_{\mathbf{s}}(x) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e + x) \ : \ \mathbf{a} \leftarrow \mathbb{Z}_q^n, e \leftarrow \chi$.
- $\mathsf{Dec}_{\mathbf{s}}(\mathbf{a}, b) = b - \langle \mathbf{a}, \mathbf{s} \rangle$.

The above encryption scheme does not have correctness: if you decrypt and encryption of $x$ you get $x + e$. This can be fixed by only using $x \in \{0, \lfloor q/2 \rfloor\}$ in which case we can remove the error $e$ by testing if the decrypted value is closer to 0 or $q/2$. However, it will be convenient to think of this as an encryption scheme that works for all $x$ but decryption only recovers something close to $x$.

We will abuse notation and write $\mathsf{Enc}_{\mathbf{s}}(x)$ to denote some arbitrary element of the form $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e + x)$. We will say that $\mathsf{Enc}_{\mathbf{s}}(x)$ has error $\beta$ if $|e| \leq \beta$.

The LWE assumption implies that encryptions of arbitrary values are indistinguishable from uniformly random vectors in $\mathbb{Z}_q^{n+1}$.

The scheme has the following properties:

1. Additive homomorphism: $\mathsf{Enc}_{\mathbf{s}}(x) + \mathsf{Enc}_{\mathbf{s}}(y) = \mathsf{Enc}_{\mathbf{s}}(x + y)$. The error goes from $\beta$ to $2\beta$.
2. Negation homomorphishm: -$\mathsf{Enc}_{\mathbf{s}}(x) = \mathsf{Enc}_{\mathbf{s}}(-x)$. The error $\beta$ stays the same.
3. Multiplication by small constant: $c \cdot \mathsf{Enc}_{\mathbf{s}}(x) = \mathsf{Enc}_{\mathbf{s}}(c \cdot x)$. The error goes from $\beta$ to $c \cdot \beta$.
4. Public encryptions: Can come up with a valid encryption of any value $x$ without knowing the secret key. Namely $(\mathbf{0}, x) \in \mathsf{Enc}_{\mathbf{s}}(x)$ with error 0.
5. Public circular encryptions: Can come up with a valid encryption of each secret key component $\mathbf{s}_i$ without knowing the secret key. Namely $(-\mathbf{1}_i, 0) \in \mathsf{Enc}_{\mathbf{s}}(\mathbf{s}_i)$ with error 0. Here $\mathbf{1}_i$ is the unit vector with a 1 in position $i$ and 0 everywhere else and $\mathbf{s}_i$ is the $i$'th position of the secret key $\mathbf{s}$. We can also come up with a valid encryption of $c\mathbf{s}_i$ for any constant $c$ without knowing the secret key; namely $(-c \cdot \mathbf{1}_i, 0) \in \mathsf{Enc}_{\mathbf{s}}(\mathbf{s}_i)$ with error 0.

Note that the public encryptions can be created without knowing the secret key $\mathbf{s}$. They are fixed vectors and do not provide any security - they reveal what value is being encrypted. However, we can re-randomize by adding in fresh encryption of 0. Because fresh encryptions of 0 are indistinguishable from uniformly random vectors, the sum is then also indistinguishable from a uniformly random vectors. This shows that the scheme has circular security: encryptions of any values $c \cdot \mathbf{s}_i$ are indistinguishable from random.

The above properties can also be used to get a public-key encryption from a symmetric-key one. The public key $\mathsf{pk}$ consists of many random encryption of 0 :

$$\mathsf{pk} = \{ct_i \leftarrow \mathsf{Enc}_{\mathbf{0}}(0)\} = \{= (\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s}\rangle + e_i)\} = (\mathbf{A}, \mathbf{b} = \mathbf{s}\mathbf{A} + \mathbf{e})$$

To encrypt a value $x$, sum up a random subset of the encryptions of 0 in the public key, which gives a fresh encryption of 0 and then add a public encryption of $x$:

$$\mathsf{Enc}_{\mathsf{pk}}(x) = \sum_{i \in I} ct_i + (\mathbf{0}, x) = \sum r_i(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s}\rangle + e_i) + (\mathbf{0}, x) = (\mathbf{A}\mathbf{r}^T, \mathbf{b} \cdot \mathbf{r}^T + x)$$

This is exactly the Regev public-key encryption from the previous lecture.

**Multiplying by Large Constant.** We now modify the scheme to allow multiplication by a large constant. We call the new schem the “prime” scheme $\mathsf{Enc}'$, to distinguish from earlier “base” scheme $\mathsf{Enc}$. To encrypt under $\mathsf{Enc}'$ we simply use the base scheme $\mathsf{Enc}$ to encrypt all the powers of 2 times $x$:

$$\mathsf{Enc}'_{\mathbf{s}}(x) = (\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(2 \cdot x), \ldots, \mathsf{Enc}_{\mathbf{s}}(2^{\lfloor \log q \rfloor} x))$$

It’s easy to see that the prime scheme still satisfies properties 1,2 above (in fact it satisfies 1-5, but we will only rely on 1,2). Moreover, it now allows us to also decrypt encryptions of small values $x \in \{0, 1\}$ by looking at the component $\mathsf{Enc}_{\mathbf{s}}(2^i \cdot x)$ where $2^i$ is the power of 2 closest to $q/2$.

We now show how to take any constant $c \in \mathbb{Z}_q$ and $\mathsf{Enc}'_{\mathbf{s}}(x)$ to get $\mathsf{Enc}'_{\mathbf{s}}(c \cdot x)$ without increasing the error too much. Let $c = \sum_{i=0}^{\lfloor \log q \rfloor} c_i \cdot 2^i$ be the binary decomposition of $c$ so that $c_i \in \{0, 1\}$. Then we define the operation:

$$c * \mathsf{Enc}'_{\mathbf{s}}(x) = \sum_{i=0}^{\lfloor \log q \rfloor} c_i \cdot \mathsf{Enc}_{\mathbf{s}}(2^i \cdot x) = \mathsf{Enc}_{\mathbf{s}}(\sum_{i=0}^{\lfloor \log q \rfloor} c_i \cdot 2^i \cdot x) = \mathsf{Enc}_{\mathbf{s}}(c \cdot x)$$

The error goes from $\beta$ to $\beta \cdot \log q$ since we just added up at most $\log q$ basic encryptions. We define the * operation to output a basic (non-prime) encryption $\mathsf{Enc}_{\mathbf{s}}(c \cdot x)$. However, we can apply if for $c, 2c, \ldots, 2^{\lfloor \log q \rfloor} c$ to get $(\mathsf{Enc}_{\mathbf{s}}(c \cdot x), \ldots, \mathsf{Enc}_{\mathbf{s}}(2^{\lfloor \log q \rfloor} c \cdot x)) = \mathsf{Enc}'_{\mathbf{s}}(c \cdot x)$.

The above allows us to compute arbitrary linear functions over encrypted data. If we have encryptions $\mathsf{Enc}'(x_1), \ldots, \mathsf{Enc}'(x_\ell)$ and some coefficients $c_i$ we can compute $\mathsf{Enc}'(\sum_{i=1}^{\ell} c_i \cdot x_i)$.

**Homomorphic Decryption.** Say we have a basic encryption of $x$

$$\mathsf{Enc}_{\mathbf{s}}(x) = (\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s}\rangle + e + x).$$

Notice that decryption $\mathsf{Dec}_{\mathbf{s}}(\mathbf{a}, b) = b - \langle \mathbf{a}, \mathbf{s} \rangle$ is a linear function of $\mathbf{s}$. Assume we have a prime encryption of the secret key components $\{\mathsf{Enc}'_{\mathbf{s}}(\mathbf{s}_i)\}_{i=0,\ldots,n}$, where we define $\mathbf{s}_0 = 1$. We can then evaluate the decryption of $(\mathbf{a}, b)$ over the encrypted secret key $\mathbf{s}$ as:

$$b*\mathsf{Enc}'_{\mathbf{s}}(1) - \sum_{i=1}^{n} \mathbf{a}_i * \mathsf{Enc}'_{\mathbf{s}}(\mathbf{s}_i) = \mathsf{Enc}_{\mathbf{s}}(b) - \sum_{i=1}^{n} \mathsf{Enc}_{\mathbf{s}}(\mathbf{a}_i \cdot \mathbf{s}_i) = \mathsf{Enc}_{\mathbf{s}}(b - \langle \mathbf{a}, \mathbf{s} \rangle) = \mathsf{Enc}_{\mathbf{s}}(x+e) = \mathsf{Enc}_{\mathbf{s}}(x)$$

What did we just do? We went from one encryption of $x$ to another encryption of $x$. That's not very interesting on its own, but the way we did it is interesting. We did it by taking the encryption of $x$ and interpreting the ciphertext as defining a linear function which we then evaluated homomorphically over encryptions of $\mathbf{s}_i$.

The error went from $\beta$ to $(n+1) \cdot \beta \cdot \log q + \beta$ (since each * operation results in error $\beta \log q$ and we're summing up $n+1$ of them, but also adding in the error $e$ from the encryption of $x$).

**Homomorphic Decrypt and Multiply.** We can use the above idea to multiply two encrypted values $x, y$ to get an encryption of $x \cdot y$. The idea is that we take some value $\mathsf{Enc}_{\mathbf{s}}(x)$ and decrypt it with the secret key $y \cdot \mathbf{s}$, we get a value $x \cdot y$. Therefore if we start with a prime encryption of $y \cdot \mathbf{s}$ and then homomorphically compute the decryption of some ciphertext $(\mathbf{a}, b) = \mathsf{Enc}_{\mathbf{s}}(x)$ we will end with an encryption of $x \cdot y$.

In more detail, we modify the encryption scheme once more and define:

$$\mathsf{Enc}''_{\mathbf{s}}(x) = (\mathsf{Enc}'_{\mathbf{s}}(x \cdot \mathbf{s}_i))_{i=0,\ldots,n} = (\mathsf{Enc}_{\mathbf{s}}(2^j \cdot x \cdot \mathbf{s}_i))_{i=0,\ldots,n; j=0,\ldots,\lfloor \log q \rfloor}$$

(recall that $\mathbf{s}_0 := 1$). Note that this encryption scheme is secure by the circular security of the basic scheme $\mathsf{Enc}$. Furthermore, it still satisfies properties 1,2.

For $\mathsf{Enc}_{\mathbf{s}}(x) = (\mathbf{a}, b)$ define the operation:

$$\begin{aligned}
\mathsf{Enc}_{\mathbf{s}}(x) * \mathsf{Enc}''_{\mathbf{s}}(y) &= b * \mathsf{Enc}'_{\mathbf{s}}(y) - \sum_{i=1}^{n} \mathbf{a}_i * \mathsf{Enc}'_{\mathbf{s}}(y \cdot \mathbf{s}_i) \\
&= \mathsf{Enc}_{\mathbf{s}}(y \cdot b) - \sum_{i=1}^{n} \mathsf{Enc}_{\mathbf{s}}(y \cdot \mathbf{a}_i \cdot \mathbf{s}_i) \\
&= \mathsf{Enc}_{\mathbf{s}}(y(b - \langle \mathbf{a}, \mathbf{s} \rangle)) = \mathsf{Enc}_{\mathbf{s}}(y(x+e)) \\
&= \mathsf{Enc}_{\mathbf{s}}(xy)
\end{aligned}$$

The error goes from $\beta$ to $(n+1) \cdot \beta \cdot \log q + y * \beta$. Therefore, we can only do the above for small $y$, say $y \in \{0, 1\}$.

We extend the above operation to multiplying two double-prime ciphertext as follows:

$$\mathsf{Enc}''_{\mathbf{s}}(x) * \mathsf{Enc}''_{\mathbf{s}}(y) = (\mathsf{Enc}_{\mathbf{s}}(2^j \cdot x \cdot \mathbf{s}_i) * \mathsf{Enc}''_{\mathbf{s}}(y))_{i,j} = (\mathsf{Enc}_{\mathbf{s}}(2^j \cdot x \cdot y \cdot \mathbf{s}_i))_{i,j} = \mathsf{Enc}''_{\mathbf{s}}(x \cdot y)$$

The error goes from $\beta$ to $(n+1) \cdot \beta \cdot \log q + y \cdot \beta$.

**Putting it all Together.** Given $\mathsf{Enc}''_{\mathbf{s}}(x), \mathsf{Enc}''_{\mathbf{s}}(y)$ where $x, y \in \{0, 1\}$ we can therefore compute a NAND gate as $\mathsf{Enc}''_{\mathbf{s}}(1) - \mathsf{Enc}''_{\mathbf{s}}(x) * \mathsf{Enc}''_{\mathbf{s}}(y) = \mathsf{Enc}''(1 - x \cdot y)$ where $\mathsf{Enc}''_{\mathbf{s}}(1)$ is a public encryption of 1 with error 0. The error goes from $\beta$ to $\beta \cdot ((n+1)\log q + 1)$.

We can compute an arbitrary circuit over encrypted data this way. If the original error is $\beta$ then the final error becomes $\beta \cdot ((n+1)\log q + 1)^d$ where $d$ is the depth of the circuit. We will be able to decrypt correctly at long as $q/4 > \beta \cdot ((n+1)\log q + 1)^d$. Therefore, by choosing the modulus $q$ large enough depending on the circuit depth $d$, we can evaluate any circuit of depth up to $d$. We will discuss parameters in more detail later on.