University of Illinois, Urbana Champaign
CS 598DK Special Topics in Cryptography

Instructor: Dakshita Khurana
Scribe: Sanket Kanjalkar, Yunqi Li
Date: October 09, 2019

LECTURE
11

Fully Homomorphic Encryption I

Informally, a cryptosystem that supports arbitrary computation on ciphertexts is known as fully homomorphic encryption (FHE)[4]. Throughout the course, we have studied encryption schemes for information transfer from one party to another party. However, if we were to compute on data using our traditional encryption schemes, it would require that data must be decrypted before it can be analyzed or manipulated. Naturally, one asked the question whether it is possible to compute on Encrypted Data. That is, it should be possible to compute a known function on encrypted data without having access to the secret key.

Such an FHE scheme can be used for privacy-preserving outsourced storage and computation. In today's lecture, we will see what is fully homomorphic encryption (FHE) scheme and how to build an FHE scheme. We will also look at intuitive ways to construct FHE for addition and multiplication.

FACT 11.1. *LWE is the only way we know to do fully homomorphic encryption as of Sept 2019.*

11.1 Recap

We provide a quick primer on the notation used in this section.

- All values which are bolded like $\boldsymbol{s}, \boldsymbol{a}$ are vectors where bolded Captial $\boldsymbol{A}$ is a matrix.
- All values which in simple like q, e are scalars. All scalars apart from field modulas q are in field Z_q.
- The values $\boldsymbol{a}, \boldsymbol{A}$ are public value seen by all parties, $\mathbf{s}$ is secret value and e, $\boldsymbol{e}$ are errors

Before stepping into how to build LWE-based FHE schemes, let's briefly recap how to build a private and public encryption scheme[1] with the LWE problem. Decisional LWE is the problem where given a matrix $A_{n,m}$, and a corresponding LWE vector result b_m, determining whether or not these and results vector of some instance of an LWE problem OR whether the vector b_m was simply drawn from sampling values uniformly randomly from Z_q^m.

DEFINITION 11.2. ***Decisional*** $LWEn, m, q, \mathcal{X}$: For all non-uniform probabilistic polynomial time adversary $\mathcal{A}$

$$|\underset{\substack{\boldsymbol{s}\leftarrow\mathbb{Z}_q^{n\times 1}\\ \boldsymbol{A}\leftarrow\mathbb{Z}_q^{n\times m}\\ \boldsymbol{e}\leftarrow\mathcal{X}^m}}{Pr}[\mathcal{A}(\boldsymbol{A}, \boldsymbol{s}^T\boldsymbol{A}+\boldsymbol{e}^T)=1] - \underset{\substack{\boldsymbol{A}\leftarrow\mathbb{Z}_q^{n\times m}\\ \boldsymbol{b}\leftarrow\mathbb{Z}_q^{m}}}{Pr}[\mathcal{A}(\boldsymbol{A}, \boldsymbol{b})=1]| = negl(n)$$

where q is a prime within $O(2^n)$, $m = O(n\log q)$[2] and norm $\parallel \boldsymbol{e} \parallel = \omega(\log n)$. Where $\omega(f(n))$ means that e should be greater than $f(n)$ asymptotically.

Next, we revise the secret key encryption (SKE) built with LWE which has $m = 1$. The secret key here is a vector $\boldsymbol{s} \in \mathbb{Z}_q^n$. We refer to the previous lecture for the definition of SKE and only give an instantiation of one using LWE.

$$KeyGen(1^n) : \boldsymbol{s} \leftarrow \mathbb{Z}_q^n$$

$$\underset{\substack{\boldsymbol{s}\leftarrow\mathbb{Z}_q^{n\times 1}\\ \boldsymbol{a}\leftarrow\mathbb{Z}_q^{n\times 1}\\ \boldsymbol{e}\leftarrow\mathcal{X}}}{Enc}(\boldsymbol{s}, \mu\in\{0,1\}) : (\boldsymbol{a}, (b = \boldsymbol{s}^T\boldsymbol{a} + e + \mu\lfloor\frac{q}{2}\rfloor) \mod q)$$

$$Dec(\boldsymbol{s}, \boldsymbol{a}, b) : b - \langle \boldsymbol{s}^T, \boldsymbol{a}\rangle = (e + \mu\lfloor\frac{q}{2}\rfloor) \mod q = \begin{cases} 0 & \text{if } b - \boldsymbol{s}^T\boldsymbol{a} \text{ mod q} \in [\frac{-q}{4}\ \frac{q}{4}] \\ 1 & \text{if } b - \boldsymbol{s}^T\boldsymbol{a} \text{ mod q} \in [\frac{q}{4}\ \frac{3q}{4}] \end{cases}$$

LWE can also be used to build public key encryption(PKE) with public key $pk = (\boldsymbol{A}, \boldsymbol{b}^T = \boldsymbol{s}^T\boldsymbol{A} + \boldsymbol{e}^T))$ and secret key sk. Just as before, interested reader can look at the definition of PKE in and here we only an instantiation of the scheme using LWE.

$$\underset{\substack{\boldsymbol{A}\leftarrow\mathbb{Z}_q^{n\times m}\\ \boldsymbol{a}\leftarrow\mathbb{Z}_q^{n\times 1}\\ \boldsymbol{e}\leftarrow\mathcal{X}^m}}{KeyGen}(1^n) : (sk = \boldsymbol{s}, pk = (\boldsymbol{A}, \boldsymbol{b}^T = \boldsymbol{s}^T\boldsymbol{A} + \boldsymbol{e}^T))$$

$$\underset{\boldsymbol{r}\leftarrow\{0,1\}^m}{Enc}(pk = (\boldsymbol{A}, \boldsymbol{b}^T), \mu\in\{0,1\}) : (\boldsymbol{c_1} = \boldsymbol{A}\boldsymbol{r}, c_2 = (\boldsymbol{b}^T\boldsymbol{r} + \mu\lfloor\frac{q}{2}\rfloor) \mod q)$$

$$Dec(sk = \boldsymbol{s}, (\boldsymbol{c_1}, c_2)) : c_2 - \boldsymbol{s}^T\boldsymbol{c_1} = \boldsymbol{e}^T\boldsymbol{r} + \mu\lfloor\frac{q}{2}\rfloor \mod q = \begin{cases} 0 & \text{if } c_2 - \boldsymbol{s}^T\boldsymbol{c_1} \text{ mod q} \in [\frac{-q}{4}\ \frac{q}{4}] \\ 1 & \text{if } c_2 - \boldsymbol{s}^T\boldsymbol{c_1} \text{ mod q} \in [\frac{q}{4}\ \frac{3q}{4}] \end{cases}$$

REMARK 11.3. Generally, we want the noise $\boldsymbol{e}$ introduced to the equations that are sampled from a distribution with zero mean and low standard variation. For the correctness of the encryption scheme, we will require that the noise distribution makes noise bound $\parallel e \parallel \leq q/4$ with high probability. As shown in the above decryption scheme, if the norm of noises is bounded by $q/4$, the boundary between the case $\mu = 1$ and $\mu = 0$ is very clear and the probability that decryption algorithm giving the wrong plaintext is very slow. In practice, $\mathcal{X}$ is usually a discrete Gaussian distribution over $\mathbb{Z}_q$. With Gaussian distribution that has to mean $\mu = 0$ and standard variation σ to be very small, according to its probability density function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$$

we can get that the error bound to go beyond $q/4$ is negligible when $\sigma \ll q/4$.

REMARK 11.4. We also recall that parameters e, $\boldsymbol{e}$ in both of the schemes above are neither in the public key nor in the secret key! That is neither the Encryption algorithms or the Decryption algorithms use it. But, it is also important for the security of the above schemes that errors are not known to other parties even though it is not a part of the secret key.

11.2 Fully Homomorphic Encryption (FHE)

Let us consider the scenario shown in 11.1. A client has a secret value of x. The client wants the server to do some computation on x without revealing what x is. To accomplish the following we do we follow the below protocol: First, a ciphertext $ct = Enc(x)$ is sent to the server along with the desired function f. The server then computes a new ciphertext $ct^* = Enc(f(x))$ by evaluating x on another function g which is publicly computable from f. After receiving ct^* from the server, the client can use its secret key sk to get the desired result of $f(x)$. Note that in the following example, the server learns the function f so it is aware of what computation is taking place, but cannot figure the values underneath.

Fully Homomorphic Encryption[5] refers to an extension of Encryption scheme with additional keywords `Fully` and `Homomorphic`. `Homomorphic` in FHE refers to homomorphism in mathematics: the encryption and decryption functions can be thought of as homomorphisms between plaintext and ciphertext domains. `Fully` refers to the fact that we can evaluate any function. There is another variant called `Levelled` Homomorphic encryptions where we restrict ourselves to evaluating only functions with certain complexity(depth).

Even though the Fully Homomorphic encryption scheme is our actual goal, in practice we also consider a simplification leveled fully homomorphic encryption scheme. Leveled FHE does not allow us to compute arbitrary functions f but only functions with a known depth d. Informally, when we already know what is the most complex(in terms of depth of the arithmetic circuit) and use that in the construction of our FHE scheme.

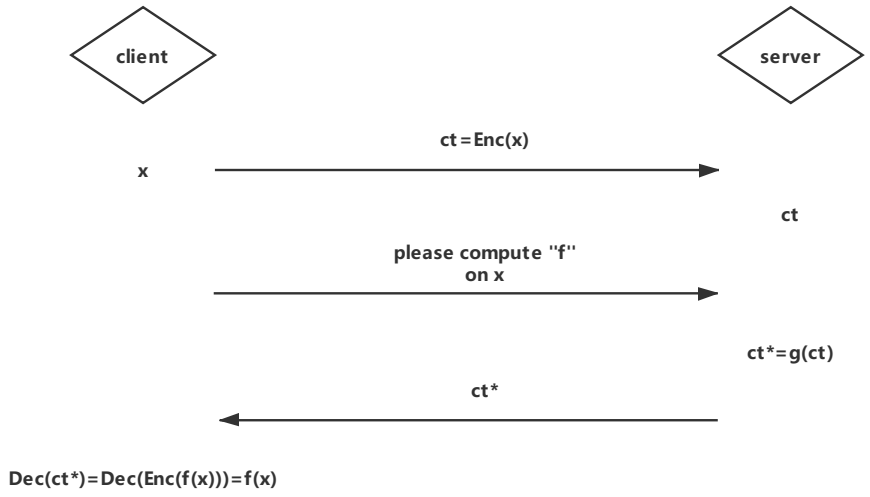

FIGURE 11.1: Outsourced Computation

FHE allows operations and analysis of encrypted data without revealing the original one, which removes the privacy barriers in several real-life applications.

DEFINITION 11.5. Let $\mathcal{C}$ be a class of circuits where for each $f \in \mathcal{C}$, $f : \{0,1\}^n \to \{0,1\}$. An encryption scheme $(KeyGen, Enc, Dec, Eval)$ is **$\mathcal{C}$-homomorphic** if $\forall f \in \mathcal{C}$, all ciphertexts $ct_1, \ldots, ct_n$, $Eval(f, ct_1, \ldots, ct_n) = ct^*$ such that if $\forall i$, $\exists m_i, r_i$ s.t. $ct_i = Enc(m_i; r_i)$, then $Dec_{sk}(ct^*) = f(m_1, \ldots, m_n)$ and the scheme is IND-CPA secure.

At a high level, given ciphertexts $ct_1, \ldots, ct_n$ that encrypt $m_1, \ldots, m_n$, FHE should allow anyone to output a ciphertext ct^* that encrypts $f(m_1, \ldots, m_n)$ for any desired function f by evaluating another function g which is publicly computable from f. Thus, the key holder could use the secret key sk to decrypt ct^* and get the result of $f(m_1, \ldots, m_n)$.

Note that each function $f : \{0,1\}^n \to \{0,1\}^k$ can be split into $f_1, \ldots, f_k$ where $\forall i$, $f_i : \{0,1\}^n \to \{0,1\}$ and also we can generalized the definition by regulating the input length of circuits in $\mathcal{C}$ from n to $poly(n)$.

In [3], they showed how to transform any additively homomorphic SKE into PKE. Since homomorphic SKE generically implies homomorphic PKE, without loss of generality, we only talk about how to build homomorphic PKE in this lecture.

REMARK 11.6. In the Leveled homomorphic setting, it is possible to hide the function f being evaluated under a Universal function evaluator. That is if we know the bound of the number of gates and depth of the circuits being evaluated, we can use the function f as an argument to another universal function evaluator.

$$Eval(U, f, args) = f(args)$$

11.3 Construction of Fully Homomorphic Encryption:

As described previously, FHE is an encryption scheme `(KeyGen, Enc, Dec)` with an additional algorithm called `Eval`. In particular, we want to construct such a `Eval` namely for two operations, Addition, and Multiplication. Constructing such an FHE scheme which must work for **all** functions f might seem like a daunting task, but can use the following fact to ease our task.

FACT 11.7. *It turns out that all functions can be expressed by arithmetic circuits consisting of only addition and multiplication gates. Therefore, we only implement our FHE operations for addition and Multiplication. We can recursively compute every gate in the arithmetic circuit homomorphically to get the output of the function.*

DEFINITION 11.8. An arithmetic circuit over a field $\mathbb{Z}_q$ is a directed acyclic graph whose vertices are called gates. Gates of incoming degree 0 are inputs to the circuit. All other gates are labeled $+$ or x.

We usually consider arithmetic circuits with fan-in 2, in which case all of the $+$ and $\times$ gates have in-degree 2. The figure 11.2 shows such a arithematic circuit.

Let us start with the simplest possible way to build an FHE for the addition operation. For simplicity, let us consider that we want to single-bit numbers and output a single bit number. This is equivalent to implementing the XOR operation.

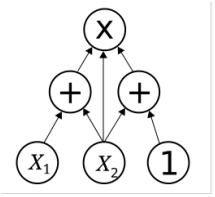

FIGURE 11.2: Arithematic circuit

11.4 FHE: Addition Operation

REMARK 11.9. It is important to know that the symbols + are being used interchangeably with $\oplus$. When we are saying addition, we XORing the messages. We maintain this notion to be consistent with the literature in FHE.

Consider an encryption of message μ_1 under the public key $(\boldsymbol{s}^T\boldsymbol{A} + \boldsymbol{e}^T, \boldsymbol{A})$. We call $\boldsymbol{s}^T\boldsymbol{A} + \boldsymbol{e}^T$ as $\boldsymbol{b}$. One intuitive naive way to implementing addition might be addition of ciphertexts.

CLAIM 11.10. *$c_1 + c_2$ is an encryption of $\mu_1 + \mu_2$*

$$c_1 = (\boldsymbol{A}\boldsymbol{r_1}, \boldsymbol{b}\boldsymbol{r_1} + \mu_1 \left\lfloor \frac{q}{2} \right\rfloor)$$

$$c_2 = (\boldsymbol{A}\boldsymbol{r_2}, \boldsymbol{b}\boldsymbol{r_2} + \mu_2 \left\lfloor \frac{q}{2} \right\rfloor)$$

$$c_{add} = c_1 + c_2 = (\boldsymbol{A}(\boldsymbol{r_1} + \boldsymbol{r_2}), \boldsymbol{b}(\boldsymbol{r_1} + \boldsymbol{r_2}) + (\mu_1 + \mu_2) \left\lfloor \frac{q}{2} \right\rfloor)$$

It is possible to extend this to multi-bit XOR outputs by simply repeating the circuit multiple times. However, it would only help in computing XOR for two k bit numbers. Let us try to decrypt the ciphertext c_{add} and check what it decrypts to:

$$\begin{aligned} Dec(sk, c_{add}) &= \boldsymbol{b}(\boldsymbol{r_1} + \boldsymbol{r_2}) + (\mu_1 + \mu_2) \left\lfloor \frac{q}{2} \right\rfloor - \boldsymbol{s}^T\boldsymbol{A}(\boldsymbol{r_1} + \boldsymbol{r_2}) \\ &= (\boldsymbol{s}^T\boldsymbol{A} + \boldsymbol{e}^T)(\boldsymbol{r_1} + \boldsymbol{r_2}) + (\mu_1 + \mu_2) \left\lfloor \frac{q}{2} \right\rfloor - \boldsymbol{s}^T\boldsymbol{A}(\boldsymbol{r_1} + \boldsymbol{r_2}) \\ &= \boldsymbol{e}^T(\boldsymbol{r_1} + \boldsymbol{r_2}) + (\mu_1 + \mu_2) \left\lfloor \frac{q}{2} \right\rfloor \end{aligned}$$

So applying the decryption algorithm we get $\mu_1 + \mu_2$ given the total error is small $|e_1 + e_2| \leq q/4$ where $e_i = \| \boldsymbol{e}^T\boldsymbol{r_i} \|$). The important observation to note here is to perform addition on two ciphertexts we need to assume hardness of LWE for stronger security parameters. Therefore, if we want to perform l addition operations, we would have to keep our $\| \boldsymbol{e}^T\boldsymbol{r_i} \|) \leq \left\lfloor \frac{q}{2} \right\rfloor / l$. The image shows the decryption algorithm works. With a very high probability, decryption of encryption of 0 would be in the blue color zone and the decryption of encryption of 1 will be in the orange zone.

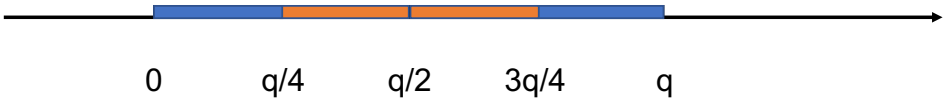

FIGURE 11.3: Decryption Error

COROLLARY 11.11. *To compute addition + (instead of XOR ⊕) of k bit numbers μ_1 and μ_2, we must modify the encryption scheme by changing the factor which is multiplied with the plaintext from $\frac{q}{2}$ to $\frac{q}{2^{k+1}}$.*

$$c_1 = (\boldsymbol{Ar_1}, \boldsymbol{br_1} + \mu_1 \left\lfloor \frac{q}{2^{k+1}} \right\rfloor)$$

$$c_2 = (\boldsymbol{Ar_2}, \boldsymbol{br_2} + \mu_2 \left\lfloor \frac{q}{2^{k+1}} \right\rfloor)$$

$$c_{add} = c_1 + c_2 = (\boldsymbol{A}(\boldsymbol{r_1} + \boldsymbol{r_2}), \boldsymbol{b}(\boldsymbol{r_1} + \boldsymbol{r_2}) + (\mu_1 + \mu_2) \left\lfloor \frac{q}{2^{k+1}} \right\rfloor)$$

On the basis of the similar argument described above, the error of the equations for 1 addition be constrained by $e_i \leq \frac{q}{2^{k+1}}$.

REMARK 11.12. It is also possible to implement a similar addition for the private key encryption scheme using LWE. That is, adding two ciphertexts c_1 and c_2 corresponding to μ_1 and μ_2 would also result in encryption of message $\mu_1 + \mu_2$.

11.5 Towards FHE multiplication:

Let us try to apply a similar logic for multiplying two ciphertexts. The components of the vectors and dimensions do not check out, but we plan to investigate what happens when we try to directly multiply the ciphertexts. By multiplication of ciphertexts, we mean component-wise multiplication of the ciphertext tuple.

$$c_1 = (\boldsymbol{Ar_1}, \boldsymbol{br_1} + \mu_1 \left\lfloor \frac{q}{2} \right\rfloor)$$

$$c_2 = (\boldsymbol{Ar_2}, \boldsymbol{br_2} + \mu_2 \left\lfloor \frac{q}{2} \right\rfloor)$$

$$c_{mult} = c_1 * c_2 = (\boldsymbol{Ar_1} * A\boldsymbol{r_2}, \boldsymbol{br_1} * \boldsymbol{br_2} + (\mu_1 * \mu_2) \left\lfloor \frac{q}{2} \right\rfloor \left\lfloor \frac{q}{2} \right\rfloor + \boldsymbol{br_1}\mu_2 \left\lfloor \frac{q}{2} \right\rfloor + \boldsymbol{br_2}\mu_1 \left\lfloor \frac{q}{2} \right\rfloor)$$

Again, we ask the reader to look over the fact that the dimensions in this system of equations do not. We are trying to investigate the possible problems to motivate to a possible solution.

PROBLEM 11.13. Applying the decryption step does not result in something clean which can be easily decrypted directly.

Intuitively, we can see that the problem is there because decrypting the first term does not get rid of the noise directly. We are going to investigate this problem deeper into the next lecture, but the high level idea is that if we naively multiply the ciphertexts and try

to decrypt them, we are not able to reason about the noise. Following is a high level idead of how to fix it, but the high-level idea is to transform a ciphertext c_2 with an operation $G^{-1}(c_2)$ before multiplying it with c_1. This transformation must ensure that the values of $G^{-1}(c_2)$ are small so that the error is contained, but also must preserve the correctness of the scheme. There our final scheme would be $c_{mult} = c_1 * G^{-1}(c_2)$.

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.

References

[1] S. Goldwasser and M. Bellare. Lecture notes on cryptography. *Summer course Cryptography and computer security at MIT*, 1999:1999, 1996.

[2] O. Regev. The learning with errors problem. *Invited survey in CCC*, 7, 2010.

[3] R. Rothblum. Homomorphic encryption: From private-key to public-key. In *Theory of cryptography conference*, pages 219–234. Springer, 2011.

[4] Wikipedia. *Fully Homomorphic Encryption.*

[5] D. J. Wu. Fully homomorphic encryption: Cryptography's holy grail. *ACM Crossroads*, 21(3):24–29, 2015.