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)[2]. Throughout the course, we have studied encryption schemes for the purpose of information transfer from other party to the other. 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. It would be great if we can outsource the computation while keeping the data encrypted. That is, it should be possible to compute a known function on encrupted data without having access to the secret key.

Such a 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 a FHE scheme achieving homomorphism. 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 an fully homomorphic encryption as of Oct 2019.*

11.1 Recap

Before stepping into how to build LWE-based FHE schemes, let's briefly recap how to build private and public encryption scheme with 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 $\mathcal{A}$

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

where q is a prime within $O(2^n)$, $m = O(n\log q)$ 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 present the secret key encryption (SKE) built with LWE which has $m = 1$:

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

LWE can also be used to build public key encryption (PKE):

- $KeyGen(1^n) : (sk = \boldsymbol{s}, pk = (\boldsymbol{A}, \boldsymbol{b}^T = \boldsymbol{s}^T\boldsymbol{A} + \boldsymbol{e}^T))$
    * $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$
    * $\boldsymbol{A} \leftarrow \mathbb{Z}_q^{n\times m}$
    * $\boldsymbol{e} \leftarrow \mathcal{X}^m$
- $Enc(pk, \mu \in \{0,1\}) : (\boldsymbol{c_1} = \boldsymbol{Ar}, c_2 = (\boldsymbol{b}^T\boldsymbol{r} + \mu\lfloor\frac{q}{2}\rfloor) \mod q)$
    * $\boldsymbol{r} \longleftarrow \{0,1\}^m$
- $Dec(sk, (\boldsymbol{c_1}, c_2)) : c_2 - \boldsymbol{s}^T\boldsymbol{c_1} = \boldsymbol{e}^T\boldsymbol{r} + \mu\lfloor\frac{q}{2}\rfloor$

11.2 Overview of Fully Homomorphic Encryption (FHE)

Let us consider the scenario shown in 11.1. A client has a secret value x. The client wants the server do some computation on x without revealing what x is. Firstly, a ciphertext $ct = Enc(x)$ is sent to the server along with the desired function f. Then the server could compute a new ciphertext $ct^* = Enc(f(x))$ by evaluating another function g on ct where g 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 $f(x)$.

A homomorphic encryption can be used for privacy-preserving outsourced storage and computation. It allows operations and analysis on encrypted data without revealing the original one, which removes the privacy barriers in several real-life applications.

In the following sections, we will cover:

- The definition of FHE
- Building additively homomorphic PKE based on LWE
- Making the additively homomorphic PKE scheme also multiplicative homomorphic

11.3 Definition of FHE

Definition 11.3. 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.

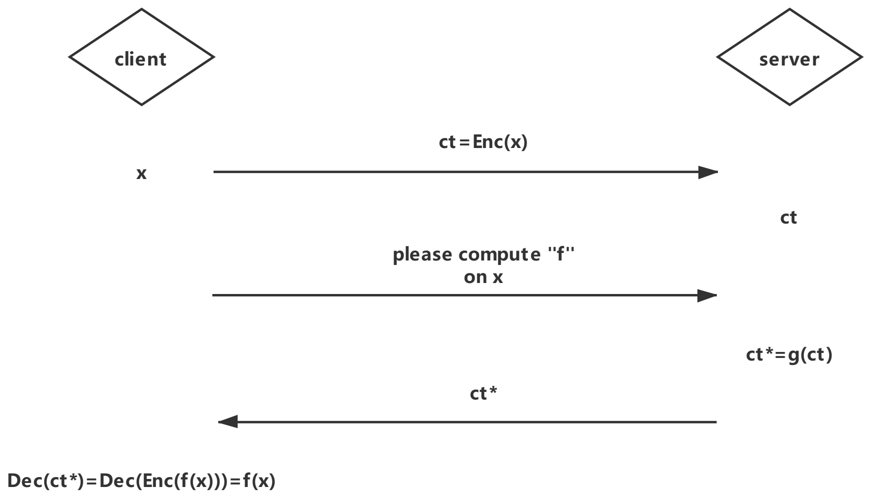

FIGURE 11.1: Outsourced Computation

At a high level, given ciphertexts $ct_1, \dots, ct_n$ that encrypt $m_1, \dots, m_n$, FHE should allow anyone to output a ciphertext ct^* that encrypts $f(m_1, \dots, m_n)$ 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 $f(m_1, \dots, m_n)$.

Note that each function $f : \{0,1\}^n \to \{0,1\}^k$ can be split into $f_1, \dots, f_k$ 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 [1], 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.

11.4 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 a FHE scheme which must work for **all** functions f might seem like daunting task, but can use the following fact to ease our task.

FACT 11.4. *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 11.5. 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 labelled + or x.

We usually consider arithmetic circuits with fan-in 2, in which case all of the + and x gates have in-degree 2.

REMARK 11.6. Even though 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 priori 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.

Let us start with the simplest possible way to build a 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.

11.5 FHE: Addition Operation

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

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

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

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

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 $max(e_i) \leq \left\lfloor \frac{q}{2} \right\rfloor / l$.

COROLLARY 11.7. *To compute addition 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 $\frac{q}{2^{k+1}}$.*

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

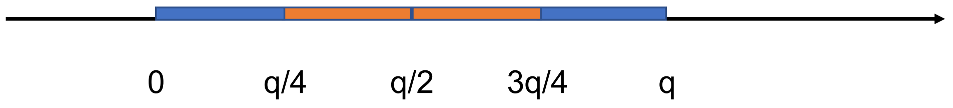

FIGURE 11.2: Decryption Error

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

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

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 11.8. 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 $\mu_1 + \mu_2$.

REMARK 11.9. A natural question which arises from the above discussion is about the similarity between XOR and addition operations.

11.6 Towards FHE multiplication

THEOREM 11.10. *Statement here*

LEMMA 11.11. *Statement here*

COROLLARY 11.12. *Statement here*

PROPOSITION 11.13. *Statement here*

FACT 11.14. *Statement here*

CLAIM 11.15. *Statement here*

DEFINITION 11.16. Statement here

EXAMPLE 11.17. Statement here

ASSUMPTION 11.18. Statement here

REMARK 11.19. Statement here

CONJECTURE 11.20. Statement here

OPEN PROBLEM 11.21. Statement here

PROBLEM 11.22. Statement here

$$a = a_1 + a_2 + \cdots + a_n. \tag{11.1}$$

For proofs, use the provided `proof` environment, illustrated below.

Proof. Proof goes here. $\square$

Acknowledgement

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

References

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

[2] Wikipedia. *Fully Homomorphic Encryption.*