ECE 443/518 – Computer Cyber Security
Lecture 23 Fully Homomorphic Encryption

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

November 5, 2025

Outline

Reading Assignment

- This lecture: Fully Homomorphic Encryption
- Next lecture: ICS 2-7,14

Outline

Fully Homomorphic Encryption

The DGHV Scheme

Limitations of Garbled Circuit

- One-time use: a garbled circuit can be evaluated only once.
  - The garbler Alice has to generate a new garbled circuit for every evaluation.
  - Otherwise the evaluator Bob will be able to learn intermediate and final bits.
- Interactive: Alice needs to interact with Bob to encrypt his input via OT.
- Not compact: Alice needs to send Bob the new garbled circuit whose size has the same complexity as the computation itself.
- Can we simply ask Alice to encrypt her data before sending it to Bob, and allow Bob to compute with it?
  - Reusable circuit, non-interactive, compact communication
  - The result should also be encrypted so Bob need help from Alice to know it

Compute with Encrypted Data

- Design a cipher (Enc and Dec) such that for any function $z = f(x, y)$, Alice and Bob can find a function F to compute $Z = F(X, y)$ together as follows:
  - Alice chooses a secret key, computes $X = Enc(key, x)$, and sends Bob only X.
  - Bob computes $Z = F(X, y)$ and sends Alice Z.
  - Alice decrypts Z as $z = Dec(key, Z)$ and it should hold that $z = f(x, y)$.
  - Bob doesn't know key so he cannot derive x and z from X and Z respectively.

Discussions

1. Alice computes $X = Enc(key, x)$ and sends Bob X.
2. Bob computes $Z = F(X, y)$ and sends Alice Z.
3. Alice computes $z = Dec(key, Z)$.

- Computation efficiency (for Alice): *Enc* and *Dec* should not depend on f
- Communication efficiency: sizes of X and Z should be linear to those of x and z respectively.
- We don't say $Z = Enc(key, z)$ in case *Enc* is probabilistic.
- *Enc* and *Dec* could use public/private key pairs.
- y could include inputs from Bob that he wants to hide, and public inputs like constants used in f.

Potential Applications

- Secure two-party computation without garbled circuit.
  - Alice encrypts her inputs and sends them to Bob.
  - Bob provides his inputs in plaintext and runs the computation.
  - Bob sends the encrypted result with Alice, who then decrypts it and shares with Bob.
- Outsourcing machine learning models and inferences.
  - Hidden model: Alice encrypts the model, Bob interacts with clients to run inferences.
  - Hidden inference: Bob has the model and helps Alice to run inferences, while Alice keeps her inputs and results secret.
  - Hidden model and inference: Alice encrypts both the model and inputs, Bob computes and returns encrypted results but knows nothing.
- And many more ...
- Only if we could design such a mechanism!

Homomorphic Encryption

- A homomorphic encryption algorithm allows certain computation to be executed on ciphertext.
  - E.g. for multiplication, $Enc(x)Enc(y) = Enc(xy)$
- RSA encryption is homomorphic for multiplication.
  - $Enc(x) = x^e \bmod p$, $Enc(y) = y^e \bmod p$
  - $Enc(xy) = (xy)^e \bmod p = x^e y^e \bmod p$
- Extend RSA to encrypt bits probabilistically.
  - $Enc(x) = (x + 2r)^e \bmod p$, $Dec(X) = X^d \bmod p \bmod 2$
    - r is a random number.
  - Now $Dec(Enc(x)Enc(y)) = xy = AND(x, y)$
    - Note that $Enc(x)Enc(y)$ could be different than $Enc(xy)$ because Enc is probabilistic.
- Homomorphic encryption for multiplication is simple.

Fully Homomorphic Encryption (FHE)

- Fully homomorphic encryption: a homomorphic encryption algorithm that support both multiplication and addition
  - Allow Bob to perform complex computations on ciphertext.
- Consider fully homomorphic encryption for bits using the same method we extend RSA.
  - $Dec(Enc(x)Enc(y)) = xy = AND(x, y)$, $Dec(Enc(x) + Enc(y)) = x + y = XOR(x, y)$
  - AND and XOR are universal for boolean circuits – Alice and Bob are able to compute with encrypted data for any function.
- While researchers quickly identified this interesting idea in 1978 after invention of RSA, it takes more than 30 years to find the first fully homomorphic encryption algorithm in 2009.

Outline

Fully Homomorphic Encryption

The DGHV Scheme

The DGHV Fully Homomorphic Encryption Scheme

- A 2010 simplification of Gentry's 2009 algorithm that works with simple integer arithmetics.
  - We'll present some basic ideas here as the whole scheme is still quite complicated.
- Consider encryption of bits
- Key p: an odd integer
- $Enc(x) = x + 2r + qp$
  - Both r and q are random numbers so Enc is probabilistic.
  - The random number r works as "noise" to prevent learning p from ciphertexts by computing their GCDs.
- $Dec(X) = X \bmod p \bmod 2$
  - We should require $2r + 1 < p$ so $X \bmod p = x + 2r$ – this is essential for Dec to decrypt correctly without knowing r or q.

DGHV Examples

- Choose $p = 13$
- With $r = 1$ and $q = 5$,
  - $Enc(0) = 0 + 2 * 1 + 5 * 13 = 67$
  - $Dec(67) = 67 \bmod 13 \bmod 2 = 2 \bmod 2 = 0$
- With $r = 4$ and $q = 6$,
  - $Enc(0) = 0 + 2 * 4 + 6 * 13 = 86$
  - $Dec(86) = 86 \bmod 13 \bmod 2 = 8 \bmod 2 = 0$
- With $r = 6$ and $q = 1$,
  - $Enc(1) = 1 + 2 * 6 + 1 * 13 = 26$
  - But $Dec(26) = 26 \bmod 13 \bmod 2 = 0 \bmod 2 = 0$
  - The "noise" r is too big so decryption doesn't work any more.
- Is DGHV homomorphic?
  - $Dec(67 + 86) = 153 \bmod 13 \bmod 2 = 10 \bmod 2 = 0$
  - $Dec(67 * 67) = 4489 \bmod 13 \bmod 2 = 4 \bmod 2 = 0$
  - But $Dec(67 * 86) = 5762 \bmod 13 \bmod 2 = 3 \bmod 2 = 1$

Somewhat Homomorphic

- Let $X = Enc(x) = x + 2r_x + q_x p$ and $Y = Enc(y) = y + 2r_y + q_y p$.
- $Dec(X + Y) = (x + y) + 2(r_x + r_y) \bmod p \bmod 2$
  - It decrypts into $x + y$ correctly if $2r_x + 2r_y$ is small enough.
- $Dec(X * Y) = xy + 2(r_x y + r_y x) + 4r_x r_y \bmod p \bmod 2$
  - It decrypts into xy correctly if $2r_x * 2r_y$ is small enough.
- Observation: a limited number of multiplications and additions can be applied on the ciphertexts.
  - Before "noises" grows too big so decryption no longer works.
  - In particular, multiplication introduces more "noises" than addition.

From Somewhat Homomorphic to Fully Homomorphic

- However, fully homomorphic encryption requries to work with arbitrary number of multiplications and additions.
- Bob needs to reduce "noises" in the ciphertexts before they grow too big in the process of computaion.
  - In order to build a fully homomorphic encryption algorithm from a somewhat homomorphic one.
- This is simple if Bob knows the key.
  - Frist decrypt the ciphertext to obtain the plaintext.
  - Then encrypt the plaintext again with a small "noise".
  - But Bob does not know the key.
- Insight: this is a problem of computing with encrypted data itself – Bob can do it!

Bootstrapping

- Alice encrypts the key with the key itself, and sends Bob the encrypted key.
  - As long as Alice doesn't change her key, this step only needs to be done once for all computations.
- To reduce the "noise" for a particular ciphertext, Bob evaluates *Dec* using the somewhat homomorphic algorithm.
  - Use the encrypted key and the ciphertext as the inputs.
  - Since *Dec* computes the plaintext from the key and the ciphertext, Bob will obtain the encrypted plaintext.
  - The encrypted plaintext is indeed a new ciphertext that decrypts to the same plaintext as the original ciphertext.
- Requirement: a proper design of *Dec*
  - Use a limited number of multiplications and additions as permitted by the somewhat homomorphic algorithm.
  - Bound the "noise" in the new ciphertext so that they do not grow too large before Bob applies the same reduction again.

DGHV Bootstrapping

- Let $f(X, p)$ be a circuit computing X mod p mod 2.
  - Inputs are bits: $X = (X_n \cdots X_2 X_1)_2$ and $p = (p_m \dots p_2 p_1)_2$.
  - Use *AND* and *XOR* gates and the constant bit 1.
- Alice encrypts each bit of p by itself: $Enc(p_1), \dots, Enc(p_m)$, and sends them to Bob.
- For simplicity, we ask Bob to also encrypt the ciphertext bits and the constants in the circuit.
  - Bob can use 1 for $Enc(1)$ and 0 for $Enc(0)$ without knowing p.
  - Bob computes $Enc(X_1), \dots, Enc(X_m)$ without Alice's help.
- Bob evaluates f with encrypted bits of ciphertext and key $Enc(X_1), \dots, Enc(X_m), Enc(p_1), \dots, Enc(p_m)$
  - Replace *AND* with multiplication and *XOR* with addition.
  - Since $x = f(X, p)$, with encrypted X and p Bob should obtain encrypted x, which is another ciphertext X' of x.
  - The "noises" in X' depend on f and its inputs – independent of the "noises" in X.

DGHV Bootstrapping Implementation

- $Enc(X_1), \ldots, Enc(X_m)$ has a "noise" of 0.
  - Since Bob use 1 for $Enc(1)$ and 0 for $Enc(0)$.
- Alice controls the "noises" in $Enc(p_1), \ldots, Enc(p_m)$.
- We need to limit the number of AND and XOR gates in f.
  - So that the "noises" do not increase beyond what plain DGHV can handle.
- Ideas to implement f to meet the need.
  - Ask Alice to encrypt an approximation of $\frac{1}{p}$ (as a fixed point number) so that $\bmod p$ can be replaced by multiplication and subtraction.
  - Ask Alice to provide additional (encrypted) data to further reduce the complexity of multiplication that would require a lot of AND gates.
- We will stop here and let you explore further by yourself.

Summary

- Fully homomorphic encryption: compute with encrypted data
  - Only owner of the encrypted data can decrypt the results.