Fully Homomorphic Encryption (FHE) - Part II

Bootstrapping

Chuanqi Zhang

The Motivation: The "Noise Budget"

So far, we have seen a system that can add and multiply encrypted data. But there is a catch...

The Motivation: The "Noise Budget"

So far, we have seen a system that can add and multiply encrypted data. But there is a catch...

- Every ciphertext has a "Noise Budget".

The Motivation: The "Noise Budget"

So far, we have seen a system that can add and multiply encrypted data. But there is a catch...

- Every ciphertext has a "Noise Budget".
- **Addition** increases the noise slowly (linear growth).

The Motivation: The "Noise Budget"

So far, we have seen a system that can add and multiply encrypted data. But there is a catch...

- Every ciphertext has a "Noise Budget".
- **Addition** increases the noise slowly (linear growth).
- **Multiplication** increases the noise more rapidly (even with [GSW13]'s Flatten fix, it could grow by a factor of the ciphertext matrix size in the worst case).

The Motivation: The "Noise Budget"

So far, we have seen a system that can add and multiply encrypted data. But there is a catch...

- Every ciphertext has a "Noise Budget".
- **Addition** increases the noise slowly (linear growth).
- **Multiplication** increases the noise more rapidly (even with [GSW13]'s Flatten fix, it could grow by a factor of the ciphertext matrix size in the worst case).
- **The Limit:** If you perform too many operations, the accumulated noise overtakes the message. Decryption rounding will fail!

The Motivation: The "Noise Budget"

So far, we have seen a system that can add and multiply encrypted data. But there is a catch...

- Every ciphertext has a "Noise Budget".
- **Addition** increases the noise slowly (linear growth).
- **Multiplication** increases the noise more rapidly (even with [GSW13]'s Flatten fix, it could grow by a factor of the ciphertext matrix size in the worst case).
- **The Limit:** If you perform too many operations, the accumulated noise overtakes the message. Decryption rounding will fail!

Somewhat Homomorphic Encryption (SWHE)

A scheme that can only perform a **limited** number of operations before the noise overflows is called Somewhat Homomorphic Encryption.

The Motivation: Refresh the Noise

To achieve **Fully** Homomorphic Encryption (for infinite operations), we need a way to "clean" the ciphertext by "refreshing" the ciphertext.

The Motivation: Refresh the Noise

To achieve **Fully** Homomorphic Encryption (for infinite operations), we need a way to "clean" the ciphertext by "refreshing" the ciphertext.

The Dilemma:

The Motivation: Refresh the Noise

To achieve **Fully** Homomorphic Encryption (for infinite operations), we need a way to "clean" the ciphertext by "refreshing" the ciphertext.

The Dilemma:

- Only the secret key can remove the noise (by running the decryption algorithm).

The Motivation: Refresh the Noise

To achieve **Fully** Homomorphic Encryption (for infinite operations), we need a way to "clean" the ciphertext by "refreshing" the ciphertext.

The Dilemma:

- Only the secret key can remove the noise (by running the decryption algorithm).
- But the Cloud/Server is doing the computing. We CANNOT give them our secret key!

The Motivation: Refresh the Noise

To achieve **Fully** Homomorphic Encryption (for infinite operations), we need a way to "clean" the ciphertext by "refreshing" the ciphertext.

The Dilemma:

- Only the secret key can remove the noise (by running the decryption algorithm).
- But the Cloud/Server is doing the computing. We CANNOT give them our secret key!

How can the Cloud/Server clean the data without ever seeing the plaintext message?

Craig Gentry's Breakthrough: Bootstrapping

Gentry's breakthrough idea: **Run the decryption algorithm homomorphically!**

Craig Gentry's Breakthrough: Bootstrapping

Gentry's breakthrough idea: **Run the decryption algorithm homomorphically!**

- We know FHE can compute *any* mathematical function on encrypted inputs.

Craig Gentry's Breakthrough: Bootstrapping

Gentry's breakthrough idea: **Run the decryption algorithm homomorphically!**

- We know FHE can compute *any* mathematical function on encrypted inputs.
- **The Trick:** The decryption algorithm is just a mathematical function!

Craig Gentry's Breakthrough: Bootstrapping

Gentry's breakthrough idea: **Run the decryption algorithm homomorphically!**

- We know FHE can compute *any* mathematical function on encrypted inputs.
- **The Trick:** The decryption algorithm is just a mathematical function!
- **Step 1:** The User encrypts their own secret key and sends this encrypted key to the Cloud.

Craig Gentry's Breakthrough: Bootstrapping

Gentry's breakthrough idea: **Run the decryption algorithm homomorphically!**

- We know FHE can compute *any* mathematical function on encrypted inputs.
- **The Trick:** The decryption algorithm is just a mathematical function!
- **Step 1:** The User encrypts their own secret key and sends this encrypted key to the Cloud.
- **Step 2:** The Cloud hardcodes the "noisy" ciphertext to define a specific decryption function.

Craig Gentry's Breakthrough: Bootstrapping

Gentry's breakthrough idea: **Run the decryption algorithm homomorphically!**

- We know FHE can compute *any* mathematical function on encrypted inputs.
- **The Trick:** The decryption algorithm is just a mathematical function!
- **Step 1:** The User encrypts their own secret key and sends this encrypted key to the Cloud.
- **Step 2:** The Cloud hardcodes the "noisy" ciphertext to define a specific decryption function.
- **Step 3:** The Cloud homomorphically evaluates this decryption function using the encrypted key as the input.

Craig Gentry's Breakthrough: Bootstrapping

Gentry's breakthrough idea: **Run the decryption algorithm homomorphically!**

- We know FHE can compute *any* mathematical function on encrypted inputs.
- **The Trick:** The decryption algorithm is just a mathematical function!
- **Step 1:** The User encrypts their own secret key and sends this encrypted key to the Cloud.
- **Step 2:** The Cloud hardcodes the "noisy" ciphertext to define a specific decryption function.
- **Step 3:** The Cloud homomorphically evaluates this decryption function using the encrypted key as the input.
- **Result:** A brand new ciphertext. The old noise is destroyed by the decryption logic, and replaced by the fresh noise of evaluating the function!

The Formulation in Mathematics

Let's formalize this mathematically.

The Formulation in Mathematics

Let's formalize this mathematically.

- Let C_{noisy} be our noisy ciphertext encrypting message μ.

The Formulation in Mathematics

Let's formalize this mathematically.

- Let C_{noisy} be our noisy ciphertext encrypting message μ.
- We define a fixed mathematical function F based on C_{noisy}:

The Formulation in Mathematics

Let's formalize this mathematically.

- Let C_{noisy} be our noisy ciphertext encrypting message μ.
- We define a fixed mathematical function F based on C_{noisy}:

$$F(x) = Decrypt(x, C_{noisy})$$

The Formulation in Mathematics

Let's formalize this mathematically.

- Let C_{noisy} be our noisy ciphertext encrypting message μ.
- We define a fixed mathematical function F based on C_{noisy}:

$$F(x) = Decrypt(x, C_{noisy})$$

- Note that if we plug the real secret key sk into F, we get the message:

$$F(sk) = \mu.$$

The Formulation in Mathematics

Let's formalize this mathematically.

- Let C_{noisy} be our noisy ciphertext encrypting message μ.
- We define a fixed mathematical function F based on C_{noisy}:

$$F(x) = Decrypt(x, C_{noisy})$$

- Note that if we plug the real secret key sk into F, we get the message:

$$F(sk) = \mu.$$

- Now, we use the homomorphic evaluation algorithm ($Eval$). We pass it our function F, and the encrypted input $Enc(sk)$:

The Formulation in Mathematics

Let's formalize this mathematically.

- Let C_{noisy} be our noisy ciphertext encrypting message μ.
- We define a fixed mathematical function F based on C_{noisy}:

$$F(x) = Decrypt(x, C_{noisy})$$

- Note that if we plug the real secret key sk into F, we get the message:

$$F(sk) = \mu.$$

- Now, we use the homomorphic evaluation algorithm ($Eval$). We pass it our function F, and the encrypted input $Enc(sk)$:

$$Eval(F, Enc(sk)) \implies Enc(F(sk))$$

The Formulation in Mathematics

Let's formalize this mathematically.

- Let C_{noisy} be our noisy ciphertext encrypting message μ.
- We define a fixed mathematical function F based on C_{noisy}:

$$F(x) = Decrypt(x, C_{noisy})$$

- Note that if we plug the real secret key sk into F, we get the message:

$$F(sk) = \mu.$$

- Now, we use the homomorphic evaluation algorithm ($Eval$). We pass it our function F, and the encrypted input $Enc(sk)$:

$$Eval(F, Enc(sk)) \implies Enc(F(sk)) = Enc(\mu)$$

The Formulation in Mathematics

Let's formalize this mathematically.

- Let C_{noisy} be our noisy ciphertext encrypting message μ.
- We define a fixed mathematical function F based on C_{noisy}:

$$F(x) = Decrypt(x, C_{noisy})$$

- Note that if we plug the real secret key sk into F, we get the message:

$$F(sk) = \mu.$$

- Now, we use the homomorphic evaluation algorithm ($Eval$). We pass it our function F, and the encrypted input $Enc(sk)$:

$$Eval(F, Enc(sk)) \implies Enc(F(sk)) = Enc(\mu) = C_{fresh}$$

How does Bootstrapping Work in [GSW13]?

Let's see how we can evaluate the function $F(x) = Decrypt(x, C_{noisy})$ in [GSW13]'s FHE scheme.

How does Bootstrapping Work in [GSW13]?

Let's see how we can evaluate the function $F(x) = Decrypt(x, C_{noisy})$ in [GSW13]'s FHE scheme.

- Recall how we use the secret key $\mathbf{v}$ to decrypt a row $\mathbf{c}_i$ of the ciphertext matrix in [GSW13]:

How does Bootstrapping Work in [GSW13]?

Let's see how we can evaluate the function $F(x) = Decrypt(x, C_{noisy})$ in [GSW13]'s FHE scheme.

- Recall how we use the secret key $\mathbf{v}$ to decrypt a row $\mathbf{c}_i$ of the ciphertext matrix in [GSW13]:

$$\mu = \lfloor \frac{\langle \mathbf{c}_i, \mathbf{v} \rangle}{v_i} \rceil = \lfloor \frac{c_{i,1}v_1 + c_{i,2}v_2 + \cdots + c_{i,n}v_n}{v_i} \rceil,$$

  where $c_{i,j}$ and v_j are the jth component of $\mathbf{c}_i$ and $\mathbf{v}$, respectively.

How does Bootstrapping Work in [GSW13]?

Let's see how we can evaluate the function $F(x) = Decrypt(x, C_{noisy})$ in [GSW13]'s FHE scheme.

- Recall how we use the secret key $\mathbf{v}$ to decrypt a row $\mathbf{c}_i$ of the ciphertext matrix in [GSW13]:

$$\mu = \lfloor \frac{\langle \mathbf{c}_i, \mathbf{v} \rangle}{v_i} \rceil = \lfloor \frac{c_{i,1}v_1 + c_{i,2}v_2 + \cdots + c_{i,n}v_n}{v_i} \rceil,$$

  where $c_{i,j}$ and v_j are the jth component of $\mathbf{c}_i$ and $\mathbf{v}$, respectively.
- To run this homomorphically, the user provides the encrypted secret key: $Enc(v_1), Enc(v_2), \ldots, Enc(v_n)$.

How does Bootstrapping Work in [GSW13]?

Let's see how we can evaluate the function $F(x) = Decrypt(x, C_{noisy})$ in [GSW13]'s FHE scheme.

- Recall how we use the secret key $\mathbf{v}$ to decrypt a row $\mathbf{c}_i$ of the ciphertext matrix in [GSW13]:

$$\mu = \lfloor \frac{\langle \mathbf{c}_i, \mathbf{v} \rangle}{v_i} \rceil = \lfloor \frac{c_{i,1}v_1 + c_{i,2}v_2 + \cdots + c_{i,n}v_n}{v_i} \rceil,$$

  where $c_{i,j}$ and v_j are the jth component of $\mathbf{c}_i$ and $\mathbf{v}$, respectively.
- To run this homomorphically, the user provides the encrypted secret key: $Enc(v_1), Enc(v_2), \ldots, Enc(v_n)$.
- The values $c_{i,j}$ from the noisy ciphertext are just known public constants.

How does Bootstrapping Work in [GSW13]?

Let's see how we can evaluate the function $F(x) = Decrypt(x, C_{noisy})$ in [GSW13]'s FHE scheme.

- Recall how we use the secret key $\mathbf{v}$ to decrypt a row $\mathbf{c}_i$ of the ciphertext matrix in [GSW13]:

$$\mu = \lfloor \frac{\langle \mathbf{c}_i, \mathbf{v} \rangle}{v_i} \rceil = \lfloor \frac{c_{i,1}v_1 + c_{i,2}v_2 + \cdots + c_{i,n}v_n}{v_i} \rceil,$$

  where $c_{i,j}$ and v_j are the jth component of $\mathbf{c}_i$ and $\mathbf{v}$, respectively.
- To run this homomorphically, the user provides the encrypted secret key: $Enc(v_1), Enc(v_2), \ldots, Enc(v_n)$.
- The values $c_{i,j}$ from the noisy ciphertext are just known public constants.
- **Our Goal:** To run this exact decryption procedure homomorphically, using the encrypted secret key!

Bootstrapping in [GSW13]: Inner Product

Let's homomorphically compute the inner product part:

Bootstrapping in [GSW13]: Inner Product

Let's homomorphically compute the inner product part:

$$\langle \mathbf{c}_i, \mathbf{v} \rangle = c_{i,1} v_1 + c_{i,2} v_2 + \cdots + c_{i,n} v_n.$$

Bootstrapping in [GSW13]: Inner Product

Let's homomorphically compute the inner product part:

$$\langle \mathbf{c}_i, \mathbf{v} \rangle = c_{i,1} v_1 + c_{i,2} v_2 + \cdots + c_{i,n} v_n.$$

- **The Setup:** Instead of raw bits v_j, the homomorphic evaluator only has access to each encrypted key bit $Enc(v_j)$.

Bootstrapping in [GSW13]: Inner Product

Let's homomorphically compute the inner product part:

$$\langle \mathbf{c}_i, \mathbf{v} \rangle = c_{i,1} v_1 + c_{i,2} v_2 + \cdots + c_{i,n} v_n.$$

- **The Setup:** Instead of raw bits v_j, the homomorphic evaluator only has access to each encrypted key bit $Enc(v_j)$.
- **Scalar Multiplication:** Since each $c_{i,j}$ is just a known public constant (from C_{noisy}), we can multiply it with the corresponding encrypted key bit:

Bootstrapping in [GSW13]: Inner Product

Let's homomorphically compute the inner product part:

$$\langle \mathbf{c}_i, \mathbf{v} \rangle = c_{i,1} v_1 + c_{i,2} v_2 + \cdots + c_{i,n} v_n.$$

- **The Setup:** Instead of raw bits v_j, the homomorphic evaluator only has access to each encrypted key bit $Enc(v_j)$.
- **Scalar Multiplication:** Since each $c_{i,j}$ is just a known public constant (from C_{noisy}), we can multiply it with the corresponding encrypted key bit:

$$c_{i,j} \cdot Enc(v_j) \implies Enc(c_{i,j} \cdot v_j)$$

Bootstrapping in [GSW13]: Inner Product

Let's homomorphically compute the inner product part:

$$\langle \mathbf{c}_i, \mathbf{v} \rangle = c_{i,1} v_1 + c_{i,2} v_2 + \cdots + c_{i,n} v_n.$$

- **The Setup:** Instead of raw bits v_j, the homomorphic evaluator only has access to each encrypted key bit $Enc(v_j)$.
- **Scalar Multiplication:** Since each $c_{i,j}$ is just a known public constant (from C_{noisy}), we can multiply it with the corresponding encrypted key bit:

$$c_{i,j} \cdot Enc(v_j) \implies Enc(c_{i,j} \cdot v_j)$$

- **Homomorphic Addition:** Next, we sum all these newly generated ciphertexts together using homomorphic addition ($\oplus$):

Bootstrapping in [GSW13]: Inner Product

Let's homomorphically compute the inner product part:

$$\langle \mathbf{c}_i, \mathbf{v} \rangle = c_{i,1} v_1 + c_{i,2} v_2 + \cdots + c_{i,n} v_n.$$

- **The Setup:** Instead of raw bits v_j, the homomorphic evaluator only has access to each encrypted key bit $Enc(v_j)$.
- **Scalar Multiplication:** Since each $c_{i,j}$ is just a known public constant (from C_{noisy}), we can multiply it with the corresponding encrypted key bit:

$$c_{i,j} \cdot Enc(v_j) \implies Enc(c_{i,j} \cdot v_j)$$

- **Homomorphic Addition:** Next, we sum all these newly generated ciphertexts together using homomorphic addition ($\oplus$):

$$Enc(c_{i,1} \cdot v_1) \oplus Enc(c_{i,2} \cdot v_2) \oplus \cdots \oplus Enc(c_{i,n} \cdot v_n)$$

Bootstrapping in [GSW13]: Inner Product

Let's homomorphically compute the inner product part:

$$\langle \mathbf{c}_i, \mathbf{v} \rangle = c_{i,1} v_1 + c_{i,2} v_2 + \cdots + c_{i,n} v_n.$$

- **The Setup:** Instead of raw bits v_j, the homomorphic evaluator only has access to each encrypted key bit $Enc(v_j)$.
- **Scalar Multiplication:** Since each $c_{i,j}$ is just a known public constant (from C_{noisy}), we can multiply it with the corresponding encrypted key bit:

$$c_{i,j} \cdot Enc(v_j) \implies Enc(c_{i,j} \cdot v_j)$$

- **Homomorphic Addition:** Next, we sum all these newly generated ciphertexts together using homomorphic addition ($\oplus$):

$$Enc(c_{i,1} \cdot v_1) \oplus Enc(c_{i,2} \cdot v_2) \oplus \cdots \oplus Enc(c_{i,n} \cdot v_n) \implies Enc(\langle \mathbf{c}_i, \mathbf{v} \rangle)$$

Bootstrapping in [GSW13]: Rounding

Another part of the decryption function F is to divide by v_i and round to integer μ.

Bootstrapping in [GSW13]: Rounding

Another part of the decryption function F is to divide by v_i and round to integer μ.

- Division and rounding are mathematically trickier to evaluate homomorphically than basic addition and multiplication.

Bootstrapping in [GSW13]: Rounding

Another part of the decryption function F is to divide by v_i and round to integer μ.

- Division and rounding are mathematically trickier to evaluate homomorphically than basic addition and multiplication.
- However, because everything is operating in a finite modulo space ($\mathbb{Z}_q$) and v_i is a known power of 2, this rounding operation can be mathematically represented as a polynomial function.

Bootstrapping in [GSW13]: Rounding

Another part of the decryption function F is to divide by v_i and round to integer μ.

- Division and rounding are mathematically trickier to evaluate homomorphically than basic addition and multiplication.
- However, because everything is operating in a finite modulo space ($\mathbb{Z}_q$) and v_i is a known power of 2, this rounding operation can be mathematically represented as a polynomial function.
- Since we possess homomorphic addition and homomorphic multiplication, we can evaluate *any* polynomial!

Bootstrapping in [GSW13]: Rounding

Another part of the decryption function F is to divide by v_i and round to integer μ.

- Division and rounding are mathematically trickier to evaluate homomorphically than basic addition and multiplication.
- However, because everything is operating in a finite modulo space ($\mathbb{Z}_q$) and v_i is a known power of 2, this rounding operation can be mathematically represented as a polynomial function.
- Since we possess homomorphic addition and homomorphic multiplication, we can evaluate *any* polynomial!
- Therefore, we can homomorphically compute the rounding part to get our final, fresh $Enc(\mu)$.

The Ultimate Requirement

There is one final, critical catch for bootstrapping to work:

The Ultimate Requirement

There is one final, critical catch for bootstrapping to work:

The scheme must be able to evaluate its own decryption function!

The Ultimate Requirement

There is one final, critical catch for bootstrapping to work:

The scheme must be able to evaluate its own decryption function!

- Homomorphically computing the inner product and rounding takes a certain number of multiplications (say D multiplications).

The Ultimate Requirement

There is one final, critical catch for bootstrapping to work:

The scheme must be able to evaluate its own decryption function!

- Homomorphically computing the inner product and rounding takes a certain number of multiplications (say D multiplications).
- If the scheme's noise overflows *before* it reaches D multiplications, it cannot bootstrap itself!

The Ultimate Requirement

There is one final, critical catch for bootstrapping to work:

The scheme must be able to evaluate its own decryption function!

- Homomorphically computing the inner product and rounding takes a certain number of multiplications (say D multiplications).
- If the scheme's noise overflows *before* it reaches D multiplications, it cannot bootstrap itself!
- [GSW13]'s Elegance: By adjusting the parameters, [GSW13]'s scheme can easily be configured to support enough multiplications to evaluate its own decryption logic.

The Ultimate Requirement

There is one final, critical catch for bootstrapping to work:

The scheme must be able to evaluate its own decryption function!

- Homomorphically computing the inner product and rounding takes a certain number of multiplications (say D multiplications).
- If the scheme's noise overflows *before* it reaches D multiplications, it cannot bootstrap itself!
- [GSW13]'s Elegance: By adjusting the parameters, [GSW13]'s scheme can easily be configured to support enough multiplications to evaluate its own decryption logic.
- Once bootstrapping finishes, the noise resets to a fixed baseline.
  So we achieve Fully Homomorphic Encryption!

Summary