Fully Homomorphic Encryption (FHE) - Part II

From Theory to Practice with OpenFHE

Chuanqi Zhang

Why Do We Need Software Libraries?

We've been looking at the math behind FHE. Can we make a real FHE in a simple Python script?

Why Do We Need Software Libraries?

We've been looking at the math behind FHE. Can we make a real FHE in a simple Python script?

- **The Expert Gap:** Most developers are not cryptographers. They need a "Black Box" where they put data in and get results out.

Why Do We Need Software Libraries?

We've been looking at the math behind FHE. Can we make a real FHE in a simple Python script?

- **The Expert Gap:** Most developers are not cryptographers. They need a "Black Box" where they put data in and get results out.
- **OpenFHE** acts as this bridge. It is a professional C++ library that translates high-level logic into low-level Lattice math.

Why Do We Need Software Libraries?

We've been looking at the math behind FHE. Can we make a real FHE in a simple Python script?

- **The Expert Gap:** Most developers are not cryptographers. They need a "Black Box" where they put data in and get results out.
- **OpenFHE** acts as this bridge. It is a professional C++ library that translates high-level logic into low-level Lattice math.

OpenFHE allows you to focus on the application, not the equations. ☺

Which FHE "Flavor" Should You Use?

In the real world, there is no single "best" FHE. You choose based on your data type. Think of it as choosing a vehicle:

Which FHE "Flavor" Should You Use?

In the real world, there is no single "best" FHE. You choose based on your data type. Think of it as choosing a vehicle:

BFV / BGV

The Calculator

For Integers.
Perfect for bank balances or counting votes. 100% exact.

Which FHE "Flavor" Should You Use?

In the real world, there is no single "best" FHE. You choose based on your data type. Think of it as choosing a vehicle:

BFV / BGV

The Calculator

For Integers.
Perfect for bank balances or counting votes. 100% exact.

CKKS

The AI Engine

For Decimals.
Approximate results. Perfect for Machine Learning and Statistics.

Which FHE "Flavor" Should You Use?

In the real world, there is no single "best" FHE. You choose based on your data type. Think of it as choosing a vehicle:

BFV / BGV

The Calculator

For Integers.
Perfect for bank balances or counting votes. 100% exact.

CKKS

The AI Engine

For Decimals.
Approximate results. Perfect for Machine Learning and Statistics.

TFHE / FHEW

The Circuit

For Booleans.
Works on bits (0 or 1). Best for logic gates (AND, OR, NOT).

Deep Dive: Why CKKS for AI?

Most modern FHE research centers on CKKS. Why?

Deep Dive: Why CKKS for AI?

Most modern FHE research centers on CKKS. Why?

- **Floating Point Support:** AI models rely heavily on continuous weights like 0.7512. BFV/BGV struggle with these; CKKS handles approximate arithmetic natively.

Deep Dive: Why CKKS for AI?

Most modern FHE research centers on CKKS. Why?

- **Floating Point Support:** AI models rely heavily on continuous weights like 0.7512. BFV/BGV struggle with these; CKKS handles approximate arithmetic natively.
- **Continuous SIMD (Batching):** While BFV/BGV can batch integers, CKKS uses a different mathematical structure to pack vectors of real/complex numbers directly into a single ciphertext.

Deep Dive: Why CKKS for AI?

Most modern FHE research centers on CKKS. Why?

- **Floating Point Support:** AI models rely heavily on continuous weights like 0.7512. BFV/BGV struggle with these; CKKS handles approximate arithmetic natively.
- **Continuous SIMD (Batching):** While BFV/BGV can batch integers, CKKS uses a different mathematical structure to pack vectors of real/complex numbers directly into a single ciphertext.
- **Perfect for Tensors:** One homomorphic operation performs the same math on the entire vector simultaneously—exactly how Neural Networks process data!

Deep Dive: Why CKKS for AI?

Most modern FHE research centers on CKKS. Why?

- **Floating Point Support:** AI models rely heavily on continuous weights like 0.7512. BFV/BGV struggle with these; CKKS handles approximate arithmetic natively.
- **Continuous SIMD (Batching):** While BFV/BGV can batch integers, CKKS uses a different mathematical structure to pack vectors of real/complex numbers directly into a single ciphertext.
- **Perfect for Tensors:** One homomorphic operation performs the same math on the entire vector simultaneously—exactly how Neural Networks process data!

The Trade-off

CKKS replaces exact decryption with approximate results. This introduces slight precision loss, which is unacceptable for strict accounting, but perfectly fine for AI models.

OpenFHE Sample Step 1 & 2: The Scheme and The Keys

FHE is not infinite magic. Every multiplication adds noise.

OpenFHE Sample Step 1 & 2: The Scheme and The Keys

FHE is not infinite magic. Every multiplication adds noise.
Level-based HE: A type of HE where the *multiplicative depth* is fixed in advance, i.e., how many multiplications we plan to do.

OpenFHE Sample Step 1 & 2: The Scheme and The Keys

FHE is not infinite magic. Every multiplication adds noise.
Level-based HE: A type of HE where the *multiplicative depth* is fixed in advance, i.e., how many multiplications we plan to do.

Step 1: CryptoContext.

- Defines the scheme (e.g., CKKS) and specifies the multiplicative depth (e.g., 5).

OpenFHE C++ API (Sketch)

```
// Step 1
CCParams<CryptoContextCKKSRNS> par;
par.SetMultiplicativeDepth(5);
auto cc = GenCryptoContext(par);
cc->Enable(PKE);
cc->Enable(ENCRYPTION);
cc->Enable(LEVELEDSHE);
```

OpenFHE Sample Step 1 & 2: The Scheme and The Keys

FHE is not infinite magic. Every multiplication adds noise.
Level-based HE: A type of HE where the *multiplicative depth* is fixed in advance, i.e., how many multiplications we plan to do.

Step 1: CryptoContext.

- Defines the scheme (e.g., CKKS) and specifies the multiplicative depth (e.g., 5).

Step 2: Key Generation.

- `keys.publicKey`: To encrypt.
- `keys.secretKey`: To decrypt.

OpenFHE C++ API (Sketch)

```
// Step 1
CCParams<CryptoContextCKKSRNS> par;
par.SetMultiplicativeDepth(5);
auto cc = GenCryptoContext(par);
cc->Enable(PKE);
cc->Enable(ENCRYPTION);
cc->Enable(LEVELEDSHE);

// Step 2
auto keys = cc->KeyGen();
```

OpenFHE Sample Step 3, 4, 5: Encrypt, Evaluate, Decrypt

Step 3: Encode & Encrypt.

- **Encode:** Packs a vector of messages (e.g., x = [1.5 2.0]) into a plaintext.

OpenFHE C++ API (Sketch)

```
// Step 3
vector<double> x = {1.5, 2.0};
auto ptxt =
cc->MakeCKKSPackedPlaintext(x);
auto ctxt =
cc->Encrypt(keys.publicKey, ptxt);
```

OpenFHE Sample Step 3, 4, 5: Encrypt, Evaluate, Decrypt

Step 3: Encode & Encrypt.

- **Encode:** Packs a vector of messages (e.g., x = [1.5 2.0]) into a plaintext.
- **Encrypt:** Use `keys.publicKey` to encrypt the plaintext into a ciphertext.

OpenFHE C++ API (Sketch)

```
// Step 3
vector<double> x = {1.5, 2.0};
auto ptxt =
cc->MakeCKKSPackedPlaintext(x);
auto ctxt =
cc->Encrypt(keys.publicKey, ptxt);
```

OpenFHE Sample Step 3, 4, 5: Encrypt, Evaluate, Decrypt

Step 3: Encode & Encrypt.

- **Encode:** Packs a vector of messages (e.g., x = [1.5 2.0]) into a plaintext.
- **Encrypt:** Use `keys.publicKey` to encrypt the plaintext into a ciphertext.

Step 4: Evaluate a Function. The Cloud chains operations to evaluate a target function, e.g., $f(x) = x^2 + x$.

OpenFHE C++ API (Sketch)

```
// Step 3
vector<double> x = {1.5, 2.0};
auto ptxt =
cc->MakeCKKSPackedPlaintext(x);
auto ctxt =
cc->Encrypt(keys.publicKey, ptxt);

// Step 4
auto ctSquare = cc->EvalMult(ctxt,
ctxt); // x^2
auto ctRes = cc->EvalAdd(ctSquare,
ctxt); // x^2 + x
```

OpenFHE Sample Step 3, 4, 5: Encrypt, Evaluate, Decrypt

Step 3: Encode & Encrypt.

- **Encode:** Packs a vector of messages (e.g., x = [1.5 2.0]) into a plaintext.
- **Encrypt:** Use `keys.publicKey` to encrypt the plaintext into a ciphertext.

Step 4: Evaluate a Function. The Cloud chains operations to evaluate a target function, e.g., $f(x) = x^2 + x$.

Step 5: Decrypt & Decode. Use `keys.secretKey` to decrypt and recover the computation result.

OpenFHE C++ API (Sketch)

```
// Step 3
vector<double> x = {1.5, 2.0};
auto ptxt =
cc->MakeCKKSPackedPlaintext(x);
auto ctxt =
cc->Encrypt(keys.publicKey, ptxt);

// Step 4
auto ctSquare = cc->EvalMult(ctxt,
ctxt); // x^2
auto ctRes = cc->EvalAdd(ctSquare,
ctxt); // x^2 + x

// Step 5
Plaintext ptxtRes;
cc->Decrypt(keys.secretKey, ctRes,
&ptxtRes);
auto result =
ptxtRes->GetRealPackedValue();
// [3.75, 6.0]
```

Case Study: Privacy-Preserving Machine Learning (PPML)

A Real-World Problem:

Case Study: Privacy-Preserving Machine Learning (PPML)

A Real-World Problem:

- A top hospital has developed an amazing AI model that detects cancer from DNA data.

Case Study: Privacy-Preserving Machine Learning (PPML)

A Real-World Problem:

- A top hospital has developed an amazing AI model that detects cancer from DNA data.
- You want to use the AI, but you refuse to send your raw DNA to a third-party server.

Case Study: Privacy-Preserving Machine Learning (PPML)

A Real-World Problem:

- A top hospital has developed an amazing AI model that detects cancer from DNA data.
- You want to use the AI, but you refuse to send your raw DNA to a third-party server.
- The hospital refuses to send their AI model to your laptop.

Case Study: Privacy-Preserving Machine Learning (PPML)

A Real-World Problem:

- A top hospital has developed an amazing AI model that detects cancer from DNA data.
- You want to use the AI, but you refuse to send your raw DNA to a third-party server.
- The hospital refuses to send their AI model to your laptop.

The FHE Solution (MLaaS: ML as a Service)

You encrypt your DNA. You send the encrypted DNA to the hospital. The hospital runs their AI on your encrypted DNA. They send back an encrypted diagnosis. **Only you can decrypt it.**

How does AI work inside FHE? (Part 1: Linear Layers)

Can an AI Neural Network run on encrypted data?

How does AI work inside FHE? (Part 1: Linear Layers)

Can an AI Neural Network run on encrypted data?

- What is a Neural Network layer, fundamentally?

How does AI work inside FHE? (Part 1: Linear Layers)

Can an AI Neural Network run on encrypted data?

- What is a Neural Network layer, fundamentally?
- It is just a **Dot Product** (Weighted Sum):

$$y = (w_1 \cdot x_1) + (w_2 \cdot x_2) + \cdots + b$$

How does AI work inside FHE? (Part 1: Linear Layers)

Can an AI Neural Network run on encrypted data?

- What is a Neural Network layer, fundamentally?
- It is just a **Dot Product** (Weighted Sum):

$$y = (w_1 \cdot x_1) + (w_2 \cdot x_2) + \cdots + b$$

- w are the model weights (known to hospital).

How does AI work inside FHE? (Part 1: Linear Layers)

Can an AI Neural Network run on encrypted data?

- What is a Neural Network layer, fundamentally?
- It is just a **Dot Product** (Weighted Sum):

$$y = (w_1 \cdot x_1) + (w_2 \cdot x_2) + \cdots + b$$

- w are the model weights (known to hospital).
- x are the inputs (encrypted by patient).

How does AI work inside FHE? (Part 1: Linear Layers)

Can an AI Neural Network run on encrypted data?

- What is a Neural Network layer, fundamentally?
- It is just a **Dot Product** (Weighted Sum):

$$y = (w_1 \cdot x_1) + (w_2 \cdot x_2) + \cdots + b$$

- w are the model weights (known to hospital).
- x are the inputs (encrypted by patient).
- **The easier side:** FHE is extremely good at additions and multiplications. OpenFHE supports fast dot-product evaluation via `EvalInnerProduct`.

How does AI work inside FHE? (Part 2: Activation Functions)

The trickier side: Neural Networks also need Activation Functions (like ReLU or Sigmoid) to learn complex patterns.

How does AI work inside FHE? (Part 2: Activation Functions)

The trickier side: Neural Networks also need Activation Functions (like ReLU or Sigmoid) to learn complex patterns.

- $\text{ReLU}(x) = \max(0, x)$.

How does AI work inside FHE? (Part 2: Activation Functions)

The trickier side: Neural Networks also need Activation Functions (like ReLU or Sigmoid) to learn complex patterns.

- $\text{ReLU}(x) = \max(0, x)$.
- **The FHE Constraint:** FHE only understands Additions ($+$) and Multiplications ($\times$). It has NO concept of "if-else", "greater than", or "maximum".

How does AI work inside FHE? (Part 2: Activation Functions)

The trickier side: Neural Networks also need Activation Functions (like ReLU or Sigmoid) to learn complex patterns.

- $\text{ReLU}(x) = \max(0, x)$.
- **The FHE Constraint:** FHE only understands Additions ($+$) and Multiplications ($\times$). It has NO concept of "if-else", "greater than", or "maximum".
- **How do we fix this?**

How does AI work inside FHE? (Part 2: Activation Functions)

The trickier side: Neural Networks also need Activation Functions (like ReLU or Sigmoid) to learn complex patterns.

- $\text{ReLU}(x) = \max(0, x)$.
- **The FHE Constraint:** FHE only understands Additions ($+$) and Multiplications ($\times$). It has NO concept of "if-else", "greater than", or "maximum".
- **How do we fix this?**
  - We use **Polynomial Approximations** (like Taylor Series or Chebyshev Polynomials).

How does AI work inside FHE? (Part 2: Activation Functions)

The trickier side: Neural Networks also need Activation Functions (like ReLU or Sigmoid) to learn complex patterns.

- $\text{ReLU}(x) = \max(0, x)$.
- **The FHE Constraint:** FHE only understands Additions ($+$) and Multiplications ($\times$). It has NO concept of "if-else", "greater than", or "maximum".
- **How do we fix this?**
  - We use **Polynomial Approximations** (like Taylor Series or Chebyshev Polynomials).
  - We replace $\max(0, x)$ with a formula like: $y = 0.5x + 0.1x^3 - 0.05x^5 \ldots$

How does AI work inside FHE? (Part 2: Activation Functions)

The trickier side: Neural Networks also need Activation Functions (like ReLU or Sigmoid) to learn complex patterns.

- $\text{ReLU}(x) = \max(0, x)$.
- **The FHE Constraint:** FHE only understands Additions ($+$) and Multiplications ($\times$). It has NO concept of "if-else", "greater than", or "maximum".
- **How do we fix this?**
  - We use **Polynomial Approximations** (like Taylor Series or Chebyshev Polynomials).
  - We replace $\max(0, x)$ with a formula like: $y = 0.5x + 0.1x^3 - 0.05x^5 \ldots$
  - Now it's just additions and multiplications again! FHE can compute it efficiently.

The “Elephant in the Room”: Performance

If FHE is so great, why is it not everywhere yet?

The "Elephant in the Room": Performance

If FHE is so great, why is it not everywhere yet?

- **Computational Overhead:** FHE is currently thousands of times slower than plaintext computing.

The “Elephant in the Room”: Performance

If FHE is so great, why is it not everywhere yet?

- **Computational Overhead:** FHE is currently thousands of times slower than plaintext computing.
- **Data Expansion:** A 1 KB file might become a 10 MB ciphertext.

The "Elephant in the Room": Performance

If FHE is so great, why is it not everywhere yet?

- **Computational Overhead:** FHE is currently thousands of times slower than plaintext computing.
- **Data Expansion:** A 1 KB file might become a 10 MB ciphertext.
- **The Good News:**
  - **Hardware:** Companies like Intel and Duality are building FHE Accelerators (ASICs) that aim to reach 10x speeds.
  - **Algorithms:** Libraries like OpenFHE are getting faster every year through mathematical breakthroughs.

Performance Comparison Between FHE Families

Every FHE scheme optimizes for a different performance metric. Here is how several widely used scheme families compare in practice:

Scheme	Data & Precision	Performance Profile	Data Expansion	Best Use Case

Performance Comparison Between FHE Families

Every FHE scheme optimizes for a different performance metric. Here is how several widely used scheme families compare in practice:

Scheme	Data & Precision	Performance Profile	Data Expansion	Best Use Case
BFV / BGV	Exact Integers	High Throughput (Using SIMD batching)	Large ($\sim 10^3\times$)	DB Lookups, Financials

Performance Comparison Between FHE Families

Every FHE scheme optimizes for a different performance metric. Here is how several widely used scheme families compare in practice:

Scheme	Data & Precision	Performance Profile	Data Expansion	Best Use Case
BFV / BGV	Exact Integers	High Throughput (Using SIMD batching)	Large ($\sim 10^3\times$)	DB Lookups, Financials
CKKS	Approximate Floats	Highest Throughput (SIMD + Fast Polynomials)	Large ($\sim 10^3\times$)	AI, ML, Statistics

Performance Comparison Between FHE Families

Every FHE scheme optimizes for a different performance metric. Here is how several widely used scheme families compare in practice:

Scheme	Data & Precision	Performance Profile	Data Expansion	Best Use Case
BFV / BGV	Exact Integers	High Throughput (Using SIMD batching)	Large ($\sim 10^3\times$)	DB Lookups, Financials
CKKS	Approximate Floats	Highest Throughput (SIMD + Fast Polynomials)	Large ($\sim 10^3\times$)	AI, ML, Statistics
TFHE / FHEW	Boolean Bits (Gates)	Ultra-low Latency per bit (but slow for large arithmetic)	Massive ($\sim 10^5\times$ per int)	Comparisons, Control Logic

Summary

1. The Software Bridge: OpenFHE
2. Example of OpenFHE Workflow
3. FHE in Privacy-Preserving Machine Learning
4. FHE Performance Discussion