Fully Homomorphic Encryption (FHE) - Part I

A Cryptographic Holy Grail and its Motivations

Chuanqi Zhang

The Three States of Data

Modern cryptography successfully secures data in two out of its three primary states. However, the third state is significantly more challenging.

The Three States of Data

Modern cryptography successfully secures data in two out of its three primary states. However, the third state is significantly more challenging.

- **Data at Rest:** Inactive data stored physically in databases, hard drives, or cloud storage.

The Three States of Data

Modern cryptography successfully secures data in two out of its three primary states. However, the third state is significantly more challenging.

- **Data at Rest:** Inactive data stored physically in databases, hard drives, or cloud storage.
  - *Solution:* Symmetric encryption (e.g., AES-256). Highly secure and efficient.

The Three States of Data

Modern cryptography successfully secures data in two out of its three primary states. However, the third state is significantly more challenging.

- **Data at Rest:** Inactive data stored physically in databases, hard drives, or cloud storage.
  - *Solution:* Symmetric encryption (e.g., AES-256). Highly secure and efficient.
- **Data in Transit:** Data moving across a network (e.g., internet, private networks).

The Three States of Data

Modern cryptography successfully secures data in two out of its three primary states. However, the third state is significantly more challenging.

- **Data at Rest:** Inactive data stored physically in databases, hard drives, or cloud storage.
  - *Solution:* Symmetric encryption (e.g., AES-256). Highly secure and efficient.
- **Data in Transit:** Data moving across a network (e.g., internet, private networks).
  - *Solution:* TLS/SSL, IPsec, End-to-End Encryption (E2EE).

The Three States of Data

Modern cryptography successfully secures data in two out of its three primary states. However, the third state is significantly more challenging.

- **Data at Rest:** Inactive data stored physically in databases, hard drives, or cloud storage.
  - *Solution:* Symmetric encryption (e.g., AES-256). Highly secure and efficient.
- **Data in Transit:** Data moving across a network (e.g., internet, private networks).
  - *Solution:* TLS/SSL, IPsec, End-to-End Encryption (E2EE).
- **Data in Use:** Data being processed, computed, or read by RAM and CPU.

The Three States of Data

Modern cryptography successfully secures data in two out of its three primary states. However, the third state is significantly more challenging.

- **Data at Rest:** Inactive data stored physically in databases, hard drives, or cloud storage.
  - *Solution:* Symmetric encryption (e.g., AES-256). Highly secure and efficient.
- **Data in Transit:** Data moving across a network (e.g., internet, private networks).
  - *Solution:* TLS/SSL, IPsec, End-to-End Encryption (E2EE).
- **Data in Use:** Data being processed, computed, or read by RAM and CPU.
  - *The Gap:* Traditionally, data **must be decrypted** into plaintext before a processor can perform any operations on it.

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The standard processing pipeline:

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The standard processing pipeline:

1. Client encrypts plaintext m to ciphertext c and sends it to the server.

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The standard processing pipeline:

1. Client encrypts plaintext m to ciphertext c and sends it to the server.
2. Server receives c and **must request the decryption key**.

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The standard processing pipeline:

1. Client encrypts plaintext m to ciphertext c and sends it to the server.
2. Server receives c and **must request the decryption key**.
3. Server decrypts: $D(c) = m$.

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The standard processing pipeline:

1. Client encrypts plaintext m to ciphertext c and sends it to the server.
2. Server receives c and **must request the decryption key**.
3. Server decrypts: $D(c) = m$.
4. Server computes function f: $y = f(m)$.

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The standard processing pipeline:

1. Client encrypts plaintext m to ciphertext c and sends it to the server.
2. Server receives c and **must request the decryption key**.
3. Server decrypts: $D(c) = m$.
4. Server computes function f: $y = f(m)$.
5. Server encrypts the result $E(y)$ and sends it back.

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The standard processing pipeline:

1. Client encrypts plaintext m to ciphertext c and sends it to the server.
2. Server receives c and **must request the decryption key**.
3. Server decrypts: $D(c) = m$.
4. Server computes function f: $y = f(m)$.
5. Server encrypts the result $E(y)$ and sends it back.

Possible vulnerabilities: Memory dumps, hardware side-channel attacks, malicious insiders, and compromised OS kernels may access m and the encryption key during step 3 and 4.

The Traditional Decryption Dilemma

The Attack Surface

No matter how strong your AES keys or TLS tunnels are, the moment you need to compute on the data, it is exposed in the server's memory.

The standard processing pipeline:

1. Client encrypts plaintext m to ciphertext c and sends it to the server.
2. Server receives c and **must request the decryption key**.
3. Server decrypts: $D(c) = m$.
4. Server computes function f: $y = f(m)$.
5. Server encrypts the result $E(y)$ and sends it back.

Motivation: Can the server compute $y = f(m)$ *without* decrypting c?

Homomorphism: Operate on Ciphertexts Directly

Homomorphism is an algebraic property that preserves the structure of operations.

Homomorphism: Operate on Ciphertexts Directly

Homomorphism is an algebraic property that preserves the structure of operations. In cryptography, it enables operations to be carried out directly on ciphertexts.

Homomorphism: Operate on Ciphertexts Directly

Homomorphism is an algebraic property that preserves the structure of operations. In cryptography, it enables operations to be carried out directly on ciphertexts.

Specifically,

Homomorphism: Operate on Ciphertexts Directly

Homomorphism is an algebraic property that preserves the structure of operations. In cryptography, it enables operations to be carried out directly on ciphertexts.

Specifically,

- Let $E(\cdot)$ be an encryption algorithm.

Homomorphism: Operate on Ciphertexts Directly

Homomorphism is an algebraic property that preserves the structure of operations. In cryptography, it enables operations to be carried out directly on ciphertexts.

Specifically,

- Let $E(\cdot)$ be an encryption algorithm.
- Let m_1 and m_2 be plaintexts.

Homomorphism: Operate on Ciphertexts Directly

Homomorphism is an algebraic property that preserves the structure of operations. In cryptography, it enables operations to be carried out directly on ciphertexts.

Specifically,

- Let $E(\cdot)$ be an encryption algorithm.
- Let m_1 and m_2 be plaintexts.
- We say that E is homomorphic with respect to operations $\circ$ and $\star$, if

$$E(m_1) \circ E(m_2) = E(m_1 \star m_2).$$

Homomorphism: Operate on Ciphertexts Directly

Homomorphism is an algebraic property that preserves the structure of operations. In cryptography, it enables operations to be carried out directly on ciphertexts.

Specifically,

- Let $E(\cdot)$ be an encryption algorithm.
- Let m_1 and m_2 be plaintexts.
- We say that E is homomorphic with respect to operations $\circ$ and $\star$, if

$$E(m_1) \circ E(m_2) = E(m_1 \star m_2).$$

Note

$\circ$ and $\star$ can be the same, but don't have to be!

Homomorphism: Operate on Ciphertexts Directly

Homomorphism is an algebraic property that preserves the structure of operations. In cryptography, it enables operations to be carried out directly on ciphertexts.

Specifically,

- Let $E(\cdot)$ be an encryption algorithm.
- Let m_1 and m_2 be plaintexts.
- We say that E is homomorphic with respect to operations $\circ$ and $\star$, if

$$E(m_1) \circ E(m_2) = E(m_1 \star m_2).$$

Example

For $m_1, m_2 \in \{0, 1\}$, $\star$ can be $+$ or $\times$.

Homomorphic Encryption Types

1. **Partially homomorphic encryption** (PHE) supports one operation type for messages, such as additions or multiplications.

Homomorphic Encryption Types

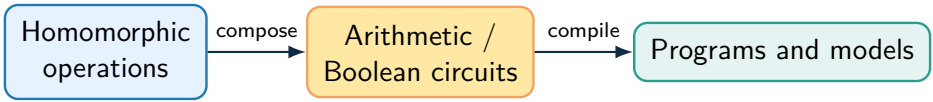
1. **Partially homomorphic encryption** (PHE) supports one operation type for messages, such as additions or multiplications.
2. **Somewhat homomorphic encryption** (SHE) supports a bounded-depth computation for both additions and multiplications, before noise becomes too large.

Homomorphic Encryption Types

1. **Partially homomorphic encryption** (PHE) supports one operation type for messages, such as additions or multiplications.
2. **Somewhat homomorphic encryption** (SHE) supports a bounded-depth computation for both additions and multiplications, before noise becomes too large.
3. **Fully homomorphic encryption** (FHE) supports arbitrary-depth computations for both additions and multiplications.

Homomorphic Encryption Types

1. **Partially homomorphic encryption** (PHE) supports one operation type for messages, such as additions or multiplications.
2. **Somewhat homomorphic encryption** (SHE) supports a bounded-depth computation for both additions and multiplications, before noise becomes too large.
3. **Fully homomorphic encryption** (FHE) supports arbitrary-depth computations for both additions and multiplications.



The Holy Grail: Fully Homomorphic Encryption

PHE and SHE are useful, but insufficient for general-purpose computing. We need a system that supports both additions and multiplications simultaneously, for an unlimited number of times.

The Holy Grail: Fully Homomorphic Encryption

PHE and SHE are useful, but insufficient for general-purpose computing. We need a system that supports both additions and multiplications simultaneously, for an unlimited number of times.

- **Turing Completeness:** With both additions (XOR) and multiplications (AND), we can construct any boolean circuit.

The Holy Grail: Fully Homomorphic Encryption

PHE and SHE are useful, but insufficient for general-purpose computing. We need a system that supports both additions and multiplications simultaneously, for an unlimited number of times.

- **Turing Completeness:** With both additions (XOR) and multiplications (AND), we can construct any boolean circuit.
- **The FHE Promise:** For any computable function f and a ciphertext $c = E(m)$, there exists a public evaluation function *Eval* such that:

$$Eval(f, c) \Rightarrow E(f(m))$$

The Holy Grail: Fully Homomorphic Encryption

PHE and SHE are useful, but insufficient for general-purpose computing. We need a system that supports both additions and multiplications simultaneously, for an unlimited number of times.

- **Turing Completeness:** With both additions (XOR) and multiplications (AND), we can construct any boolean circuit.
- **The FHE Promise:** For any computable function f and a ciphertext $c = E(m)$, there exists a public evaluation function $Eval$ such that:

$$Eval(f, c) \Rightarrow E(f(m))$$

- **The Key Paradigm Shift:** The entity computing the function f never sees the plaintext m, the result $f(m)$, or the secret key.

FHE Application in Cloud Computing

FHE Application in Cloud Computing

The Business Need:

- Organizations possess massive datasets but lack the local infrastructure to process them.
- Cloud providers (AWS, Azure, GCP) offer infinite, elastic compute power.

FHE Application in Cloud Computing

The Business Need:

- Organizations possess massive datasets but lack the local infrastructure to process them.
- Cloud providers (AWS, Azure, GCP) offer infinite, elastic compute power.

The Trust Barrier:

- Sensitive data (medical records, IP, financial transactions) cannot legally or safely be handed over to a third party.
- Data breach liability is too high.

FHE Application in Cloud Computing

The Business Need:

- Organizations possess massive datasets but lack the local infrastructure to process them.
- Cloud providers (AWS, Azure, GCP) offer infinite, elastic compute power.

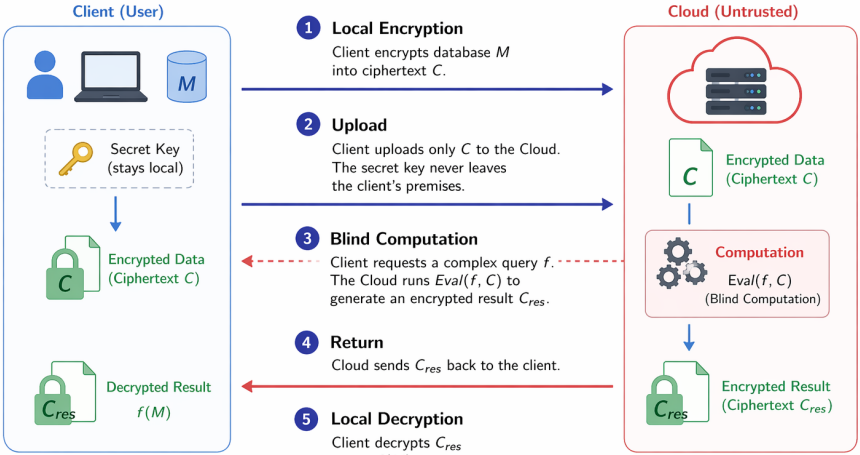
The Trust Barrier:

- Sensitive data (medical records, IP, financial transactions) cannot legally or safely be handed over to a third party.
- Data breach liability is too high.

The Goal

Outsource the **computation** without outsourcing the **trust**.

The FHE Cloud Workflow (Zero-Trust Architecture)



Result: The cloud provider is effectively removed from the trust boundary. Even if the cloud is breached, attackers only steal **mathematical noise**.

FHE Application in Machine Learning

Modern AI/ML models scale with data. However, the most valuable data is locked away due to privacy constraints.

FHE Application in Machine Learning

Modern AI/ML models scale with data. However, the most valuable data is locked away due to privacy constraints.

- **Data Silos:** Hospitals cannot share patient data with research labs. Banks cannot share transaction histories.

FHE Application in Machine Learning

Modern AI/ML models scale with data. However, the most valuable data is locked away due to privacy constraints.

- **Data Silos:** Hospitals cannot share patient data with research labs. Banks cannot share transaction histories.
- **Regulatory Compliance:** GDPR (Europe), HIPAA (US Healthcare), and CCPA impose severe penalties for mishandling PII.

FHE Application in Machine Learning

Modern AI/ML models scale with data. However, the most valuable data is locked away due to privacy constraints.

- **Data Silos:** Hospitals cannot share patient data with research labs. Banks cannot share transaction histories.
- **Regulatory Compliance:** GDPR (Europe), HIPAA (US Healthcare), and CCPA impose severe penalties for mishandling PII.
- **Solution:** FHE provides a cryptographic mechanism to unlock this data's utility without compromising its privacy.

Application Example 1: Secure Inference

Scenario: A company has a proprietary AI model in the cloud. A user wants to use the model, but their input data is highly sensitive.

Application Example 1: Secure Inference

Scenario: A company has a proprietary AI model in the cloud. A user wants to use the model, but their input data is highly sensitive.

- **Privacy Conflict:**
  - User doesn't want to reveal their data to the company.
  - Company doesn't want to send their proprietary model weights to the user.

Application Example 1: Secure Inference

Scenario: A company has a proprietary AI model in the cloud. A user wants to use the model, but their input data is highly sensitive.

- **Privacy Conflict:**
  - User doesn't want to reveal their data to the company.
  - Company doesn't want to send their proprietary model weights to the user.
- **FHE Solution:**

Application Example 1: Secure Inference

Scenario: A company has a proprietary AI model in the cloud. A user wants to use the model, but their input data is highly sensitive.

- **Privacy Conflict:**
  - User doesn't want to reveal their data to the company.
  - Company doesn't want to send their proprietary model weights to the user.
- **FHE Solution:**
  1. User encrypts their input (e.g., an X-ray image).

Application Example 1: Secure Inference

Scenario: A company has a proprietary AI model in the cloud. A user wants to use the model, but their input data is highly sensitive.

- **Privacy Conflict:**
  - User doesn't want to reveal their data to the company.
  - Company doesn't want to send their proprietary model weights to the user.
- **FHE Solution:**
  1. User encrypts their input (e.g., an X-ray image).
  2. Cloud passes the encrypted image through the neural network.

Application Example 1: Secure Inference

Scenario: A company has a proprietary AI model in the cloud. A user wants to use the model, but their input data is highly sensitive.

- **Privacy Conflict:**
  - User doesn't want to reveal their data to the company.
  - Company doesn't want to send their proprietary model weights to the user.
- **FHE Solution:**
  1. User encrypts their input (e.g., an X-ray image).
  2. Cloud passes the encrypted image through the neural network.
  3. The cloud mathematically applies the model's layers directly on the ciphertexts.

Application Example 1: Secure Inference

Scenario: A company has a proprietary AI model in the cloud. A user wants to use the model, but their input data is highly sensitive.

- **Privacy Conflict:**
  - User doesn't want to reveal their data to the company.
  - Company doesn't want to send their proprietary model weights to the user.
- **FHE Solution:**
  1. User encrypts their input (e.g., an X-ray image).
  2. Cloud passes the encrypted image through the neural network.
  3. The cloud mathematically applies the model's layers directly on the ciphertexts.
  4. Cloud returns the encrypted diagnosis.

Application Example 2: Secure Collaborative Training

Scenario: Multiple institutions want to train a global model collaboratively without revealing their local datasets.

Application Example 2: Secure Collaborative Training

Scenario: Multiple institutions want to train a global model collaboratively without revealing their local datasets.

- **FHE-enabled Federated Learning:**

Application Example 2: Secure Collaborative Training

Scenario: Multiple institutions want to train a global model collaboratively without revealing their local datasets.

- **FHE-enabled Federated Learning:**
  - Institutions compute model gradients locally.

Application Example 2: Secure Collaborative Training

Scenario: Multiple institutions want to train a global model collaboratively without revealing their local datasets.

- **FHE-enabled Federated Learning:**
  - Institutions compute model gradients locally.
  - They encrypt these gradients using FHE before sending them to an aggregator.

Application Example 2: Secure Collaborative Training

Scenario: Multiple institutions want to train a global model collaboratively without revealing their local datasets.

- **FHE-enabled Federated Learning:**
  - Institutions compute model gradients locally.
  - They encrypt these gradients using FHE before sending them to an aggregator.
  - The central server aggregates the encrypted gradients homomorphically.

Application Example 2: Secure Collaborative Training

Scenario: Multiple institutions want to train a global model collaboratively without revealing their local datasets.

- **FHE-enabled Federated Learning:**
  - Institutions compute model gradients locally.
  - They encrypt these gradients using FHE before sending them to an aggregator.
  - The central server aggregates the encrypted gradients homomorphically.
  - The updated, encrypted global model weights are sent back for decryption and local update.

Application Example 2: Secure Collaborative Training

Scenario: Multiple institutions want to train a global model collaboratively without revealing their local datasets.

- **FHE-enabled Federated Learning:**
  - Institutions compute model gradients locally.
  - They encrypt these gradients using FHE before sending them to an aggregator.
  - The central server aggregates the encrypted gradients homomorphically.
  - The updated, encrypted global model weights are sent back for decryption and local update.

- **Benefit:** Completely eliminates the risk of "model inversion attacks" where attackers reconstruct training data from gradient updates.

Summary

1. The Security Gap: Data in Use
2. Motivation: Computing on Ciphertexts
3. Application: Secure Cloud Computing
4. Application: Privacy-Preserving AI & Machine Learning