Fully Homomorphic Encryption (FHE) - Part I

Multiplicative Homomorphism via [GSW13] (Trial and Fix)

Chuanqi Zhang

The Basic Idea of [GSW13][1]

[1] "*Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*", by Craig Gentry, Amit Sahai, and Brent Waters, in CRYPTO 2013.

The Basic Idea of [GSW13][1]

Before diving into the construction of Fully Homomorphic Encryption in [GSW13], let's look at their foundational intuition.

[1] "*Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*", by Craig Gentry, Amit Sahai, and Brent Waters, in CRYPTO 2013.

The Basic Idea of [GSW13][1]

Before diving into the construction of Fully Homomorphic Encryption in [GSW13], let's look at their foundational intuition.

The Basic Encryption Identity of [GSW13]

Given a public matrix $\mathbf{C}$, find the secret vector $\mathbf{v}$ such that:

$$\mathbf{Cv} = \mu \cdot \mathbf{v} + \mathbf{e} \pmod{q}$$

where $\mathbf{e}$ is a vector of small random noise/errors and μ represents a message.

[1] "*Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*", by Craig Gentry, Amit Sahai, and Brent Waters, in CRYPTO 2013.

The Basic Idea of [GSW13][1]

Before diving into the construction of Fully Homomorphic Encryption in [GSW13], let's look at their foundational intuition.

The Basic Encryption Identity of [GSW13]

Given a public matrix $\mathbf{C}$, find the secret vector $\mathbf{v}$ such that:

$$\mathbf{Cv} = \mu \cdot \mathbf{v} + \mathbf{e} \pmod{q}$$

where $\mathbf{e}$ is a vector of small random noise/errors and μ represents a message.

- Without $\mathbf{e}$, this corresponds to calculating the *eigenvalues*.

[1] "*Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*", by Craig Gentry, Amit Sahai, and Brent Waters, in CRYPTO 2013.

The Basic Idea of [GSW13][1]

Before diving into the construction of Fully Homomorphic Encryption in [GSW13], let's look at their foundational intuition.

The Basic Encryption Identity of [GSW13]

Given a public matrix $\mathbf{C}$, find the secret vector $\mathbf{v}$ such that:

$$\mathbf{Cv} = \mu \cdot \mathbf{v} + \mathbf{e} \pmod{q}$$

where $\mathbf{e}$ is a vector of small random noise/errors and μ represents a message.

- Without $\mathbf{e}$, this corresponds to calculating the *eigenvalues*.
- With $\mathbf{e}$, this problem is at least as hard as LWE.

[1] "*Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*", by Craig Gentry, Amit Sahai, and Brent Waters, in CRYPTO 2013.

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.
- Let's isolate a single row. Let $\mathbf{c}_i$ be the i-th row of $\mathbf{C}$, v_i be the i-th component of $\mathbf{v}$, and e_i be the i-th component of $\mathbf{e}$.

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.
- Let's isolate a single row. Let $\mathbf{c}_i$ be the i-th row of $\mathbf{C}$, v_i be the i-th component of $\mathbf{v}$, and e_i be the i-th component of $\mathbf{e}$.
- **Step 1: Compute the Dot Product**

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.
- Let's isolate a single row. Let $\mathbf{c}_i$ be the i-th row of $\mathbf{C}$, v_i be the i-th component of $\mathbf{v}$, and e_i be the i-th component of $\mathbf{e}$.
- **Step 1: Compute the Dot Product**

$$x_i = \langle \mathbf{c}_i, \mathbf{v} \rangle = \mu \cdot v_i + e_i \pmod{q}$$

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.
- Let's isolate a single row. Let $\mathbf{c}_i$ be the i-th row of $\mathbf{C}$, v_i be the i-th component of $\mathbf{v}$, and e_i be the i-th component of $\mathbf{e}$.
- **Step 1: Compute the Dot Product**

$$x_i = \langle \mathbf{c}_i, \mathbf{v} \rangle = \mu \cdot v_i + e_i \pmod{q}$$

- **Step 2: Isolate the Message**

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.
- Let's isolate a single row. Let $\mathbf{c}_i$ be the i-th row of $\mathbf{C}$, v_i be the i-th component of $\mathbf{v}$, and e_i be the i-th component of $\mathbf{e}$.
- **Step 1: Compute the Dot Product**

$$x_i = \langle \mathbf{c}_i, \mathbf{v} \rangle = \mu \cdot v_i + e_i \pmod{q}$$

- **Step 2: Isolate the Message**
  We intentionally select an index i such that v_i is a large value ($\approx q/2$).

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.
- Let's isolate a single row. Let $\mathbf{c}_i$ be the i-th row of $\mathbf{C}$, v_i be the i-th component of $\mathbf{v}$, and e_i be the i-th component of $\mathbf{e}$.
- **Step 1: Compute the Dot Product**

$$x_i = \langle \mathbf{c}_i, \mathbf{v} \rangle = \mu \cdot v_i + e_i \pmod{q}$$

- **Step 2: Isolate the Message**
  We intentionally select an index i such that v_i is a large value ($\approx q/2$).
- **Step 3: Rounding**

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.
- Let's isolate a single row. Let $\mathbf{c}_i$ be the i-th row of $\mathbf{C}$, v_i be the i-th component of $\mathbf{v}$, and e_i be the i-th component of $\mathbf{e}$.
- **Step 1: Compute the Dot Product**

$$x_i = \langle \mathbf{c}_i, \mathbf{v} \rangle = \mu \cdot v_i + e_i \pmod{q}$$

- **Step 2: Isolate the Message**
  We intentionally select an index i such that v_i is a large value ($\approx q/2$).
- **Step 3: Rounding**

$$\mu = \left\lfloor \frac{x_i}{v_i} \right\rceil$$

The Basic Decryption Idea

How do we extract the message μ from the ciphertext $\mathbf{C}$ using the secret key $\mathbf{v}$?

- We know the full vector equation: $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$.
- Let's isolate a single row. Let $\mathbf{c}_i$ be the i-th row of $\mathbf{C}$, v_i be the i-th component of $\mathbf{v}$, and e_i be the i-th component of $\mathbf{e}$.
- **Step 1: Compute the Dot Product**

$$x_i = \langle \mathbf{c}_i, \mathbf{v} \rangle = \mu \cdot v_i + e_i \pmod{q}$$

- **Step 2: Isolate the Message**
  We intentionally select an index i such that v_i is a large value ($\approx q/2$).
- **Step 3: Rounding**

$$\mu = \lfloor \frac{x_i}{v_i} \rceil$$

- Since e_i is very small and v_i is very large, the noise $\frac{e_i}{v_i}$ is negligible. The rounding perfectly recovers μ.

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

- $\mathbf{C}_1\mathbf{v} = \mu_1\mathbf{v} + \mathbf{e}_1$

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

- $\mathbf{C}_1\mathbf{v} = \mu_1\mathbf{v} + \mathbf{e}_1$
- $\mathbf{C}_2\mathbf{v} = \mu_2\mathbf{v} + \mathbf{e}_2$

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

- $\mathbf{C}_1\mathbf{v} = \mu_1\mathbf{v} + \mathbf{e}_1$
- $\mathbf{C}_2\mathbf{v} = \mu_2\mathbf{v} + \mathbf{e}_2$

Homomorphic addition: matrix addition

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

- $\mathbf{C}_1\mathbf{v} = \mu_1\mathbf{v} + \mathbf{e}_1$
- $\mathbf{C}_2\mathbf{v} = \mu_2\mathbf{v} + \mathbf{e}_2$

Homomorphic addition: matrix addition

$$(\mathbf{C}_1 + \mathbf{C}_2) \cdot \mathbf{v} = \mathbf{C}_1\mathbf{v} + \mathbf{C}_2\mathbf{v}$$

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

- $\mathbf{C}_1\mathbf{v} = \mu_1\mathbf{v} + \mathbf{e}_1$
- $\mathbf{C}_2\mathbf{v} = \mu_2\mathbf{v} + \mathbf{e}_2$

Homomorphic addition: matrix addition

$$\begin{aligned}(\mathbf{C}_1 + \mathbf{C}_2) \cdot \mathbf{v} &= \mathbf{C}_1\mathbf{v} + \mathbf{C}_2\mathbf{v} \\ &= (\mu_1\mathbf{v} + \mathbf{e}_1) + (\mu_2\mathbf{v} + \mathbf{e}_2)\end{aligned}$$

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

- $\mathbf{C}_1\mathbf{v} = \mu_1\mathbf{v} + \mathbf{e}_1$
- $\mathbf{C}_2\mathbf{v} = \mu_2\mathbf{v} + \mathbf{e}_2$

Homomorphic addition: matrix addition

$$\begin{aligned}(\mathbf{C}_1 + \mathbf{C}_2) \cdot \mathbf{v} &= \mathbf{C}_1\mathbf{v} + \mathbf{C}_2\mathbf{v} \\ &= (\mu_1\mathbf{v} + \mathbf{e}_1) + (\mu_2\mathbf{v} + \mathbf{e}_2) \\ &= (\mu_1 + \mu_2)\mathbf{v} + (\mathbf{e}_1 + \mathbf{e}_2)\end{aligned}$$

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

- $\mathbf{C}_1\mathbf{v} = \mu_1\mathbf{v} + \mathbf{e}_1$
- $\mathbf{C}_2\mathbf{v} = \mu_2\mathbf{v} + \mathbf{e}_2$

Homomorphic addition: matrix addition

$$\begin{aligned}(\mathbf{C}_1 + \mathbf{C}_2) \cdot \mathbf{v} &= \mathbf{C}_1\mathbf{v} + \mathbf{C}_2\mathbf{v} \\ &= (\mu_1\mathbf{v} + \mathbf{e}_1) + (\mu_2\mathbf{v} + \mathbf{e}_2) \\ &= (\mu_1 + \mu_2)\mathbf{v} + (\mathbf{e}_1 + \mathbf{e}_2)\end{aligned}$$

- **Result:** The sum of matrices encrypts the sum of messages.

The Additive Homomorphism

Suppose we have two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ for messages μ_1, μ_2:

- $\mathbf{C}_1\mathbf{v} = \mu_1\mathbf{v} + \mathbf{e}_1$
- $\mathbf{C}_2\mathbf{v} = \mu_2\mathbf{v} + \mathbf{e}_2$

Homomorphic addition: matrix addition

$$\begin{aligned}(\mathbf{C}_1 + \mathbf{C}_2) \cdot \mathbf{v} &= \mathbf{C}_1\mathbf{v} + \mathbf{C}_2\mathbf{v} \\ &= (\mu_1\mathbf{v} + \mathbf{e}_1) + (\mu_2\mathbf{v} + \mathbf{e}_2) \\ &= (\mu_1 + \mu_2)\mathbf{v} + (\mathbf{e}_1 + \mathbf{e}_2)\end{aligned}$$

- **Result:** The sum of matrices encrypts the sum of messages.
- **Noise Growth:** The noise grows linearly $(\mathbf{e}_1 + \mathbf{e}_2)$, which is stable!

The Multiplicative Homomorphism: “Trial”

What happens if we try to multiply two ciphertext matrices?

The Multiplicative Homomorphism: "Trial"

What happens if we try to multiply two ciphertext matrices?

$$\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} = \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2)$$

The Multiplicative Homomorphism: "Trial"

What happens if we try to multiply two ciphertext matrices?

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2\end{aligned}$$

The Multiplicative Homomorphism: "Trial"

What happens if we try to multiply two ciphertext matrices?

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2\end{aligned}$$

The Multiplicative Homomorphism: "Trial"

What happens if we try to multiply two ciphertext matrices?

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

The Multiplicative Homomorphism: "Trial"

What happens if we try to multiply two ciphertext matrices?

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

- **The Crisis:** Look at the term $\mathbf{C}_1 \mathbf{e}_2$.

The Multiplicative Homomorphism: "Trial"

What happens if we try to multiply two ciphertext matrices?

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

- **The Crisis:** Look at the term $\mathbf{C}_1 \mathbf{e}_2$.
- The entries of $\mathbf{C}_1$ could be large!

The Multiplicative Homomorphism: "Trial"

What happens if we try to multiply two ciphertext matrices?

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

- **The Crisis:** Look at the term $\mathbf{C}_1 \mathbf{e}_2$.
- The entries of $\mathbf{C}_1$ could be large!
- Multiplying large entries in $\mathbf{C}_1$ with the small noise $\mathbf{e}_2$ causes a **massive noise explosion**. The message is instantly destroyed!

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers.

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

- Let $\ell = \lfloor \log_2 q \rfloor + 1$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

- Let $\ell = \lfloor \log_2 q \rfloor + 1$.
- **Tool 1:** *BitDecomp*$(\mathbf{c})$

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

- Let $\ell = \lfloor \log_2 q \rfloor + 1$.
- **Tool 1:** *BitDecomp*$(\mathbf{c})$
  - Expands a vector $\mathbf{c} \in \mathbb{Z}_q^k$ by writing each component in binary form.

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

- Let $\ell = \lfloor \log_2 q \rfloor + 1$.
- **Tool 1:** *BitDecomp*$(\mathbf{c})$
  - Expands a vector $\mathbf{c} \in \mathbb{Z}_q^k$ by writing each component in binary form.
  - Outputs an expanded vector of length $k \cdot \ell$ with entries strictly in $\{0, 1\}$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

- Let $\ell = \lfloor \log_2 q \rfloor + 1$.
- **Tool 1:** *BitDecomp*$(\mathbf{c})$
  - Expands a vector $\mathbf{c} \in \mathbb{Z}_q^k$ by writing each component in binary form.
  - Outputs an expanded vector of length $k \cdot \ell$ with entries strictly in $\{0, 1\}$.
- **Tool 2:** *Powersof2*$(\mathbf{v})$

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

- Let $\ell = \lfloor \log_2 q \rfloor + 1$.
- **Tool 1:** *BitDecomp*$(\mathbf{c})$
  - Expands a vector $\mathbf{c} \in \mathbb{Z}_q^k$ by writing each component in binary form.
  - Outputs an expanded vector of length $k \cdot \ell$ with entries strictly in $\{0, 1\}$.
- **Tool 2:** *Powersof2*$(\mathbf{v})$
  - Expands a vector $\mathbf{v} \in \mathbb{Z}_q^k$ by multiplying by powers of 2: $(v_i, 2v_i, \ldots, 2^{\ell-1}v_i)$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

- Let $\ell = \lfloor \log_2 q \rfloor + 1$.
- **Tool 1:** *BitDecomp*$(\mathbf{c})$
  - Expands a vector $\mathbf{c} \in \mathbb{Z}_q^k$ by writing each component in binary form.
  - Outputs an expanded vector of length $k \cdot \ell$ with entries strictly in $\{0, 1\}$.
- **Tool 2:** *Powersof2*$(\mathbf{v})$
  - Expands a vector $\mathbf{v} \in \mathbb{Z}_q^k$ by multiplying by powers of 2: $(v_i, 2v_i, \ldots, 2^{\ell-1}v_i)$.
  - Outputs a matching expanded vector of length $k \cdot \ell$ with entries over $\mathbb{Z}_q$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, $\mathbf{C}_1$ must be a matrix of very small numbers. [GSW13] introduces two mathematical operators to solve this.

- Let $\ell = \lfloor \log_2 q \rfloor + 1$.
- **Tool 1:** $\mathit{BitDecomp}(\mathbf{c})$
  - Expands a vector $\mathbf{c} \in \mathbb{Z}_q^k$ by writing each component in binary form.
  - Outputs an expanded vector of length $k \cdot \ell$ with entries strictly in $\{0, 1\}$.
- **Tool 2:** $\mathit{Powersof2}(\mathbf{v})$
  - Expands a vector $\mathbf{v} \in \mathbb{Z}_q^k$ by multiplying by powers of 2: $(v_i, 2v_i, \ldots, 2^{\ell-1}v_i)$.
  - Outputs a matching expanded vector of length $k \cdot \ell$ with entries over $\mathbb{Z}_q$.

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.
- $\mathit{BitDecomp}(c) = (c_0, \ldots, c_{\ell-1}) \in \{0,1\}^{\ell}$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.
- $\mathit{BitDecomp}(c) = (c_0, \ldots, c_{\ell-1}) \in \{0,1\}^\ell$.
- $c_0 \ldots c_{\ell-1}$ is the binary representation of c.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.
- $\mathit{BitDecomp}(c) = (c_0, \ldots, c_{\ell-1}) \in \{0,1\}^\ell$.
- $c_0 \ldots c_{\ell-1}$ is the binary representation of c.
- Equivalently, $c = \sum_{i=0}^{\ell-1} c_i \cdot 2^i \pmod{q}$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.
- $\mathit{BitDecomp}(c) = (c_0, \ldots, c_{\ell-1}) \in \{0,1\}^\ell$.
- $c_0 \ldots c_{\ell-1}$ is the binary representation of c.
- Equivalently, $c = \sum_{i=0}^{\ell-1} c_i \cdot 2^i \pmod{q}$.
- $\mathit{Powersof2}(v) = (v, 2v, \ldots, 2^{\ell-1}v) \pmod{q} \in \mathbb{Z}_q^\ell$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.
- $\mathit{BitDecomp}(c) = (c_0, \ldots, c_{\ell-1}) \in \{0,1\}^\ell$.
- $c_0 \ldots c_{\ell-1}$ is the binary representation of c.
- Equivalently, $c = \sum_{i=0}^{\ell-1} c_i \cdot 2^i \pmod{q}$.
- $\mathit{Powersof2}(v) = (v, 2v, \ldots, 2^{\ell-1}v) \pmod{q} \in \mathbb{Z}_q^\ell$.

$$\langle \mathit{BitDecomp}(c), \mathit{Powersof2}(v) \rangle = \sum_{i=0}^{\ell-1} \left(c_i \cdot (2^i \cdot v) \right) \pmod{q}$$

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.
- $\mathit{BitDecomp}(c) = (c_0, \ldots, c_{\ell-1}) \in \{0,1\}^\ell$.
- $c_0 \ldots c_{\ell-1}$ is the binary representation of c.
- Equivalently, $c = \sum_{i=0}^{\ell-1} c_i \cdot 2^i \pmod{q}$.
- $\mathit{Powersof2}(v) = (v, 2v, \ldots, 2^{\ell-1}v) \pmod{q} \in \mathbb{Z}_q^\ell$.

$$\begin{aligned}
\langle \mathit{BitDecomp}(c), \mathit{Powersof2}(v) \rangle &= \sum_{i=0}^{\ell-1} \left(c_i \cdot (2^i \cdot v) \right) \pmod{q} \\
&= \left(\sum_{i=0}^{\ell-1} c_i \cdot 2^i \right) \cdot v \pmod{q}
\end{aligned}$$

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.
- $\mathit{BitDecomp}(c) = (c_0, \ldots, c_{\ell-1}) \in \{0,1\}^\ell$.
- $c_0 \ldots c_{\ell-1}$ is the binary representation of c.
- Equivalently, $c = \sum_{i=0}^{\ell-1} c_i \cdot 2^i \pmod{q}$.
- $\mathit{Powersof2}(v) = (v, 2v, \ldots, 2^{\ell-1}v) \pmod{q} \in \mathbb{Z}_q^\ell$.

$$\begin{aligned}
\langle \mathit{BitDecomp}(c), \mathit{Powersof2}(v) \rangle &= \sum_{i=0}^{\ell-1} \left(c_i \cdot (2^i \cdot v) \right) \pmod{q} \\
&= \left(\sum_{i=0}^{\ell-1} c_i \cdot 2^i \right) \cdot v \pmod{q}
\end{aligned}$$

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- Let's check this equality for scalars $c \in \mathbb{Z}_q$ and $v \in \mathbb{Z}_q$.
- $\mathit{BitDecomp}(c) = (c_0, \ldots, c_{\ell-1}) \in \{0,1\}^\ell$.
- $c_0 \ldots c_{\ell-1}$ is the binary representation of c.
- Equivalently, $c = \sum_{i=0}^{\ell-1} c_i \cdot 2^i \pmod{q}$.
- $\mathit{Powersof2}(v) = (v, 2v, \ldots, 2^{\ell-1}v) \pmod{q} \in \mathbb{Z}_q^\ell$.

$$\begin{aligned}
\langle \mathit{BitDecomp}(c), \mathit{Powersof2}(v) \rangle &= \sum_{i=0}^{\ell-1} \left(c_i \cdot (2^i \cdot v) \right) \pmod{q} \\
&= \left(\sum_{i=0}^{\ell-1} c_i \cdot 2^i \right) \cdot v \pmod{q} = c \cdot v \pmod{q}
\end{aligned}$$

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- *BitDecomp* and *Powersof2* are useful, but not enough!

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- *BitDecomp* and *Powersof2* are useful, but not enough!
- Recall that $\mathbf{c}_i \in \mathbb{Z}_q^n$ is the i-th row of $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ and the decryption is based on the result of $\langle \mathbf{c}_i, \mathbf{v} \rangle$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- *BitDecomp* and *Powersof2* are useful, but not enough!
- Recall that $\mathbf{c}_i \in \mathbb{Z}_q^n$ is the i-th row of $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ and the decryption is based on the result of $\langle \mathbf{c}_i, \mathbf{v} \rangle$.
- These tools are useful in the sense that:

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- *BitDecomp* and *Powersof2* are useful, but not enough!
- Recall that $\mathbf{c}_i \in \mathbb{Z}_q^n$ is the i-th row of $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ and the decryption is based on the result of $\langle \mathbf{c}_i, \mathbf{v} \rangle$.
- These tools are useful in the sense that:
  - *BitDecomp* can transform $\mathbf{c}_i \in \mathbb{Z}_q^n$ into a binary vector $\mathbf{c}_i' \in \{0,1\}^{nl}$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- *BitDecomp* and *Powersof2* are useful, but not enough!
- Recall that $\mathbf{c}_i \in \mathbb{Z}_q^n$ is the i-th row of $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ and the decryption is based on the result of $\langle \mathbf{c}_i, \mathbf{v} \rangle$.
- These tools are useful in the sense that:
  - *BitDecomp* can transform $\mathbf{c}_i \in \mathbb{Z}_q^n$ into a binary vector $\mathbf{c}_i' \in \{0,1\}^{nl}$.
  - If we choose $\mathbf{v}' = \mathit{Powersof2}(\mathbf{v}) \in \mathbb{Z}_q^{nl}$, the result of $\langle \mathbf{c}_i', \mathbf{v}' \rangle = \langle \mathbf{c}_i, \mathbf{v} \rangle$ is maintained.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- *BitDecomp* and *Powersof2* are useful, but not enough!
- Recall that $\mathbf{c}_i \in \mathbb{Z}_q^n$ is the i-th row of $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ and the decryption is based on the result of $\langle \mathbf{c}_i, \mathbf{v} \rangle$.
- These tools are useful in the sense that:
  - *BitDecomp* can transform $\mathbf{c}_i \in \mathbb{Z}_q^n$ into a binary vector $\mathbf{c}_i' \in \{0,1\}^{nl}$.
  - If we choose $\mathbf{v}' = \mathit{Powersof2}(\mathbf{v}) \in \mathbb{Z}_q^{nl}$, the result of $\langle \mathbf{c}_i', \mathbf{v}' \rangle = \langle \mathbf{c}_i, \mathbf{v} \rangle$ is maintained.
- BUT the problems are:

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- *BitDecomp* and *Powersof2* are useful, but not enough!
- Recall that $\mathbf{c}_i \in \mathbb{Z}_q^n$ is the i-th row of $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ and the decryption is based on the result of $\langle \mathbf{c}_i, \mathbf{v} \rangle$.
- These tools are useful in the sense that:
  - *BitDecomp* can transform $\mathbf{c}_i \in \mathbb{Z}_q^n$ into a binary vector $\mathbf{c}_i' \in \{0,1\}^{nl}$.
  - If we choose $\mathbf{v}' = \mathit{Powersof2}(\mathbf{v}) \in \mathbb{Z}_q^{nl}$, the result of $\langle \mathbf{c}_i', \mathbf{v}' \rangle = \langle \mathbf{c}_i, \mathbf{v} \rangle$ is maintained.
- BUT the problems are:
  - *BitDecomp* will transform $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ into a rectangular matrix $\mathbf{C}' \in \{0,1\}^{n \times nl}$.

The "Fix" Tools: *BitDecomp* and *Powersof2*

Proposition

$$\langle \mathit{BitDecomp}(\mathbf{c}), \mathit{Powersof2}(\mathbf{v}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle \pmod{q}$$

- *BitDecomp* and *Powersof2* are useful, but not enough!
- Recall that $\mathbf{c}_i \in \mathbb{Z}_q^n$ is the i-th row of $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ and the decryption is based on the result of $\langle \mathbf{c}_i, \mathbf{v} \rangle$.
- These tools are useful in the sense that:
  - *BitDecomp* can transform $\mathbf{c}_i \in \mathbb{Z}_q^n$ into a binary vector $\mathbf{c}_i' \in \{0,1\}^{nl}$.
  - If we choose $\mathbf{v}' = \mathit{Powersof2}(\mathbf{v}) \in \mathbb{Z}_q^{nl}$, the result of $\langle \mathbf{c}_i', \mathbf{v}' \rangle = \langle \mathbf{c}_i, \mathbf{v} \rangle$ is maintained.
- BUT the problems are:
  - *BitDecomp* will transform $\mathbf{C} \in \mathbb{Z}_q^{n \times n}$ into a rectangular matrix $\mathbf{C}' \in \{0,1\}^{n \times nl}$. We will lose multiplicative homomorphism as two ciphertexts $\mathbf{C}_1, \mathbf{C}_2$ do not support multiplication anymore!
  - The problem size blows up from n to nl.

The "Fix" Tools: *BitDecomp*$^{-1}$ and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

The "Fix" Tools: $BitDecomp^{-1}$ and $Powersof2$

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

- **Tool 3:** $BitDecomp^{-1}(\mathbf{c})$

The "Fix" Tools: $BitDecomp^{-1}$ and $Powersof2$

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

- **Tool 3:** $BitDecomp^{-1}(\mathbf{c})$
  - Reassembles a binary-expanded vector $\mathbf{c} \in \{0,1\}^{k\ell}$ back into its original form over $\mathbb{Z}_q^k$.

The "Fix" Tools: $BitDecomp^{-1}$ and $Powersof2$

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

- **Tool 3:** $BitDecomp^{-1}(\mathbf{c})$
  - Reassembles a binary-expanded vector $\mathbf{c} \in \{0,1\}^{k\ell}$ back into its original form over $\mathbb{Z}_q^k$.
  - Crucially, it works *even if* $\mathbf{c}$ contains integers larger than 1.

The "Fix" Tools: $BitDecomp^{-1}$ and $Powersof2$

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

- **Tool 3:** $BitDecomp^{-1}(\mathbf{c})$
  - Reassembles a binary-expanded vector $\mathbf{c} \in \{0,1\}^{k\ell}$ back into its original form over $\mathbb{Z}_q^k$.
  - Crucially, it works *even if* $\mathbf{c}$ contains integers larger than 1.
- **The Flattening Operation:**

The "Fix" Tools: $BitDecomp^{-1}$ and $Powersof2$

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

- **Tool 3:** $BitDecomp^{-1}(\mathbf{c})$
  - Reassembles a binary-expanded vector $\mathbf{c} \in \{0,1\}^{k\ell}$ back into its original form over $\mathbb{Z}_q^k$.
  - Crucially, it works *even if* $\mathbf{c}$ contains integers larger than 1.
- **The Flattening Operation:**

$$Flatten(\mathbf{c}) = BitDecomp(BitDecomp^{-1}(\mathbf{c}))$$

The "Fix" Tools: $BitDecomp^{-1}$ and *Powersof2*

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

- **Tool 3:** $BitDecomp^{-1}(\mathbf{c})$
  - Reassembles a binary-expanded vector $\mathbf{c} \in \{0,1\}^{k\ell}$ back into its original form over $\mathbb{Z}_q^k$.
  - Crucially, it works *even if* $\mathbf{c}$ contains integers larger than 1.
- **The Flattening Operation:**

$$Flatten(\mathbf{c}) = BitDecomp(BitDecomp^{-1}(\mathbf{c}))$$

- It first computes an integer vector in $\mathbb{Z}_q^k$ that corresponds to a binary vector, and then expands it into that corresponding binary vector in $\{0,1\}^{k\ell}$.

The "Fix" Tools: $BitDecomp^{-1}$ and $Powersof2$

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

- **Tool 3:** $BitDecomp^{-1}(\mathbf{c})$
  - Reassembles a binary-expanded vector $\mathbf{c} \in \{0,1\}^{k\ell}$ back into its original form over $\mathbb{Z}_q^k$.
  - Crucially, it works *even if* $\mathbf{c}$ contains integers larger than 1.
- **The Flattening Operation:**

$$Flatten(\mathbf{c}) = BitDecomp(BitDecomp^{-1}(\mathbf{c}))$$

- It first computes an integer vector in $\mathbb{Z}_q^k$ that corresponds to a binary vector, and then expands it into that corresponding binary vector in $\{0,1\}^{k\ell}$.
- We also assume $\mathbf{v} = Powersof2(\mathbf{s}) \in \mathbb{Z}_q^{k\ell}$ for some $\mathbf{s} \in \mathbb{Z}_q^k$ to match the size.

The "Fix" Tools: $BitDecomp^{-1}$ and $Powersof2$

To prevent $\mathbf{C}_1\mathbf{e}_2$ from exploding, our goal is to constrain $\mathbf{C}_1$ to be binary while maintaining its original size.

- **Tool 3:** $BitDecomp^{-1}(\mathbf{c})$
  - Reassembles a binary-expanded vector $\mathbf{c} \in \{0,1\}^{k\ell}$ back into its original form over $\mathbb{Z}_q^k$.
  - Crucially, it works *even if* $\mathbf{c}$ contains integers larger than 1.
- **The Flattening Operation:**

$$Flatten(\mathbf{c}) = BitDecomp(BitDecomp^{-1}(\mathbf{c}))$$

- It first computes an integer vector in $\mathbb{Z}_q^k$ that corresponds to a binary vector, and then expands it into that corresponding binary vector in $\{0,1\}^{k\ell}$.
- We also assume $\mathbf{v} = Powersof2(\mathbf{s}) \in \mathbb{Z}_q^{k\ell}$ for some $\mathbf{s} \in \mathbb{Z}_q^k$ to match the size.
- The size is not increased because we can choose a smaller k such that $k\ell = n$.

Proof of Invariance for Decryption

Does flatten $\mathbf{C}$ affect the decryption correctness?

Proof of Invariance for Decryption

Does flatten $\mathbf{C}$ affect the decryption correctness?

Let's trace a single row $\mathbf{c}$ and the key $\mathbf{v} = \mathit{Powersof2}(\mathbf{s})$:

Proof of Invariance for Decryption

Does flatten $\mathbf{C}$ affect the decryption correctness?

Let's trace a single row $\mathbf{c}$ and the key $\mathbf{v} = \mathit{Powersof2}(\mathbf{s})$:

$$\langle \mathit{Flatten}(\mathbf{c}), \mathbf{v} \rangle = \langle \mathit{Flatten}(\mathbf{c}), \mathit{Powersof2}(\mathbf{s}) \rangle$$

Proof of Invariance for Decryption

Does flatten $\mathbf{C}$ affect the decryption correctness?

Let's trace a single row $\mathbf{c}$ and the key $\mathbf{v} = \mathit{Powersof2}(\mathbf{s})$:

$$\begin{aligned}\langle \mathit{Flatten}(\mathbf{c}), \mathbf{v} \rangle &= \langle \mathit{Flatten}(\mathbf{c}), \mathit{Powersof2}(\mathbf{s}) \rangle \\ &= \langle \mathit{BitDecomp}^{-1}(\mathit{Flatten}(\mathbf{c})), \mathbf{s} \rangle\end{aligned}$$

Proof of Invariance for Decryption

Does flatten $\mathbf{C}$ affect the decryption correctness?

Let's trace a single row $\mathbf{c}$ and the key $\mathbf{v} = \mathit{Powersof2}(\mathbf{s})$:

$$\begin{aligned}\langle \mathit{Flatten}(\mathbf{c}), \mathbf{v}\rangle &= \langle \mathit{Flatten}(\mathbf{c}), \mathit{Powersof2}(\mathbf{s})\rangle \\ &= \langle \mathit{BitDecomp}^{-1}(\mathit{Flatten}(\mathbf{c})), \mathbf{s}\rangle \\ &= \langle \mathit{BitDecomp}^{-1}(\mathit{BitDecomp}(\mathit{BitDecomp}^{-1}(\mathbf{c}))), \mathbf{s}\rangle\end{aligned}$$

Proof of Invariance for Decryption

Does flatten $\mathbf{C}$ affect the decryption correctness?

Let's trace a single row $\mathbf{c}$ and the key $\mathbf{v} = \mathit{Powersof2}(\mathbf{s})$:

$$\begin{aligned}
\langle \mathit{Flatten}(\mathbf{c}), \mathbf{v} \rangle &= \langle \mathit{Flatten}(\mathbf{c}), \mathit{Powersof2}(\mathbf{s}) \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathit{Flatten}(\mathbf{c})), \mathbf{s} \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathit{BitDecomp}(\mathit{BitDecomp}^{-1}(\mathbf{c}))), \mathbf{s} \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathbf{c}), \mathbf{s} \rangle
\end{aligned}$$

Proof of Invariance for Decryption

Does flatten $\mathbf{C}$ affect the decryption correctness?

Let's trace a single row $\mathbf{c}$ and the key $\mathbf{v} = \mathit{Powersof2}(\mathbf{s})$:

$$\begin{aligned}
\langle \mathit{Flatten}(\mathbf{c}), \mathbf{v} \rangle &= \langle \mathit{Flatten}(\mathbf{c}), \mathit{Powersof2}(\mathbf{s}) \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathit{Flatten}(\mathbf{c})), \mathbf{s} \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathit{BitDecomp}(\mathit{BitDecomp}^{-1}(\mathbf{c}))), \mathbf{s} \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathbf{c}), \mathbf{s} \rangle \\
&= \langle \mathbf{c}, \mathit{Powersof2}(\mathbf{s}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle
\end{aligned}$$

Proof of Invariance for Decryption

Does flatten $\mathbf{C}$ affect the decryption correctness?

Let's trace a single row $\mathbf{c}$ and the key $\mathbf{v} = \mathit{Powersof2}(\mathbf{s})$:

$$\begin{aligned}
\langle \mathit{Flatten}(\mathbf{c}), \mathbf{v} \rangle &= \langle \mathit{Flatten}(\mathbf{c}), \mathit{Powersof2}(\mathbf{s}) \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathit{Flatten}(\mathbf{c})), \mathbf{s} \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathit{BitDecomp}(\mathit{BitDecomp}^{-1}(\mathbf{c}))), \mathbf{s} \rangle \\
&= \langle \mathit{BitDecomp}^{-1}(\mathbf{c}), \mathbf{s} \rangle \\
&= \langle \mathbf{c}, \mathit{Powersof2}(\mathbf{s}) \rangle = \langle \mathbf{c}, \mathbf{v} \rangle
\end{aligned}$$

Conclusion

$\mathit{Flatten}(\mathbf{C}) \cdot \mathbf{v} = \mathbf{C} \cdot \mathbf{v}$. The matrix $\mathbf{C}$ is converted to binary, but its relationship with $\mathbf{v}$ is preserved!

The Multiplicative Homomorphism: Fixed!

Before:

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

The Multiplicative Homomorphism: Fixed!

Before:

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

The Multiplicative Homomorphism: Fixed!

Before:

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

Now:

$$\mathit{Flatten}(\mathbf{C}_1) \cdot \mathbf{C}_2 \cdot \mathbf{v} = \mathit{Flatten}(\mathbf{C}_1) \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2)$$

The Multiplicative Homomorphism: Fixed!

Before:

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

Now:

$$\begin{aligned}\mathit{Flatten}(\mathbf{C}_1) \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathit{Flatten}(\mathbf{C}_1) \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathit{Flatten}(\mathbf{C}_1)\mathbf{v}) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2\end{aligned}$$

The Multiplicative Homomorphism: Fixed!

Before:

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

Now:

$$\begin{aligned}\mathit{Flatten}(\mathbf{C}_1) \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathit{Flatten}(\mathbf{C}_1) \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathit{Flatten}(\mathbf{C}_1)\mathbf{v}) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2\end{aligned}$$

The Multiplicative Homomorphism: Fixed!

Before:

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

Now:

$$\begin{aligned}\mathit{Flatten}(\mathbf{C}_1) \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathit{Flatten}(\mathbf{C}_1) \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathit{Flatten}(\mathbf{C}_1)\mathbf{v}) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2)\end{aligned}$$

The Multiplicative Homomorphism: Fixed!

Before:

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

Now:

$$\begin{aligned}\mathit{Flatten}(\mathbf{C}_1) \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathit{Flatten}(\mathbf{C}_1) \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathit{Flatten}(\mathbf{C}_1)\mathbf{v}) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2)\end{aligned}$$

The Multiplicative Homomorphism: Fixed!

Before:

$$\begin{aligned}\mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathbf{C}_1 \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathbf{C}_1 \mathbf{v}) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathbf{C}_1 \mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathbf{C}_1 \mathbf{e}_2)\end{aligned}$$

Now:

$$\begin{aligned}\mathit{Flatten}(\mathbf{C}_1) \cdot \mathbf{C}_2 \cdot \mathbf{v} &= \mathit{Flatten}(\mathbf{C}_1) \cdot (\mu_2 \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2(\mathit{Flatten}(\mathbf{C}_1)\mathbf{v}) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2 \\ &= \mu_2(\mu_1 \mathbf{v} + \mathbf{e}_1) + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2 \\ &= \mu_1 \mu_2 \mathbf{v} + (\mu_2 \mathbf{e}_1 + \mathit{Flatten}(\mathbf{C}_1)\mathbf{e}_2)\end{aligned}$$

Noise Growth: Since $\mathit{Flatten}(\mathbf{C}_1)$ is binary, multiplying it by $\mathbf{e}_2$ is merely a *subset sum* of the errors for each row. This is affordable!

Pros and Cons of the FHE in [GSW13]

Pros and Cons of the FHE in [GSW13]

The Advantages:

Pros and Cons of the FHE in [GSW13]

The Advantages:

- *Conceptual Intuition:* Homomorphic operations map directly to standard matrix addition and multiplication.

Pros and Cons of the FHE in [GSW13]

The Advantages:

- *Conceptual Intuition:* Homomorphic operations map directly to standard matrix addition and multiplication.
- *Stable Dimension:* The size of ciphertexts does not increase throughout the computation.

Pros and Cons of the FHE in [GSW13]

The Advantages:

- *Conceptual Intuition:* Homomorphic operations map directly to standard matrix addition and multiplication.
- *Stable Dimension:* The size of ciphertexts does not increase throughout the computation.

The Disadvantages:

Pros and Cons of the FHE in [GSW13]

The Advantages:

- *Conceptual Intuition:* Homomorphic operations map directly to standard matrix addition and multiplication.
- *Stable Dimension:* The size of ciphertexts does not increase throughout the computation.

The Disadvantages:

- *Large Ciphertext Size:* Encoding a single-bit message into a large matrix ciphertext is less space-efficient.

Pros and Cons of the FHE in [GSW13]

The Advantages:

- *Conceptual Intuition:* Homomorphic operations map directly to standard matrix addition and multiplication.
- *Stable Dimension:* The size of ciphertexts does not increase throughout the computation.

The Disadvantages:

- *Large Ciphertext Size:* Encoding a single-bit message into a large matrix ciphertext is less space-efficient.
- *High Computational Cost:* Matrix multiplication is much slower than basic scalar or polynomial arithmetic.

Pros and Cons of the FHE in [GSW13]

The Advantages:

- *Conceptual Intuition:* Homomorphic operations map directly to standard matrix addition and multiplication.
- *Stable Dimension:* The size of ciphertexts does not increase throughout the computation.

The Disadvantages:

- *Large Ciphertext Size:* Encoding a single-bit message into a large matrix ciphertext is less space-efficient.
- *High Computational Cost:* Matrix multiplication is much slower than basic scalar or polynomial arithmetic.
- *Asymmetric Noise Growth:* The error growth is order-dependent ($\mathbf{C}_1\mathbf{C}_2$ vs. $\mathbf{C}_2\mathbf{C}_1$), meaning the sequence of operations affects the final noise.

Summary

1. The Basic Idea of the Encryption Scheme in [GSW13]
2. Why the Basic Idea Fails the Multiplicative Homomorphism
3. Use *Flatten* Tools to Fix the Multiplicative Homomorphism