BGV and BFV Bootstrapping: History, State-of-the-Art, and Future Perspectives

Robin Geelen

COSIC, KU Leuven
robin.geelen@esat.kuleuven.be

May 25, 2024

Outline

1. Introduction to BGV and BFV
2. Bootstrapping framework
3. Linear transformations
4. Digit extraction
5. Implementations
6. Recent developments

Outline

BGV and BFV basics

- Fully homomorphic encryption for finite fields and finite rings
- Computation over $\mathbb{Z}_t = \mathbb{Z}/t\mathbb{Z}$
  - We call t the **plaintext modulus**
  - Typical setting: $t = p^r$ for prime number p and positive integer r

BGV and BFV basics

- Fully homomorphic encryption for finite fields and finite rings
- Computation over $\mathbb{Z}_t = \mathbb{Z}/t\mathbb{Z}$
  - We call t the **plaintext modulus**
  - Typical setting: $t = p^r$ for prime number p and positive integer r
- Actual computation is done in ring extension $\mathcal{R}_t \supset \mathbb{Z}_t$
- Based on Ring-LWE encryption

BGV and BFV basics

- Ring-LWE encryption is inherently noisy
  - **Message** or **plaintext** component m
  - **Noise** or **error** component e

BGV and BFV basics

- Ring-LWE encryption is inherently noisy
  - **Message** or **plaintext** component m
  - **Noise** or **error** component e
- Message can be in least or most significant bits:

BGV:

BGV and BFV basics

- Ring-LWE encryption is inherently noisy
  - **Message** or **plaintext** component m
  - **Noise** or **error** component e
- Message can be in least or most significant bits:

BGV:

BFV:

BGV and BFV basics

- Ring-LWE encryption is inherently noisy
  - **Message** or **plaintext** component m
  - **Noise** or **error** component e
- Message can be in least or most significant bits:

BGV:

BFV:

- Only minor differences between two schemes!
- We will use BFV encoding

BFV encryption and decryption

- Ring-LWE encryption uses **cyclotomic ring** $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$
  - Example: $\Phi_m(X) = X^{m/2} + 1$ if m is a power of two
  - Ring elements are written in bold letters (e.g., secret key $\boldsymbol{s}$)

BFV encryption and decryption

- Ring-LWE encryption uses **cyclotomic ring** $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$
  - Example: $\Phi_m(X) = X^{m/2} + 1$ if m is a power of two
  - Ring elements are written in bold letters (e.g., secret key $\boldsymbol{s}$)
- Encryption uses Ring-LWE with **ciphertext modulus** $q \gg t$:

$$(\boldsymbol{c}_0, \boldsymbol{c}_1) = \left(\left\lfloor \frac{q}{t} \cdot \boldsymbol{m} \right\rceil + \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}, -\boldsymbol{a} \right) \pmod{q}$$

$$= \boxed{\boldsymbol{m}} \boxed{} \boxed{\boldsymbol{e}}$$

BFV encryption and decryption

- Ring-LWE encryption uses **cyclotomic ring** $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$
  - Example: $\Phi_m(X) = X^{m/2} + 1$ if m is a power of two
  - Ring elements are written in bold letters (e.g., secret key $\boldsymbol{s}$)
- Encryption uses Ring-LWE with **ciphertext modulus** $q \gg t$:

$$(\boldsymbol{c}_0, \boldsymbol{c}_1) = \left(\left\lfloor \frac{q}{t} \cdot \boldsymbol{m} \right\rceil + \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}, -\boldsymbol{a} \right) \pmod q$$

$$= \boxed{\boldsymbol{m}} \boxed{} \boxed{\boldsymbol{e}}$$

- Decryption via scale-and-round:

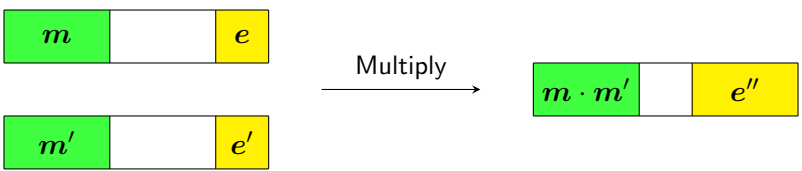
$$\boxed{\boldsymbol{m}} \boxed{} \boxed{\boldsymbol{e}} \xrightarrow{\text{Decrypt}} \boldsymbol{m} = \left\lfloor \frac{t}{q} \cdot (\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) \right\rceil$$

BFV operations

- Operations are defined for $\boldsymbol{m}, \boldsymbol{m}' \in \mathcal{R}_t$
  - Addition: $\mathsf{Enc}(\boldsymbol{m} + \boldsymbol{m}') = \mathsf{Enc}(\boldsymbol{m}) + \mathsf{Enc}(\boldsymbol{m}')$
  - Multiplication: $\mathsf{Enc}(\boldsymbol{m} \cdot \boldsymbol{m}') = \mathsf{Enc}(\boldsymbol{m}) \cdot \mathsf{Enc}(\boldsymbol{m}')$
  - Automorphism: $\mathsf{Enc}(\tau(\boldsymbol{m})) = \tau(\mathsf{Enc}(\boldsymbol{m}))$ for $\tau \in \mathrm{Aut}(\mathcal{R}/\mathbb{Z})$

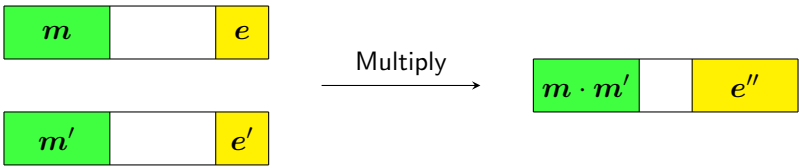
BFV operations

- Operations are defined for $\boldsymbol{m}, \boldsymbol{m}' \in \mathcal{R}_t$
  - Addition: $\mathsf{Enc}(\boldsymbol{m} + \boldsymbol{m}') = \mathsf{Enc}(\boldsymbol{m}) + \mathsf{Enc}(\boldsymbol{m}')$
  - Multiplication: $\mathsf{Enc}(\boldsymbol{m} \cdot \boldsymbol{m}') = \mathsf{Enc}(\boldsymbol{m}) \cdot \mathsf{Enc}(\boldsymbol{m}')$
  - Automorphism: $\mathsf{Enc}(\tau(\boldsymbol{m})) = \tau(\mathsf{Enc}(\boldsymbol{m}))$ for $\tau \in \mathrm{Aut}(\mathcal{R}/\mathbb{Z})$
- Each operation causes noise growth:

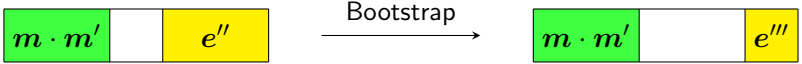


BFV operations

- Operations are defined for $\boldsymbol{m}, \boldsymbol{m}' \in \mathcal{R}_t$
  - Addition: $\mathsf{Enc}(\boldsymbol{m} + \boldsymbol{m}') = \mathsf{Enc}(\boldsymbol{m}) + \mathsf{Enc}(\boldsymbol{m}')$
  - Multiplication: $\mathsf{Enc}(\boldsymbol{m} \cdot \boldsymbol{m}') = \mathsf{Enc}(\boldsymbol{m}) \cdot \mathsf{Enc}(\boldsymbol{m}')$
  - Automorphism: $\mathsf{Enc}(\tau(\boldsymbol{m})) = \tau(\mathsf{Enc}(\boldsymbol{m}))$ for $\tau \in \mathrm{Aut}(\mathcal{R}/\mathbb{Z})$
- Each operation causes noise growth:



- Bootstrapping reduces noise:



BFV operations

	Execution time	Noise growth
Addition		
Ptxt-ctxt multiplication		
Ctxt-ctxt multiplication		
Automorphism		
Bootstrapping		

Plaintext slots

- BFV can encode multiple elements in a ciphertext
- Assume that $t = p^r$ and $\gcd(m, t) = 1$ for prime number p and positive integer r

Plaintext slots

- BFV can encode multiple elements in a ciphertext
- Assume that $t = p^r$ and $\gcd(m, t) = 1$ for prime number p and positive integer r
- Apply Chinese remainder theorem to pack ℓ elements of Galois ring $E \supseteq \mathbb{Z}_{p^r}$
  - Each copy of E is called a **plaintext slot**
  - Full plaintext space is $\mathcal{R}_{p^r} \cong E^\ell$

Plaintext slots

- BFV can encode multiple elements in a ciphertext
- Assume that $t = p^r$ and $\gcd(m, t) = 1$ for prime number p and positive integer r
- Apply Chinese remainder theorem to pack ℓ elements of Galois ring $E \supseteq \mathbb{Z}_{p^r}$
  - Each copy of E is called a **plaintext slot**
  - Full plaintext space is $\mathcal{R}_{p^r} \cong E^\ell$
- Large ℓ is usually preferred

Plaintext slots: permutations

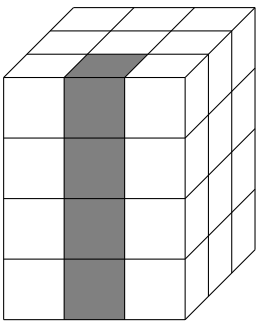
- Automorphism group is $\mathrm{Aut}(\mathcal{R}/\mathbb{Z}) \cong \mathbb{Z}_m^*$

Plaintext slots: permutations

- Automorphism group is $\mathrm{Aut}(\mathcal{R}/\mathbb{Z}) \cong \mathbb{Z}_m^*$
- It can be divided in two components
  1. Slot-wise automorphisms represented by $\langle p \rangle$
     - Generated by Frobenius map $\sigma = \tau_p \colon X \mapsto X^p$
     - There are d slot-wise automorphisms in total

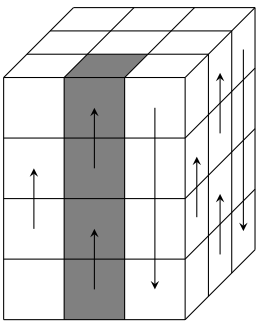
Plaintext slots: permutations

- Automorphism group is $\mathrm{Aut}(\mathcal{R}/\mathbb{Z}) \cong \mathbb{Z}_m^*$
- It can be divided in two components
  1. Slot-wise automorphisms represented by $\langle p \rangle$
     - Generated by Frobenius map $\sigma = \tau_p \colon X \mapsto X^p$
     - There are d slot-wise automorphisms in total
  2. Interslot permutations represented by $\mathbb{Z}_m^*/\langle p \rangle$
     - Let $\mathbb{Z}_m^*/\langle p \rangle = \langle g_1 \rangle \times \ldots \times \langle g_n \rangle$
     - Slots are arranged in n-dimensional **hypercube**
     - Dimension sizes are $\ell_1 \times \ldots \times \ell_n$ so that $\ell = \ell_1 \cdot \ldots \cdot \ell_n$



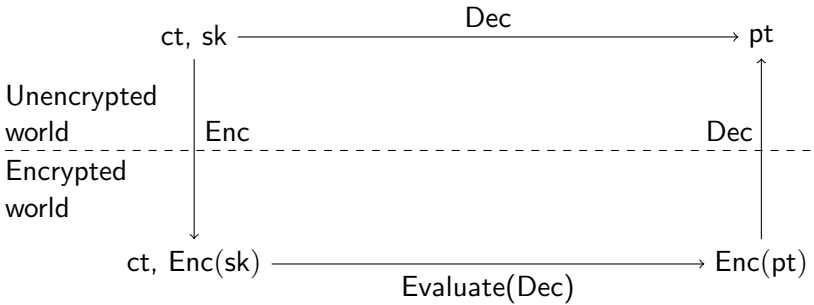
Plaintext slots: permutations

- Automorphism group is $\mathrm{Aut}(\mathcal{R}/\mathbb{Z}) \cong \mathbb{Z}_m^*$
- It can be divided in two components
  1. Slot-wise automorphisms represented by $\langle p \rangle$
     - Generated by Frobenius map $\sigma = \tau_p \colon X \mapsto X^p$
     - There are d slot-wise automorphisms in total
  2. Interslot permutations represented by $\mathbb{Z}_m^*/\langle p \rangle$
     - Let $\mathbb{Z}_m^*/\langle p \rangle = \langle g_1 \rangle \times \ldots \times \langle g_n \rangle$
     - Slots are arranged in n-dimensional **hypercube**
     - Dimension sizes are $\ell_1 \times \ldots \times \ell_n$ so that $\ell = \ell_1 \cdot \ldots \cdot \ell_n$
     - Basic operation is circular rotation ρ_i along dimension i
     - Rotation uses one or two automorphisms



Outline

Bootstrapping basics



BFV bootstrapping

- Plaintext can be **fully packed** (from E^ℓ) or **sparsely packed** (from $\mathbb{Z}_{p^r}^\ell$)

BFV bootstrapping

- Plaintext can be **fully packed** (from E^ℓ) or **sparsely packed** (from $\mathbb{Z}_{p^r}^\ell$)
- Evaluate homomorphic decryption: let's start from the equation

$$\boldsymbol{m} = \left\lfloor \frac{t}{q} \cdot (\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) \right\rceil$$

BFV bootstrapping

- Plaintext can be **fully packed** (from E^ℓ) or **sparsely packed** (from $\mathbb{Z}_{p^r}^\ell$)
- Evaluate homomorphic decryption: let's start from the equation

$$\boldsymbol{m} = \left\lfloor \frac{t}{q} \cdot (\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) \right\rceil$$

- Idea is to take $t = p^r$ and $q = p^e$ for some $e > r$, then decryption becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} \pmod{p^e} \quad \text{and} \quad \boldsymbol{m} \leftarrow \lfloor \boldsymbol{w}/p^{e-r} \rceil \pmod{p^r}$$

BFV bootstrapping

- Plaintext can be **fully packed** (from E^ℓ) or **sparsely packed** (from $\mathbb{Z}_{p^r}^\ell$)
- Evaluate homomorphic decryption: let's start from the equation

$$\boldsymbol{m} = \left\lfloor \frac{t}{q} \cdot (\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) \right\rceil$$

- Idea is to take $t = p^r$ and $q = p^e$ for some $e > r$, then decryption becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} \pmod{p^e} \quad \text{and} \quad \boldsymbol{m} \leftarrow \lfloor \boldsymbol{w}/p^{e-r} \rceil \pmod{p^r}$$

- Need homomorphic scale-and-round
  - Temporarily use $\mathcal{R}_{p^e}$ rather than $\mathcal{R}_{p^r}$
  - Coefficient-wise operation

Fully packed bootstrapping

$$\mathsf{Enc}(\boldsymbol{m})$$

$$\downarrow \text{Inner product}$$

$$\mathsf{Enc}(p^{e-r} \cdot \boldsymbol{m} + \boldsymbol{e})$$

$$\downarrow \text{Coefficients to slots}$$

$$\mathsf{Enc}(p^{e-r} \cdot \mu_{\ell i} + e_{\ell i}, \ldots, p^{e-r} \cdot \mu_{\ell(i+1)-1} + e_{\ell(i+1)-1}) \qquad \text{for } i = 0, \ldots, d-1$$

$$\downarrow \text{Digit extractions}$$

$$\mathsf{Enc}(\mu_{\ell i}, \ldots, \mu_{\ell(i+1)-1}) \qquad \text{for } i = 0, \ldots, d-1$$

$$\downarrow \text{Slots to coefficients}$$

$$\mathsf{Enc}(\boldsymbol{m})$$

Thin or slim bootstrapping

$$\mathsf{Enc}(\mu_0, \ldots, \mu_{\ell-1})$$
$$\downarrow \text{Slots to coefficients}$$
$$\mathsf{Enc}(\boldsymbol{m})$$
$$\downarrow \text{Inner product}$$
$$\mathsf{Enc}(p^{e-r} \cdot \boldsymbol{m} + \boldsymbol{e})$$
$$\downarrow \text{Coefficients to slots}$$
$$\mathsf{Enc}(p^{e-r} \cdot \mu_0 + e_0, \ldots, p^{e-r} \cdot \mu_{\ell-1} + e_{\ell-1})$$
$$\downarrow \text{Digit extraction}$$
$$\mathsf{Enc}(\mu_0, \ldots, \mu_{\ell-1})$$

Outline

Slot-to-coefficient transformation

- Given **slot-encoded** ciphertext(s) containing $\mu_0, \ldots, \mu_{N-1}$
- Compute **coefficient-encoded** ciphertext

$$\mathsf{Enc}(\boldsymbol{m}) = \mathsf{Enc}(\mu_0 + \ldots + \mu_{N-1} \cdot X^{N-1})$$

Slot-to-coefficient transformation

- Given **slot-encoded** ciphertext(s) containing $\mu_0, \ldots, \mu_{N-1}$
- Compute **coefficient-encoded** ciphertext

$$\mathsf{Enc}(\boldsymbol{m}) = \mathsf{Enc}(\mu_0 + \ldots + \mu_{N-1} \cdot X^{N-1})$$

- Required functionality: NTT-like evaluation in roots of unity

Slot-to-coefficient transformation

- Given **slot-encoded** ciphertext(s) containing $\mu_0, \ldots, \mu_{N-1}$
- Compute **coefficient-encoded** ciphertext

$$\mathsf{Enc}(\boldsymbol{m}) = \mathsf{Enc}(\mu_0 + \ldots + \mu_{N-1} \cdot X^{N-1})$$

- Required functionality: NTT-like evaluation in roots of unity
- Different method depending on cyclotomic index m
  1. Product of coprime factors
  2. Power of two

Non-power-of-two cyclotomic rings

- Start from **pairwise coprime** factorization $m = m_1 \cdot \ldots \cdot m_n$
- Hypercube is then constructed as

$$\mathbb{Z}_m^*/\langle p \rangle = \mathbb{Z}_{m_1}^*/\langle p \rangle \times \mathbb{Z}_{m_2}^* \times \ldots \times \mathbb{Z}_{m_n}^*$$

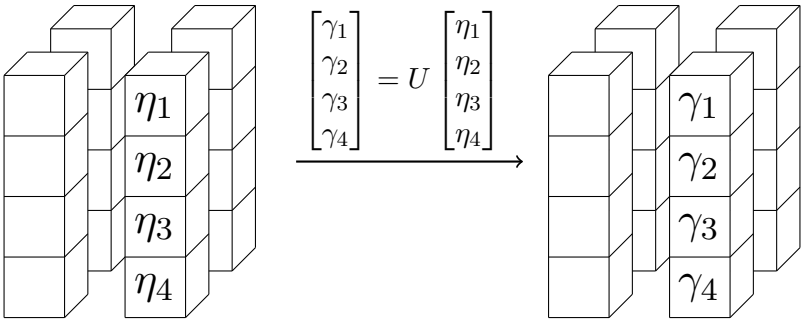
Non-power-of-two cyclotomic rings

- Start from **pairwise coprime** factorization $m = m_1 \cdot \ldots \cdot m_n$
- Hypercube is then constructed as

$$\mathbb{Z}_m^*/\langle p \rangle = \mathbb{Z}_{m_1}^*/\langle p \rangle \times \mathbb{Z}_{m_2}^* \times \ldots \times \mathbb{Z}_{m_n}^*$$

- Idea is to perform Cooley-Tukey algorithm in the slots
  - Each stage of Cooley-Tukey is evaluated in a different hypercube dimension
  - But how can we perform linear transformations in the slots?

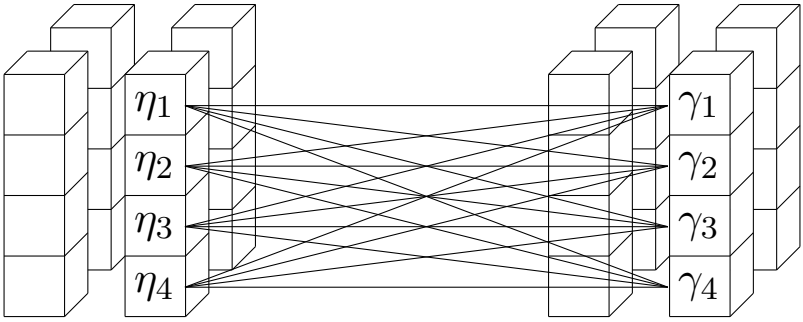
Linear transformation in hypercolumns



Input hypercube

Output hypercube

Linear transformation in hypercolumns



Linear transformation in hypercolumns

- One-dimensional E-linear transformation can be written as

$$\boldsymbol{m} \mapsto \sum_{v=0}^{\ell_i - 1} \boldsymbol{\kappa}_v \cdot \rho_i^v(\boldsymbol{m})$$

- Constants $\boldsymbol{\kappa}_v$ computed from diagonals of matrix $U \in E^{\ell_i \times \ell_i}$

Linear transformation in hypercolumns

- Remember that we have

$$\mathbb{Z}_m^*/\langle p \rangle = \mathbb{Z}_{m_1}^*/\langle p \rangle \times \mathbb{Z}_{m_2}^* \times \ldots \times \mathbb{Z}_{m_n}^*$$

- First stage of fully packed slot-to-coefficient transformation is $\mathbb{Z}_{p^r}$-linear

Linear transformation in hypercolumns

- Remember that we have

$$\mathbb{Z}_m^*/\langle p \rangle = \mathbb{Z}_{m_1}^*/\langle p \rangle \times \mathbb{Z}_{m_2}^* \times \ldots \times \mathbb{Z}_{m_n}^*$$

- First stage of fully packed slot-to-coefficient transformation is $\mathbb{Z}_{p^r}$-linear
- One-dimensional $\mathbb{Z}_{p^r}$-linear transformation can be written as

$$\boldsymbol{m} \mapsto \sum_{v=0}^{\ell_i - 1} \sum_{f=0}^{d-1} \boldsymbol{\kappa}_{v,f} \cdot \sigma^f(\rho_i^v(\boldsymbol{m}))$$

Unpacking and repacking

- Unpack one fully packed ciphertext into d sparsely packed ciphertexts
  - Plaintext comes from E^ℓ for fully packed bootstrapping
  - But digit extraction is done over $\mathbb{Z}_{p^r}^\ell$

Unpacking and repacking

- Unpack one fully packed ciphertext into d sparsely packed ciphertexts
  - Plaintext comes from E^ℓ for fully packed bootstrapping
  - But digit extraction is done over $\mathbb{Z}_{p^r}^\ell$
- Unpacking can be written as

$$\boldsymbol{m}_i = \sum_{f=0}^{d-1} \boldsymbol{\kappa}_{f+i} \cdot \sigma^f(\boldsymbol{m}) \qquad \text{for } i = 0, \ldots, d-1$$

Unpacking and repacking

- Unpack one fully packed ciphertext into d sparsely packed ciphertexts
  - Plaintext comes from E^ℓ for fully packed bootstrapping
  - But digit extraction is done over $\mathbb{Z}_{p^r}^\ell$
- Unpacking can be written as

$$\boldsymbol{m}_i = \sum_{f=0}^{d-1} \boldsymbol{\kappa}_{f+i} \cdot \sigma^f(\boldsymbol{m}) \qquad \text{for } i = 0, \ldots, d-1$$

- Inverse operation (repacking) can be written as

$$\boldsymbol{m} = \sum_{i=0}^{d-1} \boldsymbol{\kappa}'_i \cdot \boldsymbol{m}_i$$

Comparison to CKKS

- Splitting in **small stages** leads to efficient transformation
  - Small hypercube dimensions ℓ_i

Comparison to CKKS

- Splitting in **small stages** leads to efficient transformation
  - Small hypercube dimensions ℓ_i
- Unlike CKKS, we rely on **algebraic properties** of $\mathbb{Z}_m^*/\langle p \rangle$ to achieve this
  - Extra parameter p makes everything more complicated

Comparison to CKKS

- Splitting in **small stages** leads to efficient transformation
  - Small hypercube dimensions ℓ_i
- Unlike CKKS, we rely on **algebraic properties** of $\mathbb{Z}_m^*/\langle p \rangle$ to achieve this
  - Extra parameter p makes everything more complicated
  - This is because the choice of p also influences other things
    - Number of slots
    - Efficiency of digit extraction
    - Asymptotic noise growth

Comparison to CKKS

- Splitting in **small stages** leads to efficient transformation
  - Small hypercube dimensions ℓ_i
- Unlike CKKS, we rely on **algebraic properties** of $\mathbb{Z}_m^*/\langle p \rangle$ to achieve this
  - Extra parameter p makes everything more complicated
  - This is because the choice of p also influences other things
    - Number of slots
    - Efficiency of digit extraction
    - Asymptotic noise growth
  - Non-power-of-two m offers more flexibility

Power-of-two cyclotomic rings

- Three-step approach
  1. Perform slot-wise linear transformation M
  2. Multiply by matrix U (similar decomposition as CKKS)
  3. Perform slot-wise linear transformation M^{-1}

Power-of-two cyclotomic rings

- Three-step approach
  1. Perform slot-wise linear transformation M
  2. Multiply by matrix U (similar decomposition as CKKS)
  3. Perform slot-wise linear transformation M^{-1}
- Unpacking and repacking are slightly more efficient than previously

Outline

Digit extraction or removal procedure

- Given slot-encoded ciphertext containing $\mu_i = p^{e-r} \cdot \mu_i' + e_i \pmod{p^e}$
- Compute slot-encoded ciphertext containing $\mu_i' \pmod{p^r}$

Digit extraction or removal procedure

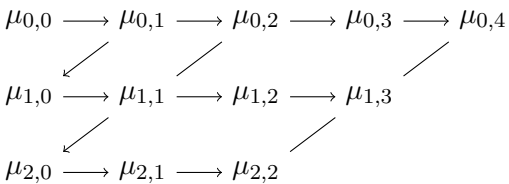
- Given slot-encoded ciphertext containing $\mu_i = p^{e-r} \cdot \mu_i' + e_i \pmod{p^e}$
- Compute slot-encoded ciphertext containing $\mu_i' \pmod{p^r}$
- This is done via **homomorphic scale-and-round**

$$\mathbb{Z}_{p^e} \to \mathbb{Z}_{p^r} : \mu_i \mapsto \left\lfloor \frac{\mu_i}{p^{e-r}} \right\rceil$$

and works correctly if $|e_i| < p^{e-r}/2$

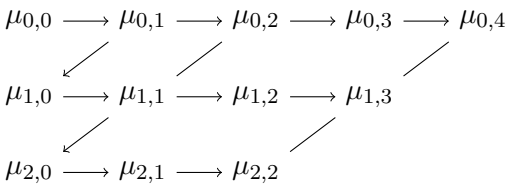
Original bit extraction [GHS12]

- **Repeated squaring** procedure for $p = 2$
  - First element $\mu_{0,0}$ given as input
  - Each element is square of its left neighbor
  - Computation of leading elements and result via diagonal



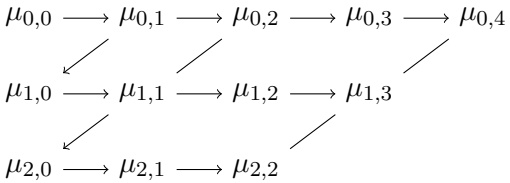
Original bit extraction [GHS12]

- **Repeated squaring** procedure for $p = 2$
  - First element $\mu_{0,0}$ given as input
  - Each element is square of its left neighbor
  - Computation of leading elements and result via diagonal
- Repeated squaring iteratively extracts the least significant bit



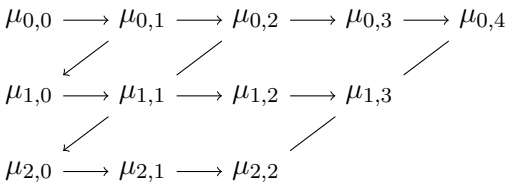
Bit extraction improvements

- Generalization to arbitrary p: digit extraction [HS15]
  - Replace squaring by **lifting polynomial** $F_e(X)$



Bit extraction improvements

- Generalization to arbitrary p: digit extraction [HS15]
  - Replace squaring by **lifting polynomial** $F_e(X)$
- Computation of $\mu_{i,j}$ directly from leading element $\mu_{i,0}$ [CH18]
  - Replace lifting polynomial by **digit extraction polynomial** $G_e(X)$



Generalization to arbitrary p

Definition

Denote by μ_0 the least significant digit of $\mu \in \mathbb{Z}_{p^e}$ in its base-p expansion, then *digit extraction* is the map

$$g_e \colon \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e} \colon \mu \mapsto \mu_0$$

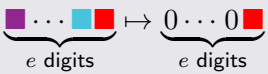
$$\underbrace{\blacksquare \cdots \blacksquare\blacksquare}_{e \text{ digits}} \mapsto \underbrace{0 \cdots 0\blacksquare}_{e \text{ digits}}$$

Generalization to arbitrary p

Definition

Denote by μ_0 the least significant digit of $\mu \in \mathbb{Z}_{p^e}$ in its base-p expansion, then *digit extraction* is the map

$$g_e \colon \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e} \colon \mu \mapsto \mu_0$$



- We can only evaluate polynomials in FHE
- So how can we realize digit extraction with a polynomial?

Polyfunctions

Definition

Function $f : \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e}$ is a *polyfunction* if there exists $F(X) \in \mathbb{Z}[X]$ s.t.

$$F(a) = f(a) \pmod{p^e}$$

for each $a \in \mathbb{Z}$. We call $F(X)$ a *representation* of f.

Polyfunctions

Definition

Function $f : \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e}$ is a *polyfunction* if there exists $F(X) \in \mathbb{Z}[X]$ s.t.

$$F(a) = f(a) \pmod{p^e}$$

for each $a \in \mathbb{Z}$. We call $F(X)$ a *representation* of f.

If $e = 1$

- $\mathbb{Z}_{p^e}$ is a field
- Every function is a polyfunction
- Unique lowest-degree representation

Polyfunctions

Definition

Function $f : \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e}$ is a *polyfunction* if there exists $F(X) \in \mathbb{Z}[X]$ s.t.

$$F(a) = f(a) \pmod{p^e}$$

for each $a \in \mathbb{Z}$. We call $F(X)$ a *representation* of f.

If $e = 1$

- $\mathbb{Z}_{p^e}$ is a field
- Every function is a polyfunction
- Unique lowest-degree representation

If $e > 1$

- $\mathbb{Z}_{p^e}$ is not a field
- Not every function is a polyfunction
- No unique representation

Digit extraction representations

- Digit extraction g_e is a polyfunction with (non-unique) representation $G_e(X)$

Digit extraction representations

- Digit extraction g_e is a polyfunction with (non-unique) representation $G_e(X)$

Representations of g_e for $p = 2$ and $e = 8$

- Halevi and Shoup [HS15] perform repeated squaring and find

$$G_8^{HS}(X) = X^{2^7}$$

- Chen and Han [CH18] find a lowest degree representation

$$G_8^{CH}(X) = 13X^8 + 96X^7 + 84X^6 + 32X^5 + 32X^4$$

Digit extraction representations

- Digit extraction g_e is a polyfunction with (non-unique) representation $G_e(X)$

Representations of g_e for $p = 2$ and $e = 8$

- Halevi and Shoup [HS15] perform repeated squaring and find

$$G_8^{HS}(X) = X^{2^7}$$

- Chen and Han [CH18] find a lowest degree representation

$$G_8^{CH}(X) = 13X^8 + 96X^7 + 84X^6 + 32X^5 + 32X^4$$

Their difference satisfies $\underbrace{G_8^{HS}(X) - G_8^{CH}(X)}_{\text{Null polynomial}} \equiv 0 \pmod{2^8}$

Null polynomials and equivalent representations

- A **null polynomial** $O(X)$ evaluates the zero function modulo p^e

$$g_e \quad \Longleftrightarrow \quad \{G_e(X) + O(X)\}$$

- Add null polynomial to find FHE-friendly representation

Null polynomials and equivalent representations

- A **null polynomial** $O(X)$ evaluates the zero function modulo p^e

$$g_e \quad \Longleftrightarrow \quad \{G_e(X) + O(X)\}$$

- Add null polynomial to find FHE-friendly representation
- But how do we find these null polynomials?

Finding null polynomials

- Example: all multiples of $X^p - X$ are null polynomials modulo p

Finding null polynomials

- Example: all multiples of $X^p - X$ are null polynomials modulo p
- More complicated for $e > 1$
  - Define **falling factorial polynomials** as $(X)_i = X(X-1) \cdot \ldots \cdot (X-i+1)$
  - Evaluation of $(X)_i$ at any integer is divisible by $i!$

Finding null polynomials

- Example: all multiples of $X^p - X$ are null polynomials modulo p
- More complicated for $e > 1$
  - Define **falling factorial polynomials** as $(X)_i = X(X-1) \cdot \ldots \cdot (X-i+1)$
  - Evaluation of $(X)_i$ at any integer is divisible by $i!$
  - The set of all null polynomials contains
    - $(X)_i$ if p^e divides $i!$
    - $p^n \cdot (X)_i$ otherwise (with n s.t. p^{e-n} divides $i!$)
    - $\mathbb{Z}$-linear combinations of the above

Optimization metrics

- We can optimize polynomial representations by adding null polynomials
- It can be interesting to optimize the following properties
  - Small polynomial degree
  - Few non-zero coefficients
  - Small coefficient size

Optimizing polynomial degree

- Let $O(X)$ be a monic null polynomial of the lowest degree
- Apply **Euclidean division** on any representation $G_e(X)$ to obtain

$$G_e(X) = O(X) \cdot Q(X) + G'_e(X)$$

Optimizing polynomial degree

- Let $O(X)$ be a monic null polynomial of the lowest degree
- Apply **Euclidean division** on any representation $G_e(X)$ to obtain

$$G_e(X) = O(X) \cdot Q(X) + G'_e(X)$$

- Note that $G'_e(X)$ is a representation of degree less than $\deg(O(X)) \leq p \cdot e$

Optimizing non-zero coefficients

- Digit extraction is a **symmetric function**
- Representation with only even- or odd-exponent terms
  - If $p > 2$ then $g_e(-a) = -g_e(a)$, and thus

$$G'_e(X) = (G_e(X) - G_e(-X))/2$$

Optimizing non-zero coefficients

- Digit extraction is a **symmetric function**
- Representation with only even- or odd-exponent terms
  - If $p > 2$ then $g_e(-a) = -g_e(a)$, and thus

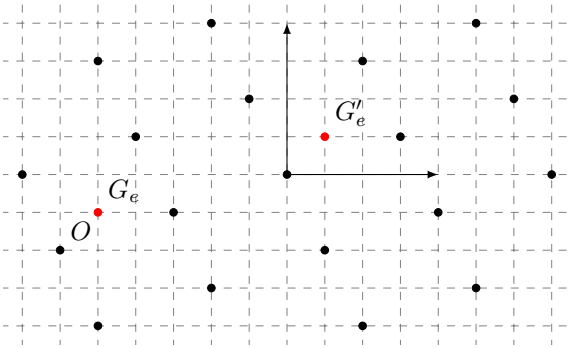
$$G'_e(X) = (G_e(X) - G_e(-X))/2$$

  - If $p = 2$ then $g_e(-a) = g_e(a)$, and thus

$$G'_e(X) = (G_{e+1}(X) + G_{e+1}(-X))/2$$

Optimizing coefficient size

- Null polynomials with degree bound n form an $n+1$-dimensional lattice
- Solve **closest vector problem** to obtain $G'_e(X) = G_e(X) - O(X)$



Example

Representations of g_e for $p = 2$ and $e = 8$

- Halevi and Shoup [HS15] perform repeated squaring and find

$$G_8^{HS}(X) = X^{2^7}$$

- Chen and Han [CH18] find a lowest degree representation

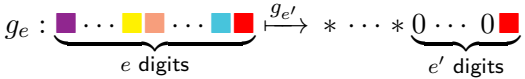
$$G_8^{CH}(X) = 13X^8 + 96X^7 + 84X^6 + 32X^5 + 32X^4$$

- Our techniques [GIKV23] find the polynomial

$$G_8(X) = 13X^8 - 12X^6$$

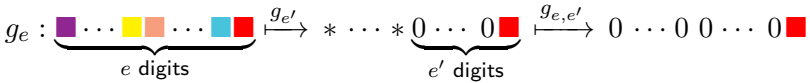
Function composition

- Compose digit extraction function as $g_e = g_{e,e'} \circ g_{e'}$ for some $e' < e$



Function composition

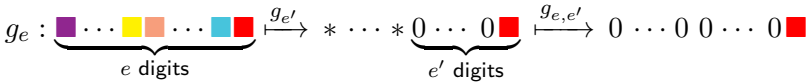
- Compose digit extraction function as $g_e = g_{e,e'} \circ g_{e'}$ for some $e' < e$



- Relevant domain of $g_{e,e'}$ is Range$(g_{e'}) \subset \mathbb{Z}_{p^e}$
  $\implies$ More null polynomials defined over this range

Function composition

- Compose digit extraction function as $g_e = g_{e,e'} \circ g_{e'}$ for some $e' < e$



- Relevant domain of $g_{e,e'}$ is Range$(g_{e'}) \subset \mathbb{Z}_{p^e}$
  $\implies$ More null polynomials defined over this range
- Complexity gain over regular approach
  - Non-scalar multiplications: $\mathcal{O}(\sqrt{pe}) \Rightarrow \mathcal{O}(\sqrt{p}\sqrt[4]{e})$
  - Scalar multiplications: $\mathcal{O}(pe) \Rightarrow \mathcal{O}(p\sqrt{e})$
- Total degree increases with roughly a factor of p

Example

Function composition for $p = 2$, $e = 25$ and $e' = 8$

- We already found the optimized polynomial

$$G_8(X) = 13X^8 - 12X^6$$

- Starting from $G_8(X)$, digit extraction modulo 2^{25} can be done with

$$G_{25,8}(X) = 6X^5 - 15X^4 + 10X^3$$

$\Longrightarrow$ The composition $G_{25,8}(G_8(X))$ gives g_{25}

Outline

Implementation in FHE libraries

- Two implementations with different trade-offs
  - HElib [HS15]: non-power-of-two m and small p
  - SEAL [CH18]: power-of-two m and small p
    - Bootstrapping not publicly released

Implementation in FHE libraries

- Two implementations with different trade-offs
  - HElib [HS15]: non-power-of-two m and small p
  - SEAL [CH18]: power-of-two m and small p
    - Bootstrapping not publicly released
- Note: recently proposed techniques not included in table

	HElib	SEAL
Sufficient plaintext slots	🙂	😐
Efficient linear transformations	🙂	😐
Easy to understand and use	😐	🙂
Optimized NTT algorithm	😐	🙂

Benchmarks for thin or slim bootstrapping [GIKV23]

Cyclotomic index m		$127 \cdot 337$	$101 \cdot 451$
Number of slots		2016	1000
Params (p, r, e)		$(2, 8, 15)$	$(17, 4, 6)$
Capacity (bits)	Initial	1151	1136
	Linear maps	137	174
	Digit extract	260	435
	Remaining	754	527
Execution time (sec)	Linear maps	35	32
	Digit extract	35	46
	Total	70	78

Outline

Recently proposed techniques

	Power-of-two m	Large p	Digit extract	Functional bootstrap
[OPP23]	✓	✗	✓	✗
[LMS24]	✓	✗	✓	✓
[LW24]	✓	✓	✗	✓
[KSS24]	✓	✓	✗	✗
[MHWW24a]	✗	✓	✓	✗
[MHWW24b] [Gee24]	✓	✓	✓	✗

Recently proposed techniques

	Power-of-two m	Large p	Digit extract	Functional bootstrap
[OPP23]	✓	✗	✓	✗
[LMS24]	✓	✗	✓	✓
[LW24]	✓	✓	✗	✓
[KSS24]	✓	✓	✗	✗
[MHWW24a]†	✗	✓	✓	✗
[MHWW24b] [Gee24]	✓	✓	✓	✗

†See Eurocrypt: Accelerating BGV Bootstrapping for Large p Using Null Polynomials over $\mathbb{Z}_{p^e}$

Conclusion and future perspectives

- Non-power-of-two m and small p most widely used today

Conclusion and future perspectives

- Non-power-of-two m and small p most widely used today
- Shift towards **power-of-two cyclotomics**
  - Six out of seven recent papers already use it

Conclusion and future perspectives

- Non-power-of-two m and small p most widely used today
- Shift towards **power-of-two cyclotomics**
  - Six out of seven recent papers already use it
  - Can be combined with large-p bootstrapping
    - Large number of slots possible
    - Efficient digit extraction
    - Reasonable noise growth

Conclusion and future perspectives

- Non-power-of-two m and small p most widely used today
- Shift towards **power-of-two cyclotomics**
  - Six out of seven recent papers already use it
  - Can be combined with large-p bootstrapping
    - Large number of slots possible
    - Efficient digit extraction
    - Reasonable noise growth
- Integration in FHE libraries?

References I


Jacob Alperin-Sheriff and Chris Peikert.
Practical Bootstrapping in Quasilinear Time.
In *Annual Cryptology Conference*, pages 1–20. Springer, 2013.


Hao Chen and Kyoohyung Han.
Homomorphic Lower Digits Removal and Improved FHE Bootstrapping.
In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–337. Springer, 2018.


Robin Geelen.
Revisiting the Slot-to-Coefficient Transformation for BGV and BFV.
Cryptology ePrint Archive, 2024.

References II


Craig Gentry, Shai Halevi, and Nigel P Smart.
Better Bootstrapping in Fully Homomorphic Encryption.
In *International Workshop on Public Key Cryptography*, pages 1–16. Springer, 2012.


Robin Geelen, Ilia Iliashenko, Jiayi Kang, and Frederik Vercauteren.
On Polynomial Functions Modulo p^e and Faster Bootstrapping for Homomorphic Encryption.
In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 257–286. Springer, 2023.


Robin Geelen and Frederik Vercauteren.
Bootstrapping for BGV and BFV Revisited.
Journal of Cryptology, 36(2):12, 2023.

References III

Shai Halevi and Victor Shoup.
Bootstrapping for HElib.
In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 641–670. Springer, 2015.

Jaehyung Kim, Jinyeong Seo, and Yongsoo Song.
Simpler and Faster BFV Bootstrapping for Arbitrary Plaintext Modulus from CKKS.
Cryptology ePrint Archive, 2024.

Dongwon Lee, Seonhong Min, and Yongsoo Song.
Functional Bootstrapping for FV-style Cryptosystems.
Cryptology ePrint Archive, 2024.

Zeyu Liu and Yunhao Wang.
Relaxed Functional Bootstrapping: A New Perspective on BGV/BFV Bootstrapping.
Cryptology ePrint Archive, 2024.

References IV

Shihe Ma, Tairong Huang, Anyu Wang, and Xiaoyun Wang.
Accelerating BGV Bootstrapping for Large p Using Null Polynomials over $\mathbb{Z}_{p^e}$.
In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 403–432. Springer, 2024.

Shihe Ma, Tairong Huang, Anyu Wang, and Xiaoyun Wang.
Faster BGV Bootstrapping for Power-of-Two Cyclotomics through Homomorphic NTT.
Cryptology ePrint Archive, 2024.

Hiroki Okada, Rachel Player, and Simon Pohmann.
Homomorphic Polynomial Evaluation Using Galois Structure and Applications to BFV Bootstrapping.
In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 69–100. Springer, 2023.

1-4 July 2024 in Leuven

COSIC (KU Leuven) is organizing the 17th edition of the international COSIC course on cybersecurity and cryptography. The course will be held in Leuven from 1-4 July 2024.

Thank you for listening!