Faster Amortized FHEW Bootstrapping using Ring Automorphisms

Gabrielle De Micheli (UCSD)
Joint work with: Duhyeong Kim (Intel), Daniele Micciancio and Adam Suhl (UCSD)

PKC 2024

April 2024

Motivation/Goal

Main approaches to FHE Bootstrapping:

Bootstrapping	BGV/BFV	FHEW/TFHE
Message space	Large ($\mathbb{Z}_p^n$)	Small ($\mathbb{Z}_2$)
Latency	Slow (several minutes)	Fast ($<$ 1 second)
Amortized Time	Fast	Slow

Our work:

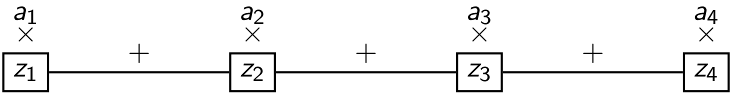
- New algorithm to bootstrap n FHEW ciphertexts with very small overhead ($\ll n$) to a single FHEW bootstrapping.
- Previous work [M., Sorrell, ICALP 2018]: theoretically promising, but impractical due to very high overhead.
- Our work: Similar asymptotic amortized cost but **much smaller** overhead.

Bootstrapping for LWE Ciphertext

Bootstrapping: **Homomorphic** evaluation of **decryption** circuit

- $\mathrm{Dec}_s(a_1, \ldots, a_n, b) = \lfloor b + \sum_{i=1}^{n} a_i \cdot s_i \rceil$
- Bootstrapping keys: $\mathrm{Enc}(z_1), \ldots, \mathrm{Enc}(z_n)$
- $\mathrm{Bootstrap}(a_1, \ldots, a_n, b) =$

$$\left\lfloor b + \sum_{i=1}^{n} a_i \cdot \mathrm{Enc}(z_i) \right\rceil = \mathrm{Enc}\left(\left\lfloor b + \sum_{i=1}^{n} a_i \cdot z_i \right\rceil \right) = \mathrm{Enc}\,(m)$$



FHEW Bootstrapping [Ducas, M., Eurocrypt'15]

- Homomorphic decryption "in the exponent".
- $\text{Enc}(z_i) = \text{RGSW}(X^{z_i})$ (i.e., RGSW register).
- A two-step process:
  1. *Inner Product*[1]:

$$b + \sum_{i=1}^{n} a_i \cdot \text{Enc}(z_i) = \text{RGSW}(X^{b+\sum_{i=1}^{n} a_i \cdot z_i})$$

  2. *Rounding (msbExtract)*:

$$\text{RGSW}(X^{b+\sum_{i=1}^{n} a_i \cdot z_i}) \to \text{LWE}(m)$$

[1] Possible optimization: use RLWE ciphertexts with an external product.

Limitation of LWE bootstrapping

- Cost: $O(n)$ homomorphic multiplications per message.
- Bootstrap n messages: total cost $O(n^2)$ crypto ops.
- Infeasible computational cost in practical FHE parameters (e.g., $n = 2^{14}$).

Amortized Bootstrapping

- Utilize RLWE decryption instead of LWE decryption:

$$\mathrm{Dec}_{\mathbf{z}}(\mathbf{a}, \mathbf{b}) = \lfloor \mathbf{a} \cdot \mathbf{z} + \mathbf{b} \rceil$$

- A three-step process:
  1. *Ring packing:*

     $$\{\mathrm{LWE}_s(m_i)\}_{i=0}^{d-1} \Longrightarrow \mathrm{RLWE}_{\mathbf{z}}(m(X)) \in \mathcal{R}_q^2$$

     with some packing key.
  2. *Inner Product:* For $(\mathbf{a}, \mathbf{b}) = \mathrm{RLWE}_{\mathbf{z}}(m(X))$, homomorphically compute

     $$\mathbf{a} \cdot \boxed{\mathbf{z}} + \mathbf{b}$$

  3. *Rounding (msbExtract):* Compute homomorphic rounding for each coefficient of $\mathbf{a} \cdot \boxed{\mathbf{z}} + \mathbf{b}$.

Amortized Bootstrapping

- Utilize RLWE decryption instead of LWE decryption:

$$\mathsf{Dec}_{\mathbf{z}}(\mathbf{a}, \mathbf{b}) = \lfloor \mathbf{a} \cdot \mathbf{z} + \mathbf{b} \rceil$$

- A three-step process:

2. *Inner Product:* For $(\mathbf{a}, \mathbf{b}) = \mathsf{RLWE}_{\mathbf{z}}(m(X))$, homomorphically compute

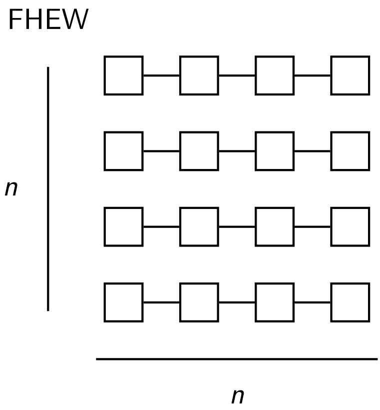
$$\mathbf{a} \cdot \boxed{\mathbf{z}} + \mathbf{b}$$

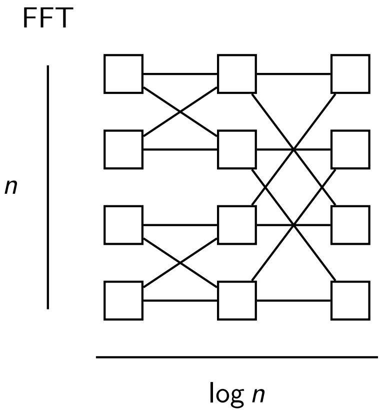
FHEW vs FFT-based Solution

Goal: Compute $\mathbf{a} \cdot \boxed{\mathbf{z}} + \mathbf{b}$ homomorphically.

One Solution: Fast Fourier Transform (over finite field)

- homomorphic operations: addition/subtraction and multiplication by "twiddle" factors powers of a primitive root of unity.
- less homomorphic operations $O(n \log n)$ compared to $O(n^2)$.





Previous work [M., Sorrell, ICALP'18]

The first work on amortized FHEW bootstrapping:

- Bottleneck: RGSW registers only support **homomorphic addition**.
- In theory: bootstrap n messages with $O(3^\ell \cdot n^{1+1/\ell})$ crypto ops, which gives $O(3^\ell \cdot n^{1/\ell})$ amortized cost.
- In practice: Even worse than $O(n)$ sequential FHEW, due to very high overhead.

Our **key observation:**

- we "can" efficiently perform homomorphic scalar multiplication in RGSW register,
- and hence we "can" apply FFT for homomorphic INTT.
- better asymptotic complexity, smaller overhead.

A (very) brief description of FFT

General idea:

- evaluate degree-n polynomials at appropriate values (roots of unity) $O(n \log n)$
- compute pointwise multiplication $O(n)$

Evaluation:

- compute a remainder tree
- each layer corresponds to a reduction of polynomials modulo other polynomials (operations: $a_0 + a_1\zeta + \cdots + a_k\zeta^k$)
- optimize algorithm: regroup the number of layers (radix).

Pointwise multiplication: similar operations needed.

What operations are needed homomorphically?

Notation for encrypted data: $\square$

Question: What operations do we need?

- scalar multiplication : $a \times \square \rightarrow \square$
- addition: $\square + \square \rightarrow \square$

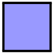
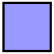
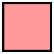
The feasability of these homomorphic operations depends on the encryption schemes considered: GadgetRLWE, RGSW.

More on the homomorphic operations ...

Message encoding: scalar values $v \in \mathbb{Z}_q$ are encoded in the exponent, *i.e.*, mapping it to the monomial X^v.

- In our algorithm, we will work with both RGSW and RLWE' **registers**, i.e., RGSW/RLWE' encryptions of X^v for $v \in \mathbb{Z}_q$.

With the schemes:

- scalar multiplication: $a\times$  $\rightarrow$ automorphisms on GadgetRLWE.
- addition:  $+$  $\rightarrow$ GadgetRLWE $\times$ RGSW multiplication.

Ring automorphisms

- scalar multiplication: $a\times$ ■ $\rightarrow$ automorphisms on GadgetRLWE.

Automorphisms: bijective map from the ring $\mathcal{R}$ to itself : $\mathbf{a}(X) \mapsto \mathbf{a}(X^t)$, $t \in \mathbb{Z}_q^*$.

Automorphisms in RLWE/RLWE' :

- RLWE ciphertext : $(\mathbf{a}(X), \mathbf{b}(X))$ under **sk**.
- $\psi_t : \mathcal{R} \rightarrow \mathcal{R}$, $\mathbf{a}(X) \mapsto \mathbf{a}(X^t)$.
- apply ψ_t to RLWE components:

$$(\mathbf{a}(X^t), \mathbf{b}(X^t)) = \mathrm{RLWE}_{\mathbf{sk}(X^t)}(\mathbf{m}(X^t))$$

- apply key switching function to get $\mathrm{RLWE}_{\mathbf{sk}(X)}(\mathbf{m}(X^t))$

Gadget RLWE × RGSW Multiplication

- addition: $\square + \square \rightarrow$ GadgetRLWE × RGSW multiplication

Multiplication RLWE $\star$ RGSW $\rightarrow$ RLWE:

$$\mathrm{RLWE}_{\mathbf{sk}}(\mathbf{m}_1) \star \mathrm{RGSW}_{\mathbf{sk}}(\mathbf{m}_2) = \mathbf{a} \odot RLWE'_{\mathbf{sk}}(\mathbf{s} \cdot \mathbf{m}_2) + \mathbf{b} \odot RLWE'_{\mathbf{sk}}(\mathbf{m}_2)$$

- This operation allows to **multiply** two ciphertexts, which in the exponent acts like an **addition**.

An additional scheme-switching technique

- If we want to **multiply** a ciphertext by a scalar, we use automorphisms (for RLWE' ciphertexts only!): $X^{a \times c} = (X^c)^a$ (allows homomorphic exponentiation).
- If we want to **add** two ciphertexts (in the exponent) we use RLWE$'$ $\times$ RGSW multiplication: $X^{c_1 + c_2} = X^{c_1} \times X^{c_2}$

A necessary scheme-switch:

- In our algorithm, we primarily use RLWE' registers.
- RGSW is only needed for multiplication.
- We also introduce a novel **scheme-switching** method from RLWE' to RGSW.

Using Fast Fourier Transform

The **second step** of amortized boostrapping is: let $(\mathbf{a}, \mathbf{b}) = \mathsf{RLWE}_{\mathbf{z}}(m(X))$: homomorphically compute decryption, *i.e.*, compute

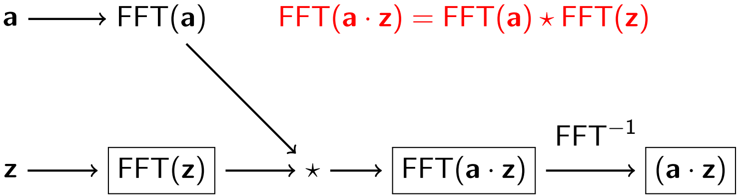
$$\mathbf{a} \cdot \mathbf{z} + \mathbf{b}$$

Main goal: compute a single polynomial multiplication $\mathbf{a} \cdot \mathbf{z}$ using FFT.

Important: We have an **encryption** of $\mathbf{z}$.

FFT-based multiplication algorithm:

$$\mathsf{FFT}(\mathbf{a} \cdot \mathbf{z}) = \mathsf{FFT}(\mathbf{a}) \star \mathsf{FFT}(\mathbf{z})$$



What needs to be computed (homomorphically)

1. Compute an FFT of $\mathbf{a}$ in cleartext form.
2. Evaluation key: contains RGSW registers of $\mathsf{FFT}(\mathbf{z})$.
3. Homomorphically compute $\mathsf{FFT}(\mathbf{a} \cdot \mathbf{z})$ (**pointwise multiplication**)
4. Compute **inverse FFT**: $\mathsf{FFT}^{-1}(\mathbf{a} \cdot \mathbf{z})$.

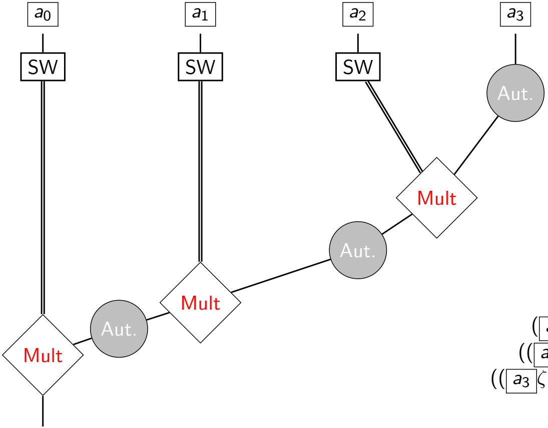
Operations performed:

$\boxed{a_i}\zeta^j \rightarrow$ to scalar multiply in the exponent, use **automorphisms**.
$\boxed{a_i\zeta^j} + \boxed{a_{i'}\zeta^{j'}} \rightarrow$ to add two encrypted data, use **scheme-switching** GadgetRLWE $\rightarrow$ RGSW and then perform a **multiplication** GadgetRLWE $\times$ RGSW.

One layer of FFT: evaluation example

Goal: compute $\boxed{a_0} + \boxed{a_1}\zeta + \boxed{a_2}\zeta^2 + \boxed{a_3}\zeta^3$

- re-write as $\left((\boxed{a_3}\zeta + \boxed{a_2})\zeta + \boxed{a_1}\right)\zeta + \boxed{a_0}$
- $\boxed{a_i}$ are RLWE' ciphertexts



$\boxed{a_3}\zeta$

$\boxed{a_3}\zeta + \boxed{a_2}$

$(\boxed{a_3}\zeta + \boxed{a_2})\zeta$

$(\boxed{a_3}\zeta + \boxed{a_2})\zeta + \boxed{a_1}$

$((\boxed{a_3}\zeta + \boxed{a_2})\zeta + \boxed{a_1})\zeta$

$((\boxed{a_3}\zeta + \boxed{a_2})\zeta + \boxed{a_1})\zeta + \boxed{a_0}$

Overview of the amortized bootstrapping scene

Scheme	Amortized cost	Modulus	Pros	Cons
FHEW/TFHE	$\tilde{O}(n)$	Polynomial	–	Cost
[MS18]	$\tilde{O}(3^{\frac{1}{\epsilon}} \cdot n^{\epsilon})$	Polynomial	Promising...	Large overhead
Our work	$O(\frac{1}{\epsilon} \cdot n^{\epsilon})$	Polynomial	smaller overhead	non-power-2 cycl/Impractical
Guimarães, Pereira, Van Leeuwen, AC23	$O(\frac{1}{\epsilon} \cdot n^{\epsilon})$	Polynomial	smaller overhead	Impractical
Liu, Wang, EC23	$\tilde{O}(n^{.75})$	Polynomial	–	Worse complexity
Liu, Wang, EC23	$\tilde{O}(1)$	Polynomial	Good complexity	Impractical
Liu, Wang, AC23	$\tilde{O}(1)$	Super-polynomial	Best practical perf.	Large modulus

Conclusion and future work

Where we stand now:

- New methods to amortize FHEW bootstrapping overcoming practical limitations of [MS'18].
- No clear winning candidate in terms of practical performance.
- Performance gap between amortized FHEW and BGV/BFV.

What about an efficient implementation?

- Much needed: (better) support for general cyclotomics (other than powers-of-two) in FHE libraries.

Thank you !

What encryption schemes to use ?

Let $\mathcal{R}_q = q^{th}$ prime cyclotomic ring, q prime.

- **GadgetRLWE (RLWE'):** consider a gadget vector

  $$\mathbf{v} = (v_0, v_1, \cdots, v_{k-1}) \in \mathcal{R}_q^k.$$

  GadgetRLWE is expressed as a vector of RLWE ciphertexts:

  $$\text{RLWE'}(\mathbf{m}) = (\text{RLWE}(v_0 \cdot \mathbf{m}), \cdots, \text{RLWE}(v_{k-1} \cdot \mathbf{m}))$$

- **RGSW:** For a message $\mathbf{m} \in \mathcal{R}_q$ and a secret key $\mathbf{z} \leftarrow \chi$, we define

  $$\text{RGSW}_{\mathbf{z}}(\mathbf{m}) = (\text{RLWE'}(\mathbf{z} \cdot \mathbf{m}), \text{RLWE'}(\mathbf{m})) \in \mathcal{R}_q^{2\times 2k}$$

An important operation: $\odot$

- Main operation in our algorithm: **scalar multiplication by arbitrary ring elements.**
- One uses RLWE' with gadget vector $\mathbf{v} = (v_0, v_1, \cdots, v_{k-1})$.
- The scalar multiplication: $\mathcal{R} \odot \mathrm{RLWE}'$ corresponds to $\odot : \mathcal{R} \times \mathrm{RLWE}' \to \mathrm{RLWE}$ defined as

$$\mathbf{t} \odot RLWE'_{\mathbf{sk}}(\mathbf{m}) := \sum_{i=0}^{k-1} \mathbf{t}_i \cdot RLWE_{\mathbf{sk}}(v_i \cdot \mathbf{m})$$

$$= RLWE_{\mathbf{sk}}\left(\sum_{i=0}^{k-1} v_i \cdot \mathbf{t}_i \cdot \mathbf{m}\right) = RLWE_{\mathbf{sk}}(\mathbf{t} \cdot \mathbf{m})$$

where $\sum_i v_i \mathbf{t}_i = \mathbf{t}$ is the gadget decomposion of $\mathbf{t}$ into "short" vectors $\mathbf{t}_i$.

RLWE'-to-RGSW scheme switching

Input $\mathrm{RLWE}'_{\mathbf{sk}}(m)$

Output $\mathrm{RGSW}_{\mathbf{sk}}(\mathbf{m}) = (\mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{sk} \cdot \mathbf{m}), \mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{m}))$

- Goal: compute $\mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{sk} \cdot \mathbf{m})$.

1. Use $\mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{sk^2})$ given as part of the evaluation key.
2. Operate in parallel on each of the $\mathrm{RLWE}_{\mathbf{sk}}(v_i \cdot \mathbf{m})$, lifting each $\mathrm{RLWE}_{\mathbf{sk}}(v_i \cdot \mathbf{m})$ to $\mathrm{RLWE}_{\mathbf{sk}}(v_i \cdot \mathbf{sk} \cdot \mathbf{m})$.
3. For each $\mathrm{RLWE}_{\mathbf{sk}}(v_i \cdot \mathbf{m}) := (\mathbf{a}, \mathbf{b})$, compute

$$\mathbf{a} \odot RLWE'_{\mathbf{sk}}(\mathbf{sk}^2) + (\mathbf{b}, 0).$$

Scheme switching (cont.)

- $(\mathbf{b}, 0) =$ noiseless RLWE encryption of $\mathbf{b} \cdot \mathbf{sk}$ under secret key $\mathbf{sk}$.
- Above computation gives

$$\begin{aligned} \mathbf{a} \odot RLWE'_{\mathbf{sk}}(\mathbf{sk}^2) + (\mathbf{b}, 0) &= \mathrm{RLWE}_{\mathbf{sk}}(\mathbf{a} \cdot \mathbf{sk}^2 + \mathbf{b} \cdot \mathbf{sk}) \\ &= \mathrm{RLWE}_{\mathbf{sk}}((\mathbf{a} \cdot \mathbf{sk} + \mathbf{b}) \cdot \mathbf{sk}) \\ &= \mathrm{RLWE}_{\mathbf{sk}}((v_i \cdot \mathbf{m} + \mathbf{e}) \cdot \mathbf{sk}) \end{aligned}$$

- We get $\mathrm{RLWE}_{\mathbf{sk}}(v_i \cdot \mathbf{sk} \cdot \mathbf{m})$, but with an **additional error** $\mathbf{e} \cdot \mathbf{sk}$.
- Choose the secret key $\mathbf{sk}$ with small norm (e.g., binary) so that this multiplicative error growth remains small.

We are now ready for our homomorphic FFT!

Homomorphic pointwise multiplication

Goal: compute $\mathrm{FFT}(\mathbf{a} \cdot \mathbf{z})$.

What we have so far:

- $\mathrm{FFT}(\mathbf{a})$ = list of polynomials $\tilde{\mathbf{a}}_i = \mathbf{a}(x) \pmod{x^k - \zeta_i}$: computed in the clear for different values of ζ.
- encryption of $\mathrm{FFT}(\mathbf{z})$ = list of polynomials $\tilde{\mathbf{z}}_i = \mathbf{z}(x) \pmod{x^k - \zeta_i}$ as part of the evaluation key.

What we do: we multiply $\tilde{\mathbf{a}}_i(x)$ and $\tilde{\mathbf{z}}_i(x)$ modulo $(x^k - \zeta_i)$ (for all i).

Example:

$$a(x) \pmod{x^k - \zeta} = \tilde{a}_0 + \tilde{a}_1 x + \tilde{a}_2 x^2$$

$$z(x) \pmod{x^k - \zeta} = \boxed{\tilde{z}_0} + \boxed{\tilde{z}_1} x + \boxed{\tilde{z}_2} x^2$$

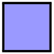
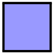
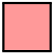
As $(x^k - \zeta)$ has such a nice form (!), we get a "simple" formula:

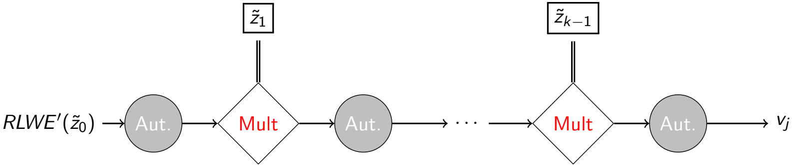
Constant term: $\tilde{a}_0 \boxed{\tilde{z}_0} + \zeta(\tilde{a}_1 \boxed{\tilde{z}_2} + \cdots)$

More generally, the j-th coefficient of $\tilde{\mathbf{a}}_i \cdot \tilde{\mathbf{z}}_i$ is equal to an **inner product**:

$$v_j = \langle (\boxed{\tilde{z}_0}, \cdots, \boxed{\tilde{z}_{k-1}}), (\tilde{a}_j, \tilde{a}_{j-1}, \ldots, \tilde{a}_0, \zeta\tilde{a}_{k-1}, \ldots, \zeta\tilde{a}_{j+1}) \rangle$$

- Compute this inner product homomorphically in a telescoping manner (see formula in paper).

Operations: $a\times$  , and  $+$ .



$$\left(\boxed{\tilde{z}_0}\,\tilde{a}_j\tilde{a}_{j-1}^{-1} + \boxed{\tilde{z}_1}\right)\tilde{a}_{j-1}\tilde{a}_{j-2}\cdots$$

The next step: inverse FFT

After pointwise multiplication, we have:

$$\boxed{\mathrm{FFT}(\mathbf{a} \cdot \mathbf{z})} = \text{list of RLWE' encryptions of } X^{v_j}$$

v_j: all coefficients of all products $\tilde{\mathbf{a}}_i \cdot \tilde{\mathbf{z}}_i$.
What we want now: $\boxed{\mathbf{a} \cdot \mathbf{z}}$, i.e. RLWE encryptions of the coefficients of $\mathbf{a} \cdot \mathbf{z}$.

Goal: compute an homomorphic inverse FFT.

- can be reduced to a forward FFT with an additional multiplication.

Operations for Homomorphic inverse FFT

Input RLWE' registers, outputs of pointwise multiplication

Output RLWE encryptions of $\mathbf{a} \cdot \mathbf{z}$.

1. Split registers into groups.
2. For each group, perform a standard (not primitive) forward FFT.
3. Homomorphically multiply each output register in each group by a power of the root of unity. (automorphism).

Focus on operations in forward FFT (step 2):

- FFT works with a remainder tree,
- At each layer, a child node is produced by taking an input polynomial and reducing it modulo $X^k - \zeta$.
- Each reduction results in a computation of the form $\sum_i \boxed{a_i} \zeta^i$.

Analysis of our algorithm

How to evaluate the performance of our algorithm?

- we count the number of $\odot$ operations, i.e, the number of $\mathcal{R} \odot$ RLWE' operations.
- we quantify the error growth in our algorithm (necessary for correctness).
  - in previous work, error analysis is done for power-of-2 cyclotomics.
  - in this work, we use prime cyclotomic rings
  - new error growth analysis (in the paper).
- The amortized cost per message is $O(n^{1/\ell} \cdot \log n \cdot \ell)$ homomorphic operations (in terms of the number of $\mathcal{R} \odot \mathrm{RLWE}'$ operations).

Concurrent and follow-up works

1. *Amortized Bootstrapping Revisited: Simpler, Asymptotically-faster, Implemented*, Antonio Guimarães, Hilder V. L. Pereira and Barry van Leeuwen at **Asiacrypt 2023**
   - Very similar algorithm with same asymptotic amortized cost.
   - Some technical differences:
     - Uses circular rings (Ours: cyclotomic rings),
     - Focuses on RGSW Register (Ours: RLWE).
2. *Batch Bootstrapping I:: A New Framework for SIMD Bootstrapping in Polynomial Modulus*, Feng-Hao Liu and Han Wang at **Eurocrypt 2023**
3. *Batch Bootstrapping II: Bootstrapping in Polynomial Modulus Only Requires O(1) FHE Multiplications in Amortization*, Feng-Hao Liu and Han Wang at **Eurocrypt 2023**
4. *Amortized Functional Bootstrapping in less than 7ms, with* $\tilde{O}(1)$ *polynomial multiplications*, Zeyu Liu and Yunhao Wang, at **Asiacrypt 2023**