SHIP
A SHALLOW AND HIGHLY PARALLELIZABLE CKKS BOOTSTRAPPING ALGORITHM

JUNG HEE CHEON, GUILLAUME HANROT,
JONGMIN KIM & DAMIEN STEHLÉ

MADRID --- MAY 6, 2025

Eprint 2025/784

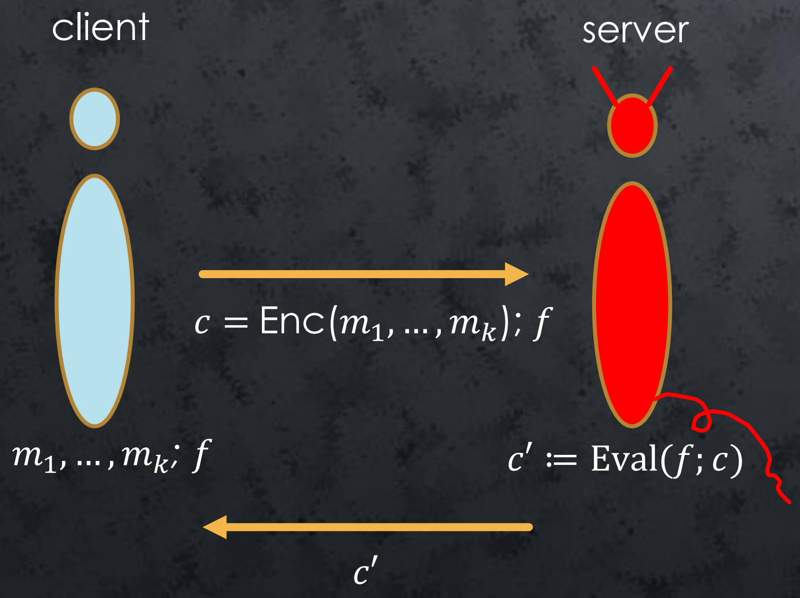
HEAAN CRYPTO LAB

MAIN RESULT

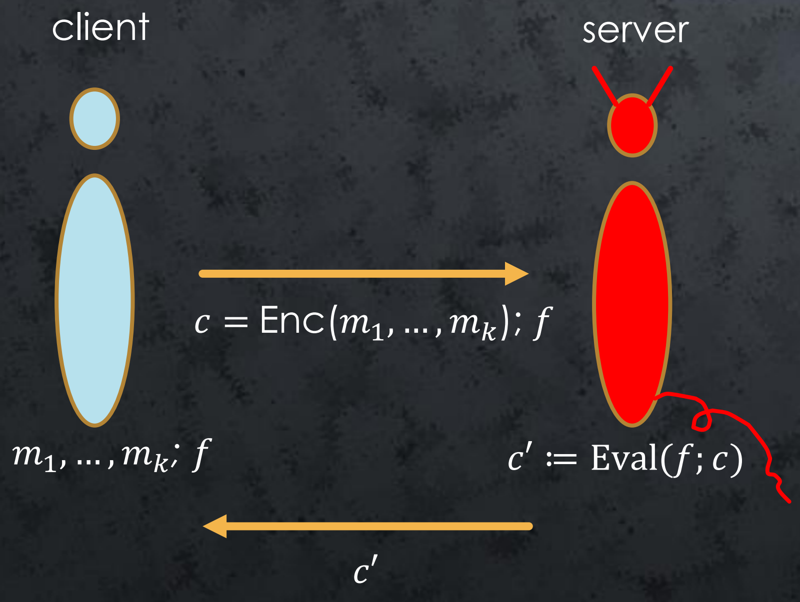
A new **bootstrapping (BTS)** algorithm for **CKKS**

- **Small multiplicative depth** ⇨ smaller Ring-LWE ring degree
- **High-grain parallelizability** ⇨ works well in multi-threaded environment

FHE & BTS


client
server
$c = \mathsf{Enc}(m_1, \ldots, m_k);\ f$
$m_1, \ldots, m_k;\ f$
$c' := \mathrm{Eval}(f; c)$
c'

FHE & BTS



Gentry's blueprint for building an **FHE**:

- Start with an encryption scheme that is homomorphic for some circuits

- Find a **bootstrapping** algorithm, i.e., a plaintext-preserving procedure that allows to extend homomorphism to all circuits

For all known FHE schemes, BTS drives the cost

CKKS

Cleartexts: vectors of $\mathbb{C}^{N/2}$

- up to some precision
- for some power-of-two N

Plaintexts: elements of $R_q = \mathbb{Z}[X] / (X^N + 1)$

- $\mathrm{ptxt} = \mathrm{DFT}^{-1}(\mathrm{ctxt})$

Ciphertexts: pairs over $R_q = \mathbb{Z}_q[X] / (X^N + 1)$

$$ct = (a, b) \in R_q^2 : \quad a \cdot s + b \approx m \ [q]$$

secret key
ternary

plaintext $\ll q$

CKKS

Cleartexts: vectors of $\mathbb{C}^{N/2}$

- up to some precision
- for some power-of-two N

Plaintexts: elements of $R_q = \mathbb{Z}[X] \,/\, (X^N + 1)$

- $\mathrm{ptxt} = \mathrm{DFT}^{-1}(\mathrm{ctxt})$

Ciphertexts: pairs over $R_q = \mathbb{Z}_q[X] \,/\, (X^N + 1)$

$$ct = (a, b) \in R_q^2 : \quad a \cdot s + b \approx m \; [q]$$

secret key
ternary

plaintext $\ll q$

Operations

• **mult in //**	**consumes 1 level**	**decreases q**
• add in //, conj in //	consume 0 level	keep q
• rotate coords	consumes 0 level	keeps q
• **BTS**	**regains levels**	**increases q**

LATENCY OF THE CKKS BTS

Parameters

ring degree	2^{16}	2^{15}
largest modulus	1555 bits	771bits
precision	22.0 bits	16.7 bits
non-BTS levels	9	1

Can we decrease the latency?

	1 core	8 cores	16 cores	32 cores
Param16	8.7 s	1.9 s	1.4 s	1.1 s
Param15	3.4 s	0.90 s	0.64 s	0.62 s

CPU: two 24-core AMD EPYC 7473X @2.8GHz with AVX2 & OpenMP

128-bit security & BTS failure probability $\leq 2^{-128}$

CONVENTIONAL CKKS BTS

$(a, b) \in R_{q_0}^2$: $\quad a \cdot s + b \approx m \ [q_0]$

$(a', b') \in R_Q^2$: $\quad a' \cdot s + b' \approx m \ [Q]$ for some $Q \gg q_0$

CONVENTIONAL CKKS BTS

$(a,b) \in R_{q_0}^2$: $\quad a \cdot s + b \approx m \ [q_0]$

$(a',b') \in R_Q^2$: $\quad a' \cdot s + b' \approx m \ [Q]$ for some $Q \gg q_0$

1. **S2C**: Inverse DFT — consumes 2-3 levels
2. **ModRaise**: Viewing $(a,b) \in R_{q_0}^2$ as $(a,b) \in R_Q^2$ for $m' = m + q_0 \cdot I$
3. **C2S**: DFT — consumes 2-3 levels
4. **EvalMod**: Remove $q_0 \cdot I$ in $m' = m + q_0 \cdot I$ — consumes 8-12 levels

CONVENTIONAL CKKS BTS

$(a, b) \in R_{q_0}^2$: $\quad a \cdot s + b \approx m \ [q_0]$

$m \ [Q]$ for some $Q \gg q_0$

1. **S2C**: consumes 2-3 levels
2. **ModRais**
3. **C2S**: consumes 2-3 levels
4. **EvalMod**: consumes 8-12 levels

Many levels

⬇

High modulus consumption

⬇

Needs a large degree N for security

⬇

Higher latency

BOOTSTRAPPING VIA ROOTS OF UNITY

$$\omega = \exp\left(\frac{2i\pi}{q_0}\right) \in \mathbb{C}$$

Input: $(a, b) \in R_{q_0}^2$: $\quad a \cdot s + b \approx m \ [q_0]$

Goal: $(a', b') \in R_Q^2$: $\quad a' \cdot s + b' \approx \mathrm{DFT}^{-1}\left((\omega^{m_0}, \omega^{m_1}, \dots, \omega^{m_{N-1}})\right) \quad [Q]$

Why is it sufficient?

$m_i \ll q_0$

1. Slots are correct: $\quad \mathrm{Im}(\omega^{m_i}) = \sin\left(\frac{2\pi}{q_0} m_i\right) \approx \frac{2\pi}{q_0} m_i$

2. To put the m_i's in coeffs, use S2C.

BOOTST ROOTS OF UNITY

Cleartexts have only $\frac{N}{2} < N$ slots…

Let's ignore that for the talk

$$\omega = \exp\left(\frac{2i\pi}{q_0}\right) \in \mathbb{C}$$

Input: $(a, b$ $]$

Goal: $(a', b') \in$ $s + b' \approx \mathrm{DFT}^{-1}\left((\omega^{m_0}, \omega^{m_1}, \ldots, \omega^{m_{N-1}})\right) \quad [Q]$

Why is it sufficient?

$m_i \ll q_0$

1. Slots are correct: $\mathrm{Im}(\omega^{m_i}) = \sin\left(\frac{2\pi}{q_0} m_i\right) \approx \frac{2\pi}{q_0} m_i$
2. To put the m_i's in coeffs, use S2C.

REDUCING TO A BINARY PRODUCT TREE

$$\omega = \exp\left(\frac{2i\pi}{q_0}\right) \in \mathbb{C}$$

Input: $(a, b) \in R_{q_0}^2$: $a \cdot s + b \approx m \ [q_0]$

$j - i$ is mod N

$$\omega^{m_i} = \omega^{(a \cdot s + b)_i} = \omega^{\,b_i + \sum_j (a_{j-i} \cdot s_j)}$$

entry-wise product

$$(\omega^{m_i})_i = (\omega^{b_i})_i \odot \bigodot_{j:\, s_j \neq 0} ((\omega^{a_{j-i} \cdot s_j})_i)$$

If s has a small Hamming weight, this is a **binary product tree** with $\log h$ levels

$\Rightarrow$ To minimize depth, we use $h = 31 \ll N$.

REDU... O A BINARY PRODUCT TREE

Reduction mod $X^N + 1$
creates signs

Let's ignore that for the talk

$$\omega = \exp\left(\frac{2i\pi}{q_0}\right) \in \mathbb{C}$$

Inp ... $+\, b \approx\ m\ [q_0]$

$j - i$ is mod N

$$\ldots \omega^{(a \cdot s + b)_i} = \omega^{\,b_i + \sum_j (a_{j-i} \cdot s_j)}$$

entry-wise prod...

$$(\omega^{m_i})_i = (\omega^{b_i})_i \odot \bigodot_{j:\, s_j \neq 0} ((\omega^{a_{j-i} \cdot s_j})_i)$$

Similar bootstrapping strategy considered in concurrent work by Coron & Köstler (eprint 2025/651)

If s has a small Hamming weight, this is a **binary product tree** with $\log$...

⇒ To minimize depth, we use $h = 31 \ll N$.

COLUMN METHOD

New goal: compute the h terms $(\omega^{a_{j-i}})_i$ for all j with $s_j \neq 0$

Assumption: there is one non-zero s_j in every block of $\approx N/h$ coordinates

$$\bigodot_{j \text{ in block}} (\omega^{a_{j-i} \cdot s_j})_i = \sum_{j \text{ in block}} (\omega^{a_{j-i}})_i \cdot s_j$$

COLUMN METHOD

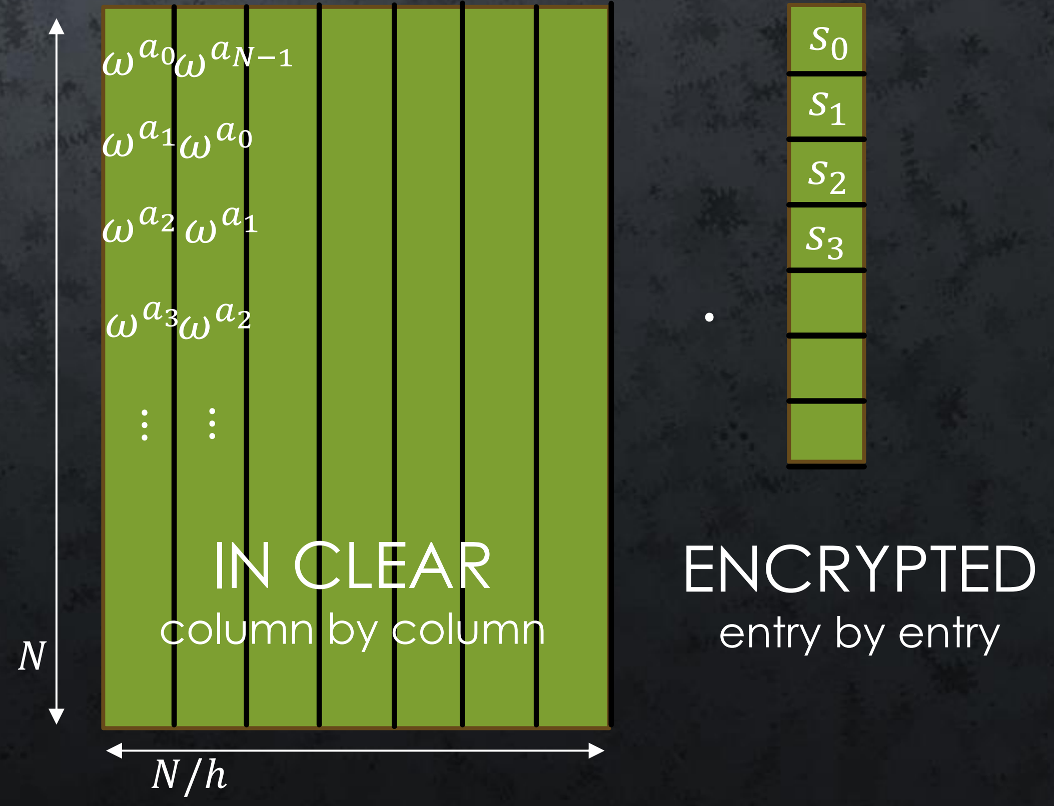
New goal: compute the h terms $(\omega^{a_{j-i}})_i$ for all j with $s_j \neq 0$

Assumption: there is one non-zero s_j in every block of $\approx N/h$ coordinates

$$\bigodot_{j \text{ in block}} (\omega^{a_{j-i} \cdot s_j})_i = \sum_{j \text{ in block}} (\omega^{a_{j-i}})_i \cdot s_j$$

no level consumption
simple operations

$\Omega(N^2)$ key size
$\Omega(N^2)$ cost

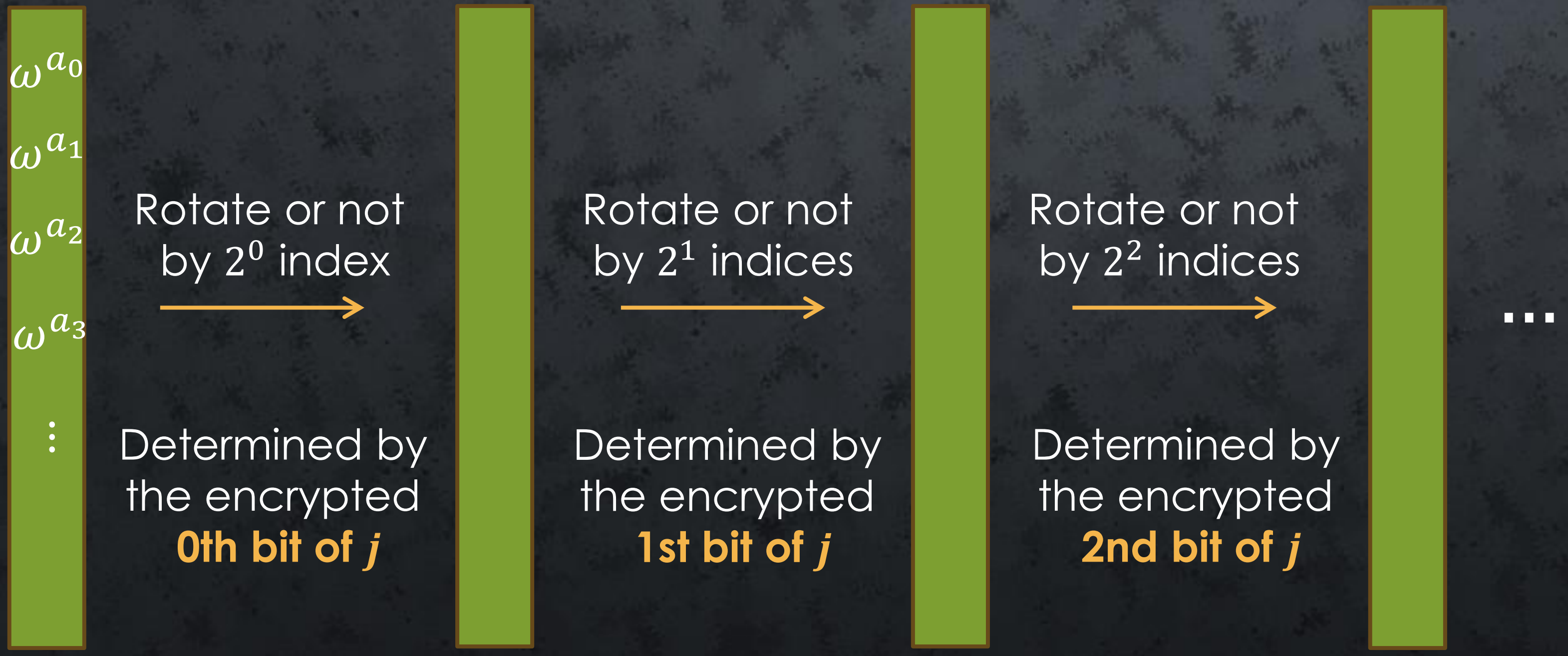


BLIND ROTATION METHOD

For this talk, let's assume that $s_j \in \{0,1\}$

New goal: compute the h terms $(\omega^{a_{j-i} \cdot s_j})_i = (\omega^{a_{j-i}})_i$ for all j with $s_j \neq 0$

Approach: for each such j : blindly rotate $(\omega^{a_i})_i$ by j indices



BLIND ROTATION METHOD

For this talk, let's assume that $s_j \in \{0,1\}$

New goal: compute the h terms $(\omega^{a_{j-i}\cdot s_j})_i = (\omega^{a_{j-i}})_i$ for all j with $s_j \neq 0$

Approach: for each such j : blindly rotate $(\omega^{a_i})_i$ by j indices

ω^{a_0}
ω^{a_1}
ω^{a_2}
ω^{a_3}
$\vdots$

Rotate or not by 2^0 index →
Determined by the encrypted **0th bit of j**

Rotate or not by 2^1 indices →
Determined by the encrypted **1st bit of j**

Rotate or not by 2^2 indices →
Determined by the encrypted **2nd bit of j**

...

no level consumption
$\tilde{O}(N)$ key size
$\tilde{O}(N)$ cost

more complex operations

The bits of j can be incorporated in rotation keys ⇒ no level consumption

SHIP

More fun in the paper ☺

- too few slots for N coefficients
- reduction mod $X^N + 1$ creates signs
- how to handle ternary s_j's
- S2C permutes the slots/coeffs
- column and blind-rotate can be combined

1- For every non-zero entry j of s:
blind-rotate by j positions

2- Binary product tree to compute

$$(\omega^{m_i})_i = \left(\omega^{b_i}\right)_i \odot \bigodot_{j:\, s_j \neq 0} \left((\omega^{a_{j-i} \cdot s_j})_i\right)$$

3- Perform S2C (DFT) to get back to coeffs

ANALYSIS: LEVELS

1- For every non-zero entry j of s:
blind-rotate by j positions

0 level

2- Binary product tree to compute

$$(\omega^{m_i})_i = (\omega^{b_i})_i \odot \bigodot_{j:\, s_j \neq 0} \left((\omega^{a_{j-i} \cdot s_j})_i\right)$$

$\mathbf{log(}\boldsymbol{h}\mathbf{+1)=5}$ **levels**

3- Perform S2C (DFT) to get back to coeffs

1 or 2 levels

Bonus: the top levels are smaller than in conventional CKKS BTS,
as there is no need to represent $q_0 \cdot I$ as part of the plaintext

ANALYSIS: PARALLELIZABILITY

1- For every non-zero entry j of s:
blind-rotate by j positions

$h = 31$ independent tracks

2- Binary product tree to compute

$$(\omega^{m_i})_i = \left(\omega^{b_i}\right)_i \odot \bigodot_{j: s_j \neq 0} \left((\omega^{a_{j-i} \cdot s_j})_i\right)$$

Binary product tree

3- Perform S2C (DFT) to get back to coeffs

Matrix-vector product

PERFORMANCE

Parameters for conventional BTS

- ring degree $N = 2^{16}$ $N = 2^{15}$
- Precision 22.0 bits 16.7 bits
- non-BTS levels 9 1

Parameters for SHIP

- ring degree $N = 2^{13}$ $N = 2^{14}$
- precision 4.45 bits 16.9 bits
- non-BTS levels 1 1

	1 core	8 cores	16 cores	32 cores
Param16	8.7 s	1.9 s	1.4 s	1.1 s
Param15	3.4 s	0.90 s	0.64 s	0.62 s
SHIP13	3.0 s	0.45 s	0.29 s	0.22 s
SHIP14	4.9 s	0.70 s	0.42 s	0.33 s

CPU: two 24-core AMD EPYC 7473X @2.8GHz with AVX2 & OpenMP
128-bit security & BTS failure probability $\leq 2^{-128}$

PERFORMANCE

Parameters for conventional BTS

- ring degree $N = 2^{16}$ $N = 2^{15}$
- Precision 22.0 bits 16.7 bits
- non-BTS levels 9 1

Parameters for SHIP

- ring degree $N = 2^{13}$ $N = 2^{14}$
- precision 4.45 bits 16.9 bits
- non-BTS levels 1 1

	1 core	8 cores	16 cores	32 cores
Param16	**8.7 s**	**1.9 s**	**1.4 s**	**1.1 s**
Param15	**3.4 s**	**0.90 s**	**0.64 s**	**0.62 s**
SHIP13	**3.0 s**	**0.45 s**	**0.29 s**	**0.22 s**
SHIP14	**4.9 s**	**0.70 s**	**0.42 s**	**0.33 s**

For bootstrapping bits, we get down to 0.17s

CPU: two 24-core AMD EPYC 7473X @2.8GHz with AVX2 & OpenMP
128-bit security & BTS failure probability $\leq 2^{-128}$

WRAP-UP

Main contribution

A new **bootstrapping** algorithm for **CKKS**.

- **Small multiplicative depth:** $\mathbf{1 + \log(h+1) = 6}$
  ⇨ full-slot BTS in ring degree $N = 2^{13}$
- **High-grain parallelizability:**
  - $h = 31$ fully independent dominating tasks
  - Other components can also be parallelized

QUESTIONS?

Eprint 2025/784

damien.stehle @ cryptolab.co.kr