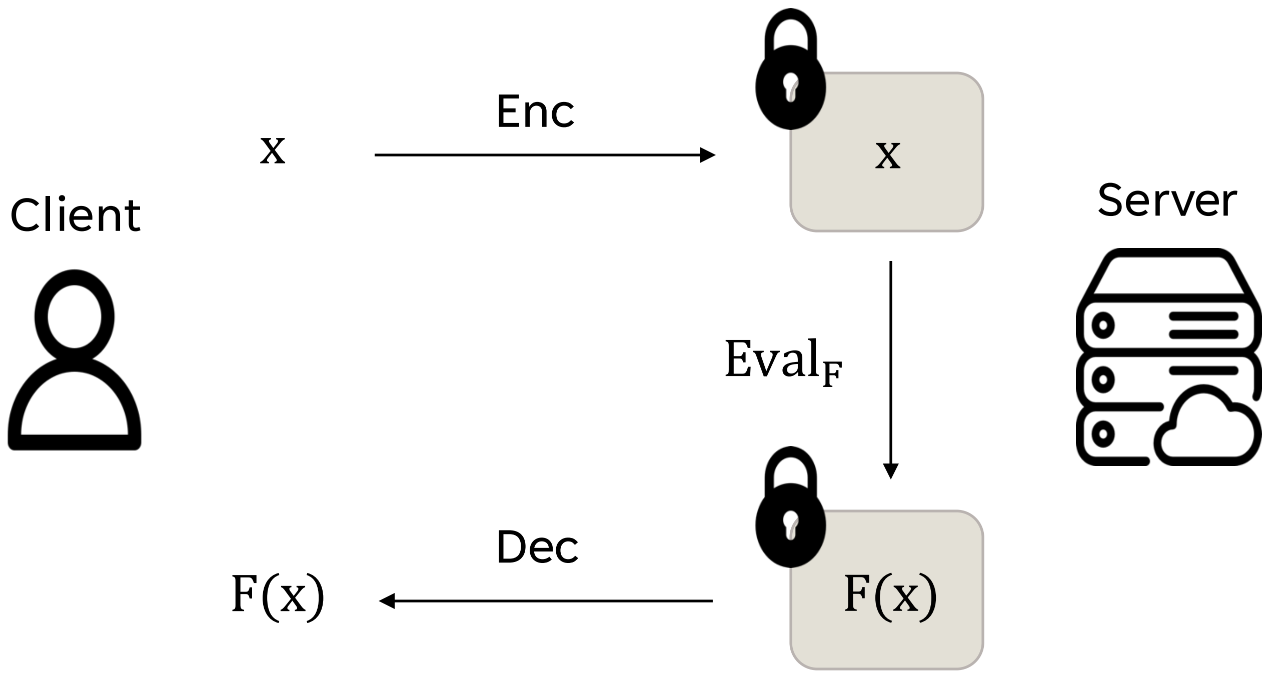
GRAFTING: DECOUPLED SCALE FACTORS AND MODULUS IN RNS-CKKS

Jung Hee Cheon[1,2], Hyeongmin Choe[2], Minsik Kang[1], Jaehyung Kim[3]

Seonghak Kim[2], Johannes Mono[2,4], Taeyeong Noh[2]

FULLY HOMOMORPHIC ENCRYPTION (FHE)



- FHE enables computations on encrypted data without decryption.
- Provides efficient privacy-preserving computation.
- CKKS supports approximate computations on real/complex numbers.

RNS-CKKS

- CKKS encodes $\vec{z} \in \mathbb{C}^{N/2}$ with scale factor Δ as:
  - Plaintext: $\Delta m = \lfloor \Delta \cdot DFT^{-1}(\vec{z}) \rceil \in R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$.

RNS-CKKS

- CKKS encodes $\vec{z} \in \mathbb{C}^{N/2}$ with scale factor Δ as:
  - Plaintext: $\Delta m = \lfloor \Delta \cdot DFT^{-1}(\vec{z}) \rceil \in R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$.

- CKKS Ciphertext: a pair over $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$

$$ct(m) = (a, b) \in R_Q^2: \quad a \cdot s + b = \Delta m + e \ \ (mod\ Q),$$

where s: secret, e: error, $\Delta m \ll Q$.

RNS-CKKS

- CKKS encodes $\vec{z} \in \mathbb{C}^{N/2}$ with scale factor Δ as:
  - Plaintext: $\Delta m = \lfloor \Delta \cdot DFT^{-1}(\vec{z}) \rceil \in R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$.

- CKKS Ciphertext: a pair over $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$

$$ct(m) = (a, b) \in R_Q^2: \quad a \cdot s + b = \Delta m + e \ \ (mod\ Q),$$

  where s: secret, e: error, $\Delta m \ll Q$.

- For modulus $Q = \prod_{i=0}^{\ell} q_i$, the CRT: $\mathbb{Z}_Q \cong \prod_{i=0}^{\ell} \mathbb{Z}_{q_i}$ allows

$$R_Q \cong R_{q_0} \times R_{q_1} \times \cdots \times R_{q_\ell}$$

  - Computation cost grows linearly with level ℓ.

RNS-CKKS

- CKKS encodes $\vec{z} \in \mathbb{C}^{N/2}$ with scale factor Δ as:
  - Plaintext: $\Delta m = \lfloor \Delta \cdot DFT^{-1}(\vec{z}) \rceil \in R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$.

- CKKS Ciphertext: a pair over $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$

$$ct(m) = (a, b) \in R_Q^2: \quad a \cdot s + b = \Delta m + e \pmod{Q},$$

where s: secret, e: error, $\Delta m \ll Q$.

- For modulus $Q = \prod_{i=0}^{\ell} q_i$, the CRT: $\mathbb{Z}_Q \cong \prod_{i=0}^{\ell} \mathbb{Z}_{q_i}$ allows

$$R_Q \cong R_{q_0} \times R_{q_1} \times \cdots \times R_{q_\ell}$$

  - Computation cost grows linearly with level ℓ.

$\Rightarrow$ Filling Q with machine's word-size primes is the most efficient!

MODULUS RIGIDLY TIED TO SCALE FACTOR

For $ct(m_i) = (a_i, b_i)$ $(i = 1, 2)$, CKKS multiplication proceeds as follows:

MODULUS RIGIDLY TIED TO SCALE FACTOR

For $ct(m_i) = (a_i, b_i)$ $(i = 1, 2)$, CKKS multiplication proceeds as follows:

- Tensor & Relinearization:

$$
\begin{aligned}
(d_0, d_1) \in R_Q^2: \quad d_0 + d_1 s &\approx (b_1 + a_1 s)(b_2 + a_2 s) \\
&\approx \Delta^2 m_1 m_2 + \Delta(m_1 e_2 + m_2 e_1) + e_{relin} \ \ (mod\ Q).
\end{aligned}
$$

MODULUS RIGIDLY TIED TO SCALE FACTOR

For $ct(m_i) = (a_i, b_i)$ $(i = 1, 2)$, CKKS multiplication proceeds as follows:

- Tensor & Relinearization:

$$(d_0, d_1) \in R_Q^2: \quad d_0 + d_1 s \approx (b_1 + a_1 s)(b_2 + a_2 s)$$
$$\approx \Delta^2 m_1 m_2 + \Delta(m_1 e_2 + m_2 e_1) + e_{relin} \ (mod\ Q).$$

- Rescale:

$$(c_0, c_1) = \left(\left\lfloor \frac{d_0}{q_\ell} \right\rceil, \left\lfloor \frac{d_1}{q_\ell} \right\rceil\right) \in R_{Q/q_\ell}^2, \quad c_0 + c_1 s \approx \frac{\Delta^2}{q_\ell} m_1 m_2 + \frac{\Delta}{q_\ell}(m_1 e_2 + m_2 e_1) \ \left(mod\, \frac{Q}{q_\ell}\right).$$

MODULUS RIGIDLY TIED TO SCALE FACTOR

For $ct(m_i) = (a_i, b_i)$ $(i = 1, 2)$, CKKS multiplication proceeds as follows:

- Tensor & Relinearization:

$$(d_0, d_1) \in R_Q^2: \quad d_0 + d_1 s \approx (b_1 + a_1 s)(b_2 + a_2 s)$$
$$\approx \Delta^2 m_1 m_2 + \Delta(m_1 e_2 + m_2 e_1) + e_{relin} \ \ (mod\ Q).$$

- Rescale:

$$(c_0, c_1) = \left(\left\lfloor \frac{d_0}{q_\ell} \right\rceil, \left\lfloor \frac{d_1}{q_\ell} \right\rceil \right) \in R_{Q/q_\ell}^2, \qquad c_0 + c_1 s \approx \frac{\Delta^2}{q_\ell} m_1 m_2 + \frac{\Delta}{q_\ell}(m_1 e_2 + m_2 e_1) \ \left(mod\, \frac{Q}{q_\ell}\right).$$

∴ Hence, each modulus should match the scale factor Δ:

$q_1 \approx \cdots \approx q_\ell \approx \Delta$ for multiplication levels ℓ.

STRUCTURE ON CKKS MODULUS CHAIN

- Modulus chain in RNS-CKKS is constructed as follows:

$$Q_0 \mid Q_1 \mid \cdots \mid Q_L,$$

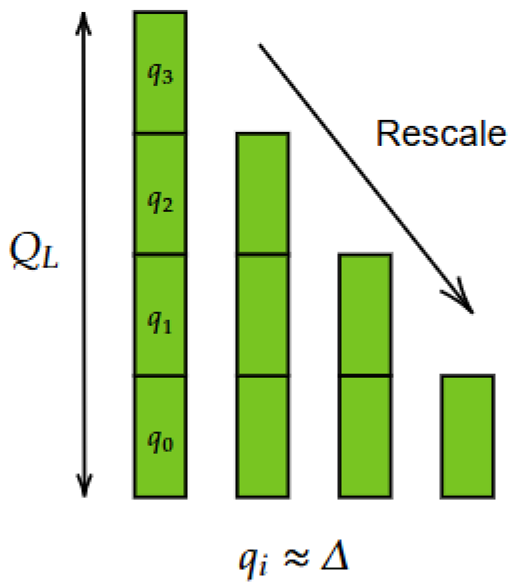
  - $Q_\ell = q_0 q_1 \cdots q_\ell \approx \Delta^\ell \cdot q_0$ for each level ℓ.

STRUCTURE ON CKKS MODULUS CHAIN

- Modulus chain in RNS-CKKS is constructed as follows:

$$Q_0 \mid Q_1 \mid \cdots \mid Q_L,$$

  - $Q_\ell = q_0 q_1 \cdots q_\ell \approx \Delta^\ell \cdot q_0$ for each level ℓ.



STRUCTURE ON CKKS MODULUS CHAIN

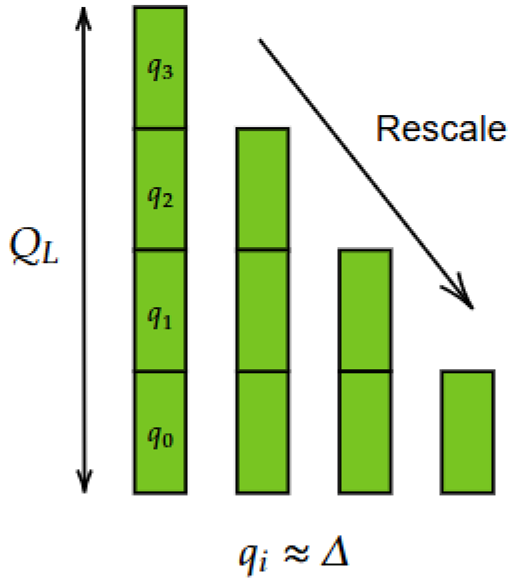
- Modulus chain in RNS-CKKS is constructed as follows:

$$Q_0 \mid Q_1 \mid \cdots \mid Q_L,$$

  - $Q_\ell = q_0 q_1 \cdots q_\ell \approx \Delta^\ell \cdot q_0$ for each level ℓ.

- Key-switching keys are positioned at modulus PQ_L
  - The auxiliary modulus P allows key-switching at any level ℓ.
  - The main property enabling this is:

$$Q_\ell \mid Q_L$$



STRUCTURE ON CKKS MODULUS CHAIN

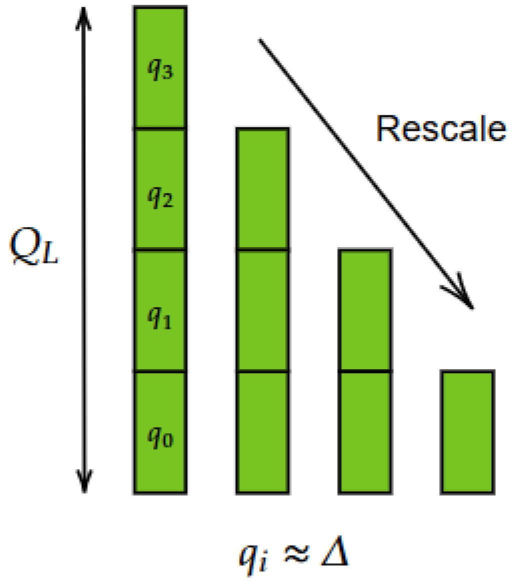
- Modulus chain in RNS-CKKS is constructed as follows:

$$Q_0 \mid Q_1 \mid \cdots \mid Q_L,$$

  - $Q_\ell = q_0 q_1 \cdots q_\ell \approx \Delta^\ell \cdot q_0$ for each level ℓ.

- Key-switching keys are positioned at modulus PQ_L
  - The auxiliary modulus P allows key-switching at any level ℓ.
  - The main property enabling this is:

$$Q_\ell \mid Q_L$$



Why don't we set $Q_\ell \mid Q_L$ — not necessarily $Q_\ell \mid Q_{\ell+1}$,
while each Q_ℓ is filled up with machine's word-size primes?

GRAFTING:
A NOVEL MODULUS MANAGEMENT SYSTEM

RATIONAL RESCALE WITH SPROUT

- We set the top-modulus as $Q_{top} = q_0 q_1 \cdots q_{L-1} \cdot r_{top}$.
  - Each q_i is machine word-size prime.
  - r_{top} is called a *sprout*, reusable modulus factor of Q_{top}.

RATIONAL RESCALE WITH SPROUT

- We set the top-modulus as $Q_{top} = q_0 q_1 \cdots q_{L-1} \cdot r_{top}$.
  - Each q_i is machine word-size prime.
  - r_{top} is called a *sprout,* reusable modulus factor of Q_{top}.

- Every possible modulus is of the form $Q = q_0 q_1 \cdots q_\ell \cdot r$.
  - Here, $\ell < L$ and $r \mid r_{top}$ to ensure $Q \mid Q_{top}$.

RATIONAL RESCALE WITH SPROUT

- We set the top-modulus as $Q_{top} = q_0 q_1 \cdots q_{L-1} \cdot r_{top}$.
  - Each q_i is machine word-size prime.
  - r_{top} is called a *sprout*, reusable modulus factor of Q_{top}.

- Every possible modulus is of the form $Q = q_0 q_1 \cdots q_\ell \cdot r$.
  - Here, $\ell < L$ and $r \mid r_{top}$ to ensure $Q \mid Q_{top}$.

- Rescale from modulus Q to $Q'(< Q)$ proceeds as:

$$\mathrm{Rescale}_{Q \mapsto Q'}\left(ct = (a, b) \in R_Q^2\right) = \left(\left\lfloor \frac{Q'}{Q} \cdot a \right\rceil, \left\lfloor \frac{Q'}{Q} \cdot b \right\rceil\right) \in R_{Q'}^2$$

RATIONAL RESCALE WITH SPROUT

- We set the top-modulus as $Q_{top} = q_0 q_1 \cdots q_{L-1} \cdot r_{top}$.
  - Each q_i is machine word-size prime.
  - r_{top} is called a *sprout*, reusable modulus factor of Q_{top}.

- Every possible modulus is of the form $Q = q_0 q_1 \cdots q_\ell \cdot r$.
  - Here, $\ell < L$ and $r \mid r_{top}$ to ensure $Q \mid Q_{top}$.

- Rescale from modulus Q to $Q'(< Q)$ proceeds as:

$$\mathrm{Rescale}_{Q \mapsto Q'}\left(ct = (a, b) \in R_Q^2\right) = \left(\left\lfloor \frac{Q'}{Q} \cdot a \right\rceil, \left\lfloor \frac{Q'}{Q} \cdot b \right\rceil\right) \in R_{Q'}^2$$

⇒ We call it *Rational Rescale*, a generalized Rescale in RNS-CKKS.

UNIVERSAL SPROUT

- To enable Rational Rescale with arbitrary bit-lengths,

$Q = q_0 q_1 \cdots q_\ell \cdot r$ should represent integers of any bit-lengths.

UNIVERSAL SPROUT

- To enable Rational Rescale with arbitrary bit-lengths,

  $Q = q_0 q_1 \cdots q_\ell \cdot r$ should represent integers of any bit-lengths.

  - Sprout r_{top} should satisfy the following:

    For word-size ω, $0 \leq \forall\delta < \omega$, $\exists r \mid r_{top}$ such that $r \approx 2^\delta$.

UNIVERSAL SPROUT

- To enable Rational Rescale with arbitrary bit-lengths,

$Q = q_0 q_1 \cdots q_\ell \cdot r$ should represent integers of any bit-lengths.

  - Sprout r_{top} should satisfy the following:

For word-size ω, $0 \leq \forall\delta < \omega$, $\exists r \mid r_{top}$ such that $r \approx 2^\delta$.

$\Rightarrow$ We call such r_{top} a universal sprout.

UNIVERSAL SPROUT

- To enable Rational Rescale with arbitrary bit-lengths,

$$Q = q_0 q_1 \cdots q_\ell \cdot r \text{ should represent integers of any bit-lengths.}$$

  - Sprout r_{top} should satisfy the following:

$$\text{For word-size } \omega,\ 0 \le \forall\delta < \omega,\ \exists r \mid r_{top} \text{ such that } r \approx 2^{\delta}.$$

$$\Rightarrow \text{We call such } r_{top} \text{ a universal sprout.}$$

- Example (for $\omega = 60$):
  - $r_{top} = 2^{15} \cdot r_1 \cdot r_2$ with r_1 16-bit and r_2 29-bit NTT primes:

UNIVERSAL SPROUT

- To enable Rational Rescale with arbitrary bit-lengths,

$$Q = q_0 q_1 \cdots q_\ell \cdot r \text{ should represent integers of any bit-lengths.}$$

  - Sprout r_{top} should satisfy the following:

$$\text{For word-size } \omega,\ 0 \le \forall\delta < \omega,\ \exists r \mid r_{top} \text{ such that } r \approx 2^{\delta}.$$

$$\Rightarrow \text{We call such } r_{top} \text{ a universal sprout.}$$

- Example (for $\omega = 60$):
  - $r_{top} = 2^{15} \cdot r_1 \cdot r_2$ with r_1 16-bit and r_2 29-bit NTT primes:

	$0 \le \log_2 r < 16$	$16 \le \log_2 r < 29$	$29 \le \log_2 r < 44$	$44 \le \log_2 r < 60$
r	$2^0, 2^1, \dots, 2^{15}$	$r_1, 2r_1, \dots, 2^{15} r_1$	$r_2, 2r_2, \dots, 2^{15} r_2$	$r_1 r_2, 2r_1 r_2, \dots, 2^{15} r_1 r_2$

UNIVERSAL SPROUT

- To enable Rational Rescale with arbitrary bit-lengths,

$$Q = q_0 q_1 \cdots q_\ell \cdot r \text{ should represent integers of any bit-lengths.}$$

  - Sprout r_{top} should satisfy the following:

$$\text{For word-size } \omega,\ 0 \le \forall\delta < \omega,\ \exists r \mid r_{top} \text{ such that } r \approx 2^\delta.$$

$\Rightarrow$ We call such r_{top} a universal sprout.

- Example (for $\omega = 60$):
  - $r_{top} = 2^{15} \cdot r_1 \cdot r_2$ with r_1 16-bit and r_2 29-bit NTT primes:

	$0 \le \log_2 r < 16$	$16 \le \log_2 r < 29$	$29 \le \log_2 r < 44$	$44 \le \log_2 r < 60$
r	$2^0, 2^1, \dots, 2^{15}$	$r_1, 2r_1, \dots, 2^{15}r_1$	$r_2, 2r_2, \dots, 2^{15}r_2$	$r_1r_2, 2r_1r_2, \dots, 2^{15}r_1r_2$

1) Embed $\mathbb{Z}_{2^{15}} \hookrightarrow \mathbb{Z}_p$ and 2) composite NTT with $\mathbb{Z}_{r_1} \times \mathbb{Z}_{r_2} \cong \mathbb{Z}_{r_1 r_2}$

UNIVERSAL SPROUT

- To enable Rational Rescale with arbitrary bit-lengths,

$Q = q_0 q_1 \cdots q_\ell \cdot r$ should represent integers of any bit-lengths.

  - Sprout r_{top} should satisfy the following:

For word-size ω, $0 \le \forall\delta < \omega$, $\exists r \mid r_{top}$ such that $r \approx 2^\delta$.

$\Rightarrow$ We call such r_{top} a universal sprout.

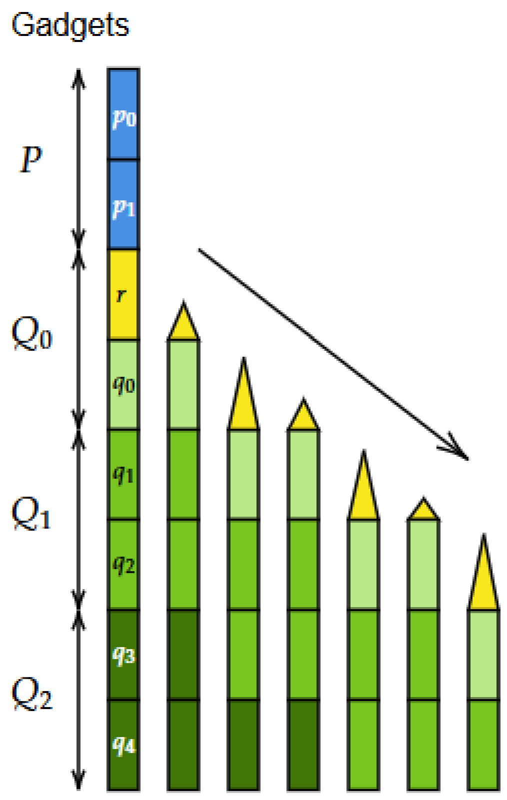
- Example (for $\omega = 60$):
  - $r_{top} = 2^{15} \cdot r_1 \cdot r_2$ with r_1 16-bit and r_2 29-bit NTT primes:

	$0 \le \log_2 r < 16$	$16 \le \log_2 r < 29$	$29 \le \log_2 r < 44$	$44 \le \log_2 r < 60$
r	$2^0, 2^1, \dots, 2^{15}$	$r_1, 2r_1, \dots, 2^{15}r_1$	$r_2, 2r_2, \dots, 2^{15}r_2$	$r_1r_2, 2r_1r_2, \dots, 2^{15}r_1r_2$

1) Embed $\mathbb{Z}_{2^{15}} \hookrightarrow \mathbb{Z}_p$ and 2) composite NTT with $\mathbb{Z}_{r_1} \times \mathbb{Z}_{r_2} \cong \mathbb{Z}_{r_1 r_2}$

$\Rightarrow$ Universal sprout, within 2 machine words.

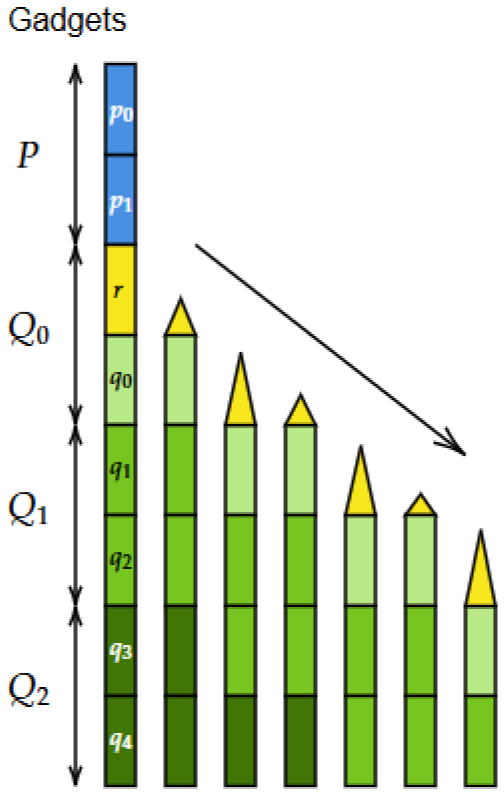
MODULUS RESURRECTION DURING COMPUTATIONS



Modulus descending scenario
during HE computations

MODULUS RESURRECTION DURING COMPUTATIONS

When rescaling $Q = q_0 q_1 \cdots q_\ell \cdot r$ by q to obtain $Q' \approx Q/q$,



Modulus descending scenario during HE computations

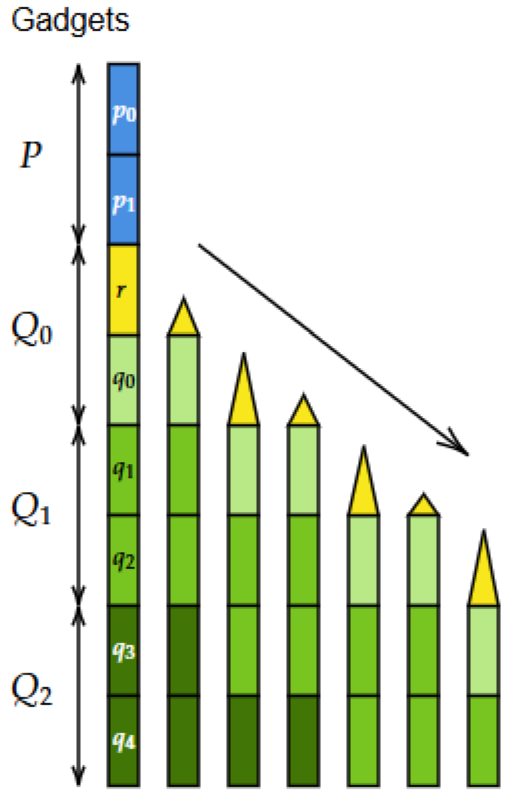
MODULUS RESURRECTION DURING COMPUTATIONS

When rescaling $Q = q_0 q_1 \cdots q_\ell \cdot r$ by q to obtain $Q' \approx Q/q$,

- If $q \leq r$,

$$Q' \approx q_0 q_1 \cdots q_\ell \cdot (r/q).$$

  - Choose $r' \mid r_{top}$ such that $r' \approx r/q$.



Modulus descending scenario during HE computations

MODULUS RESURRECTION DURING COMPUTATIONS

When rescaling $Q = q_0 q_1 \cdots q_\ell \cdot r$ by q to obtain $Q' \approx Q/q$,

- If $q \leq r$,

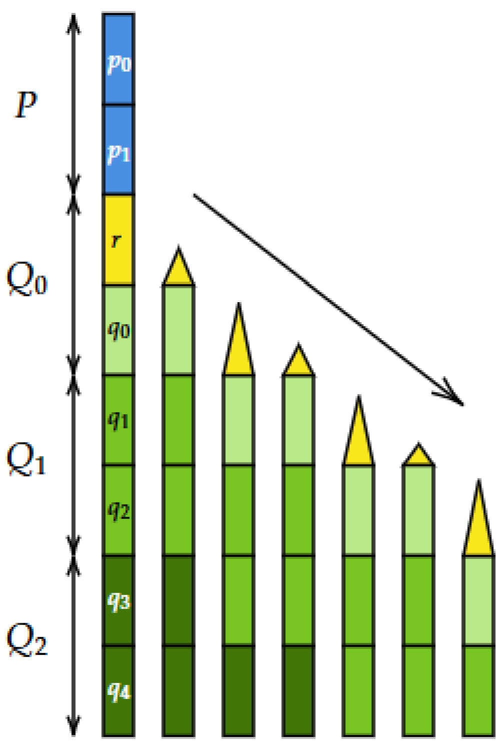
$$Q' \approx q_0 q_1 \cdots q_\ell \cdot (r/q).$$

  - Choose $r' \mid r_{top}$ such that $r' \approx r/q$.

- If $q > r$,

$$Q' \approx q_0 q_1 \cdots q_{\ell-1} \cdot (q_\ell r/q).$$

  - Choose $r' \mid r_{top}$ such that $r' \approx (2^\omega r)/q$.
  - Some factors of r_{top} are resurrected during Rational Rescale.



Modulus descending scenario during HE computations

MODULUS RESURRECTION DURING COMPUTATIONS

When rescaling $Q = q_0 q_1 \cdots q_\ell \cdot r$ by q to obtain $Q' \approx Q/q$,

- If $q \le r$,

$$Q' \approx q_0 q_1 \cdots q_\ell \cdot (r/q).$$

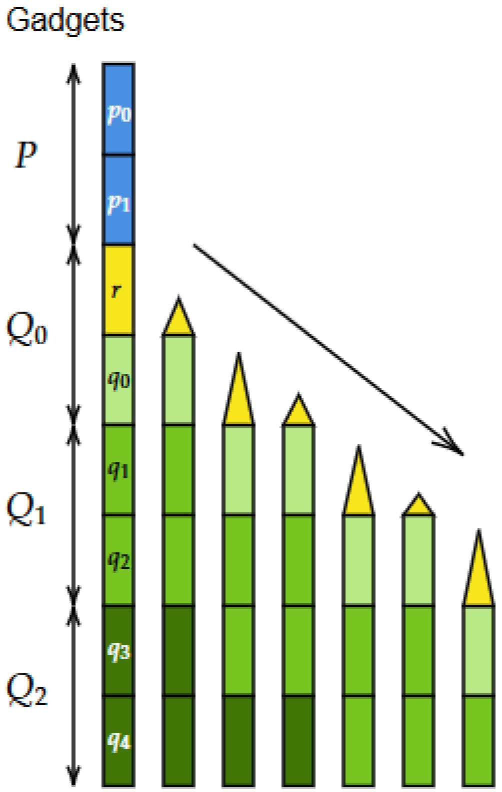
  - Choose $r' \mid r_{top}$ such that $r' \approx r/q$.

- If $q > r$,

$$Q' \approx q_0 q_1 \cdots q_{\ell-1} \cdot (q_\ell r/q).$$

  - Choose $r' \mid r_{top}$ such that $r' \approx (2^\omega r)/q$.
  - Some factors of r_{top} are resurrected during Rational Rescale.

$\Rightarrow$ We call this process Modulus Resurrectio



Modulus descending scenario during HE computations

APPLICATION OF GRAFTING & EXPERIMENTAL RESULTS

WHEN APPLIED TO STANDARD CKKS / BIT-CKKS

Parameter	N	log PQ	# factors	Mult (ms)	Boot (s)	Key size (MB)
HEaaN [Cry22]	2^{15}	777	22	102.20	14.5	115.34
		780	13	57.28 **(1.8x)**	7.6 **(1.9x)**	44.04 **(62% ↓)**
	2^{16}	1555	30	329.38	37.0	157.29
			27	247.45 **(1.3x)**	35.5 **(1.1x)**	146.80 **(7% ↓)**
Sec. Guide. [BCC+24]	2^{16}	1734	35	360.84	86.5	220.20
			29	179.87 **(2.0x)**	71.7 **(1.2x)**	157.29 **(29% ↓)**

- Mult up to 2.0x, BTS up to 1.9x, key size reduced by up to 62%.
  - Parameter w/ many small scale factors → accelerates well!

[Cry22] CryptoLab, HEaaN library, 2022.
[BCC+24] Jean-Philippe Bossuat et. al., Security guidelines for implementing homomorphic encryption, CIC 2025.
[BCKS24] Youngjin Bae et. al., Bootstrapping bits with CKKS, Eurocrypt 2024.

WHEN APPLIED TO STANDARD CKKS / BIT-CKKS

Parameter	N	log PQ	# factors	Mult (ms)	Boot (s)	Key size (MB)
HEaaN [Cry22]	2^{15}	777	22	102.20	14.5	115.34
		780	13	57.28 (1.8x)	7.6 (1.9x)	44.04 (62% ↓)
	2^{16}	1555	30	329.38	37.0	157.29
			27	247.45 (1.3x)	35.5 (1.1x)	146.80 (7% ↓)
Sec. Guide. [BCC+24]	2^{16}	1734	35	360.84	86.5	220.20
			29	179.87 (2.0x)	71.7 (1.2x)	157.29 (29% ↓)
Bit-CKKS [BCKS24]	2^{14}	424	14	16.1	5.18	47.71
		426	8	16.2 **(1.8x)**	2.74 **(1.9x)**	16.52 **(65% ↓)**
	2^{16}	1598	46	884.8	102.10	144.70
		1522	25	428.1 **(2.1x)**	56.41 **(1.8x)**	109.05 **(25% ↓)**

- Mult up to 2.1x, BTS up to 1.9x, key size reduced by up to 65%.
  - Parameter w/ many small scale factors → accelerates well!

[Cry22] CryptoLab, HEaaN library, 2022.
[BCC+24] Jean-Philippe Bossuat et. al., Security guidelines for implementing homomorphic encryption, CIC 2025.
[BCKS24] Youngjin Bae et. al., Bootstrapping bits with CKKS, Eurocrypt 2024.

APPLICATION TO HOMOMORPHIC COMPARISON [CKK20]

- **Δ-Changeability**: Decoupling allows changing Δ freely even during computation!

[CKK20] Jung Hee Cheon et. al., Efficient homomorphic comparison methods with optimal complexity, Asiacrypt 2020.

APPLICATION TO HOMOMORPHIC COMPARISON [CKK20]

- **Δ-Changeability**: Decoupling allows changing Δ freely even during computation!

- **Application?**
  - Adaptive Precision Computation in **Plain World** allows better latency/memory:
    - ML training: FP16 ⇔ FP32
    - Iterative solvers: FP8 → FP16 → FP32

[CKK20] Jung Hee Cheon et. al., Efficient homomorphic comparison methods with optimal complexity, Asiacrypt 2020.

APPLICATION TO HOMOMORPHIC COMPARISON [CKK20]

- **Δ-Changeability**: Decoupling allows changing Δ freely even during computation!

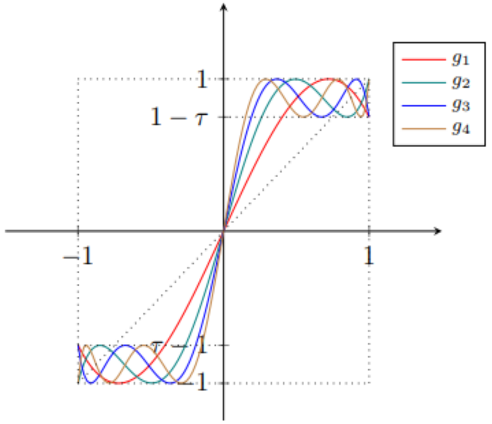
- **Application?**
  - Adaptive Precision Computation in **Plain World** allows better latency/memory:
    - ML training: FP16 ⇔ FP32
    - Iterative solvers: FP8 → FP16 → FP32

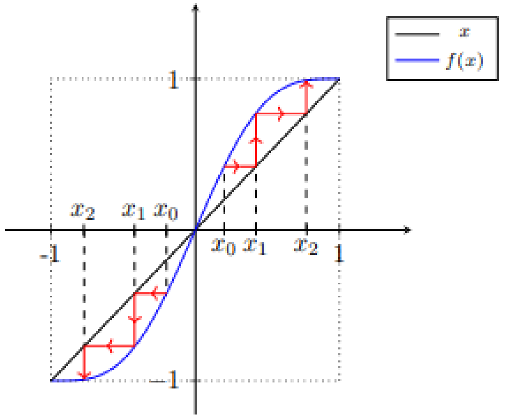
  - In **Encrypted World?**
    - Grafting now allows changing **precision** by changing **Δ**

[CKK20] Jung Hee Cheon et. al., Efficient homomorphic comparison methods with optimal complexity, Asiacrypt 2020.

APPLICATION TO HOMOMORPHIC COMPARISON [CKK20]

- Homomorphic Comparison [CKK20] use iterative method:
  - Evaluate $f^k \circ g^\ell : I_\epsilon \to I_{1-2^{-\alpha}}$, iteratively narrowing the interval.
    - $I_\epsilon := [-1, -\epsilon] \cup [\epsilon, 1]$
    - f and g: deg-7 polynomials

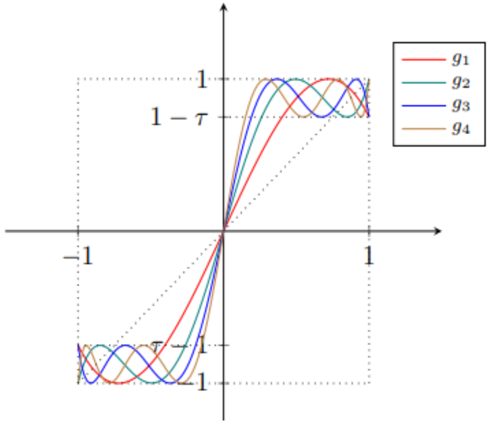


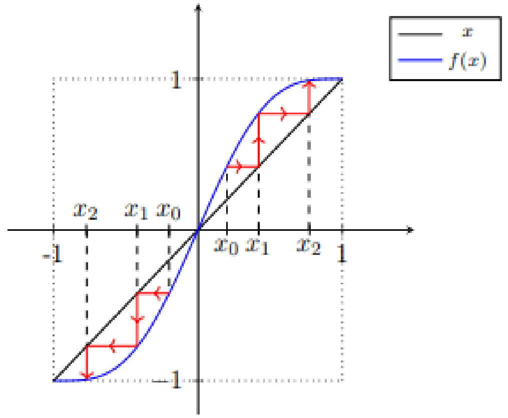


[CKK20] Jung Hee Cheon et. al., Efficient homomorphic comparison methods with optimal complexity, Asiacrypt 2020.

APPLICATION TO HOMOMORPHIC COMPARISON [CKK20]

- Homomorphic Comparison [CKK20] use iterative method:
  - Evaluate $f^k \circ g^\ell : I_\epsilon \to I_{1-2^{-\alpha}}$, iteratively narrowing the interval.
    - $I_\epsilon \coloneqq [-1,-\epsilon] \cup [\epsilon, 1]$
    - f and g: deg-7 polynomials





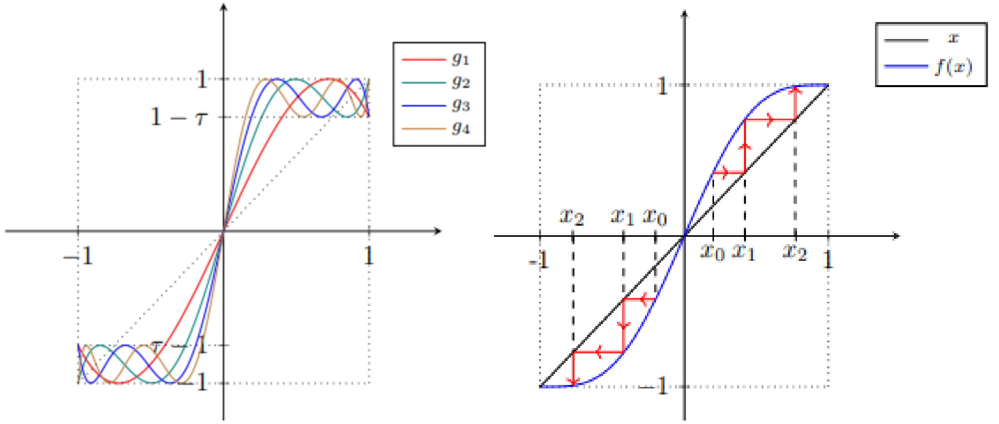
➔ We can save modulus by using smaller Δ for early iterations!

[CKK20] Jung Hee Cheon et. al., Efficient homomorphic comparison methods with optimal complexity, Asiacrypt 2020.

APPLICATION TO HOMOMORPHIC COMPARISON [CKK20]

- Homomorphic Comparison [CKK20] use iterative method:
  - Evaluate $f^k \circ g^\ell : I_\epsilon \to I_{1-2^{-\alpha}}$, iteratively narrowing the interval.
    - $I_\epsilon := [-1, -\epsilon] \cup [\epsilon, 1]$
    - f and g: deg-7 polynomials

➔ We can save modulus by using smaller Δ for early iterations!



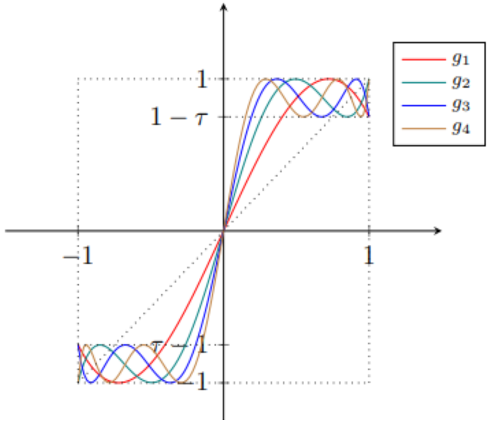
Comparison Function	$f^{(2)} \circ g^{(4)}$	
Methods	Original	Changing Δ (28, 30, 42 bits)
Consumed Modulus (bit)	$(42 \times 3) \times 6 = 756$	$(28 \times 3) \times 4 + (30 \times 3) + (42 \times 3) = 552$ **(27% ↓)**
Precision (bit)	23.1	23.1

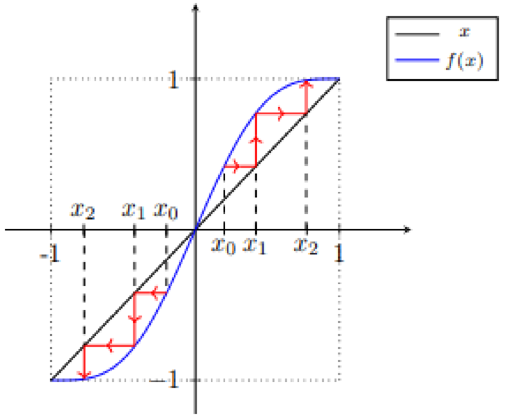
⌘ Bit-precision $:= -\log_2|\text{max error}|$ from 100 iterations

[CKK20] Jung Hee Cheon et. al., Efficient homomorphic comparison methods with optimal complexity, Asiacrypt 2020.

APPLICATION TO HOMOMORPHIC COMPARISON [CKK20]

- Homomorphic Comparison [CKK20] use iterative method:
  - Evaluate $f^k \circ g^\ell : I_\epsilon \to I_{1-2^{-\alpha}}$, iteratively narrowing the interval.
    - $I_\epsilon := [-1, -\epsilon] \cup [\epsilon, 1]$
    - f and g: deg-7 polynomials





➔ We can save modulus by using smaller Δ for early iterations!

Comparison Function	$f^{(2)} \circ g^{(4)}$		$f^{(2)} \circ g^{(8)}$	
Methods	Original	Changing Δ (28, 30, 42 bits)	Original	Changing Δ (31, 42 bits)
Consumed Modulus (bit)	(42 × 3) × 6 = 756	(28 × 3) × 4 + (30 × 3) + (42 × 3) = 552 **(27% ↓)**	(42 × 3) × 10 = 1260	(31 × 3) × 9 + (42 × 3) = 963 **(24% ↓)**
Precision (bit)	23.1	23.1	23.5	23.3

⌘ Bit-precision := $-\log_2|\max \text{error}|$ from 100 iterations

[CKK20] Jung Hee Cheon et. al., Efficient homomorphic comparison methods with optimal complexity, Asiacrypt 2020.

SUMMARY

- Grafting redesigns modulus usage in RNS-CKKS
  - Enable arbitrary bit-lengths Rescale,
  - No additional key-switching keys required.

SUMMARY

- Grafting redesigns modulus usage in RNS-CKKS
  - Enable arbitrary bit-lengths Rescale,
  - No additional key-switching keys required.

$\Rightarrow$ Grafting improves RNS-CKKS:

1. **Performance** from machine word-size RNS moduli
   - Up to 2.1x faster multiplication and 1.9x faster bootstrapping.
   - Up to 62% reduction in ciphertext/key-switching key size.

SUMMARY

- Grafting redesigns modulus usage in RNS-CKKS
  - Enable arbitrary bit-lengths Rescale,
  - No additional key-switching keys required.

$\Rightarrow$ Grafting improves RNS-CKKS:

1. **Performance** from machine word-size RNS moduli
   - Up to 2.1x faster multiplication and 1.9x faster bootstrapping.
   - Up to 62% reduction in ciphertext/key-switching key size.

2. **Flexibility** from decoupling
   - Scale/precision adjustable independently of ciphertext modulus.
   - Modulus saving for iterative methods, *e.g.*, homomorphic comparison.

EPRINT: 2024/1014

THANK YOU!