

Leveraging Discrete CKKS to Bootstrap in High Precision

Hyeongmin Choe

CryptoLab, Inc.

Jaehyung Kim

Stanford University

Damien Stehlé

CryptoLab, Inc.

Elias Suvanto

University of Luxembourg



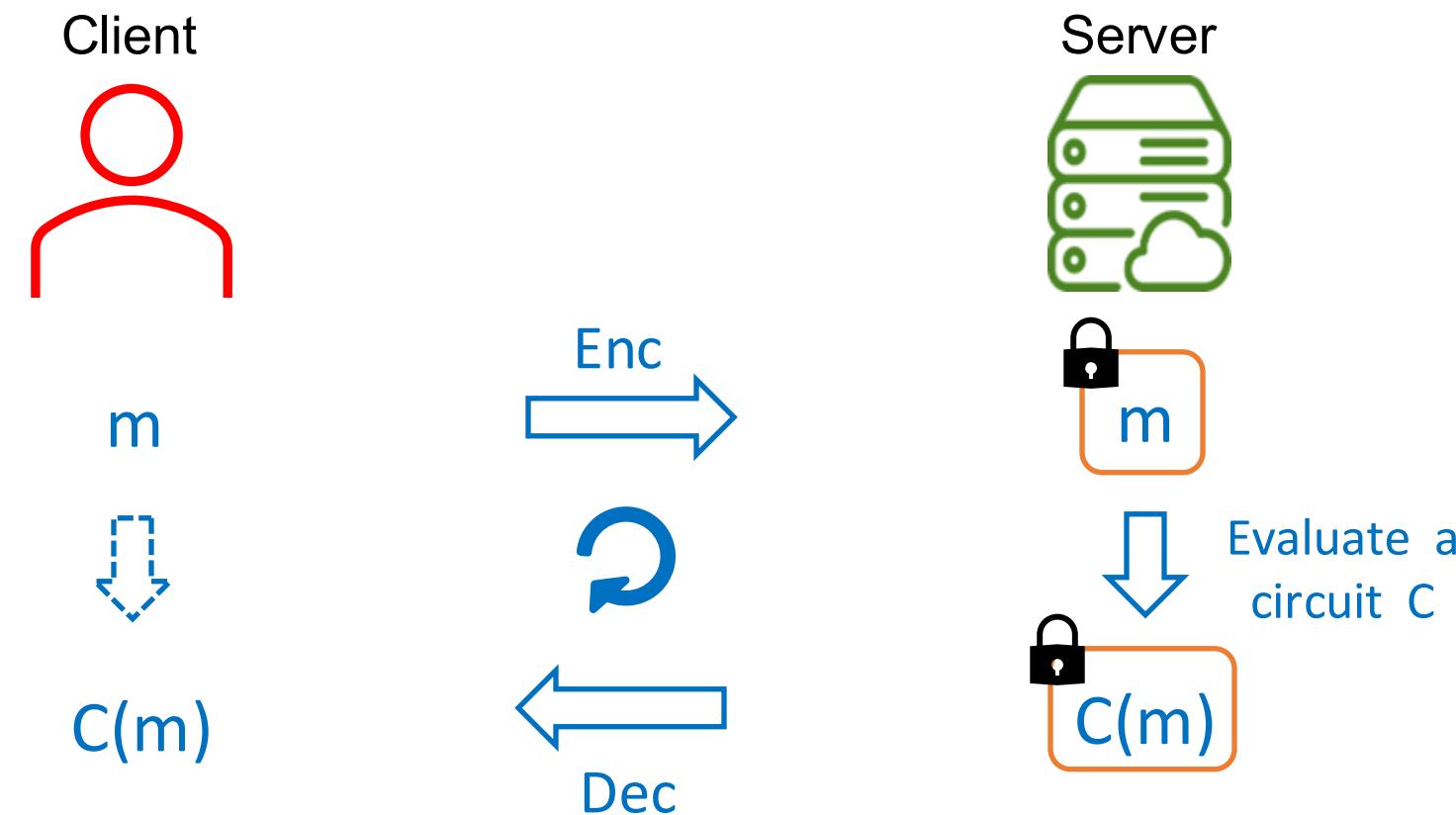
October 14th
ACM CCS 2025

C R Y P T O L A B
주식회사 크립토랩



Fully Homomorphic Encryption

- **FHE** enables computations on encrypted data without decryption.
 - Allows *Evaluation on Data* while keeping *Privacy*
 - One of the most popular Privacy Enhancing Technologies (PETs)





- CKKS FHE Scheme
 - Support arithmetic computation on $(\mathbb{C}, +, \times)$ Δ: scale factor
 - Encoding: $m \in \mathbb{C}^N$: $Ecd_{\Delta}(m) = [\Delta \cdot iDFT(m)] \in \mathbb{Z}[x]/(x^N + 1)$
 - Q: Ciphertext modulus, **decreases after each Mult**:
 - $Ecd_{\Delta}(m_1) \cdot Ecd_{\Delta}(m_2) \approx \Delta \cdot Ecd_{\Delta}(m_1 \odot m_2)$: Q becomes Q/Δ .



- CKKS FHE Scheme

- Support arithmetic computation on $(\mathbb{C}, +, \times)$ Δ: scale factor

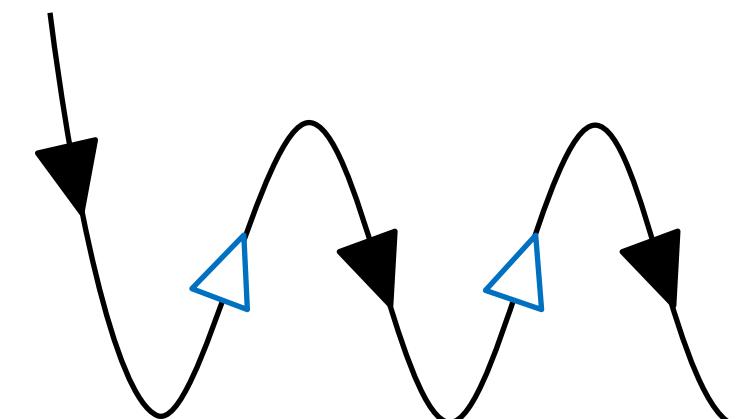
- **Encoding:** $m \in \mathbb{C}^N$: $Ecd_{\Delta}(m) = [\Delta \cdot iDFT(m)] \in \mathbb{Z}[x]/(x^N + 1)$

- **Q:** Ciphertext modulus, **decreases after each Mult:**

- $Ecd_{\Delta}(m_1) \cdot Ecd_{\Delta}(m_2) \approx \Delta \cdot Ecd_{\Delta}(m_1 \odot m_2)$: Q becomes Q/Δ .

- **Bootstrapping** allows arbitrary computation by refreshing Q

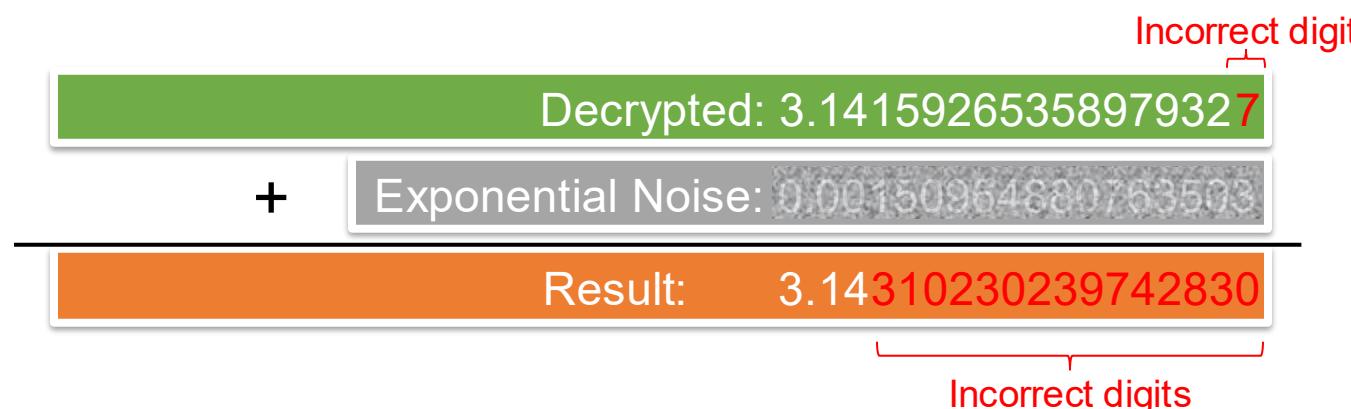
- Usually, ≤ 20 bits of precision
 - Our target: 60 - 100 bits





Why High-Precision?

- For $IND\text{-}CPA^D$ Security, Circuit Privacy or Threshold Decryption, we add **exponential noise** after evaluations (a.k.a. noise flooding):
 - 1) (High-Precision) CKKS computation,
 - 2) Decrypt,
 - 3) Add exponential noise.

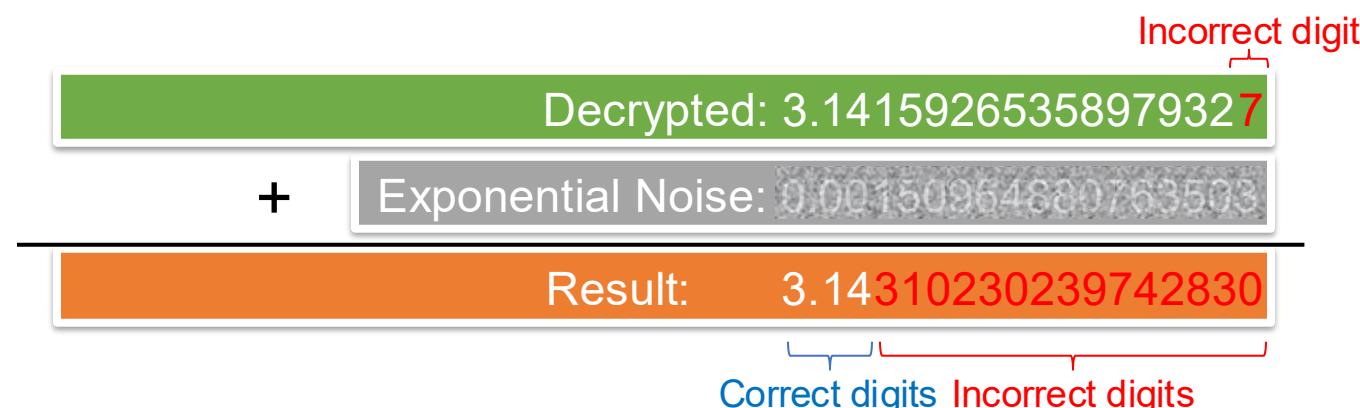




Why High-Precision?

- For $IND\text{-}CPA^D$ Security, Circuit Privacy or Threshold Decryption, we add **exponential noise** after evaluations (a.k.a. noise flooding):
 - 1) (High-Precision) CKKS computation,
 - 2) Decrypt,
 - 3) Add exponential noise.

Then, only a few bits of the *correct message* remain:





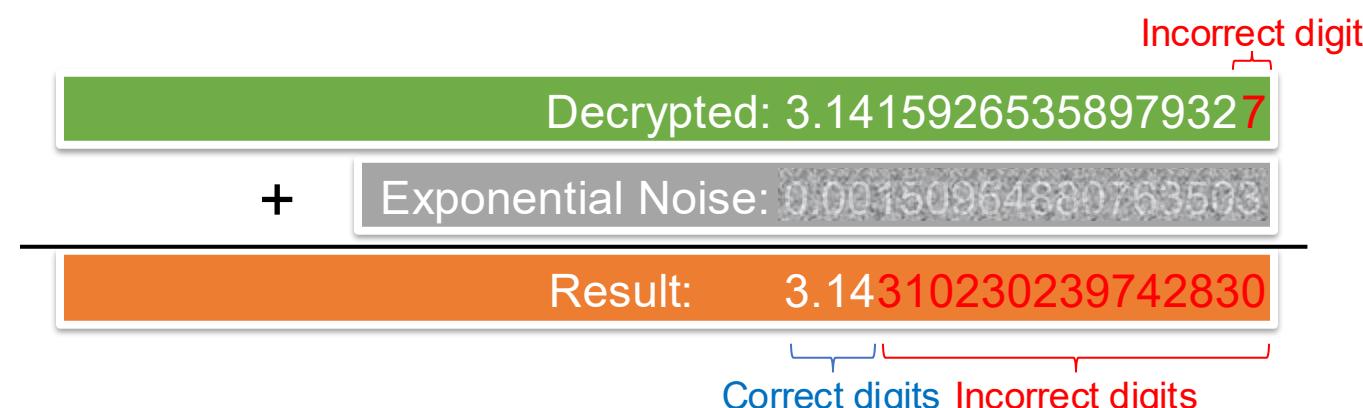
Why High-Precision?

- For $IND\text{-}CPA^D$ Security, Circuit Privacy or Threshold Decryption, we add **exponential noise** after evaluations (a.k.a. noise flooding):

≈ 64 bits for $\lambda = 128^1$

- 1) (High-Precision) CKKS computation, → Need precision $\geq (64 + \alpha)$ bits
- 2) Decrypt,
- 3) Add exponential noise.

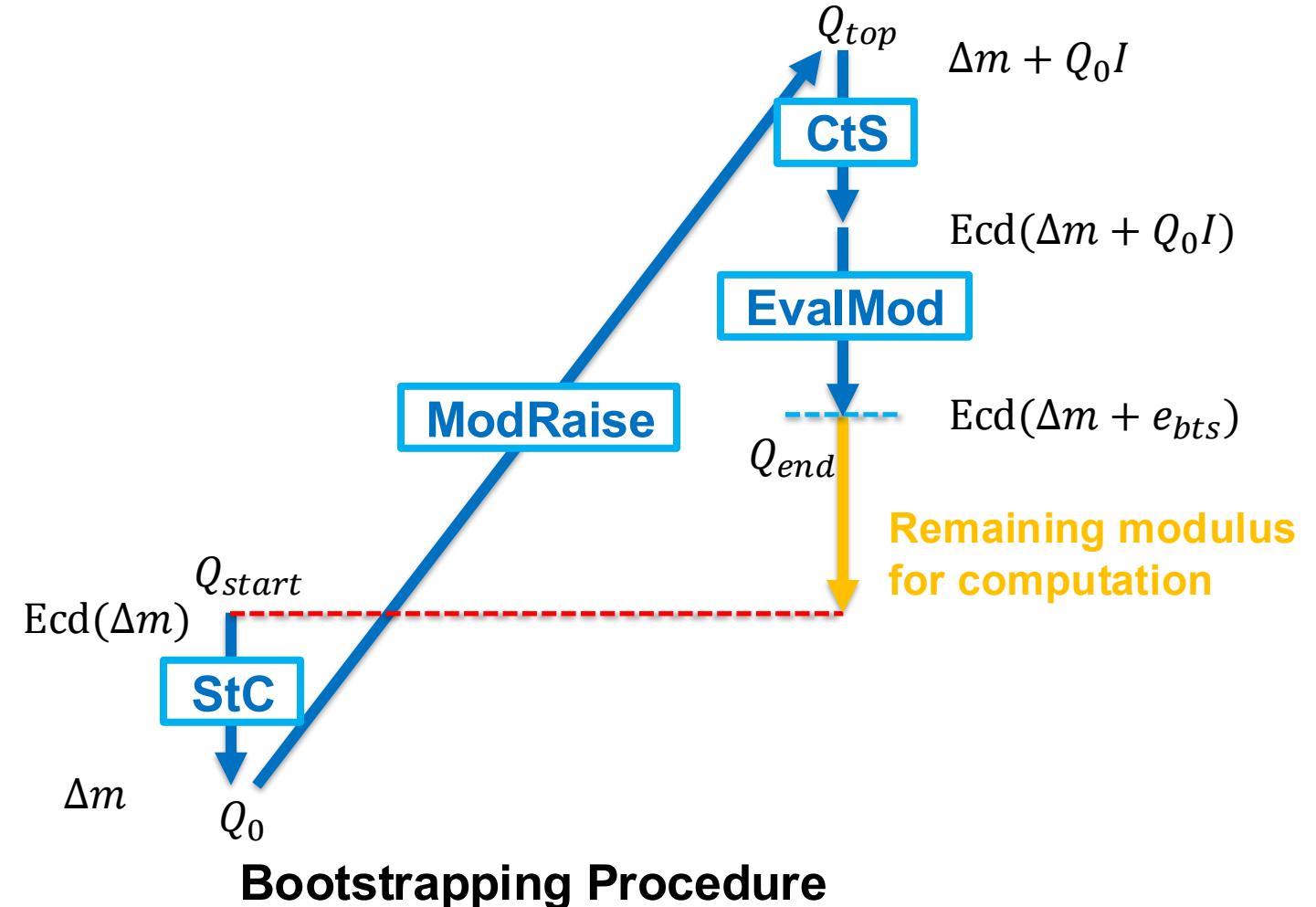
Then, only a few bits of the *correct message* remain:





Standard CKKS Bootstrapping

- CKKS Bootstrapping: Increase modulus for further computation



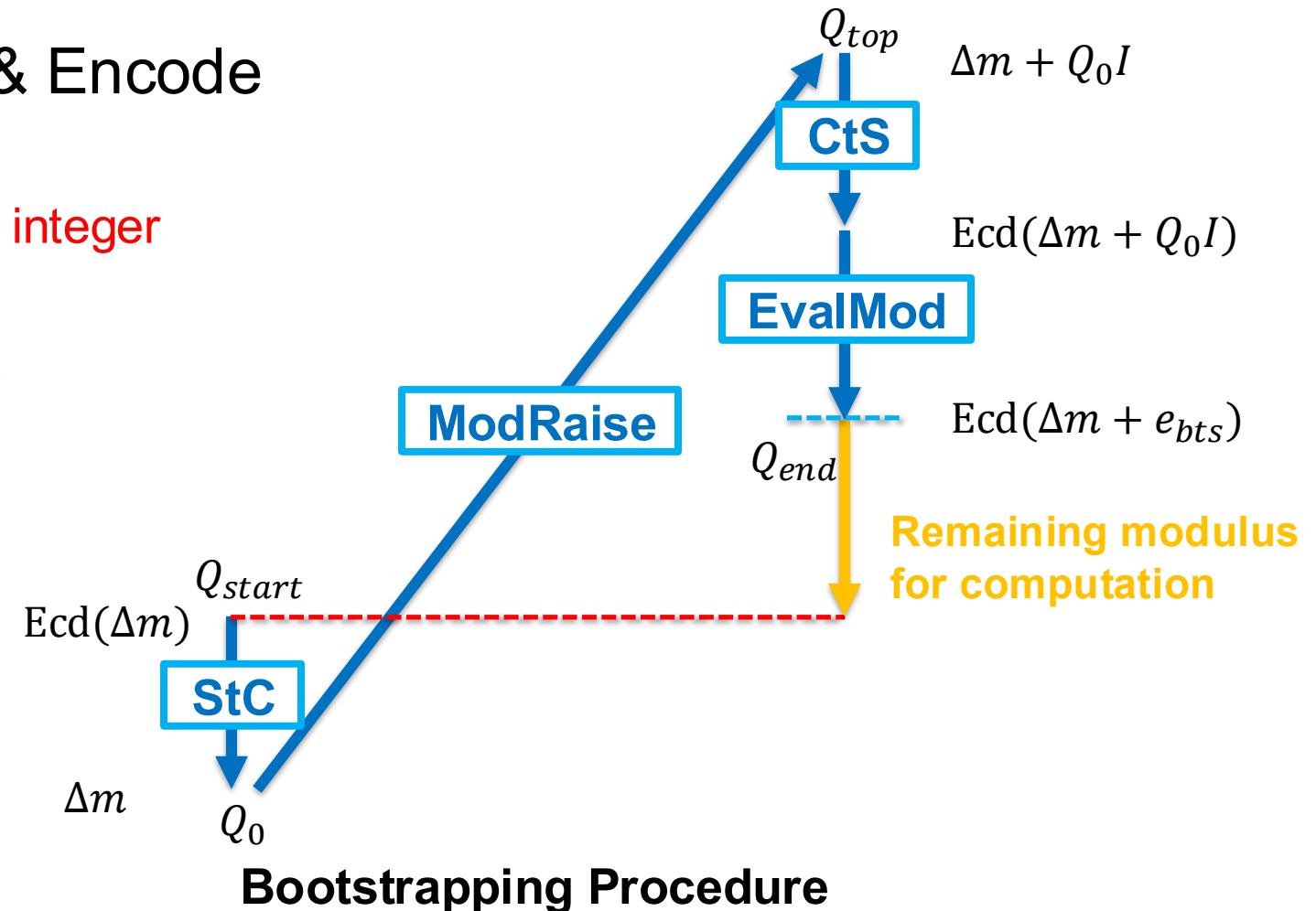


Standard CKKS Bootstrapping

- **CKKS Bootstrapping:** Increase modulus for further computation

- **StC & CtS:** homomorphic Decode & Encode

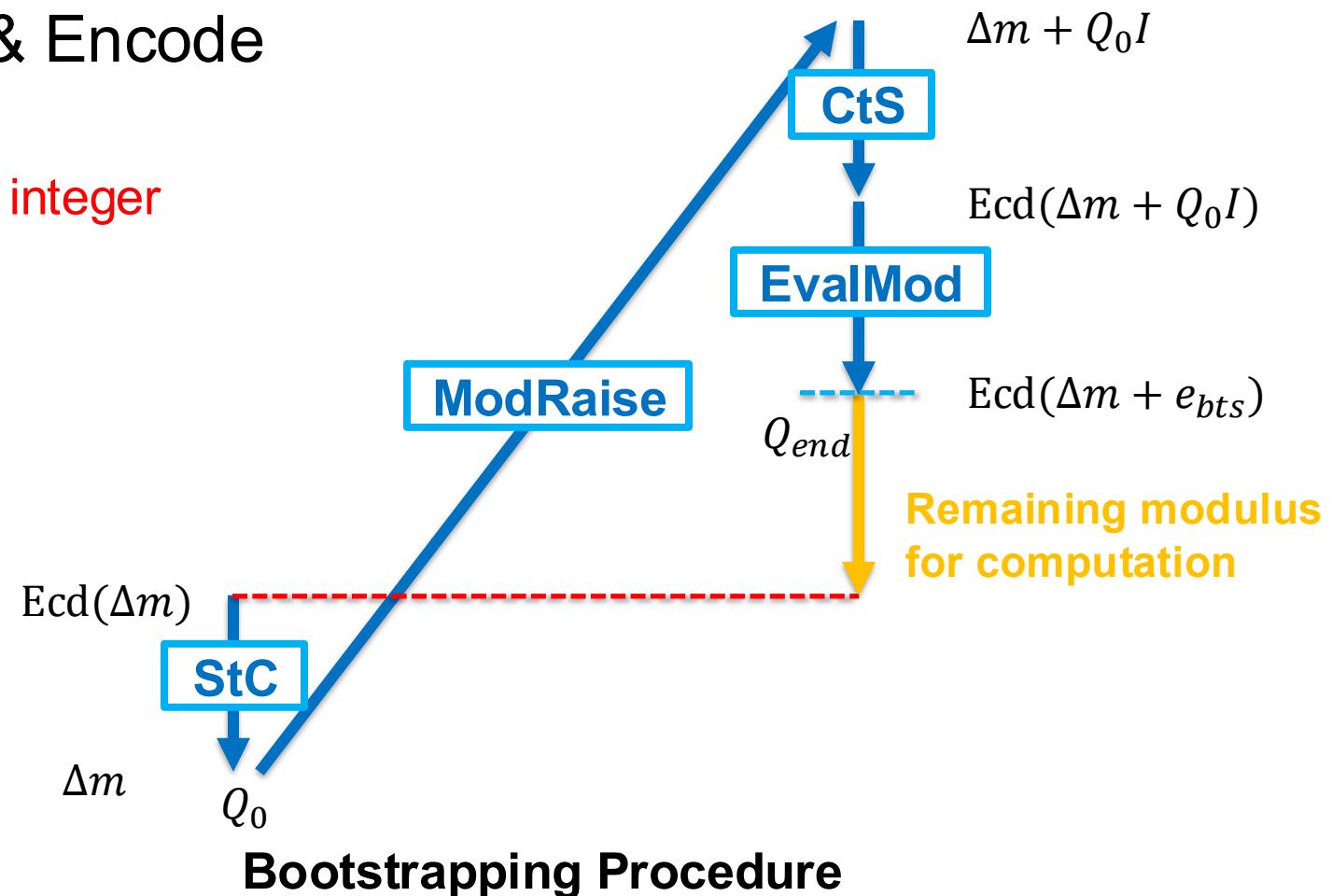
- **EvalMod:** $\Delta m + Q_0 I \mapsto \Delta m$
 - Polynomial approx. of $x \mapsto (x \bmod Q_0)$





Standard CKKS Bootstrapping

- **CKKS Bootstrapping:** Increase modulus for further computation
 - **StC & CtS:** homomorphic Decode & Encode
 - **EvalMod:** $\Delta m + Q_0 I \mapsto \Delta m$
 - Polynomial approx. of $x \mapsto (x \bmod Q_0)$
- **High-precision?**
 - EvalMod is the bottleneck:
 - Polynomial degree $\sim \Theta(\log t)$
 - Modulus consumption $\sim \Theta(t \log t)$





Standard CKKS Bootstrapping

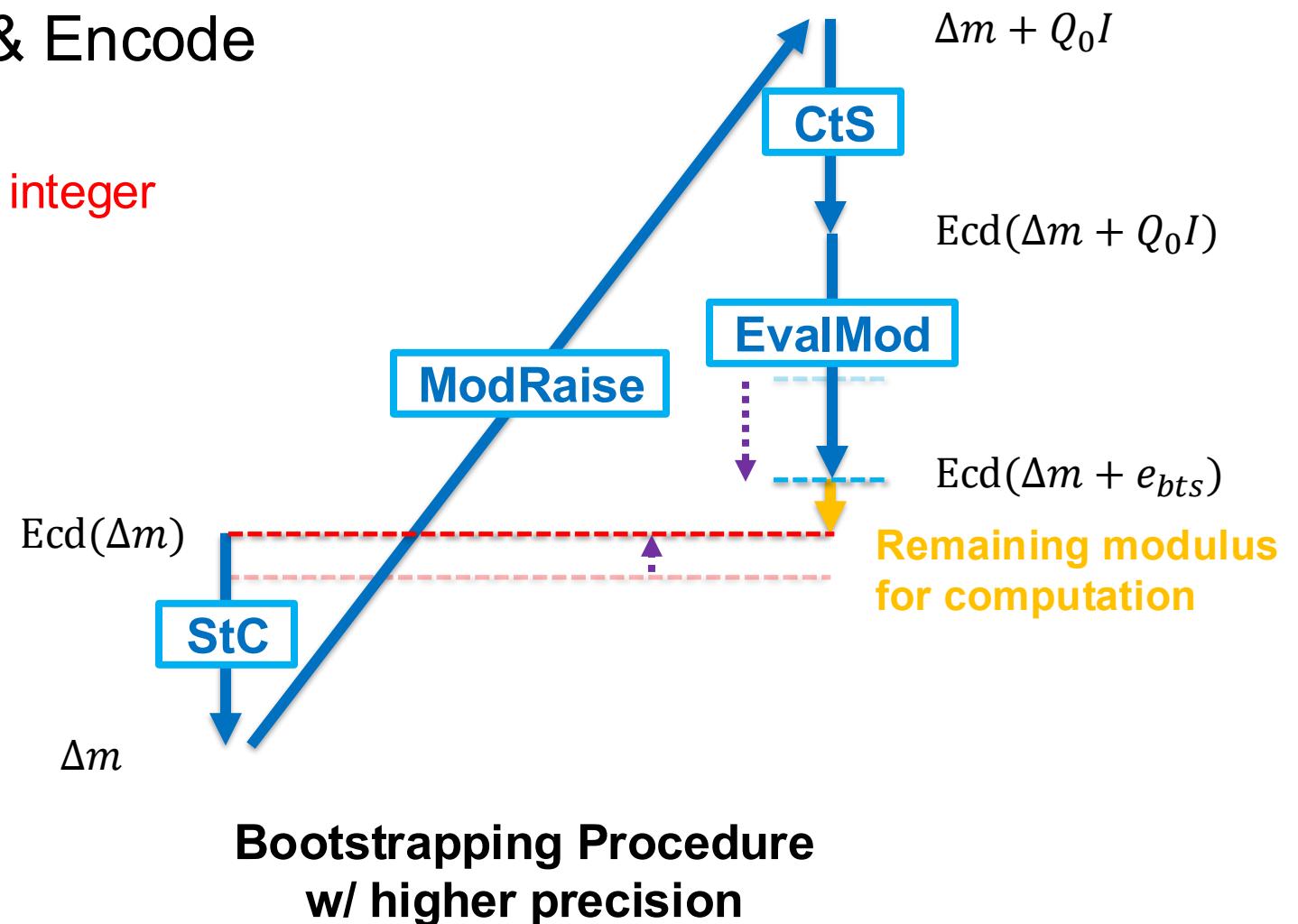
- **CKKS Bootstrapping:** Increase modulus for further computation

- **StC & CtS:** homomorphic Decode & Encode

- **EvalMod:** $\Delta m + Q_0 I \mapsto \Delta m$
 - Polynomial approx. of $x \mapsto (x \bmod Q_0)$

- **High-precision?**

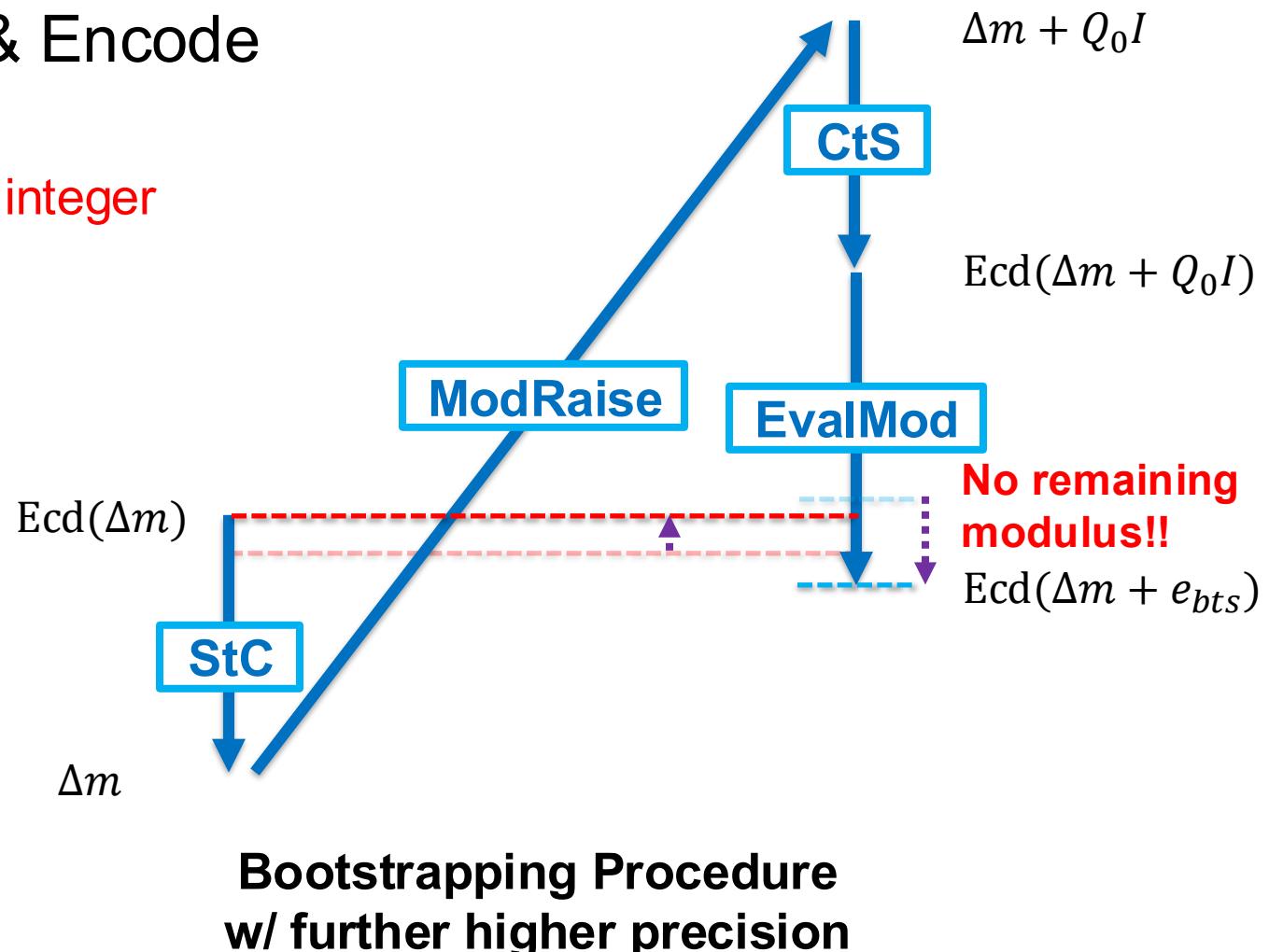
- EvalMod is the bottleneck:
 - Polynomial degree $\sim \Theta(\log t)$
 - Modulus consumption $\sim \Theta(t \log t)$





Standard CKKS Bootstrapping

- **CKKS Bootstrapping:** Increase modulus for further computation
 - **StC & CtS:** homomorphic Decode & Encode
 - **EvalMod:** $\Delta m + Q_0 I \mapsto \Delta m$
 - Polynomial approx. of $x \mapsto (x \bmod Q_0)$
- **High-precision?**
 - EvalMod is the bottleneck:
 - Polynomial degree $\sim \Theta(\log t)$
 - Modulus consumption $\sim \Theta(t \log t)$



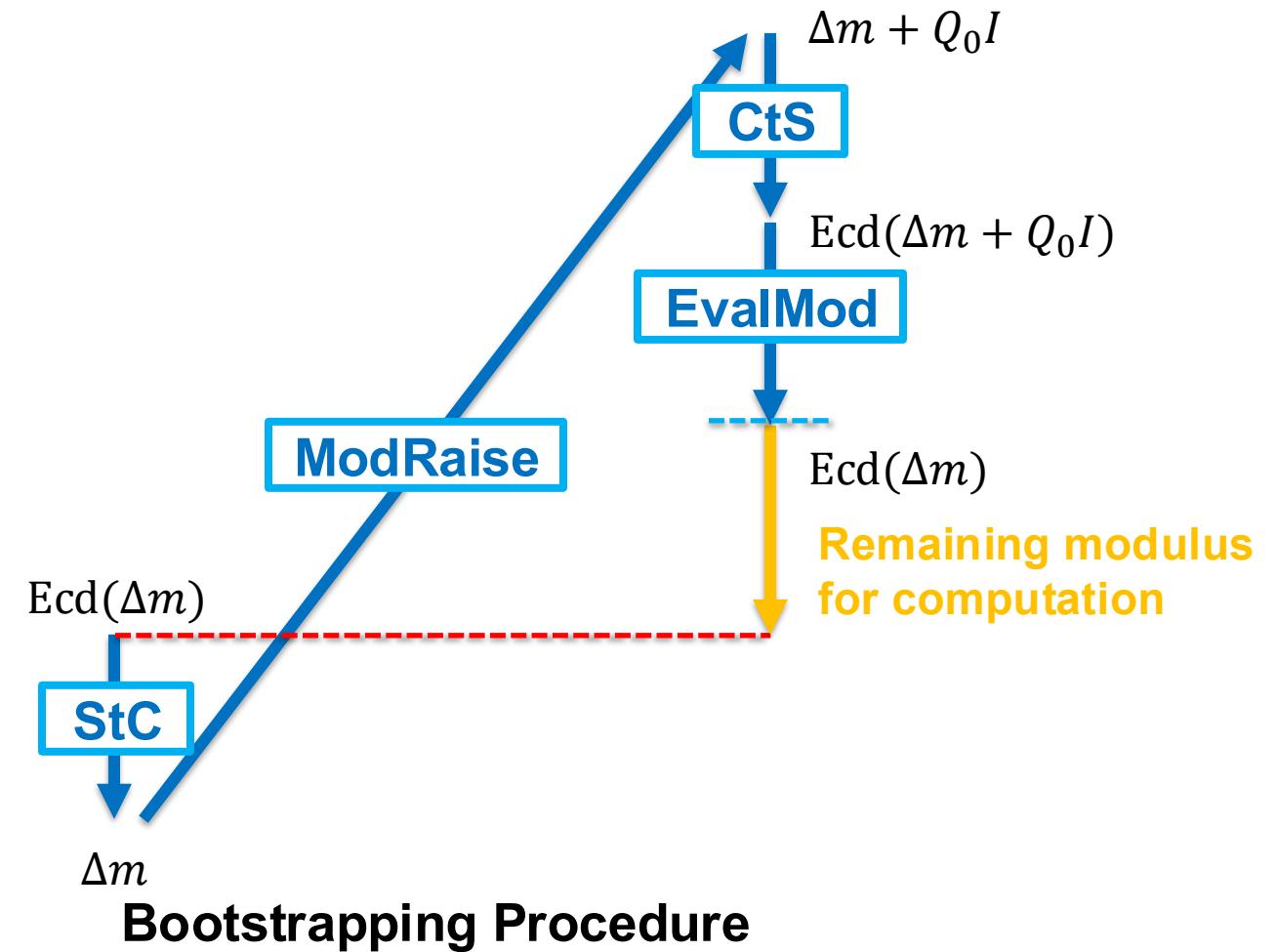


CKKS Bootstrapping via Integer Cleaning

- **EvalRound**^{2,3} instead of EvalMod

- EvalMod: $x \mapsto (x \bmod Q_0)$

$$\Delta m + Q_0 I \mapsto \Delta m$$



2. S. Kim, M. Park, J. Kim, T. Kim, and C. Min, "EvalRound Algorithm in CKKS Bootstrapping," Asiacrypt 2022.

3. H. Sung, S. Seo, T. Kim, and C. Min, "EvalRound+ Bootstrapping and its Rigorous Analysis for CKKS Scheme," ePrint Archive 2024.

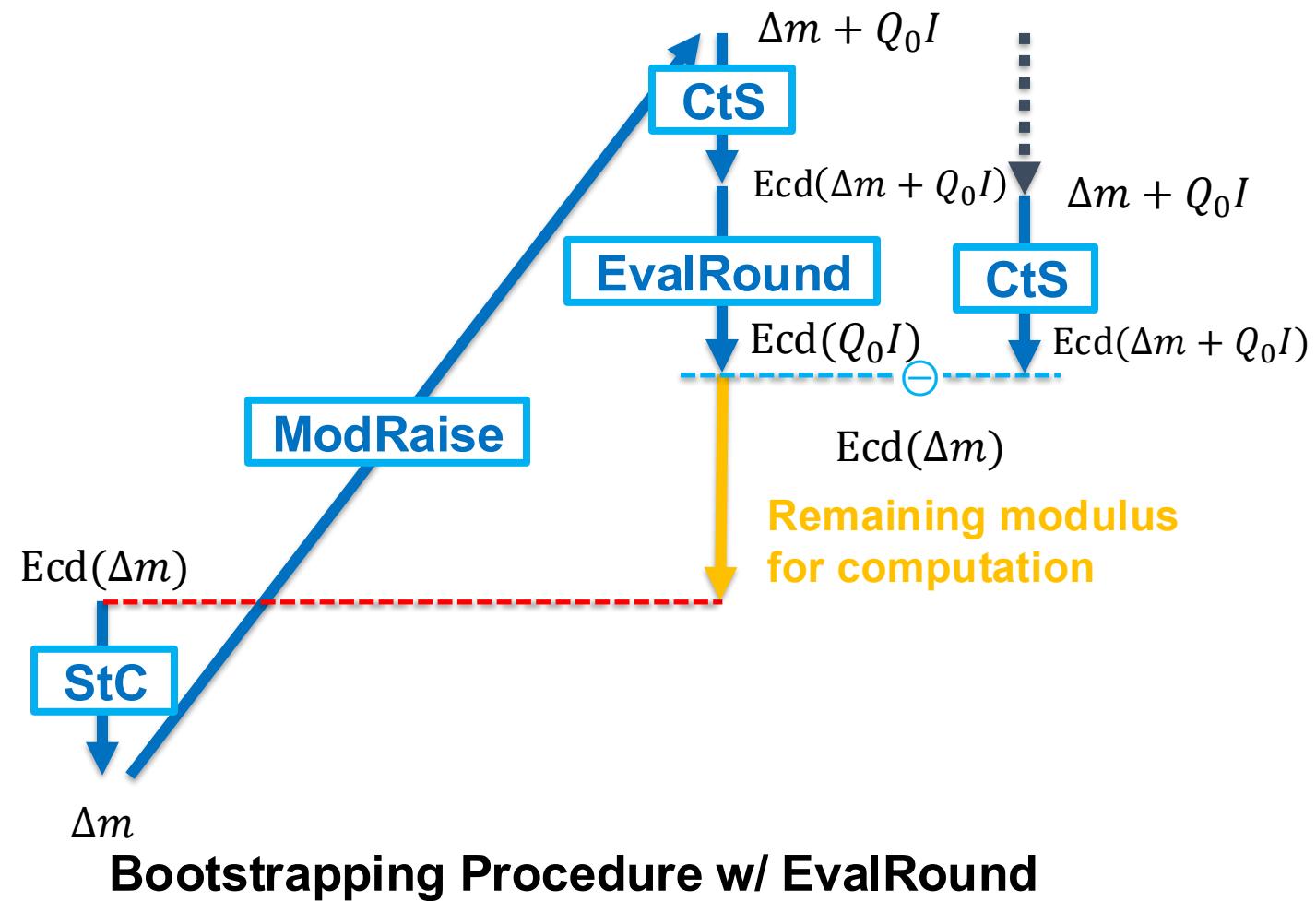


CKKS Bootstrapping via Integer Cleaning

- **EvalRound**^{2,3} instead of EvalMod

- EvalRound: $x \mapsto x - (x \bmod Q_0)$

$$\Delta m + Q_0 I \mapsto Q_0 I$$



2. S. Kim, M. Park, J. Kim, T. Kim, and C. Min, "EvalRound Algorithm in CKKS Bootstrapping," Asiacrypt 2022.

3. H. Sung, S. Seo, T. Kim, and C. Min, "EvalRound+ Bootstrapping and its Rigorous Analysis for CKKS Scheme," ePrint Archive 2024.



CKKS Bootstrapping via Integer Cleaning

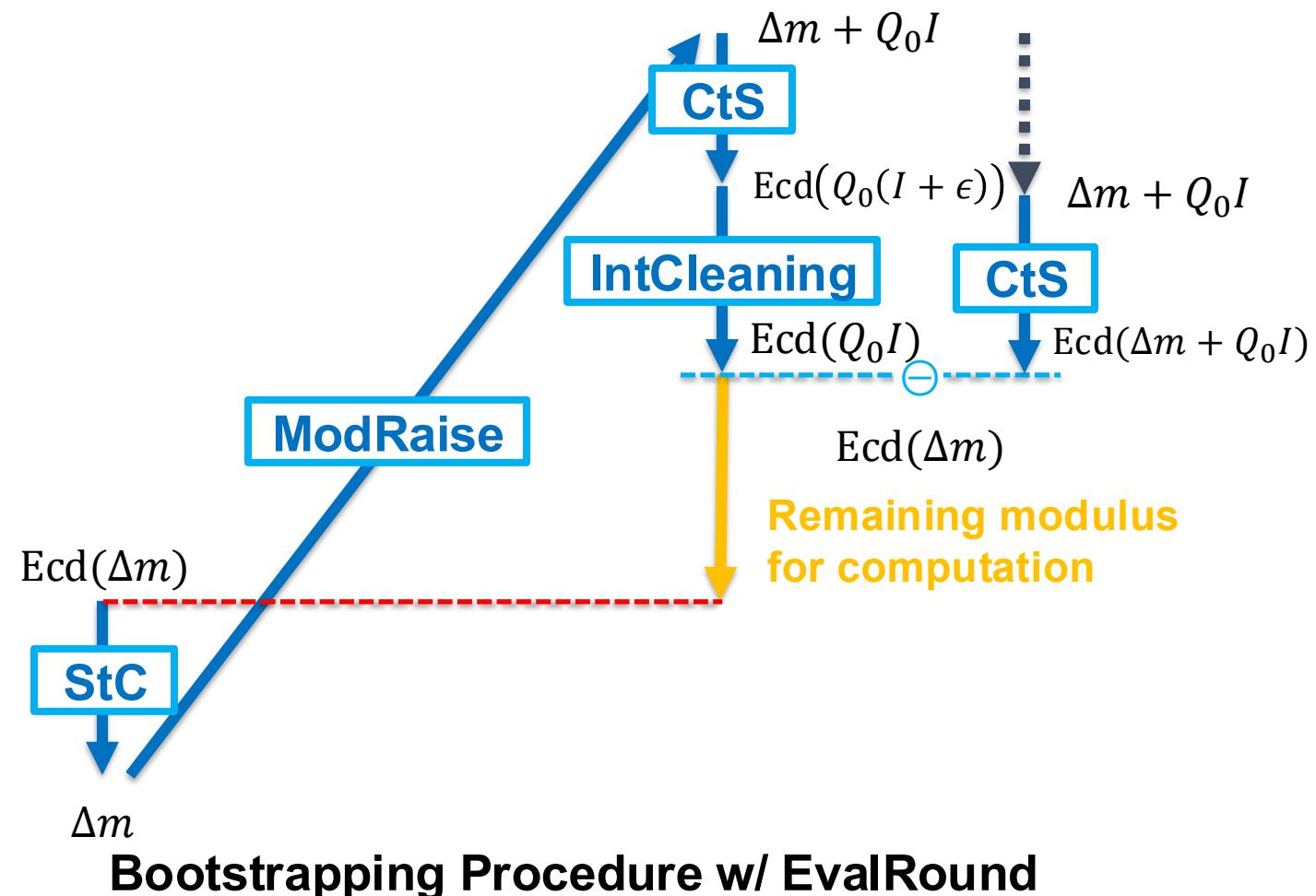
- **EvalRound**^{2,3} instead of EvalMod

- EvalRound: $x \mapsto x - (x \bmod Q_0)$

$$\Delta m + Q_0 I \mapsto Q_0 I$$

- **EvalRound = Integer Cleaning?**

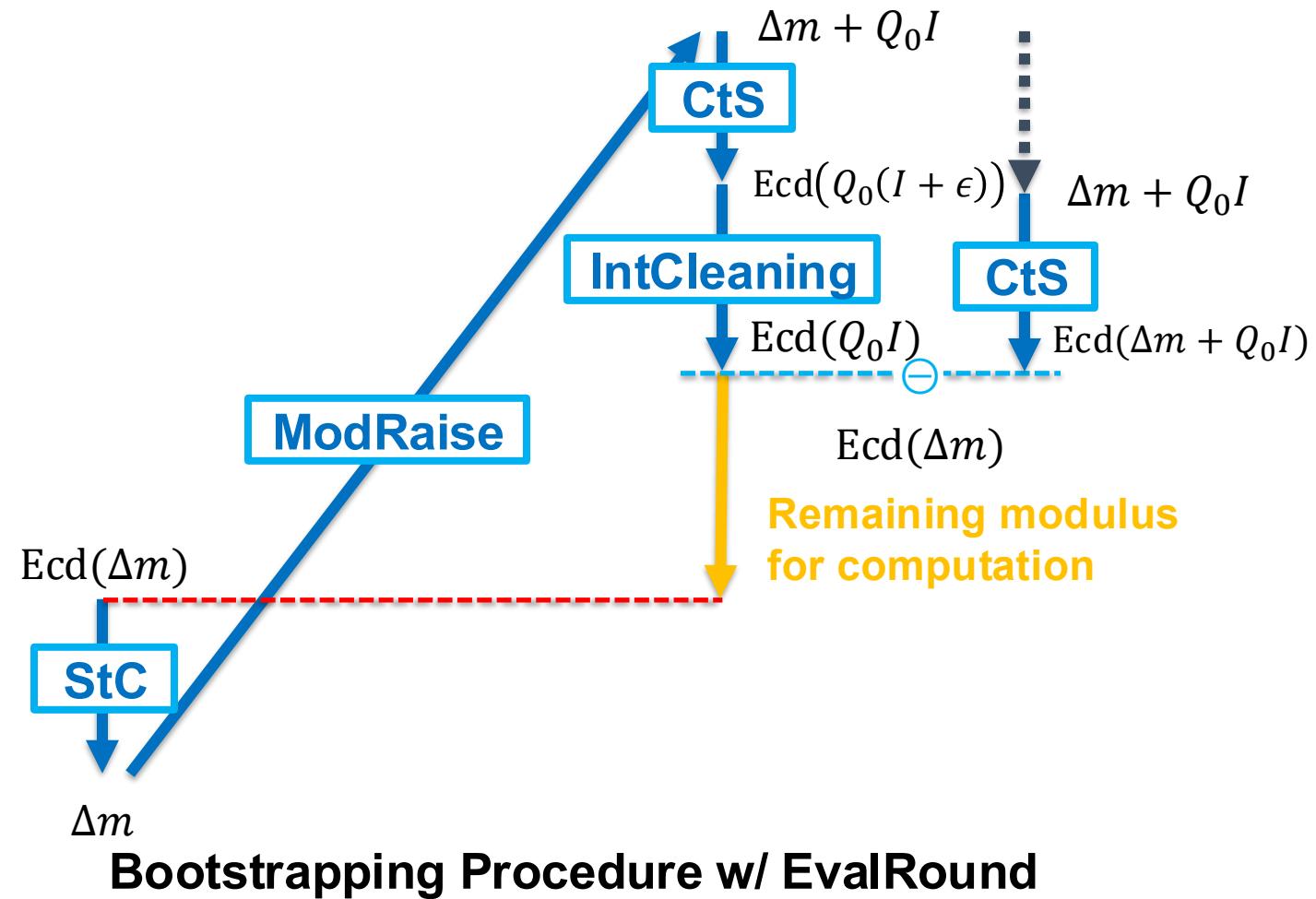
- $Q_0(I + \epsilon) \mapsto Q_0 \cdot I$
 - I : integer, $\epsilon = \Delta m / Q_0$
 - Regard Q_0 as a scale factor
 - It cleans *noisy integer* $I + \epsilon$.





High-precision CKKS Bootstrapping

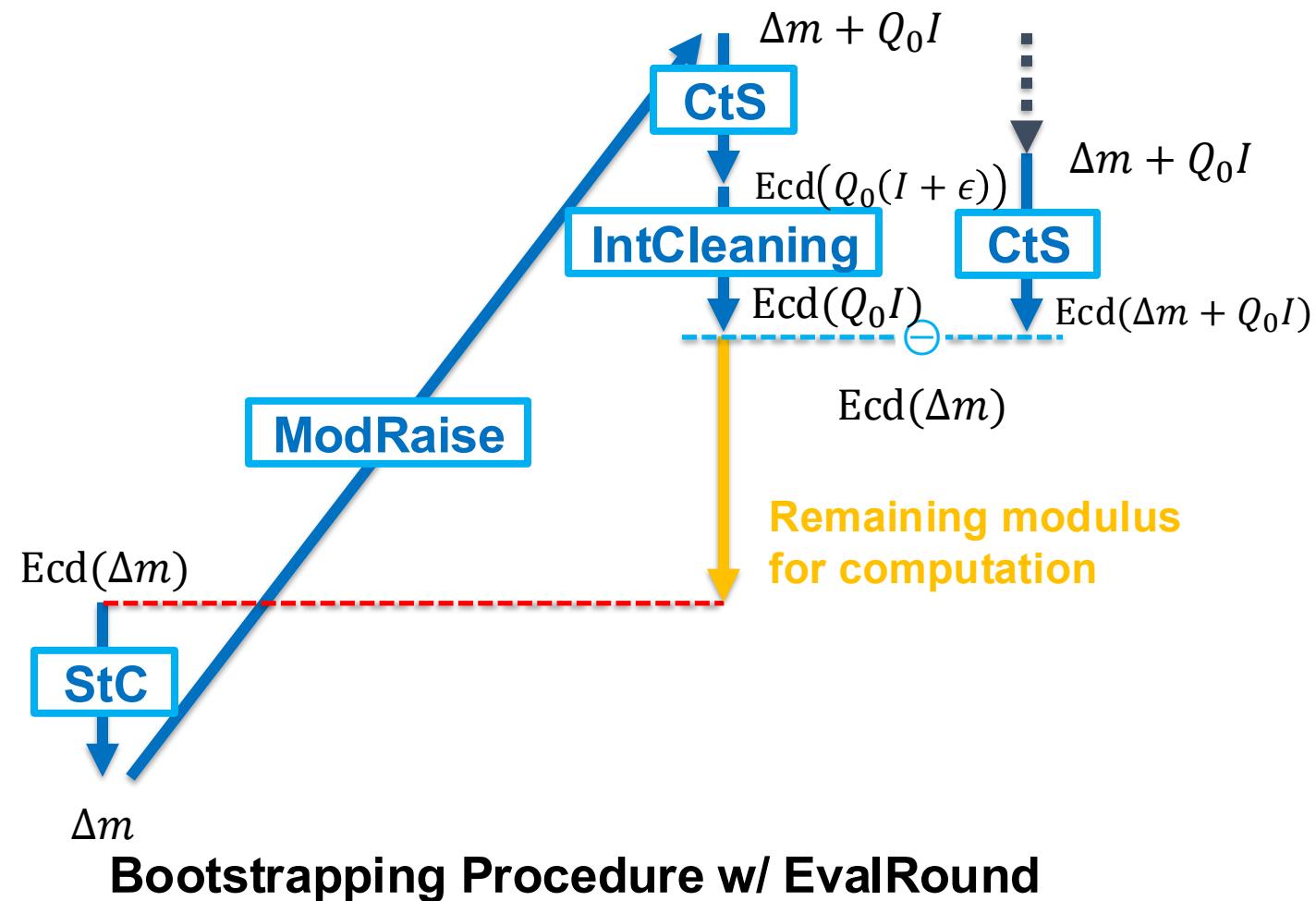
- We introduce new integer cleaning method
a.k.a. **Vectorized Cleaning**





High-precision CKKS Bootstrapping

- We introduce new integer cleaning method
a.k.a. **Vectorized Cleaning** w/
 - Low modulus consumption



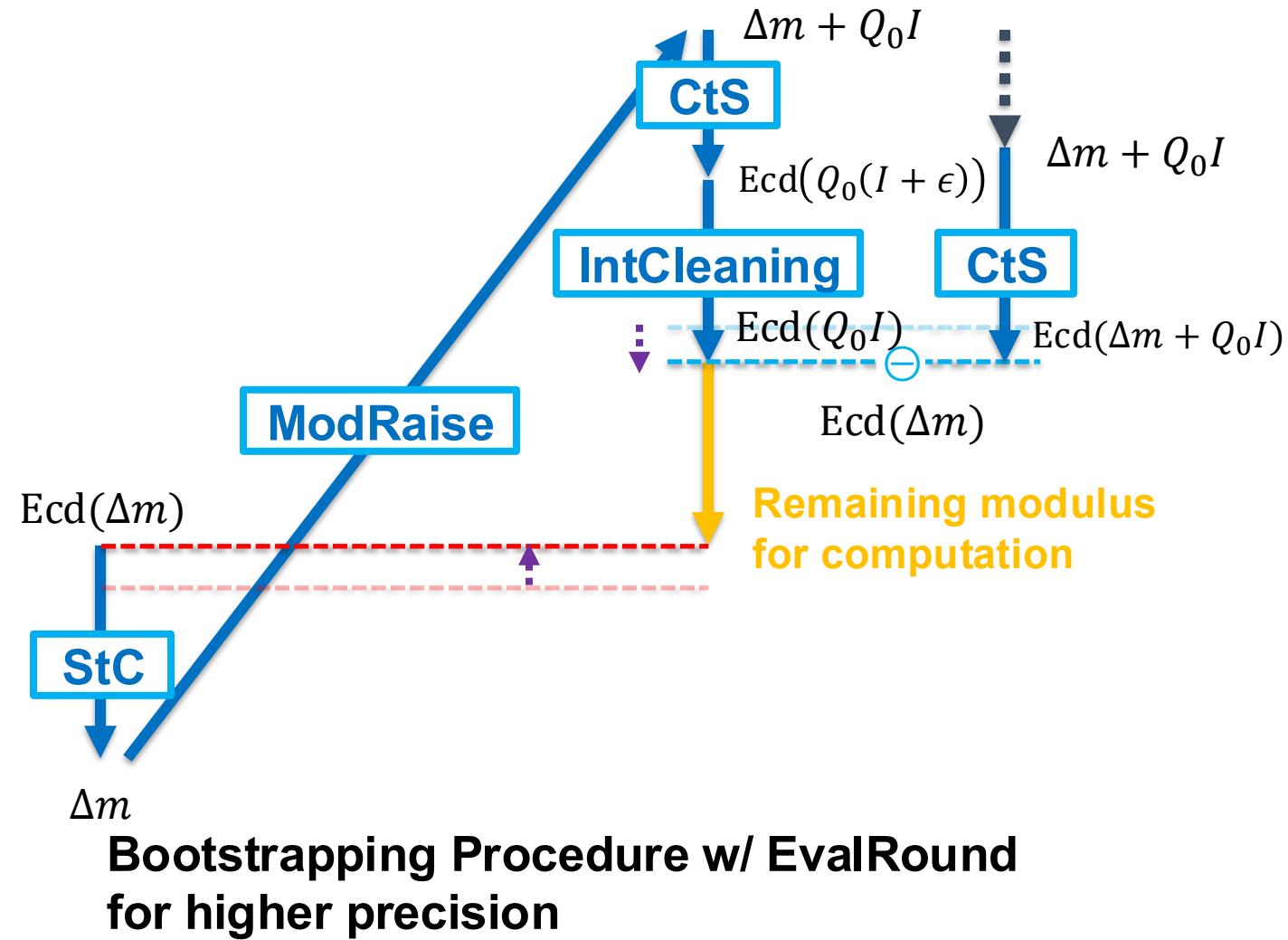


High-precision CKKS Bootstrapping

- We introduce new integer cleaning method

a.k.a. **Vectorized Cleaning** w/

- Low modulus consumption
- Low even for high precisions





Vectorized Cleaning

- For a digit- β representation $I = \sum_{\ell=0}^k I_\ell \beta^\ell$,

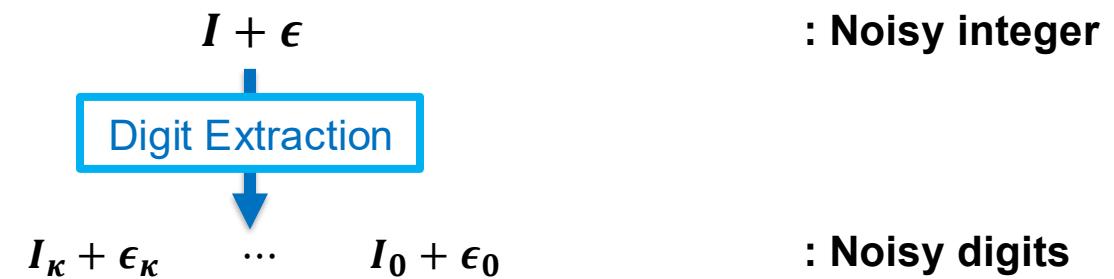


Vectorized Cleaning

- For a digit- β representation $I = \sum_{\ell=0}^{\kappa} I_\ell \beta^\ell$,

① Digit Extraction

- $I + \epsilon \mapsto \{I_\ell + \epsilon_\ell\}_{\ell=0..{\kappa}}$





Vectorized Cleaning

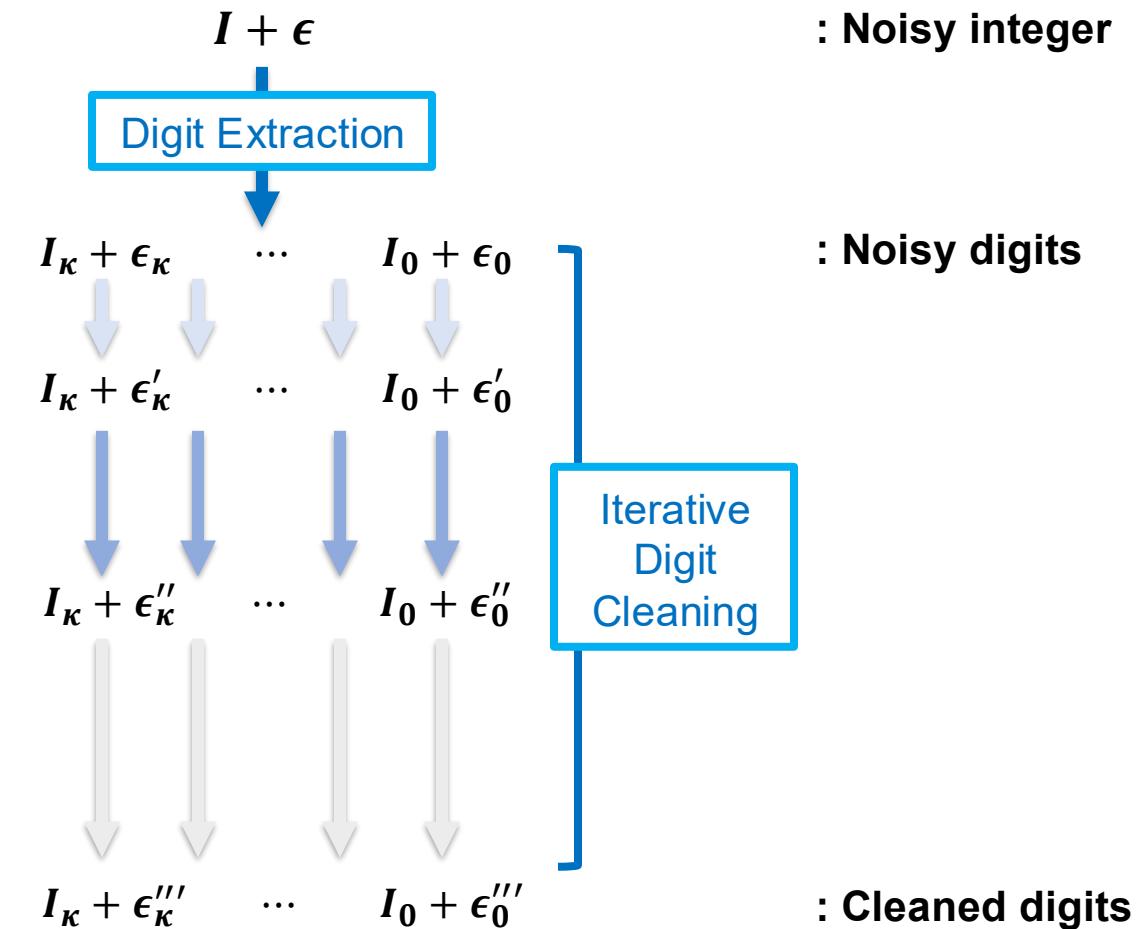
- For a digit- β representation $I = \sum_{\ell=0}^{\kappa} I_\ell \beta^\ell$,

① Digit Extraction

- $I + \epsilon \mapsto \{I_\ell + \epsilon_\ell\}_{\ell=0..K}$

② Iterative Digit Cleaning

- $I_\ell + \epsilon_\ell \mapsto I_\ell + \epsilon'_\ell \mapsto I_\ell + \epsilon''_\ell \ (\mapsto \dots)$
- $\epsilon''_\ell \ll \epsilon'_\ell \ll \epsilon_\ell$





Vectorized Cleaning

- For a digit- β representation $I = \sum_{\ell=0}^{\kappa} I_\ell \beta^\ell$,

① Digit Extraction

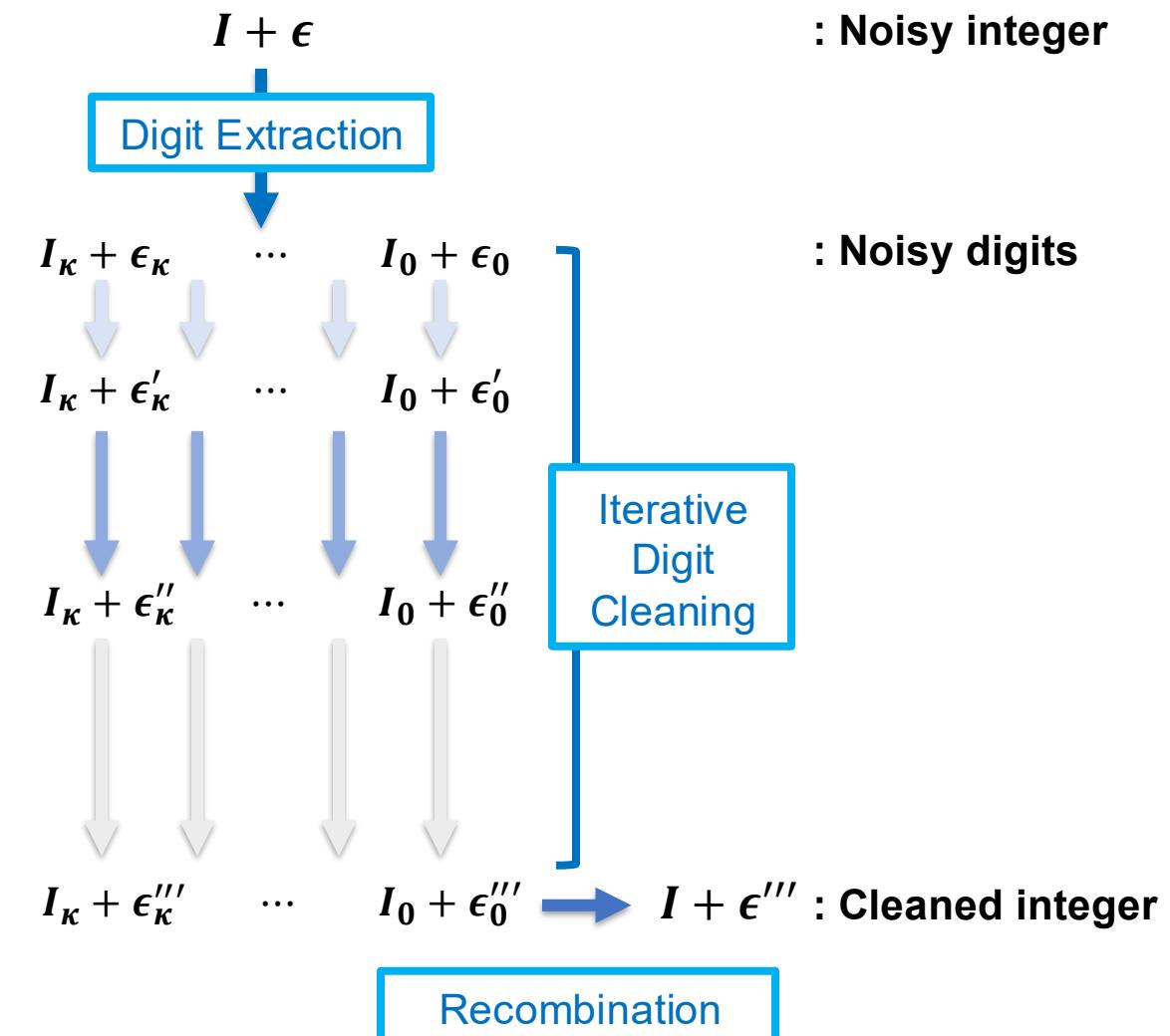
- $I + \epsilon \mapsto \{I_\ell + \epsilon_\ell\}_{\ell=0..K}$

② Iterative Digit Cleaning

- $I_\ell + \epsilon_\ell \mapsto I_\ell + \epsilon'_\ell \mapsto I_\ell + \epsilon''_\ell \ (\mapsto \dots)$
 - $\epsilon''_\ell \ll \epsilon'_\ell \ll \epsilon_\ell$

③ Recombination

- $I + \epsilon''' = \sum_{\ell=0}^{\kappa} (I_\ell + \epsilon''_\ell) \beta^\ell$
 - $\epsilon''' \ll \epsilon$

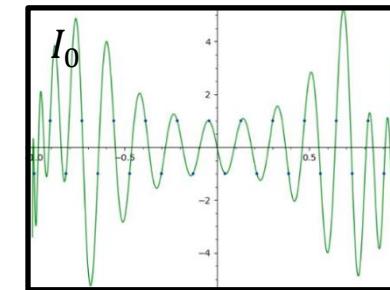
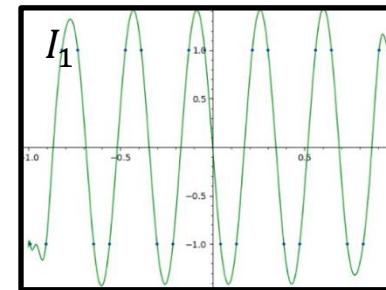
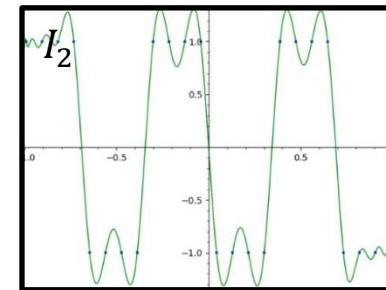
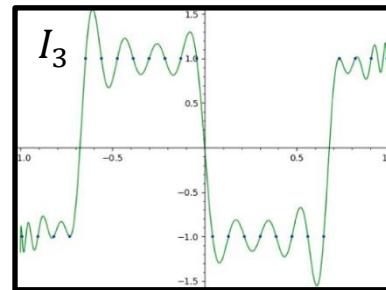
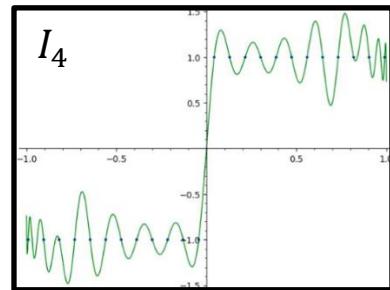
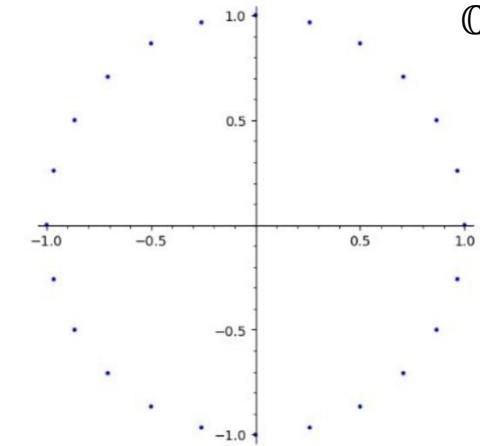




Vectorized Cleaning

Digit Extractions

- For $I \in [-K, K]$,
 - Integer \rightarrow Roots of unity \rightarrow Digits ($j^2 = -1$)
 - 1. Approximate complex exponential $I \mapsto e^{2\pi j \cdot I/(2K+1)}$
 - 2. Interpolate $e^{2\pi j \cdot I/(2K+1)} \mapsto I_\ell$ or $e^{2\pi j \cdot I_\ell/\beta}$ for all $0 \leq \ell \leq \kappa$

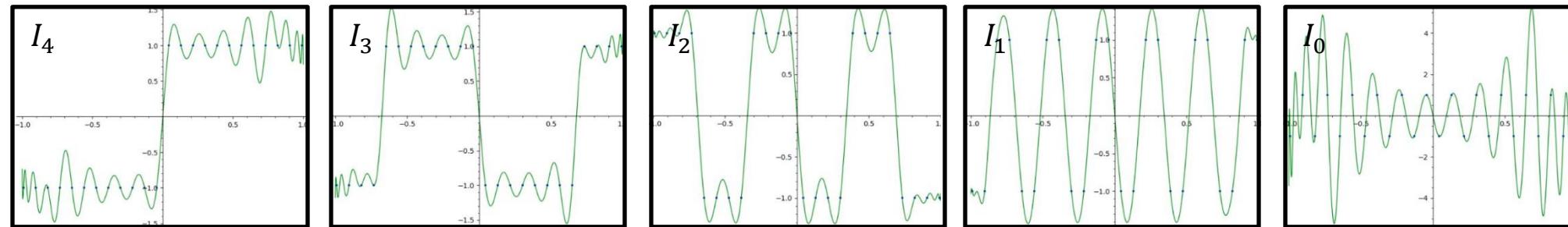
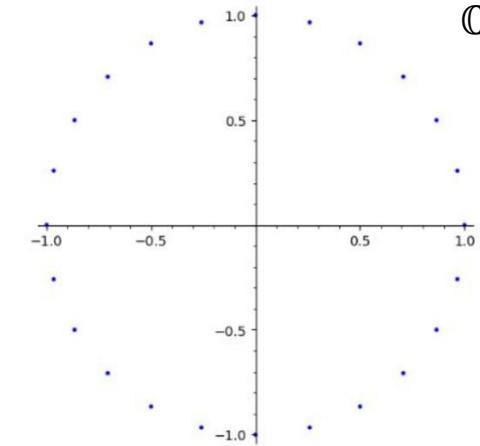




Vectorized Cleaning

Digit Extractions

- For $I \in [-K, K]$,
 - Integer \rightarrow Roots of unity \rightarrow Digits $(j^2 = -1)$
 1. Approximate complex exponential $I \mapsto e^{2\pi j \cdot I/(2K+1)}$
 2. Interpolate $e^{2\pi j \cdot I/(2K+1)} \mapsto I_\ell$ or $e^{2\pi j \cdot I_\ell/\beta}$ for all $0 \leq \ell \leq \kappa$



- Or directly interpolate $I_\kappa I_{\kappa-1} \cdots I_1 I_0(\beta) \mapsto I_\ell$ or $e^{2\pi j \cdot I_\ell/\beta}$ for all $0 \leq \ell \leq \kappa$

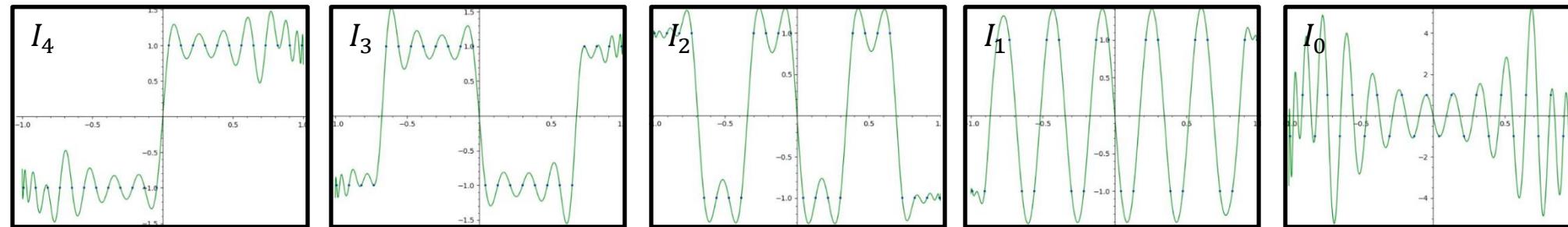
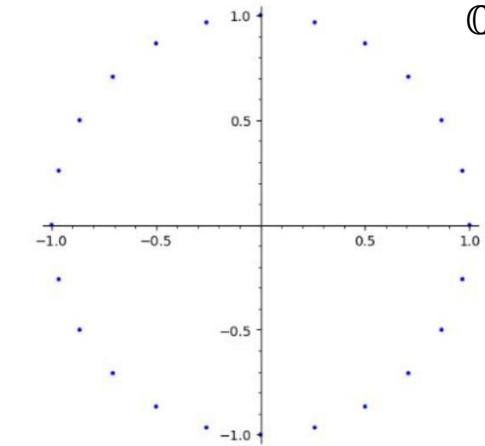


Vectorized Cleaning

Digit Extractions

- For $I \in [-K, K]$,
 - Integer \rightarrow Roots of unity \rightarrow Digits
 - 1. Approximate complex exponential $I \mapsto e^{2\pi j \cdot I/(2K+1)}$
 - 2. Interpolate $e^{2\pi j \cdot I/(2K+1)} \mapsto I_\ell$ or $e^{2\pi j \cdot I_\ell/\beta}$ for all $0 \leq \ell \leq \kappa$

Better latency



- Or directly interpolate $I_\kappa I_{\kappa-1} \cdots I_1 I_0(\beta) \mapsto I_\ell$ or $e^{2\pi j \cdot I_\ell/\beta}$ for all $0 \leq \ell \leq \kappa$

Less modulus consumption



Vectorized Cleaning

Iterative Digit Cleaning

- For each digit, we iteratively apply:
 - Cleaning functions from BKSS24⁴, CKKL24⁵ maps $b + \epsilon \mapsto b + O(\epsilon^2)$,
 - $h_1(x) = 3x^2 - 2x^3$ for $b \in \{0, 1\}$ ($\beta = 2$)
 - $f_1(x) = (\bar{x}^2 + 4x - 2x^2\bar{x})/3$ for $b \in \{e^{-2\pi j/3}, 1, e^{2\pi j/3}\}$ ($\beta = 3$)

4. Y. Bae, J. Kim, D. Stehlé, and E. Suvanto, “Bootstrapping Small Integers With CKKS,” Asiacrypt 2024.

5. H. Chung, H. Kim, Y. Kim, and Y. Lee, “Amortized Large Look-up Table Evaluation with Multivariate Polynomials for Homomorphic Encryption,” ePrint Archive 2024.



Vectorized Cleaning

Iterative Digit Cleaning

- For each digit, we iteratively apply:
 - Cleaning functions from BKSS24⁴, CKKL24⁵ maps $b + \epsilon \mapsto b + O(\epsilon^2)$,
 - $h_1(x) = 3x^2 - 2x^3$ for $b \in \{0, 1\}$ ($\beta = 2$)
 - $f_1(x) = (\bar{x}^2 + 4x - 2x^2\bar{x})/3$ for $b \in \{e^{-2\pi j/3}, 1, e^{2\pi j/3}\}$ ($\beta = 3$)
 - Start with $b + \epsilon$ w/ scale factor Δ
 - $b + O(\epsilon^2)$ w/ scale factor Δ^2
 - $b + O(\epsilon^4)$ w/ scale factor Δ^4
 - $b + O(\epsilon^8)$ w/ scale factor $\Delta^8 \dots$

4. Y. Bae, J. Kim, D. Stehlé, and E. Suvanto, “Bootstrapping Small Integers With CKKS,” Asiacrypt 2024.

5. H. Chung, H. Kim, Y. Kim, and Y. Lee, “Amortized Large Look-up Table Evaluation with Multivariate Polynomials for Homomorphic Encryption,” ePrint Archive 2024.



Vectorized Cleaning

Iterative Digit Cleaning

- For each digit, we iteratively apply:
 - Cleaning functions from BKSS24⁴, CKKL24⁵ maps $b + \epsilon \mapsto b + O(\epsilon^2)$,
 - $h_1(x) = 3x^2 - 2x^3$ for $b \in \{0, 1\}$ ($\beta = 2$)
 - $f_1(x) = (\bar{x}^2 + 4x - 2x^2\bar{x})/3$ for $b \in \{e^{-2\pi j/3}, 1, e^{2\pi j/3}\}$ ($\beta = 3$)
 - Start with $b + \epsilon$ w/ scale factor Δ
 - $b + O(\epsilon^2)$ w/ scale factor Δ^2
 - $b + O(\epsilon^4)$ w/ scale factor Δ^4
 - $b + O(\epsilon^8)$ w/ scale factor $\Delta^8 \dots$

First cleaning consumes $4 \log_2 \Delta$ bits.
It doubles for later iterations... quite a bit!

We introduce **Thrifty approach** which
consumes only $\log_2 \Delta$ bits!

4. Y. Bae, J. Kim, D. Stehlé, and E. Suvanto, "Bootstrapping Small Integers With CKKS," Asiacrypt 2024.

5. H. Chung, H. Kim, Y. Kim, and Y. Lee, "Amortized Large Look-up Table Evaluation with Multivariate Polynomials for Homomorphic Encryption," ePrint Archive 2024.



Experiments

- We implemented for $N = 2^{16}$ and failure probability $\leq 2^{-128}$.
- We used Grafting⁶ implemented in C++.

| Parameter set | # iter | Bit precision | Remaining modulus (bits) | Time (s) | Throughput (bits/s) |
|-------------------|--------|---------------|--------------------------|----------|---------------------|
| FGb ^{*7} | - | ≈ 20 | 550 | 8.5 | 64.6 |
| Ours | 2 | | 742 | 14.3 | 51.8 |

Low to High Precision CKKS Bootstrapping

6. J. H. Cheon, H. Choe, M. Kang, J. Kim, S. Kim, J. Mono, and T. Noh, "Grafting: Decoupled Scale Factors and Modulus in RNS-CKKS," ACM CCS 2025.
7. Parameter set with $N = 2^{16}$ from CryptoLab's HEaaN library. Tuned for failure probability of $\leq 2^{-128}$ and better throughput.



Experiments

- We implemented for $N = 2^{16}$ and failure probability $\leq 2^{-128}$.
- We used Grafting⁶ implemented in C++.

– $\log_2|\max \text{error}|$
from 100 iterations

Remaining
modulus / Time

| Parameter set | # iter | Bit precision | Remaining modulus (bits) | Time (s) | Throughput (bits/s) |
|-------------------|--------|---------------|--------------------------|----------|---------------------|
| FGb ^{*7} | - | | 550 | 8.5 | 64.6 |
| Ours | 2 | ≈ 20 | 742 | 14.3 | 51.8 |

Low to High Precision CKKS Bootstrapping

6. J. H. Cheon, H. Choe, M. Kang, J. Kim, S. Kim, J. Mono, and T. Noh, “Grafting: Decoupled Scale Factors and Modulus in RNS-CKKS,” ACM CCS 2025.
7. Parameter set with $N = 2^{16}$ from CryptoLab’s HEaaN library. Tuned for failure probability of $\leq 2^{-128}$ and better throughput.



Experiments

- We implemented for $N = 2^{16}$ and failure probability $\leq 2^{-128}$.
- We used Grafting⁶ implemented in C++.

– $\log_2|\max \text{error}|$
from 100 iterations

Remaining
modulus / Time

| Parameter set | # iter | Bit precision | Remaining modulus (bits) | Time (s) | Throughput (bits/s) |
|----------------------------------|--------|---------------|--------------------------|----------|---------------------|
| FGb ^{*7} | - | ≈ 20 | 550 | 8.5 | 64.6 |
| Ours | 2 | | 742 | 14.3 | 51.8 |
| 2x FGb* (Meta-BTS ⁸) | - | ≈ 40 | 530 | 17.0 | 31.1 |
| Ours | 2 | | 640 | 17.9 | 35.8 |

≈ 15% ↑

Low to High Precision CKKS Bootstrapping

6. J. H. Cheon, H. Choe, M. Kang, J. Kim, S. Kim, J. Mono, and T. Noh, "Grafting: Decoupled Scale Factors and Modulus in RNS-CKKS," ACM CCS 2025.

7. Parameter set with $N = 2^{16}$ from CryptoLab's HEaaN library. Tuned for failure probability of $\leq 2^{-128}$ and better throughput.

8. Y. Bae, J. H. Cheon, W. Cho, J. Kim, and T. Kim, "META-BTS: Bootstrapping Precision Beyond the Limit," ACM CCS 2022.



Experiments

- We implemented for $N = 2^{16}$ and failure probability $\leq 2^{-128}$.
- We used Grafting⁶ implemented in C++.

– $\log_2|\max \text{error}|$
from 100 iterations

Remaining
modulus / Time

| Parameter set | # iter | Bit precision | Remaining modulus (bits) | Time (s) | Throughput (bits/s) |
|----------------------------------|--------|---------------|--------------------------|----------|---------------------|
| FGb ^{*7} | - | ≈ 20 | 550 | 8.5 | 64.6 |
| Ours | 2 | | 742 | 14.3 | 51.8 |
| 2x FGb* (Meta-BTS ⁸) | - | ≈ 40 | 530 | 17.0 | 31.1 |
| Ours | 2 | | 640 | 17.9 | 35.8 |
| 4x FGb* (Meta-BTS ⁸) | - | ≈ 80 | 490 | 34.0 | 14.4 |
| Ours | 3 | | 494 | 20.9 | 23.7 |

≈ 15% ↑

≈ 64% ↑

Low to High Precision CKKS Bootstrapping

6. J. H. Cheon, H. Choe, M. Kang, J. Kim, S. Kim, J. Mono, and T. Noh, "Grafting: Decoupled Scale Factors and Modulus in RNS-CKKS," ACM CCS 2025.

7. Parameter set with $N = 2^{16}$ from CryptoLab's HEaaN library. Tuned for failure probability of $\leq 2^{-128}$ and better throughput.

8. Y. Bae, J. H. Cheon, W. Cho, J. Kim, and T. Kim, "META-BTS: Bootstrapping Precision Beyond the Limit," ACM CCS 2022.

Thank You!



IACR ePrint: <https://ia.cr/2025/1786>

CKKS.org Blogpost: https://ckks.org/blog/2025/high_prec_bootstrap



Bonus – Thrifty Approach

- Notation:

$$ct(x, Q) \text{ implies } \langle ct, sk \rangle = x \bmod Q, \quad ct^{(\ell)}(x, Q) \text{ implies } \langle ct^{(\ell)}, \underbrace{sk \otimes \cdots \otimes sk}_{\ell - \text{tensor}} \rangle = x \bmod Q.$$

- Evaluate $h_1(x) = 3x^2 - 2x^3$: $ct(\Delta x, Q) \Rightarrow ct(\Delta^2 h_1(x), ?)$.

- $x = b + \epsilon$ with $\epsilon = O(\Delta^{-1}) \rightarrow h_1(x) + O(\Delta^{-2}) = b + O(\epsilon^2 + \Delta^{-2}) = b + O(\epsilon^2)$

Basic (a.k.a. Black-box)

- 1) $ct_1(\Delta^2 \textcolor{red}{x}, Q) = \Delta \cdot ct$
- 2) $ct_2(\Delta^2 \textcolor{red}{x}^2 + \textcolor{red}{e}, Q/\Delta^2) = \text{Mult}(ct_1, ct_1)$
- 3) $ct_3(\Delta^2 \textcolor{red}{x}^3 + \textcolor{red}{e}', Q/\Delta^4) = \text{Mult}(ct_1, ct_2)$
- 4) $ct_{res}(\Delta^2 \textcolor{red}{h}_1(x) + \textcolor{red}{e}'', Q/\Delta^4) = 3ct_2 - 2ct_3$

- All error $\textcolor{red}{e}, \textcolor{red}{e}', \textcolor{red}{e}''$ are $O(1)$.
- Δ should be an integer.



Bonus – Thrifty Approach

- Notation:

$$ct(x, Q) \text{ implies } \langle ct, sk \rangle = x \bmod Q, \quad ct^{(\ell)}(x, Q) \text{ implies } \langle ct^{(\ell)}, \underbrace{sk \otimes \cdots \otimes sk}_{\ell - \text{tensor}} \rangle = x \bmod Q.$$

- Evaluate $h_1(x) = 3x^2 - 2x^3$: $ct(\Delta x, Q) \Rightarrow ct(\Delta^2 h_1(x), ?)$.

- $x = b + \epsilon$ with $\epsilon = O(\Delta^{-1}) \rightarrow h_1(x) + O(\Delta^{-2}) = b + O(\epsilon^2 + \Delta^{-2}) = b + O(\epsilon^2)$

Basic (a.k.a. Black-box)

- 1) $ct_1(\Delta^2 \mathbf{x}, Q) = \Delta \cdot ct$
- 2) $ct_2(\Delta^2 \mathbf{x}^2 + e, Q/\Delta^2) = \text{Mult}(ct_1, ct_1)$
- 3) $ct_3(\Delta^2 \mathbf{x}^3 + e', Q/\Delta^4) = \text{Mult}(ct_1, ct_2)$
- 4) $ct_{res}(\Delta^2 h_1(x) + e'', Q/\Delta^4) = 3ct_2 - 2ct_3$

- All error e, e', e'' are $O(1)$.
- Δ should be an integer.

Inverse Rescale Approach

- 1) $ct_1(\Delta^2 x, \Delta Q) = \Delta \cdot ct$
- 2) $ct_2(\Delta^2 x^2 + e, Q/\Delta) = \text{Mult}(ct_1, ct_1)$
- 3) $ct_3(\Delta^2 x^3 + e', Q/\Delta^3) = \text{Mult}(ct_1, ct_2)$
- 4) $ct_{res}(\Delta^2 h_1(x) + e'', Q/\Delta^3) = 3ct_2 - 2ct_3$

$\boxed{\langle ct, sk \rangle = \Delta x \bmod Q}$
 $\rightarrow \langle \Delta \cdot ct, sk \rangle = \Delta^2 x \bmod \Delta Q$



Bonus – Thrifty Approach

- Notation:

$$ct(x, Q) \text{ implies } \langle ct, sk \rangle = x \bmod Q, \quad ct^{(\ell)}(x, Q) \text{ implies } \langle ct^{(\ell)}, \underbrace{sk \otimes \cdots \otimes sk}_{\ell - \text{tensor}} \rangle = x \bmod Q.$$

- Evaluate $h_1(x) = 3x^2 - 2x^3$: $ct(\Delta x, Q) \Rightarrow ct(\Delta^2 h_1(x), ?)$.

- $x = b + \epsilon$ with $\epsilon = O(\Delta^{-1}) \rightarrow h_1(x) + O(\Delta^{-2}) = b + O(\epsilon^2 + \Delta^{-2}) = b + O(\epsilon^2)$

Basic (a.k.a. Black-box)

- 1) $ct_1(\Delta^2 \mathbf{x}, Q) = \Delta \cdot ct$
- 2) $ct_2(\Delta^2 \mathbf{x}^2 + e, Q/\Delta^2) = \text{Mult}(ct_1, ct_1)$
- 3) $ct_3(\Delta^2 \mathbf{x}^3 + e', Q/\Delta^4) = \text{Mult}(ct_1, ct_2)$
- 4) $ct_{res}(\Delta^2 h_1(x) + e'', Q/\Delta^4) = 3ct_2 - 2ct_3$

- All error e, e', e'' are $O(1)$.
- Δ should be an integer.

Inverse Rescale Approach

- 1) $ct_1(\Delta^2 x, \Delta Q) = \Delta \cdot ct$
- 2) $ct_2(\Delta^2 x^2 + e, Q/\Delta) = \text{Mult}(ct_1, ct_1)$
- 3) $ct_3(\Delta^2 x^3 + e', Q/\Delta^3) = \text{Mult}(ct_1, ct_2)$
- 4) $ct_{res}(\Delta^2 h_1(x) + e'', Q/\Delta^3) = 3ct_2 - 2ct_3$

$\langle ct, sk \rangle = \Delta x \bmod Q$
 $\rightarrow \langle \Delta \cdot ct, sk \rangle = \Delta^2 x \bmod \Delta Q$

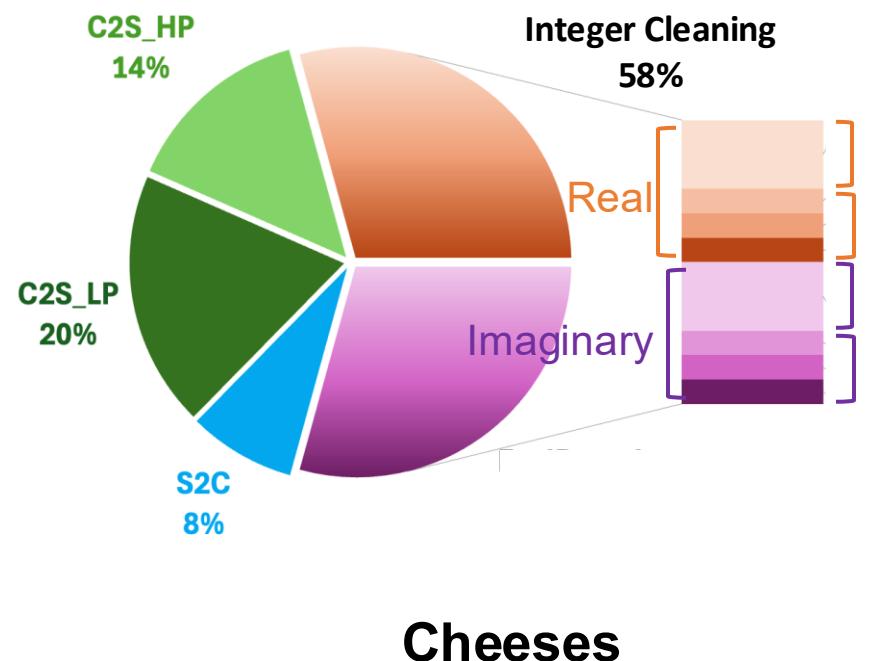
Thrifty Approach

- 1) $ct_1^{(2)}(\Delta^2 x^2, Q) = ct \otimes ct$
- 2) $ct_2^{(2)}(\Delta^3 x^2, Q) = \Delta \cdot ct_1^{(2)}$
- 3) $ct_3^{(3)}(\Delta^3 x^3, Q) = ct \otimes ct \otimes ct$
- 4) $ct_4^{(3)}(\Delta^3 h_1(x), Q) = 3ct_2^{(2)} - 2ct_3^{(3)}$
- 5) $ct_{res}(\Delta^2 h_1(x) + e, Q/\Delta)$
 $= \text{Rescale}(\text{Relin}(ct_4^{(3)}))_{12}$



Bonus – Cheese et Bar

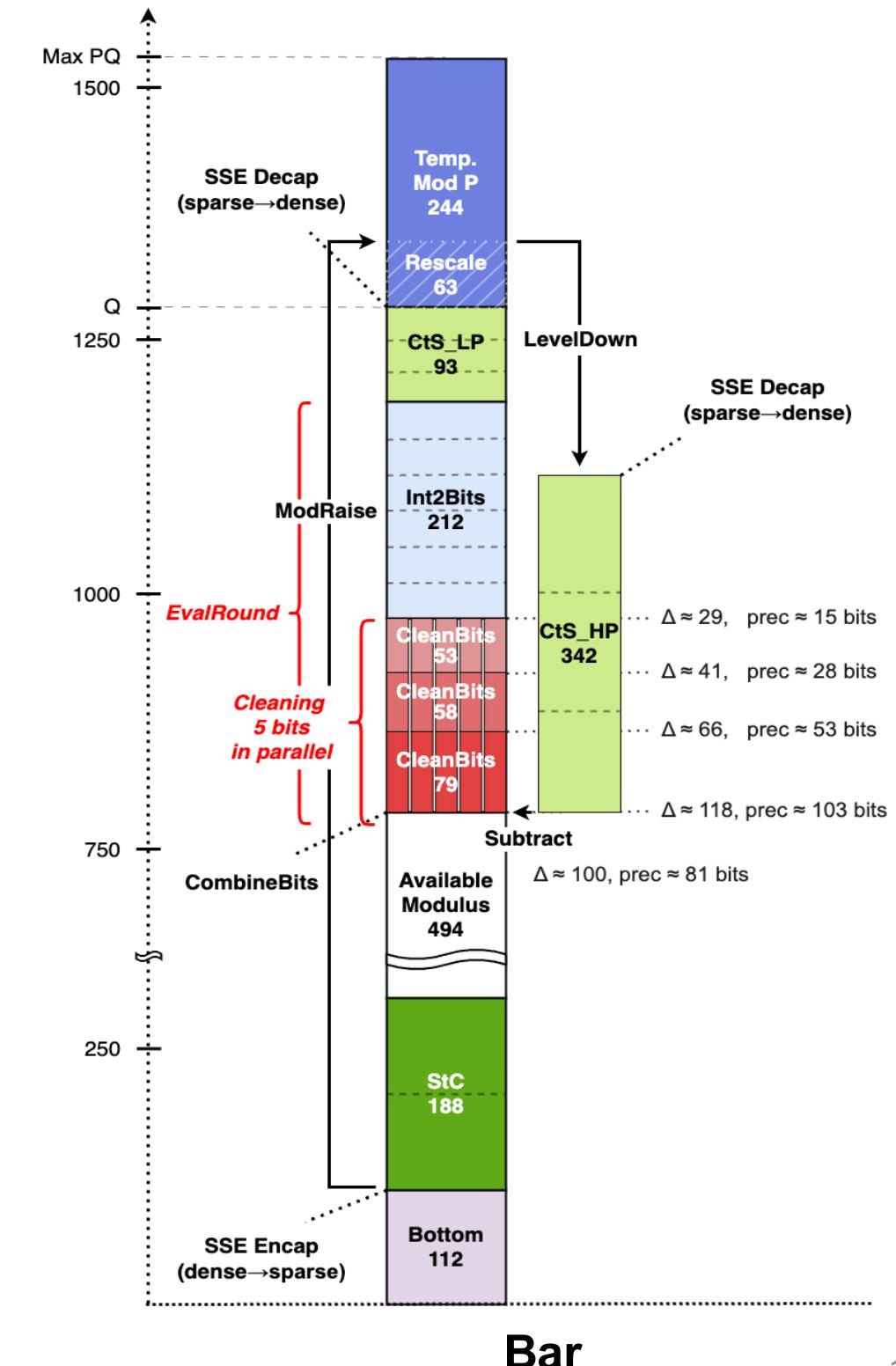
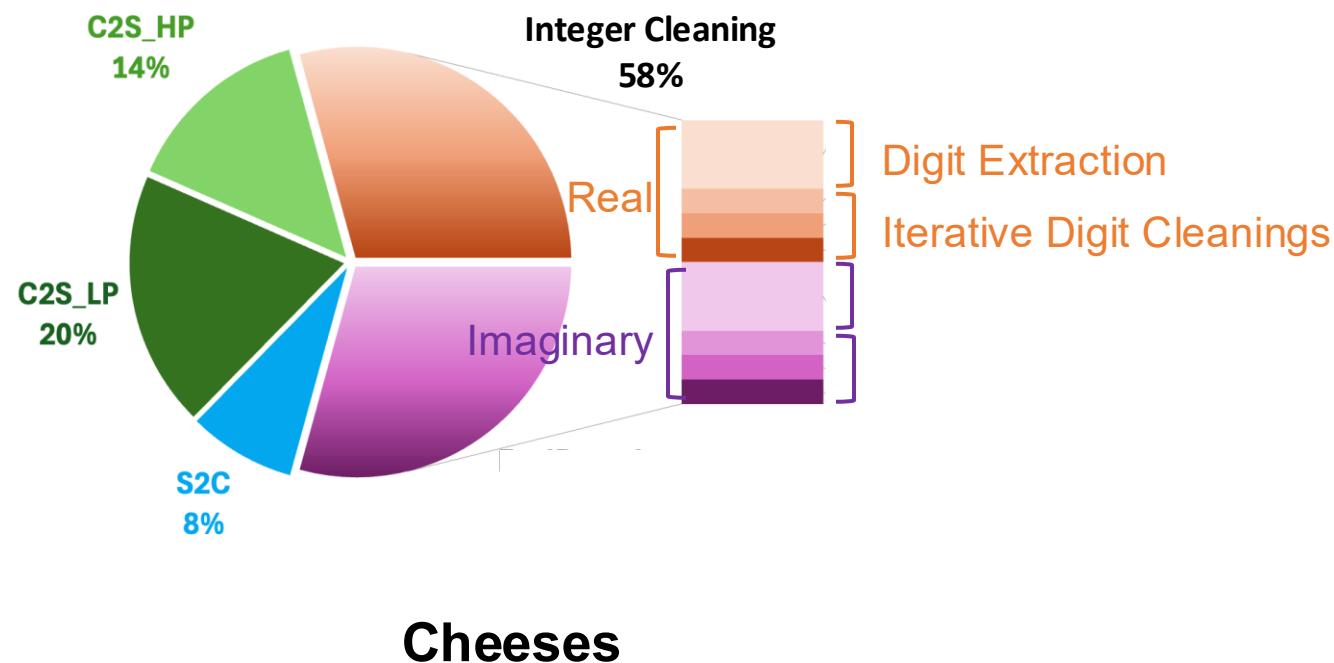
- We implemented for $N = 2^{16}$ and fail prob. $\leq 2^{-128}$.
- We used Grafting⁶ implemented in C++.





Bonus – Cheese et Bar

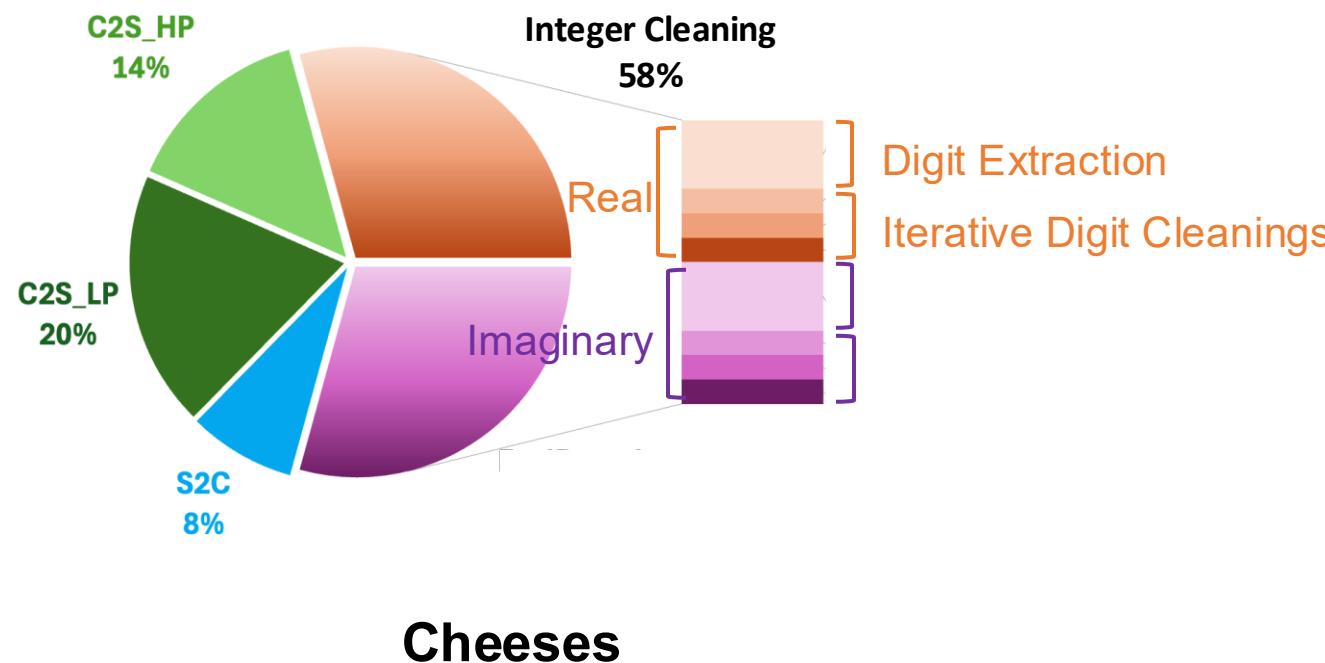
- We implemented for $N = 2^{16}$ and fail prob. $\leq 2^{-128}$.
- We used Grafting⁶ implemented in C++.





Bonus – Cheese et Bar

- We implemented for $N = 2^{16}$ and fail prob. $\leq 2^{-128}$.
- We used Grafting⁶ implemented in C++.



Scale factors of $\log_2 \Delta = 29 \sim 35$.
So, Grafting helps a lot !
Learn more in the next talk ☺

