

HEaaN2

A High-Performance CKKS Library

Presented by Seonghak Kim on the behalf of Crypto Lab. Inc.

Core Dev Team : S. Jeon, M. Kim, S. Kim, J. Lee, J. Lee, T. Noh, Erkhes N.

Former Members : J. Ju, S. Jung, S. Kim, J. Mono, E. Suvanto

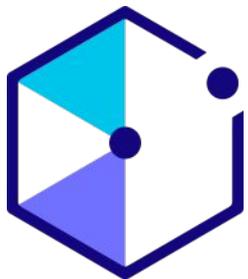


TABLE
OF
CONTENTS

Overview of HEaaN2

CKKS Engineering : Performance vs Abstraction

Design Challenges

- Evict Centralized Parameters
- Redefine Levels
- Allow vast design space

Conclusion

Overview of HEaaN2



What is HEaaN2

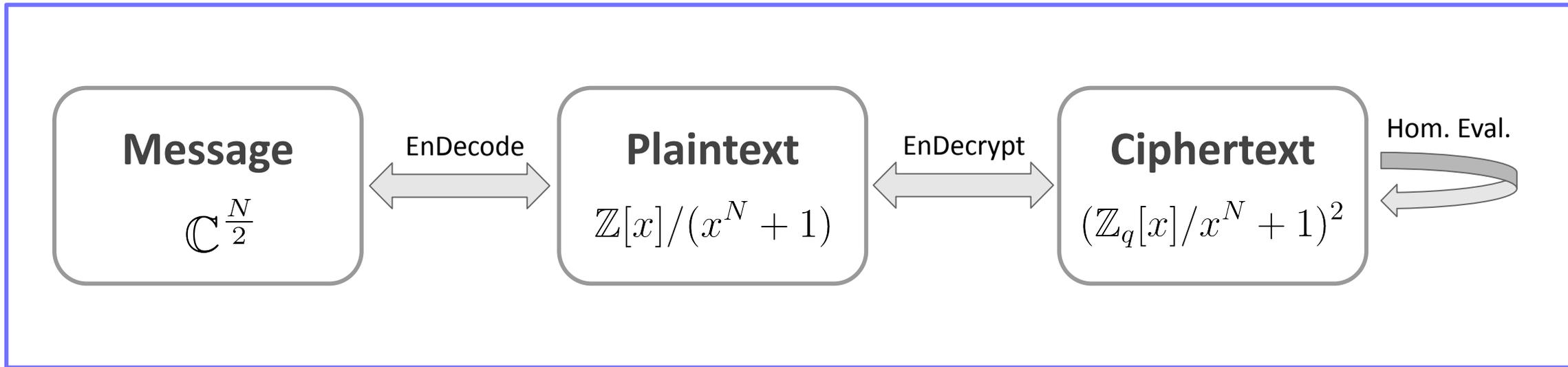
Overview of HEaaN2

The graphic features a dark blue background with glowing digital lines and data points. At the top center is the HEaaN2 logo, a hexagon with a keyhole and a key inside. Below it, the text reads "HEaaN² High-Performance CKKS Homomorphic Encryption Library" and "State-of-the-Art Performance for Encrypted Numerical Computation". A central glowing cylinder labeled "CKKS" is shown with a beam of light passing through it, flanked by two boxes containing the mathematical expressions $[x_1, x_2, \dots]$ and $[x'_1, x'_2, \dots]$. At the bottom, there are icons for a GPU, a fan, a circuit board, a globe, and a bar chart, with the text "GPU acceleration · SIMD packing · optimized modulus chain" below them.

- *CKKS only.*
- *Closed-source.*
- *SOTA performance.*
- *Available at : heaan.io*

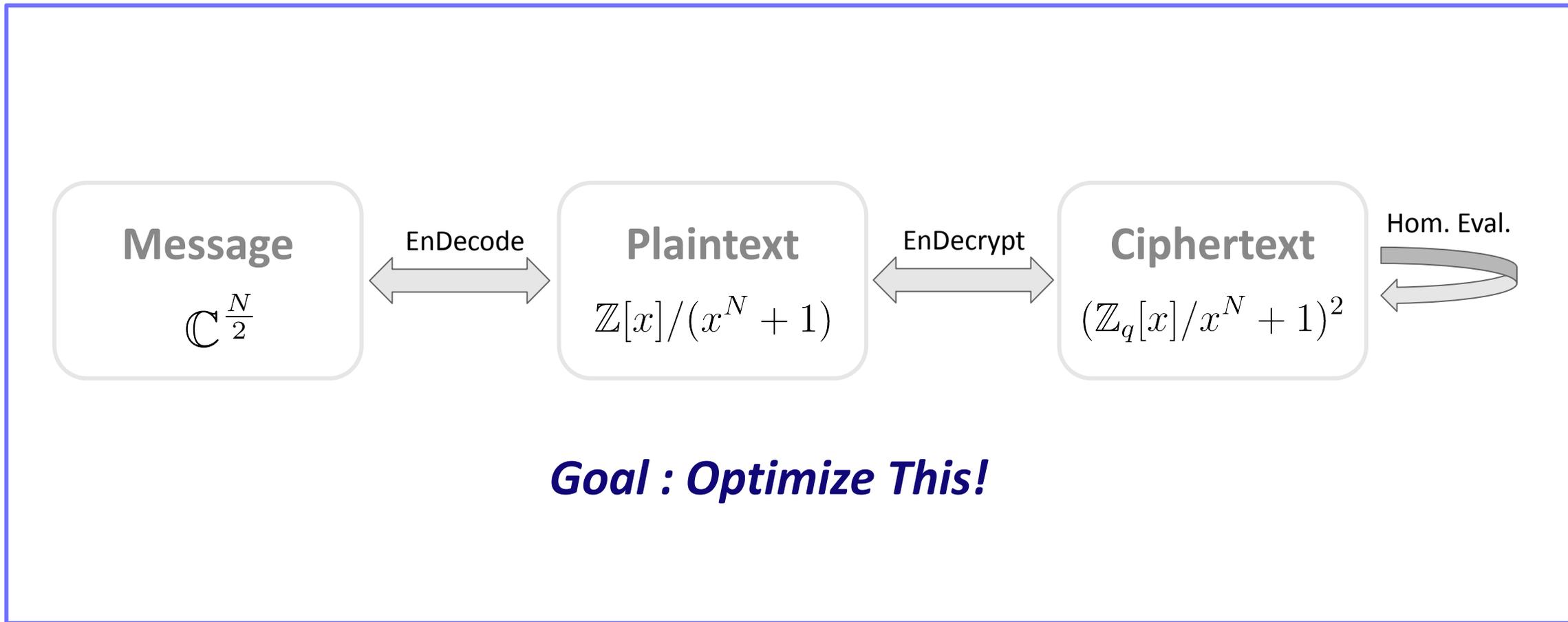


Basic CKKS



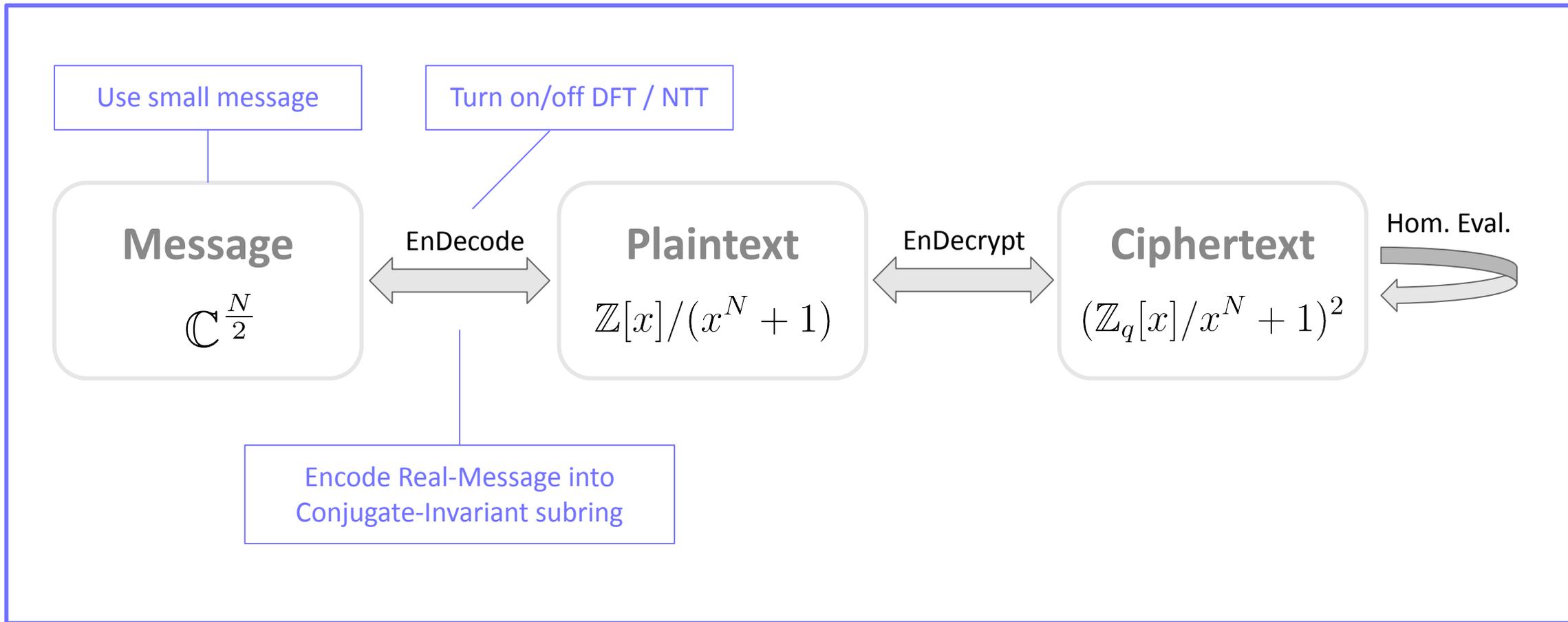


Extended CKKS for High Performance



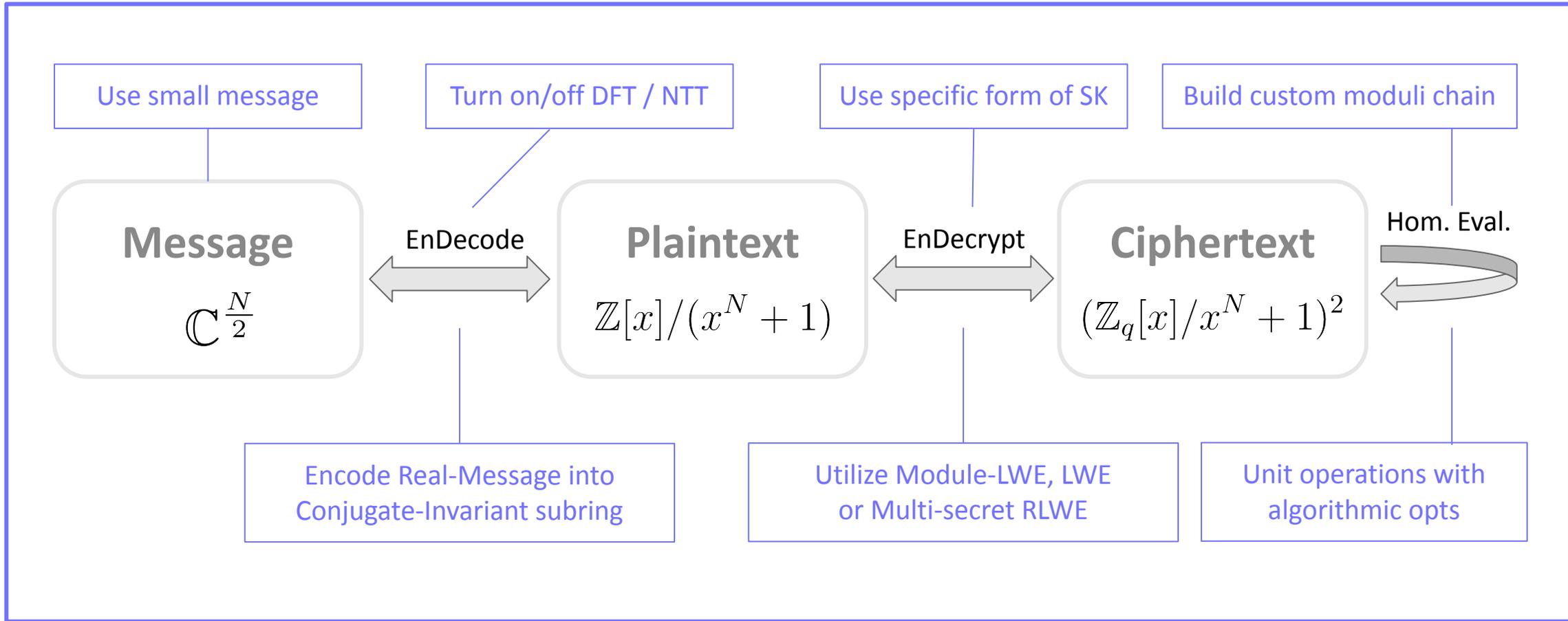


Extended CKKS for High Performance



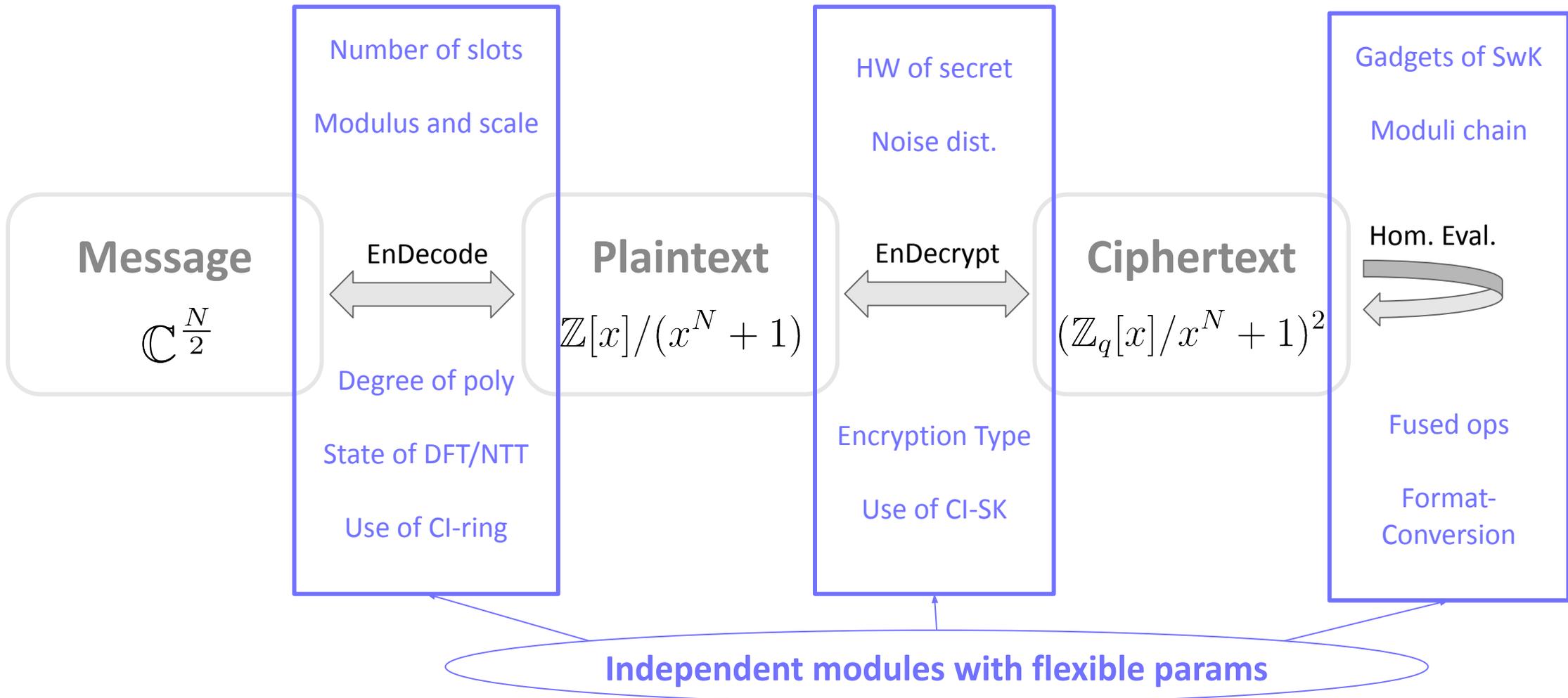


Extended CKKS for High Performance





Overview of HEaaN2





Performance of HEaaN2

Overview of HEaaN2

CPU, 16 threads	OpenFHE	HEaaN2	Ratio
Addition	3.85 ms	0.09 ms	42.8x
Multiplication	121.4 ms	10.69 ms	11.4x
Bootstrap	11,341 ms	665 ms	17.1x
Levels gained by BTS	4	15	3.75x

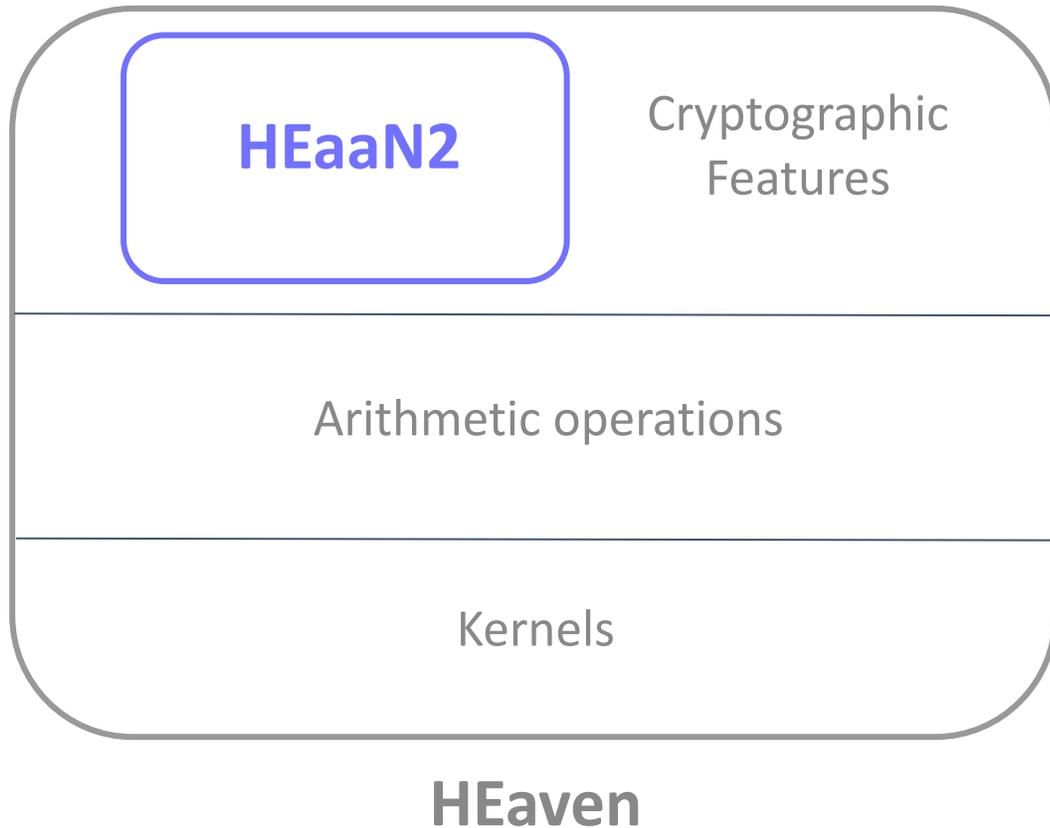
GPU(RTX 5090)	Cheddar	HEaaN2 v0.1.1	Theodosian	HEaaN2 v0.2.0
Bootstrap	23.18 ms	14.17 ms	12.75 ms	9.62 ms

GPU(A100)	NVIDIA Cerium	HEaaN2 v0.1.1	HEaaN2 v0.2.0
Bootstrap	34.2 ms	29.1 ms	20.9 ms

OpenFHE : v1.4.2
 CPU : INTEL XEON GOLD 6544Y @3.6GHz
 All parameters are logN=16, logPQ < 1730, prec > 18 bit,
 Details adjusted to be disavantageous on HEaaN2.



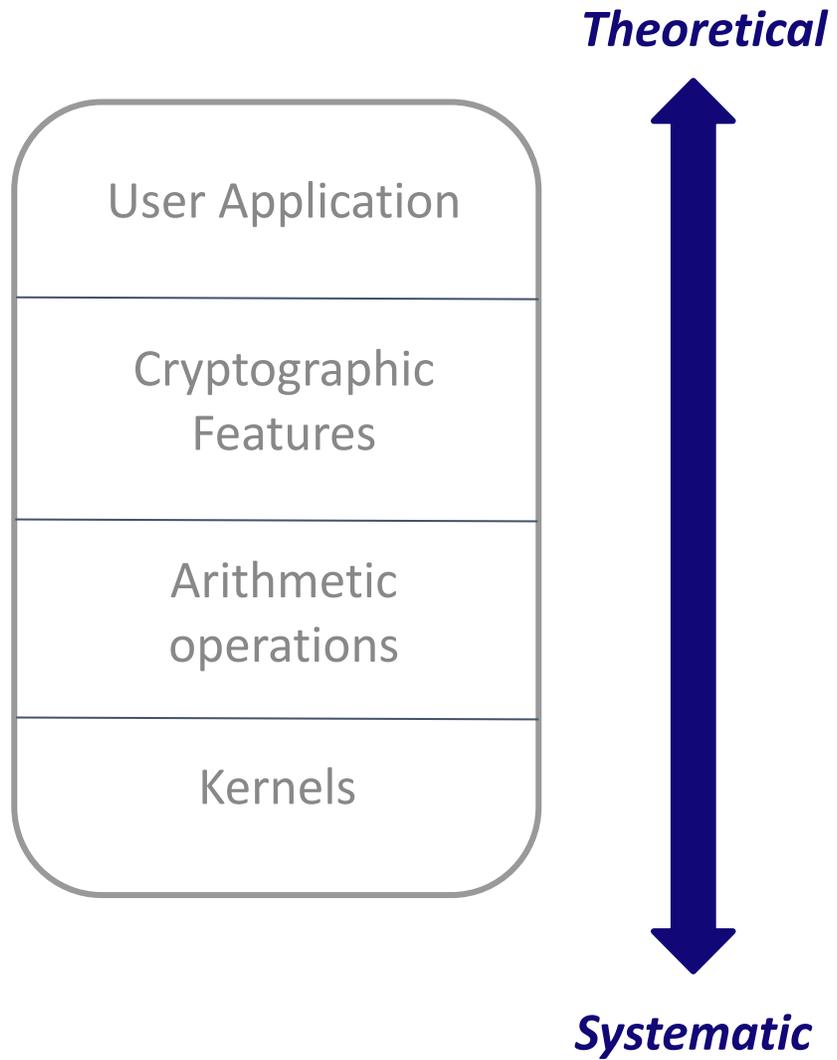
Overview of HEaaN2



- In this version, we have :
 - Grafting
 - Custom parameters
 - Flexible moduli chain & scale
 - Conjugate-Invariant subring
 - Batching
- In future, we may release :
 - LWE, MLWE, Ring packing
 - Multi-secret RLWE
 - High-precision FHE
 - Bootstrapping on discrete data
 - 32bit-word support
 - **More to come!**

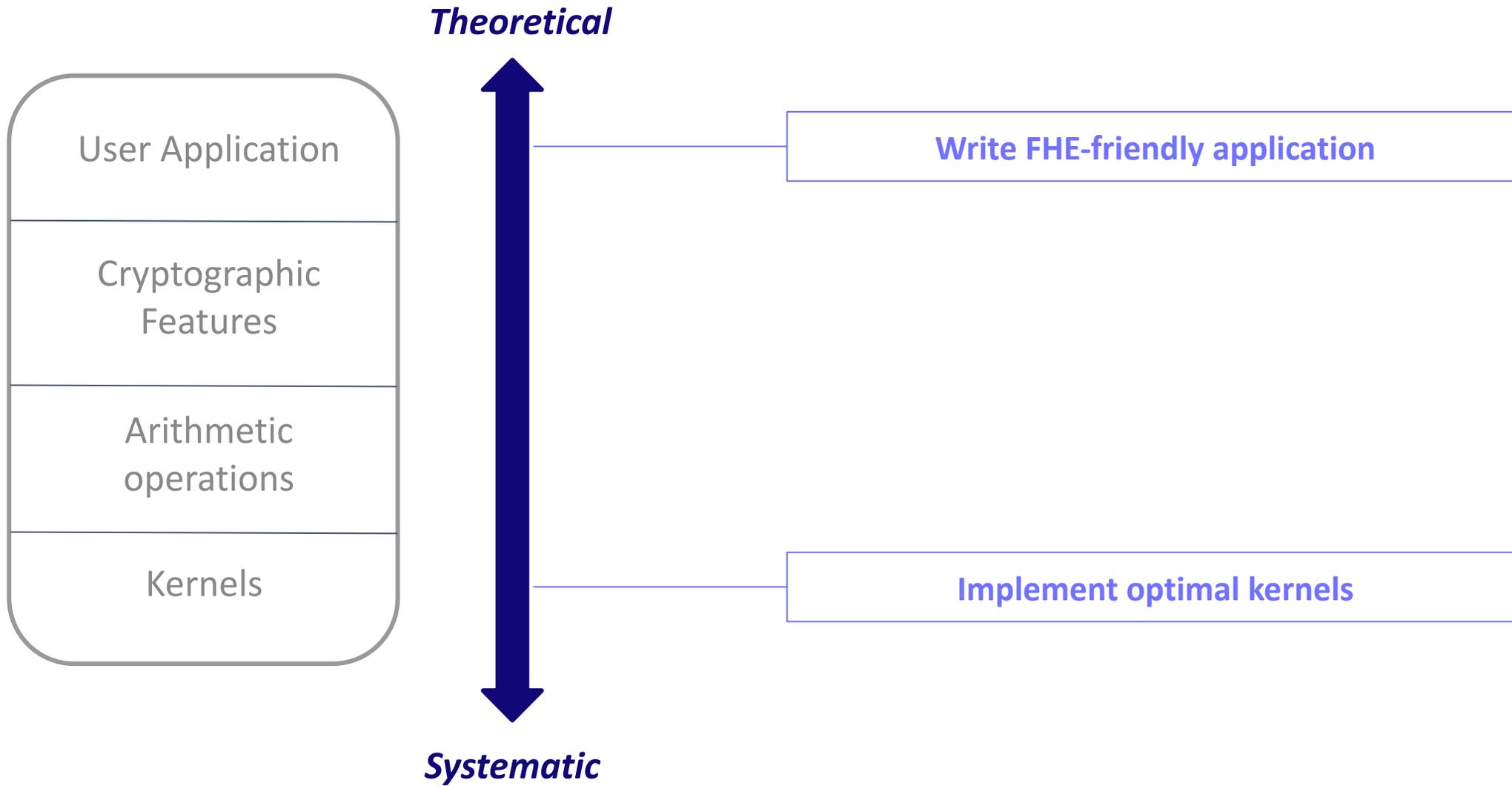
CKKS Engineering :

Performance vs Abstraction



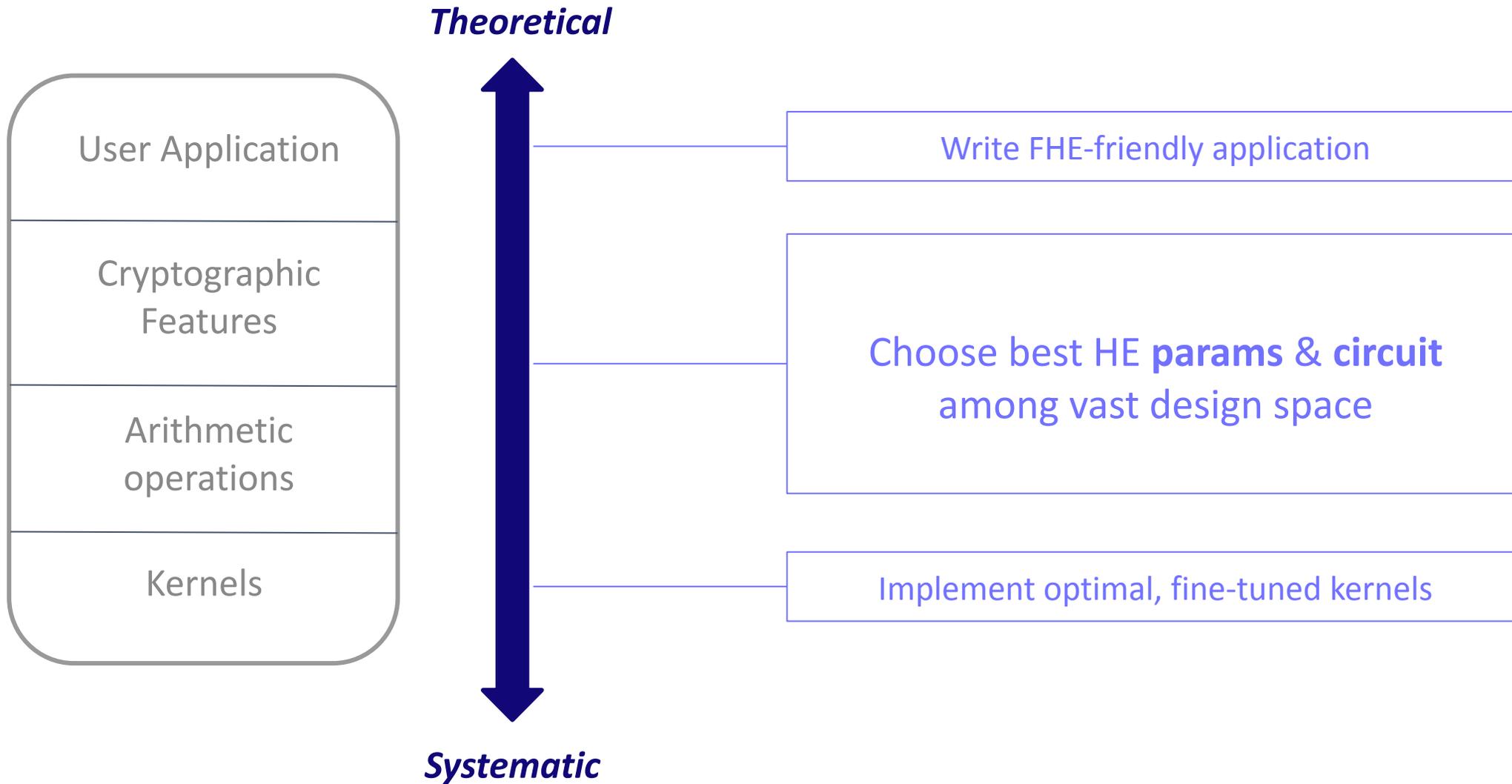


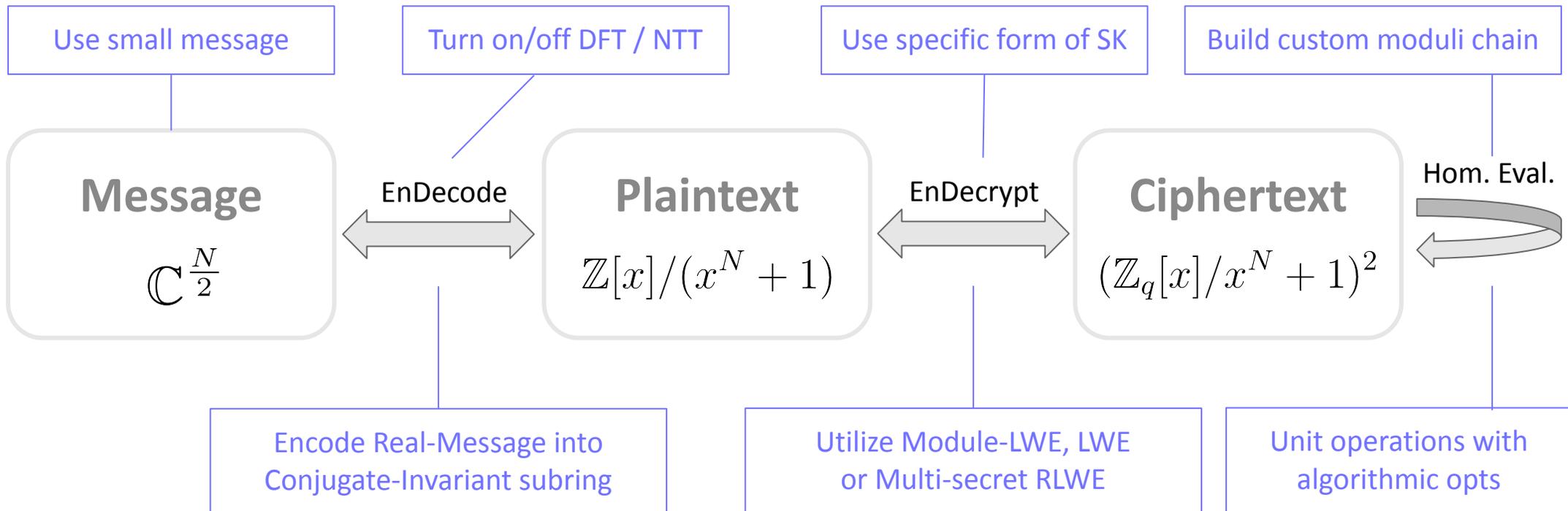
CKKS Engineering : Performance vs Abstraction

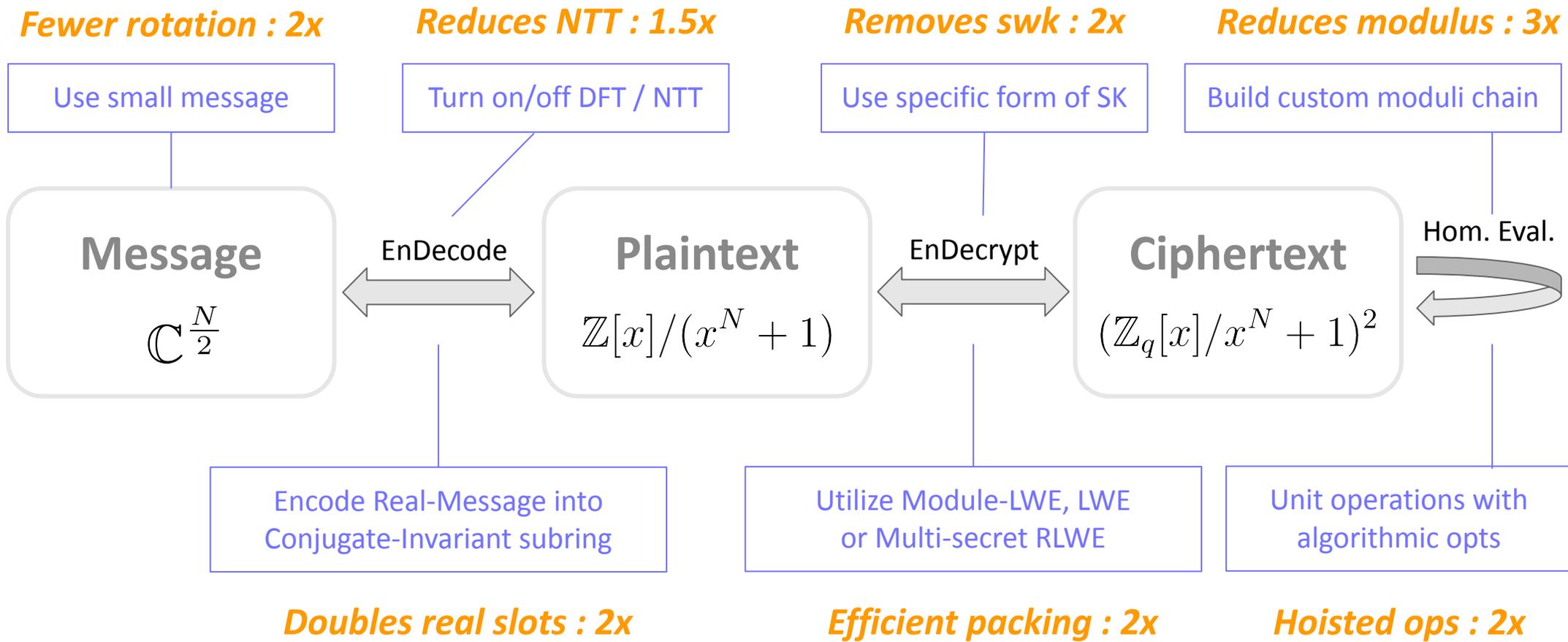




Performance vs Abstraction



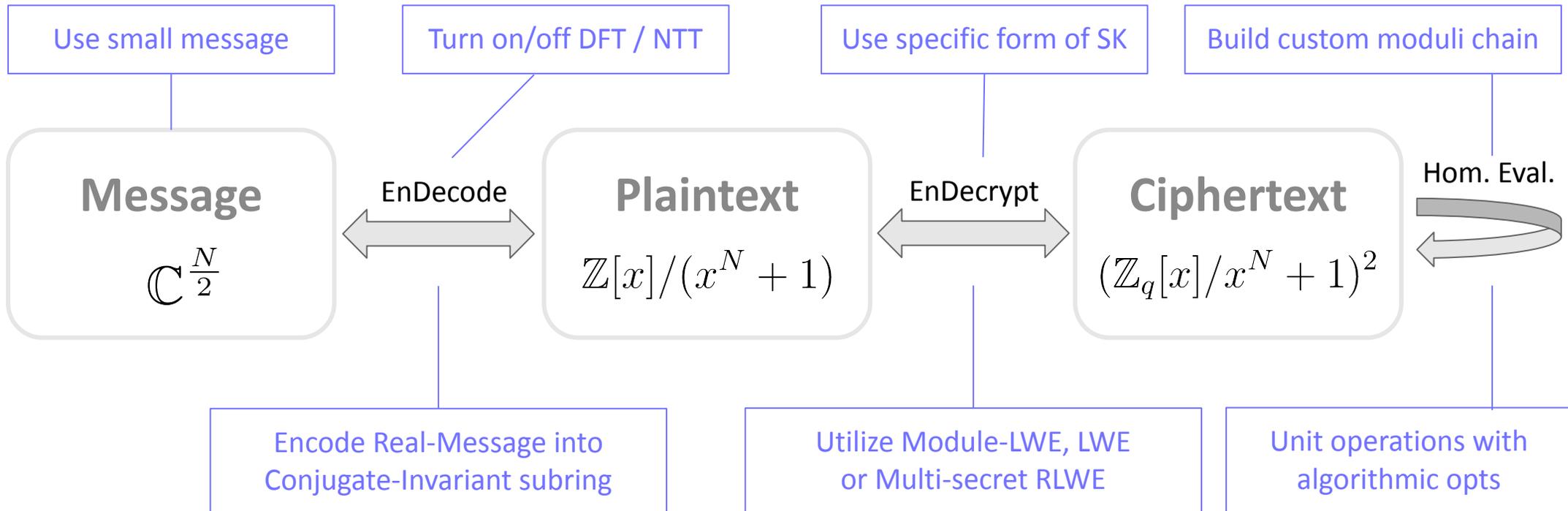




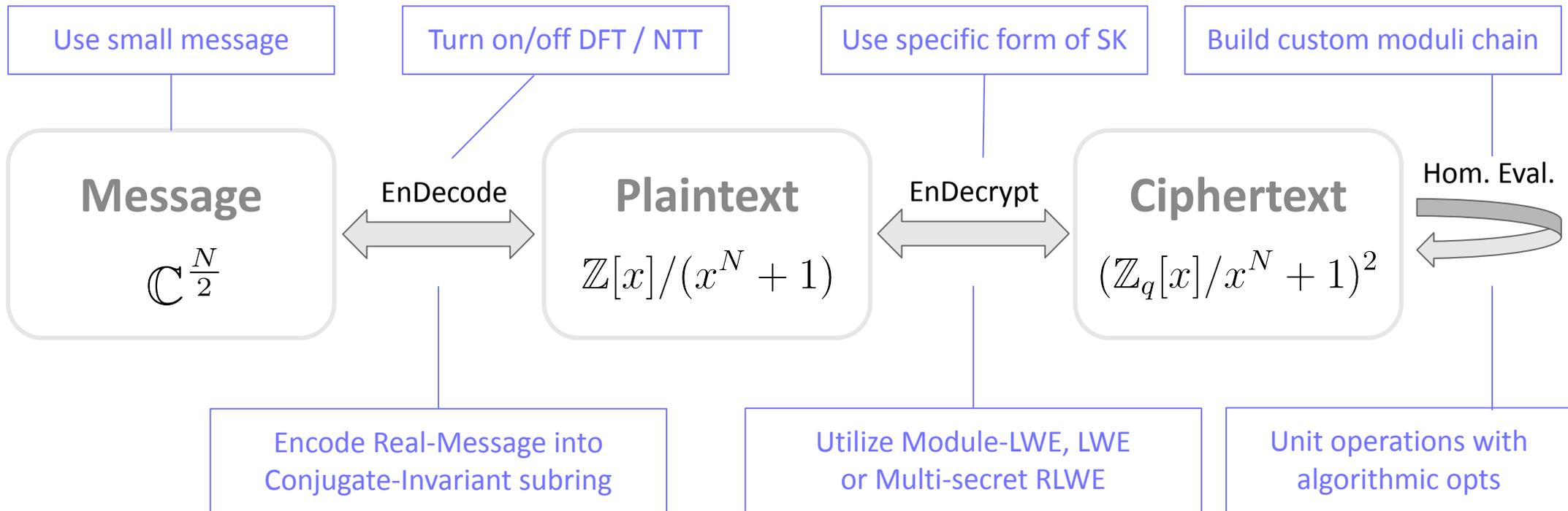


**Breaks assumption
on encoding**

**No centralized
Secret / Moduli chain**



**Breaks assumption
on format of Ctxts**

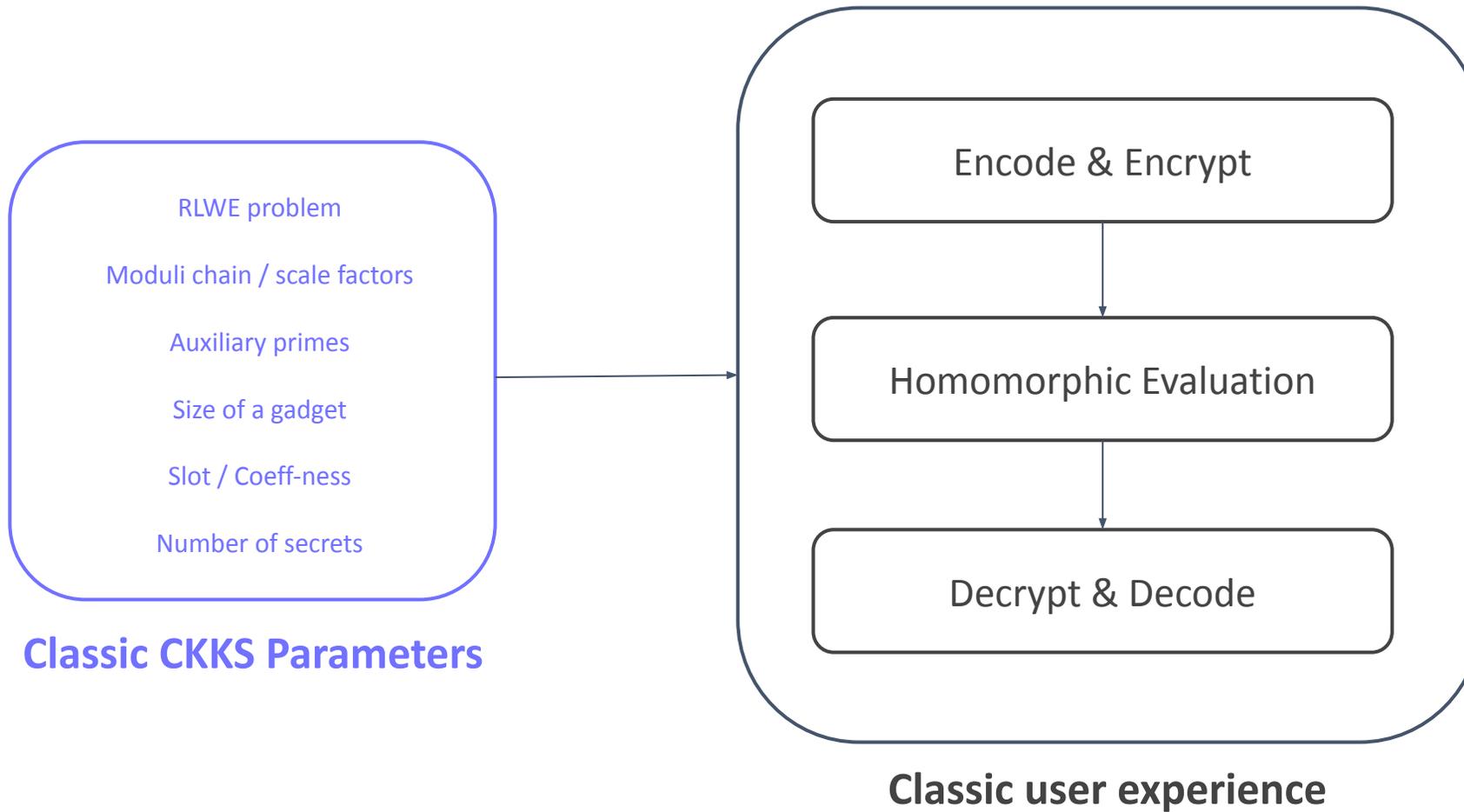


Choose best design between Performance and Abstraction!

Design Challenges

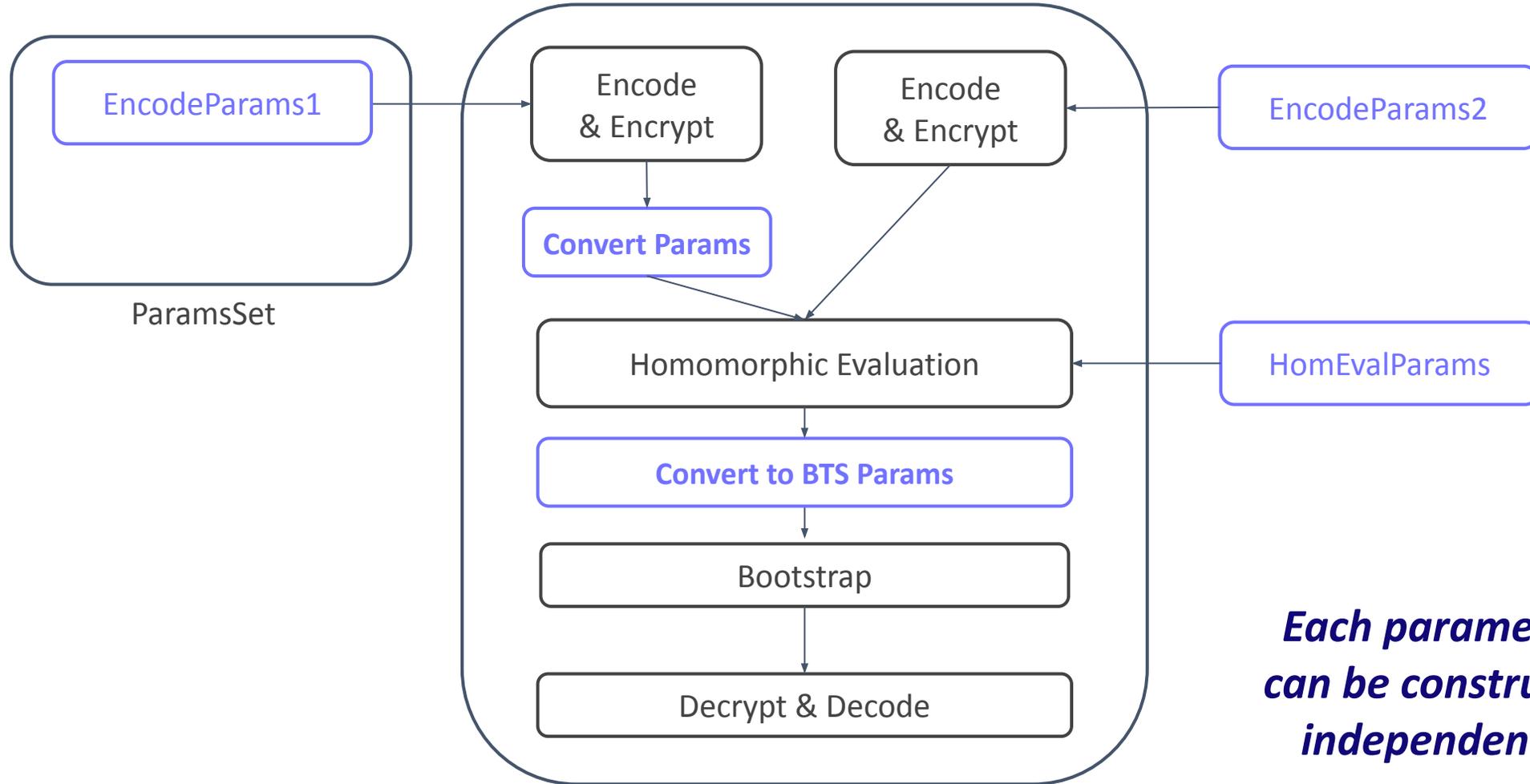


Design Challenges 1 : Evict Centralized Parameters





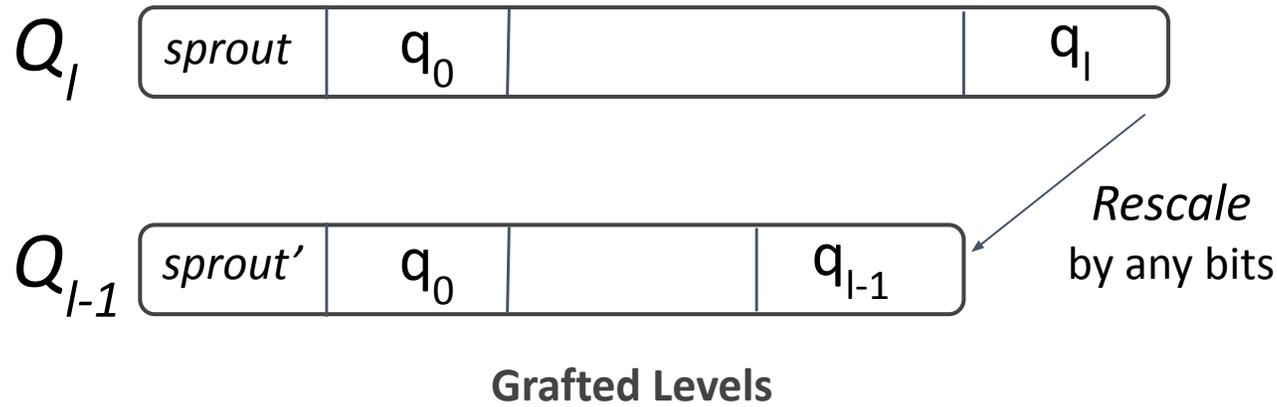
Design Challenges 1 : Evict Centralized Parameters



Each parameters can be constructed independently.



Design Challenges 2 : Redefine Levels



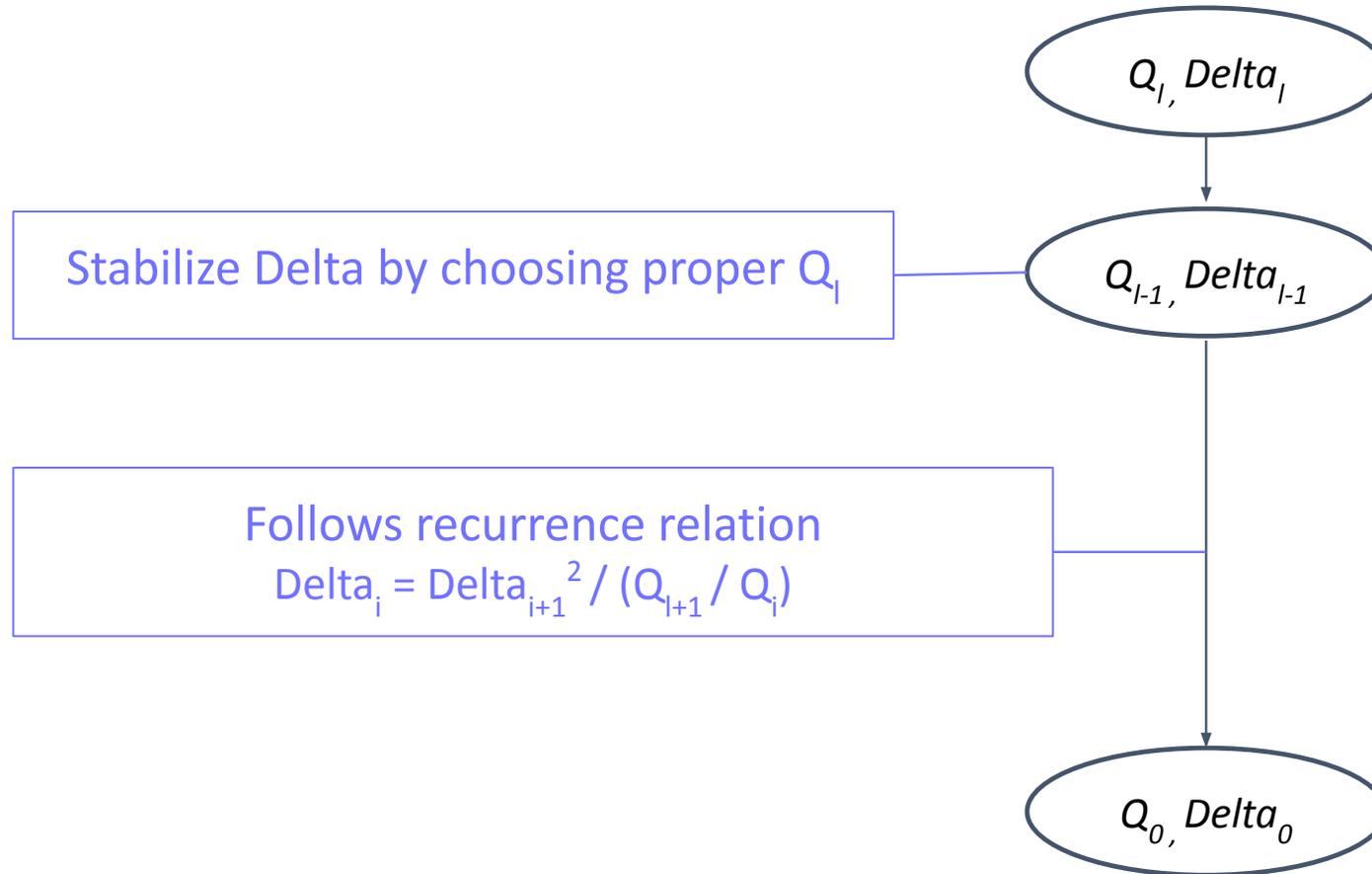
Hidden Limitation of Classic Levels

- Moduli chain is constructed in a single sequence.
- Moduli in the chain divides the next one.
- RNS system is truncated in a fixed order.
- One rescaling removes 1 word.
- Rescaling amount is NTT-able.

*[CCS25] Grafting: Decoupled Scale Factors and Modulus in RNS-CKKS, Cheon et al.



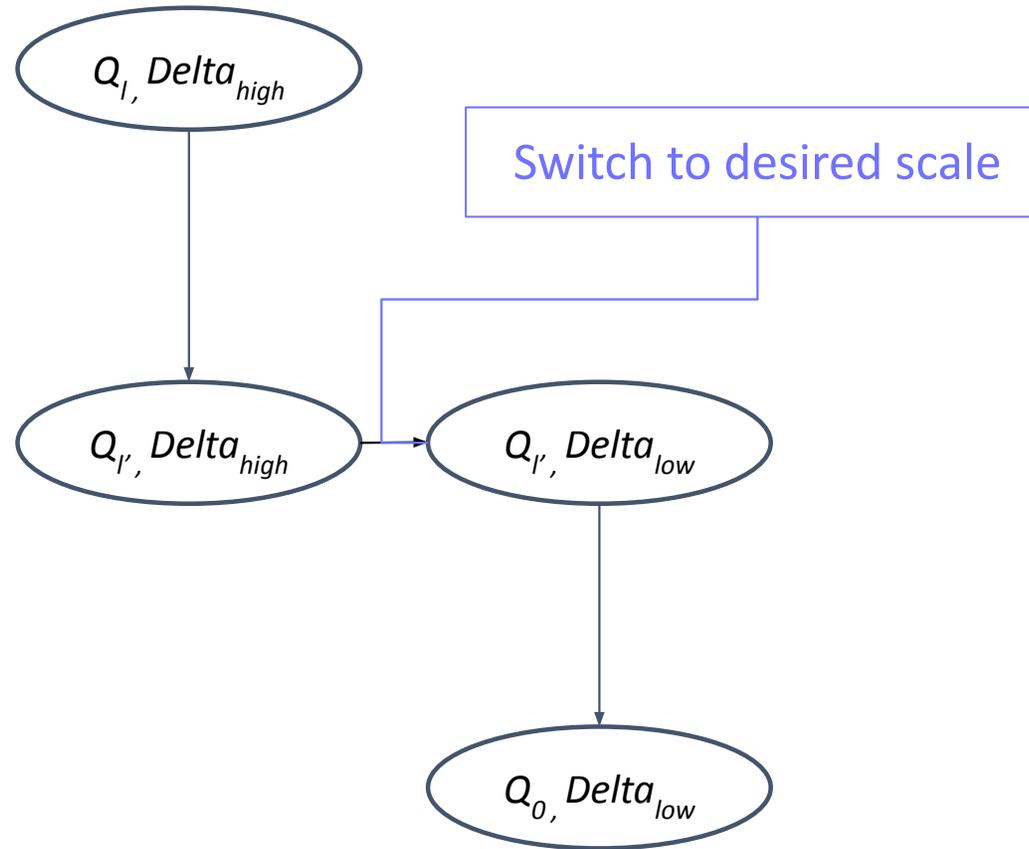
Design Challenges 2 : Redefine Levels



A simple HEaaN2 Levels



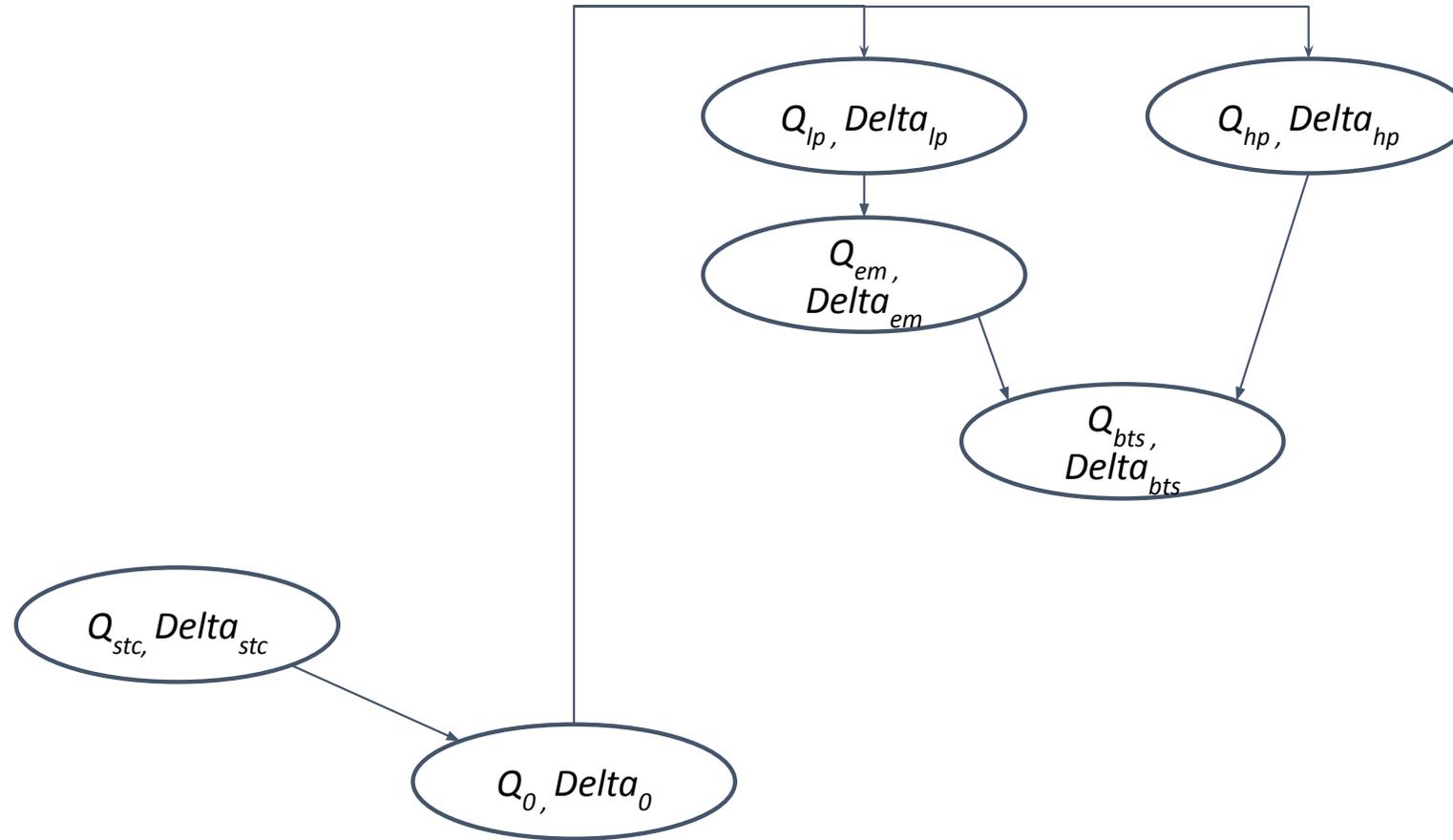
Design Challenges 2 : Redefine Levels



A modulus-optimized Levels



Design Challenges 2 : Redefine Levels



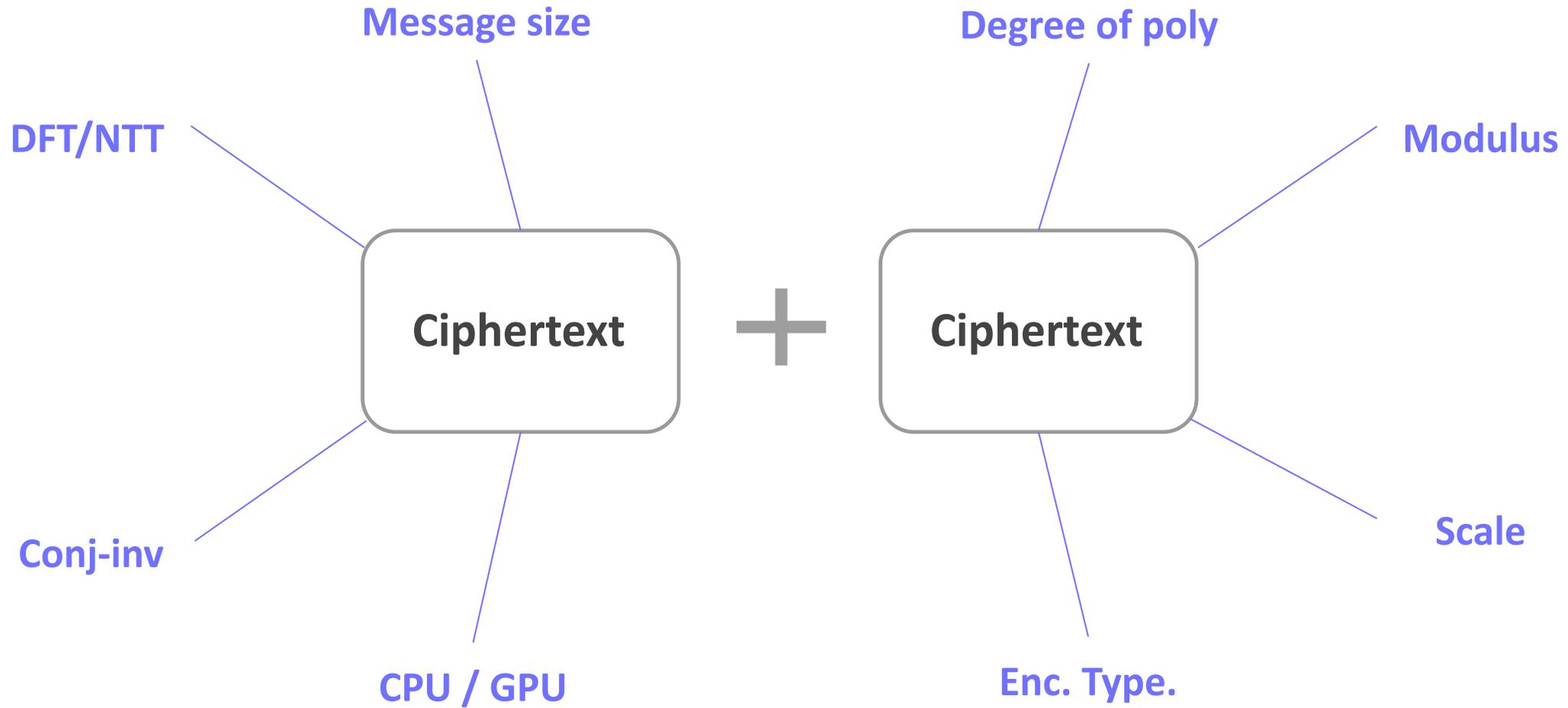
Example of advanced use of Levels*

*[CCS25] Leveraging Discrete CKKS to Bootstrap in High Precision, Choe et al.



Exponential growth of design space

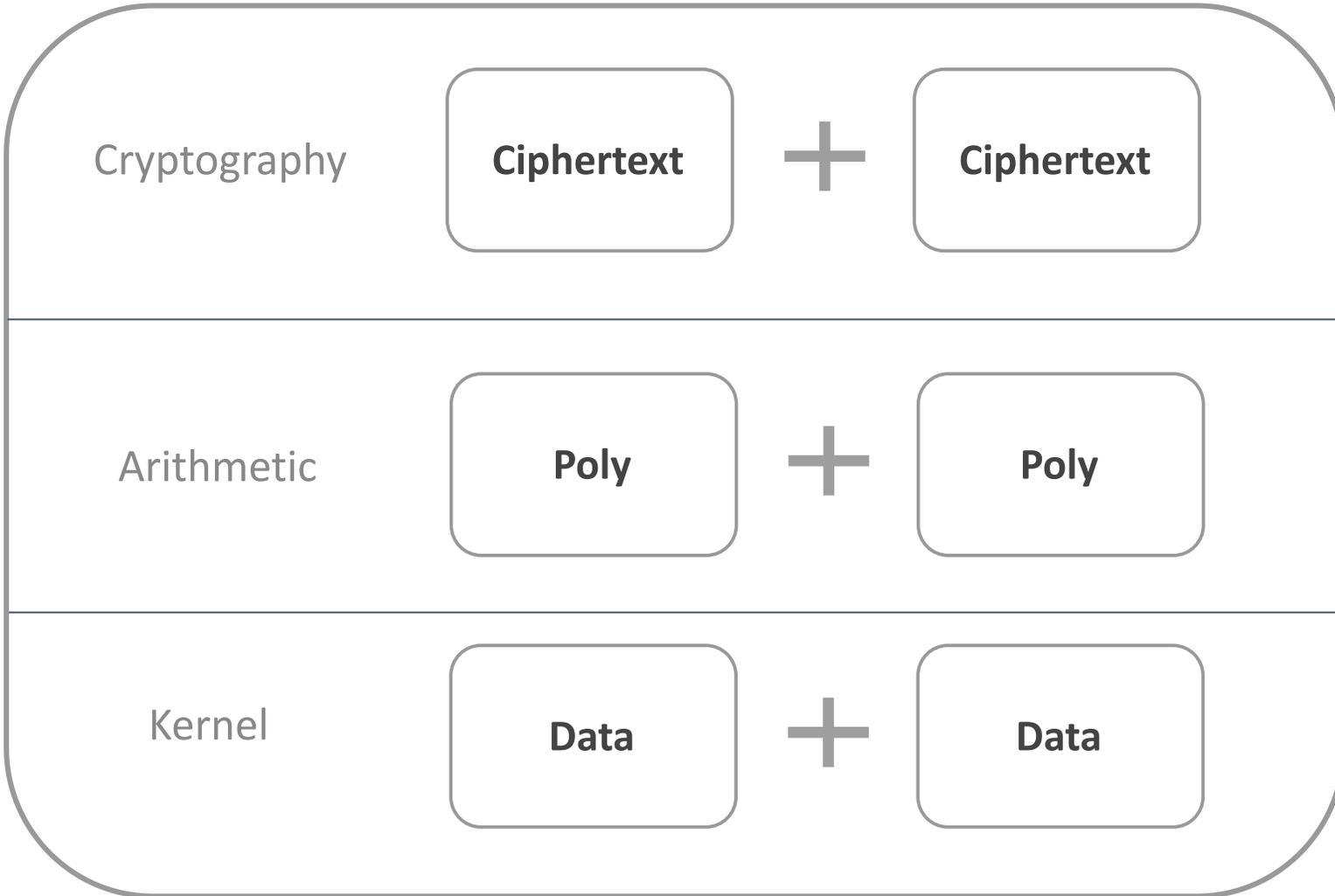
Design Challenges 3 : Allow vast design space





Distribute complexity through layers

Design Challenges 3 : Allow vast design space



Encoding
 size of message
 scale
 dft

Encryption
 power
 rank
 num secrets

PolyRing
 degree
 modulus
 NTT

Device

Conclusion



Conclusion



HEaaN2

Sign In →

CODE.HEAAN

Bring Your Own Encrypted Idea

Ultra-fast CKKS bootstrapping with GPU acceleration
Build encrypted applications in your browser

9.6ms

Bootstrapping of CKKS Ciphertext to 15 Levels
measured on RTX 5090

20.9ms

Bootstrapping of CKKS Ciphertext to 15 Levels
measured on A100

- *CKKS only.*
- *Closed-source.*
- *SOTA performance.*
- *Available at : heaan.io*



Again, what is HEaaN2

Conclusion

```

EXPLORER
  USER
  .cache
  .config
  .local
  devkit
  docs
  examples
  include
  lib
  python
  gpu_run_results
    output_job-user176251-20260306094727-q...
    output_job-user176251-20260306095632-c...
  .bash_history
  .bashrc
  .clangd
  .sudo_as_admin_successful

  OUTPUT
  TIMELINE

gpu_run_results > output_job-user176251-20260306094727-quickstart.log
1 ===== GPU Worker Specification (a100) =====
16 =====
17
18
19 Encoding message into plaintext...
20 Done.
21 Encrypting plaintext into ciphertext...
22 Done.
23 Current Level : 15
24 Generating switching keys...
25 Done.
26 Performing homomorphic evaluations...
27 Level before squaring: 15
28 - Squaring...
29   | Scaling factor before rescale: 7.55579e+22
30   | Scaling factor after rescale: 2.74876e+11
31 Level after squaring: 14
32 - Rotating...
33 - Conjugating...
34 - Multiply by a real constant...
35 Done.
36   | Scaling factor after mul by constant: 7.55579e+22
37   | Scaling factor after rescale: 2.74876e+11
38 - Multiply by a Gaussian integer...
39 Done.
40   | scaling factor after mul by Gaussian integer: 2.74876e+11
41 Level after all operations: 14
42 Leveling down to 0...
43 Done.
44 Generating bootstrap key...
45 Done.
46 Warming up the bootstrapper...
47 Done.
48 Bootstrapping...
49 Done.
50 Level after bootstrapping: 15
51 Bootstrap latency (ms) : 20.816
52 Decrypting the result...
53 Done.
54 Decoding the result...
55 Done.
56 msg1 : (-0.382255, 2.40703) (3.43279, -0.998938) (-5.16589, 0.3804) (-1.15347, -3.08616) (-1.5665, -1.76749)
57 msg2 : (-0.382256, 2.40703) (3.43279, -0.998937) (-5.16589, 0.380399) (-1.15347, -3.08616) (-1.5665, -1.76749)

```

- **CKKS only.**
- **Closed-source.**
- **SOTA performance.**
- **Available at : heaan.io**



Again, what is HEaaN2

Conclusion

The screenshot shows a code editor with the following content:

```

313 - // Square via mulRescale
314 - auto san_sq = ICiphertext::make();
315 - heval.mulRescale(*san_ct, *san_ct, *san_sq, *relin_key);
316 - // Decrypt and check
317 - auto san_dec = IPlaintext::make();
318 - encryptor.decrypt(*san_sq, *sk, *san_dec);
319 - Message san_res;
320 - encoder.decode(*san_dec, san_res);
321 - san_res.to(Device::CPU);
322 - double got = san_res[0].real();
323 - double err = std::abs(got - 9.0); // expect 3^2 = 9
324 - if (err < 1.0) {
325 -     std::cout << "    Relin key OK (sanity error: " << err << " )"
326 -     << std::endl;
327 -     break;
328 - }
329 - std::cout << "    Attempt " << (attempt + 1)
330 -     << " "; sanity failed (got " << got
331 -     << ", expected 9), regenerating..." << std::endl;
332 - }
333 -
296 // =====
297 // STEP 4-6: Process test images
298 // =====

```

Below the code, there are diagnostic messages and a bash command:

```

● Searched for 1 pattern (ctrl+o to expand)
● Good, only one definition. The diagnostic was stale. Let me build and test.
● Bash(make MnistQuadratic -j$(nproc) 2>&1 && echo "=== 1 image ===" && cd ../classifier && cp
quad_test_images.txt quad_test_images.txt.bak && cp quad_test_labels.txt
└─ Running─

```

The bash command is:

```

make MnistQuadratic -j$(nproc) 2>&1 && echo "=== 1 image ===" && cd ../classifier && cp
quad_test_images.txt quad_test_images.txt.bak && cp quad_test_labels.txt
> quad_test_labels.txt && cd .. /build && for i in 1 2 3; do echo "— Run $i —" &&
./MnistQuadratic 2>&1 | grep -E "prediction|accuracy"; done && echo "=== 10 images ===" && cd
../classifier && cp quad_test_images.txt.bak quad_test_images.txt && cp quad_test_labels.txt.bak
quad_test_labels.txt && cd .. /build && ./MnistQuadratic 2>&1 | grep -E "accuracy|match"

```

Command contains quoted characters in flag names

```

Do you want to proceed?
> 1. Yes
2. No

```

Esc to cancel · Tab to amend · ctrl+e to explain

Terminal output:

```

[0] @:claude>
[4/6] Encrypting input...
[5/6] Layer 1 + Quadratic + Layer 2...

```

Terminal output (right side):

```

./MnistQuadratic
Class | Plaintext | Encrypted | Error
-----|-----|-----|-----
0 | -8.797037 | -8.797039 | 2.36e-06
1 | 2.077547 | 2.077545 | 2.12e-06
2 | -6.659669 | -6.659672 | 3.64e-06
3 | -10.810706 | -10.810708 | 2.05e-06
* 4 | 29.768277 | 29.768266 | 1.06e-05
5 | -11.671589 | -11.671602 | 1.28e-05
6 | -8.939387 | -8.939389 | 2.66e-06
7 | 12.129476 | 12.129474 | 2.27e-06
8 | 4.393008 | 4.393007 | 1.26e-06
9 | 5.839725 | 5.839727 | 1.68e-06

Plaintext prediction: 4
Encrypted prediction: 4
True label: 4

Test sample 2 (true label: 8)

```

- **CKKS only.**

- **Closed-source.**

- **SOTA performance.**

- **Available at : heaan.io**

Thank you for listening.

