Fully Homomorphic Encryption

Overview, applications and new directions

Ilaria Chillotti

College de France, Paris (France)

November 27, 2025

Outline

- Fully Homomorphic Encryption
- LWE based FHE schemes
- Homomorphic properties & bootstrapping (TFHE)
- Applications
- New directions

Fully Homomorphic Encryption

FHE

Fully Homomorphic Encryption

- Computations over encrypted data (bits, integers, approximate)
- Noisy ciphertexts

Timeline

1978 - Privacy Homomorphisms (Rivest, Adleman, Dertouzoz)

Partially Homomorphic schemes

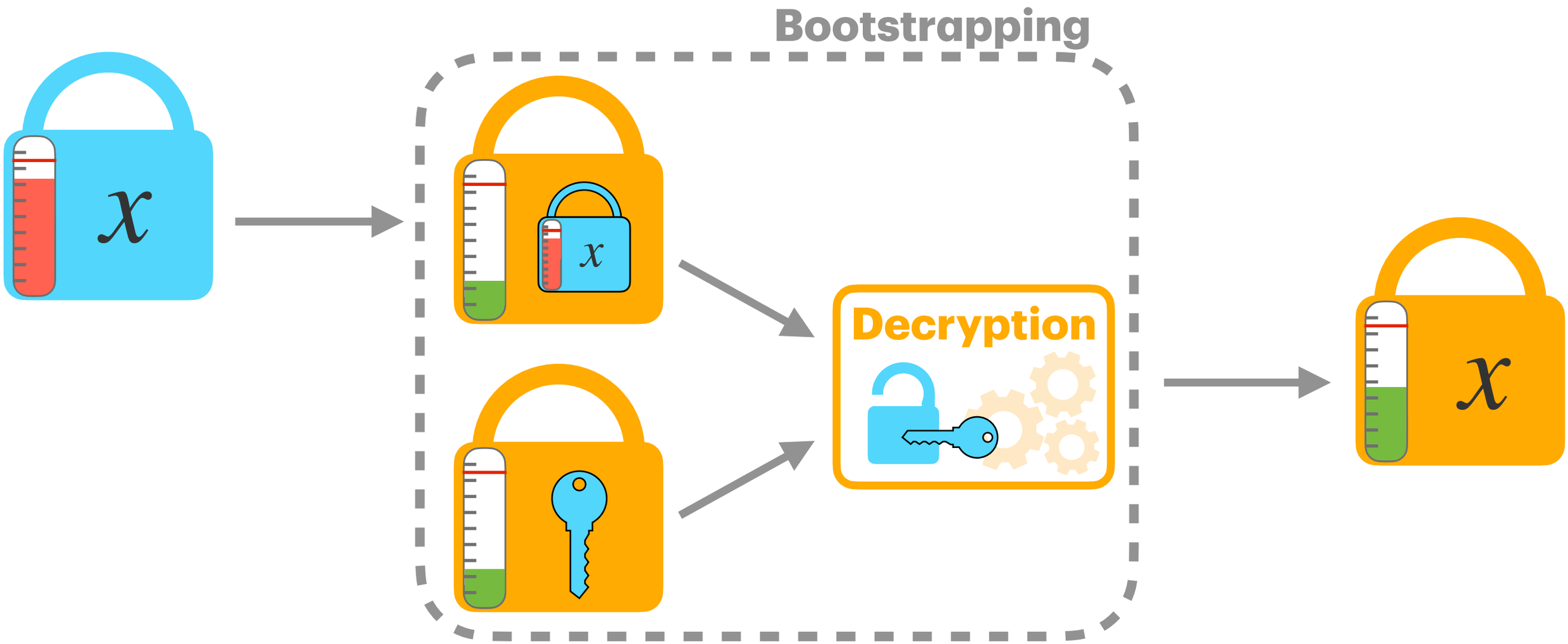
- RSA,
- ElGamal,
- Paillier,
- Goldwasser-Micali, ...

2005 - Somewhat Homomorphic (Boneh, Goh, Nissim)

2009 - First FHE scheme & bootstrapping (Gentry)

Bootstrapping

The idea (Gentry 2009)



"Fully homomorphic encryption using ideal lattices", C. Gentry, STOC 2009

Bootstrapping

How often do we have to use the bootstrapping in practice?

Leveled approach

If f can be evaluated with a **small depth** circuit

The deepest the circuit:

- ❌ The largest the ciphertexts
- ❌ The slowest the computations

Bootstrapped approach

If f requires to be evaluated with a **deep depth** circuit, or if the function is **unknown**

- ✅ No depth limitations (bootstrap when necessary)
- ❌ Bootstrapping is the most costly operation

Timeline

1978 - Privacy Homomorphisms (Rivest, Adleman, Dertouzoz)

Partially Homomorphic schemes

- RSA,
- ElGamal,
- Paillier,
- Goldwasser-Micali, ...

2005 - Somewhat Homomorphic (Boneh, Goh, Nissim)

2009 - First FHE scheme & bootstrapping (Gentry)

2010 - FHE over the integers: DGHV (van Dijk, Gentry, Halevi, Vaikuntanathan)

2011 - LWE based FHE: **BGV** (Brakerski, Gentry, Vaikuntanathan)

2012 - **B/FV** (Brakerski / Fan, Vercauteren) &
2013 - **GSW** (Gentry, Sahai, Waters)

NTRU based FHE: LTV
(Lopez-Alt, Tromer, Vaikuntanathan)

2014 - **FHEW/DM** (Ducas, Micciancio)

2016 - **TFHE/CGGI** (Chillotti, Gama, Georgieva, Izabachène)
HEAAN/CKKS (Cheon, Song, Song, Kim)

...

LWE based FHE schemes

LWE - Learning With Errors

The problem(s) - Regev 2005

Ring LWE variant
[SSTX09], [LPR10]

$$\vec{s} = (s_0\ s_1\ s_2\ s_3\ s_4 \cdots s_{n-1}) \in \mathbb{Z}^n$$

$$\sum_{i=0}^{n-1} a_i \cdot s_i + e = u$$

$$a_i \in \mathbb{Z}_q \qquad e \in \mathbb{Z}_q \qquad u \in \mathbb{Z}_q$$

LWE Sample

$$(a_0\ a_1\ a_2\ a_3\ a_4 \cdots a_{n-1}\ u)$$

Computational LWE: Given many LWE samples, hard to find the secret key

Decisional LWE: Hard to distinguish between LWE samples and random samples

"On lattices, learning with errors, random linear codes, and cryptography", O. Regev, STOC 2005
"Efficient public key encryption based on ideal lattices", D. Stehlé, R. Steinfeld, K. Tanaka, K. Xagawa, ASIACRYPT 2009
"On ideal lattices and learning with errors over rings", V. Lyubashevsky, C. Peikert, O. Regev, EUROCRYPT 2010

LWE - Learning With Errors

Encryption: from message to ciphertext

$$\vec{s} = (s_0\ s_1\ s_2\ s_3\ s_4\ \cdots\ s_{n-1}) \in \mathbb{Z}^n$$

$$(\quad \cdots \quad)$$

$$0\ \ 0\ \ 0\ \ 0\ \ 0 \quad \cdots \quad 0\ \ \Delta m$$

Encryption

$$(\quad \cdots \quad) \in \mathrm{LWE}_{\vec{s}}(\Delta m)$$

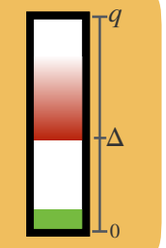
$$a_0\ a_1\ a_2\ a_3\ a_4 \quad \cdots \quad a_{n-1}\ \ b = u + \Delta m$$

LWE - Learning With Errors

Decryption: from ciphertext to message

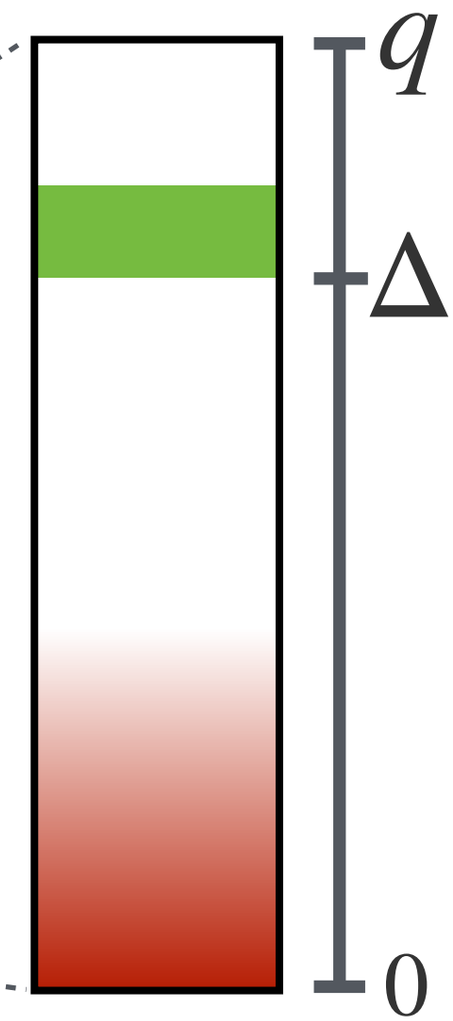
$$\vec{s} = (s_0\ s_1\ s_2\ s_3\ s_4 \cdots s_{n-1}) \in \mathbb{Z}^n$$

LWE with message in the LSB
noise rescaled to a certain Δ



$$(a_0\ a_1\ a_2\ a_3\ a_4 \quad \cdots \quad a_{n-1}\ b)$$

$$b - \sum_{i=0}^{n-1} a_i \cdot s_i =$$



RLWE - Ring Learning With Errors

Encryption: from message to ciphertext

$$S = \sum_{i=0}^{N-1} s_i X^i = \boxed{s_0}\boxed{s_1}\boxed{s_2}\boxed{s_3}\cdots\boxed{s_{n-2}}\boxed{s_{n-1}} \in \mathbb{Z}[X]/(X^N+1)$$

$$\left(\boxed{}\boxed{}\boxed{}\boxed{}\cdots\boxed{}\boxed{}\quad \boxed{}\boxed{}\boxed{}\boxed{}\cdots\boxed{}\boxed{}\right)$$

$$0\ 0\ 0\ 0 \quad \cdots \quad 0\ 0 \quad \Delta m_0 \quad \cdots \quad \Delta m_{N-1}$$

$$\Delta M = \sum_{i=0}^{N-1} \Delta m_i X^i$$

Encryption

$$\left(\boxed{}\boxed{}\boxed{}\boxed{}\cdots\boxed{}\boxed{}\quad \boxed{}\boxed{}\boxed{}\boxed{}\cdots\boxed{}\boxed{}\right) \in \mathrm{RLWE}_S(\Delta M)$$

$$A = \sum_{i=0}^{N-1} a_i X^i \in \mathbb{Z}_q[X]/(X^N+1) \qquad B = \sum_{i=0}^{N-1} b_i X^i = A \cdot S + E + \Delta M \in \mathbb{Z}_q[X]/(X^N+1)$$

RLWE - Ring Learning With Errors

Decryption: from ciphertext to message

$$S = \sum_{i=0}^{N-1} s_i X^i = \boxed{s_0 \; s_1 \; s_2 \; s_3 \; \cdots \; s_{n-2} \; s_{n-1}} \in \mathbb{Z}[X]/(X^N + 1)$$

$$\left(\boxed{\cdots}_{A} \quad \boxed{\cdots}_{B} \right)$$

$$B - A \cdot S = \boxed{\cdots}$$

RLWE - Ring Learning With Errors

Decryption: from ciphertext to message

$$S = \sum_{i=0}^{N-1} s_i X^i = \boxed{s_0 \; s_1 \; s_2 \; s_3 \; \cdots \; s_{n-2} \; s_{n-1}} \in \mathbb{Z}[X]/(X^N + 1)$$

$$\left(\boxed{\cdots}_{A} \quad \boxed{\cdots}_{B} \right)$$

$$B - A \cdot S = \boxed{\cdots}$$

TFHE - Coefficient packing
CKKS - Slot packing (SIMD)

Homomorphic Properties & Bootstrapping (TFHE)

Homomorphic Properties

Noise increase

LWE

- Addition ▲
- Constant Multiplication ▲

RLWE

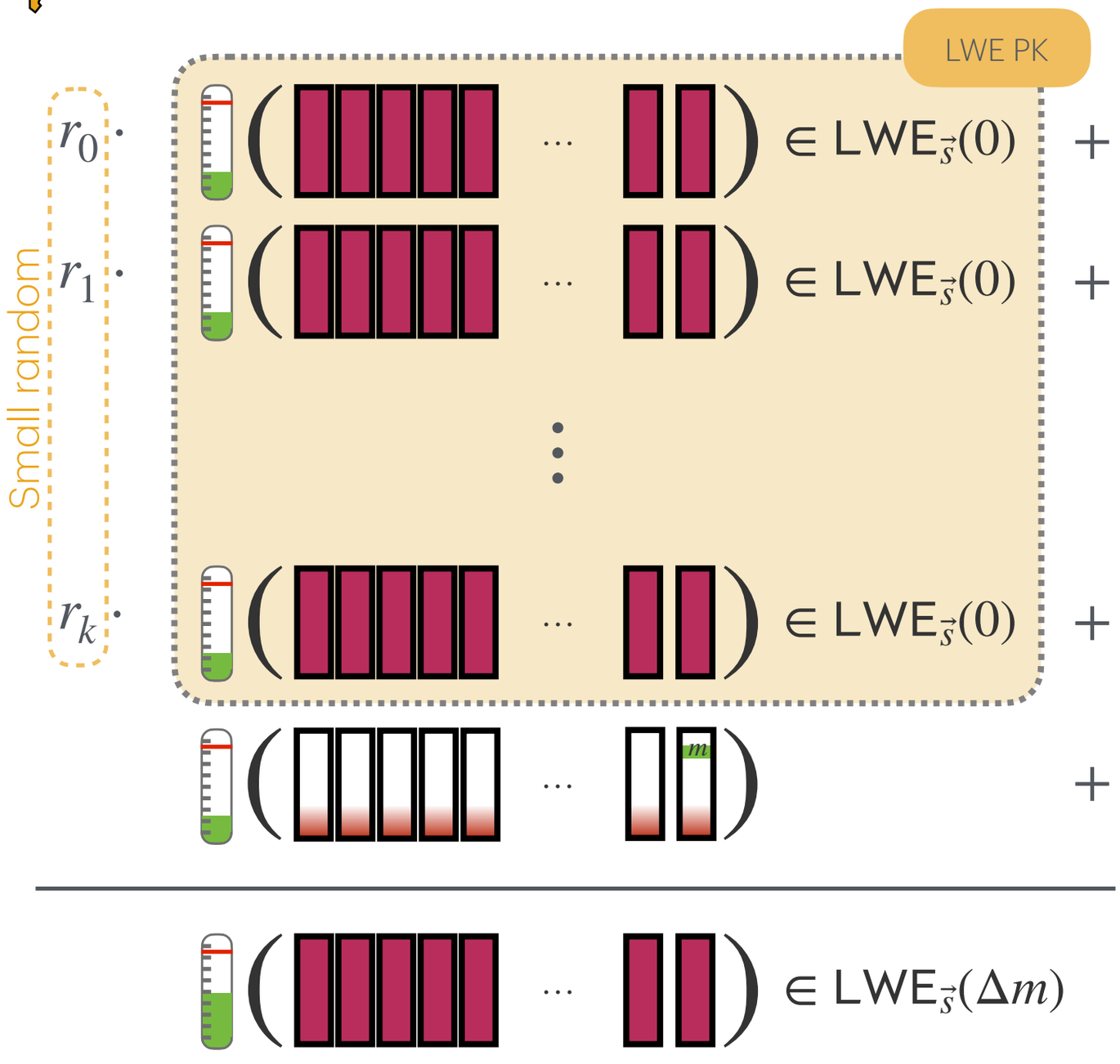
- Addition ▲
- Constant Multiplication ▲

LWE Public Key Encryption

Similar for RLWE

Random combination of encryptions of zero

$\vec{s} = (s_0\ s_1\ s_2\ s_3\ s_4 \cdots s_{n-1}) \in \mathbb{Z}^n$



Homomorphic Properties

Noise increase

LWE

- Addition 🔺
- Constant Multiplication 🔺

⚠️Only small constants

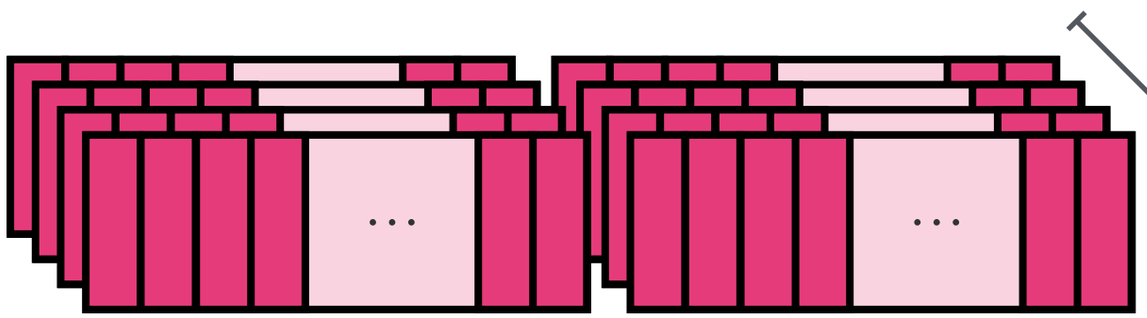
RLWE

- Addition 🔺
- Constant Multiplication 🔺

RLev & RGSW

The power of redundancy

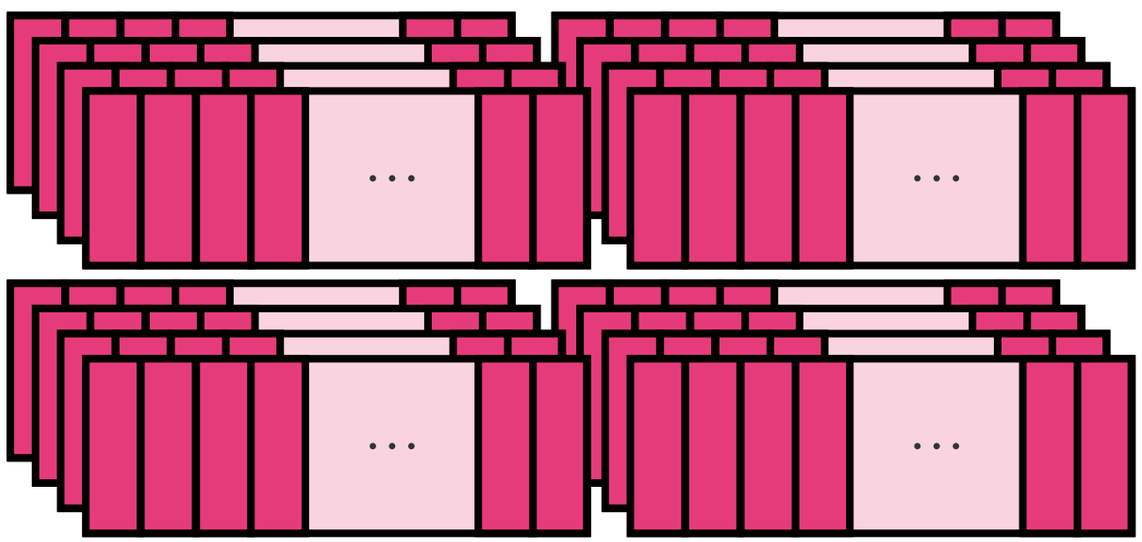
$$S = \boxed{s_0 \; s_1 \; s_2 \; s_3 \; \cdots \; s_{n-2} \; s_{n-1}} \in \mathbb{Z}_N[X]$$



$$\Delta_j = \frac{q}{\beta j}$$

Base of decomposition for larger constants

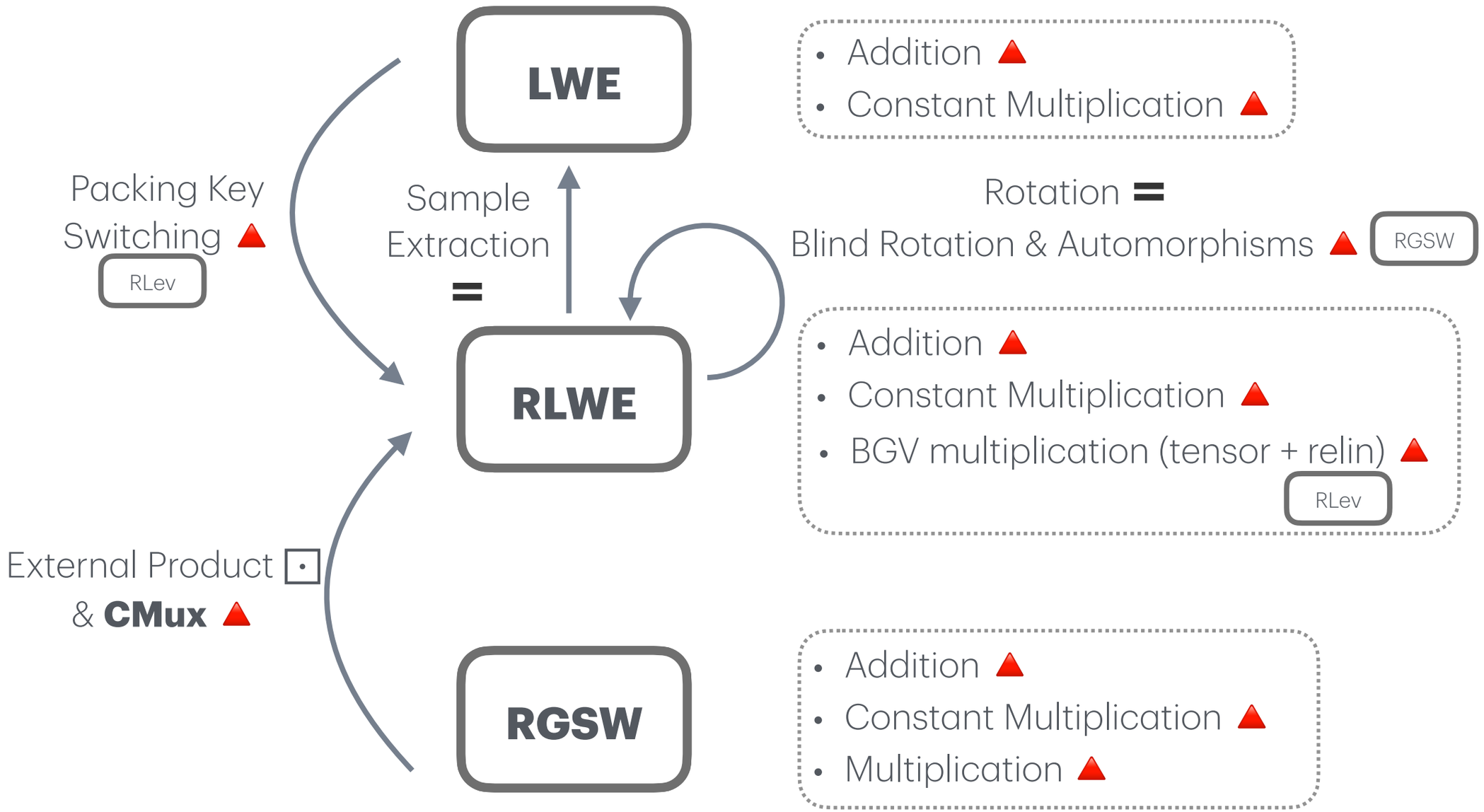
$$\in \mathsf{RLev}_S(M) = (\mathsf{RLWE}_S(\Delta_1 M) \times \ldots \times \mathsf{RLWE}_S(\Delta_\ell M))$$



$$\in \mathsf{RGSW}_S(M) = (\mathsf{RLev}_S(-S \cdot M) \times \mathsf{RLev}_S(M))$$

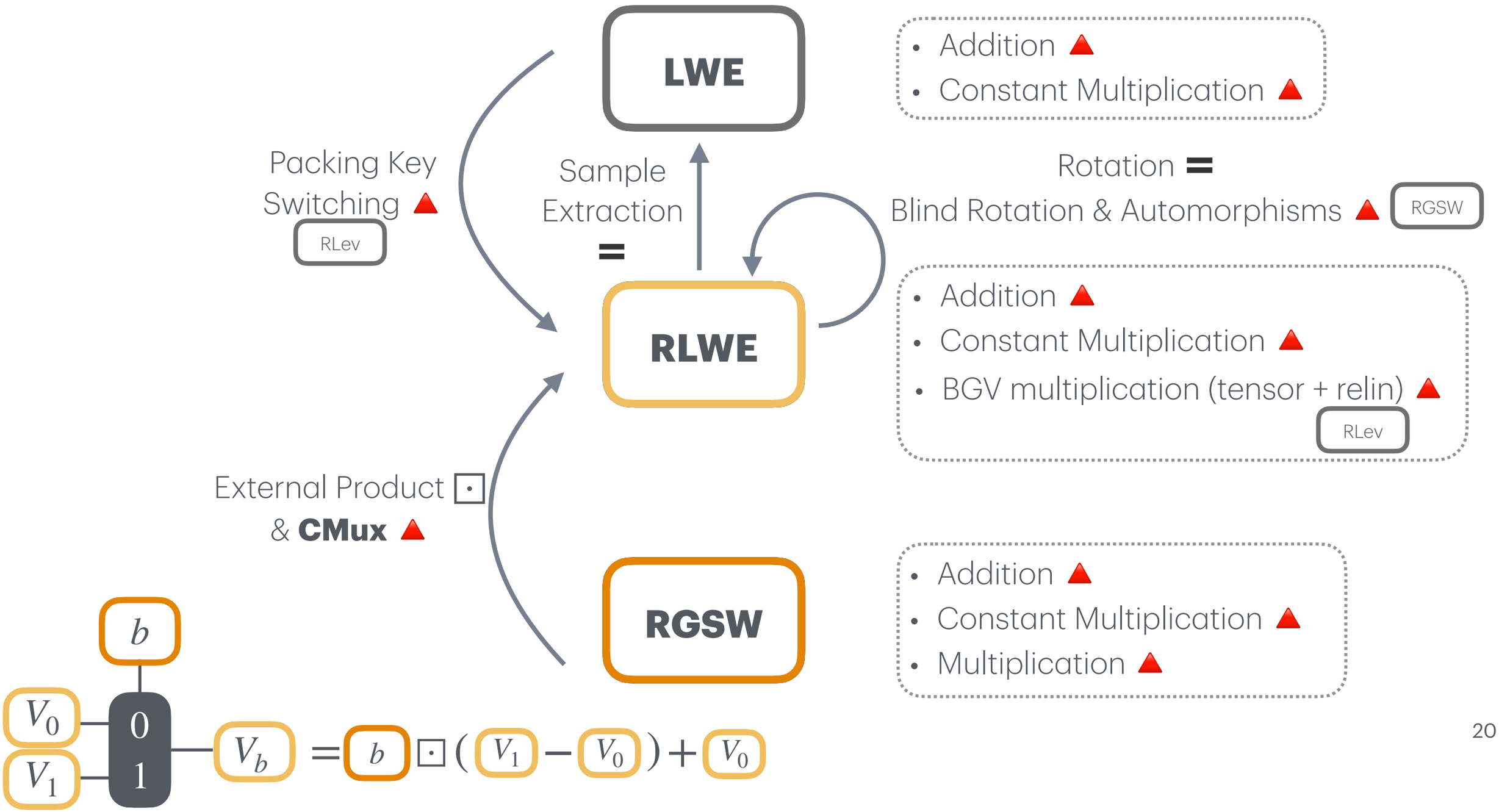
Homomorphic Properties

Noise increase



Homomorphic Properties

Noise increase



Homomorphic Properties

Noise increase

All ciphertexts can be **Key-Switched** 🔑 → 🗝

Packing Key Switching ▲

RLev

LWE

Sample Extraction =

- Addition ▲
- Constant Multiplication ▲

Rotation =
Blind Rotation & Automorphisms ▲

RGSW

RLWE

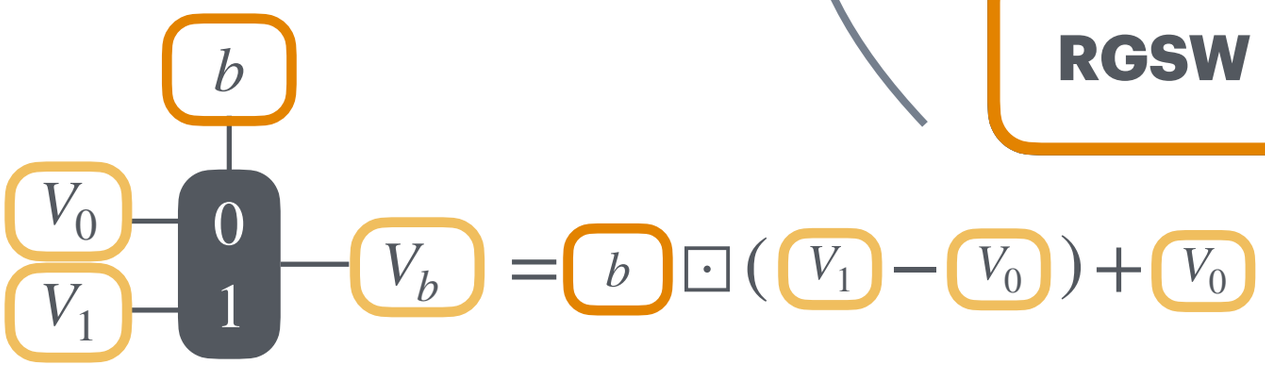
- Addition ▲
- Constant Multiplication ▲
- BGV multiplication (tensor + relin) ▲

RLev

External Product ⊡ & **CMux** ▲

RGSW

- Addition ▲
- Constant Multiplication ▲
- Multiplication ▲

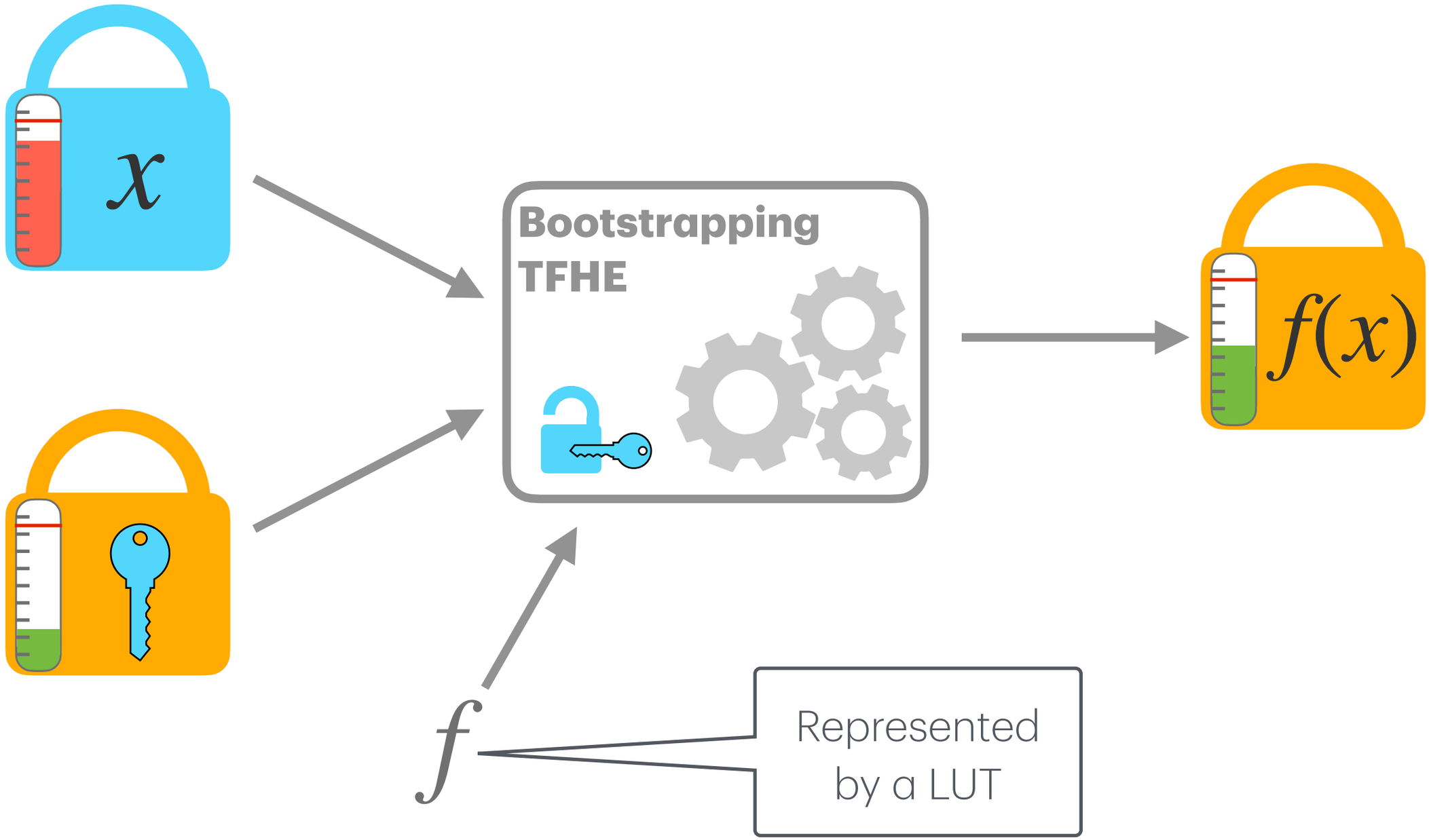


Homomorphic Properties

Bootstrappings

TFHE Bootstrapping

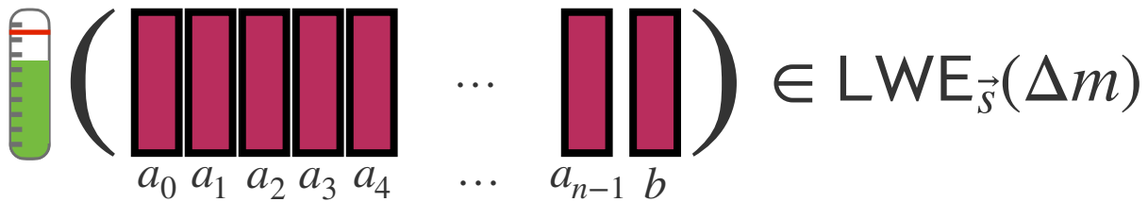
The idea



"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

TFHE Bootstrapping

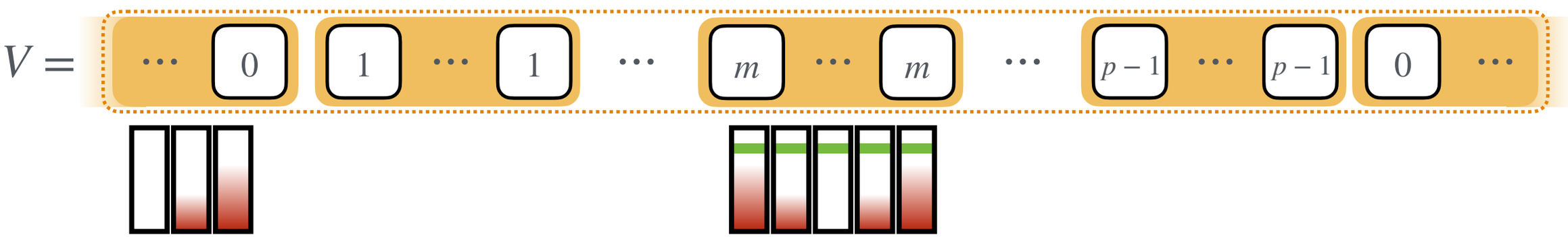
How it works (intuition)



Decryption

$$\Delta m + e$$

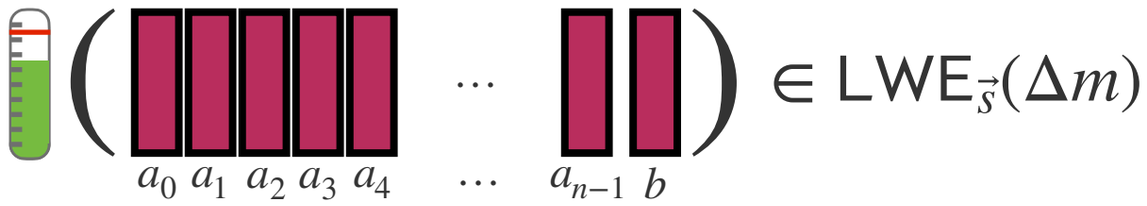
2 Rounding with a Look-Up Table (encoded in a polynomial)



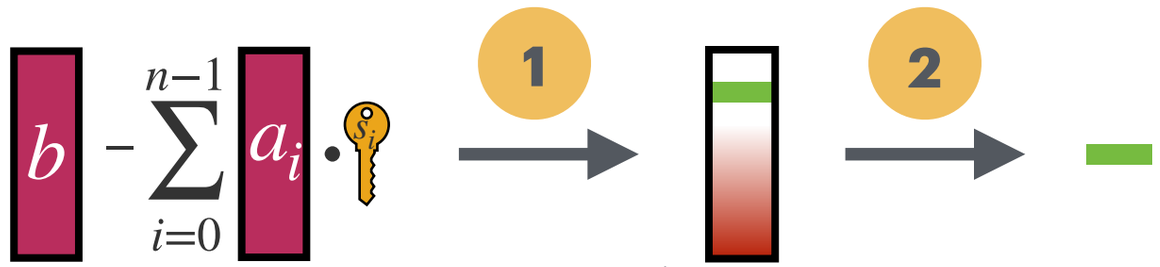
"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

TFHE Bootstrapping

How it works (intuition)

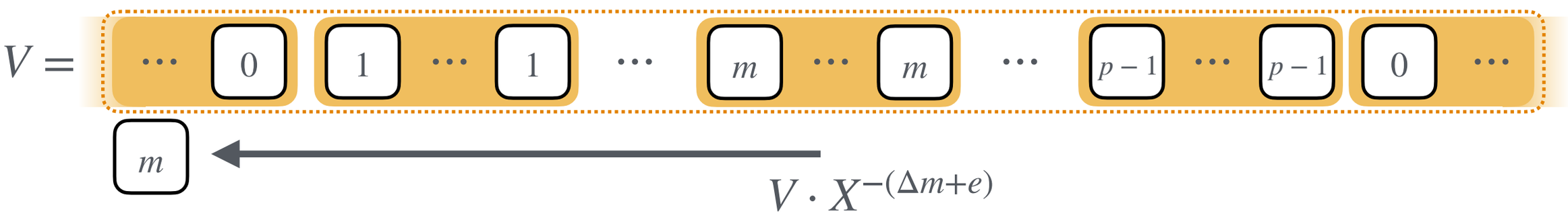


Decryption



$\Delta m + e$

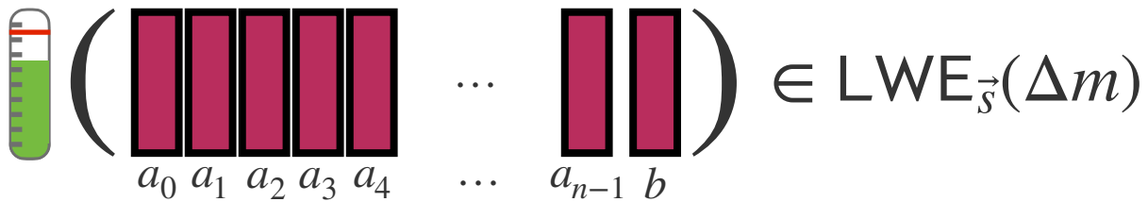
2 Rounding with a Look-Up Table (encoded in a polynomial)



"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

TFHE Bootstrapping

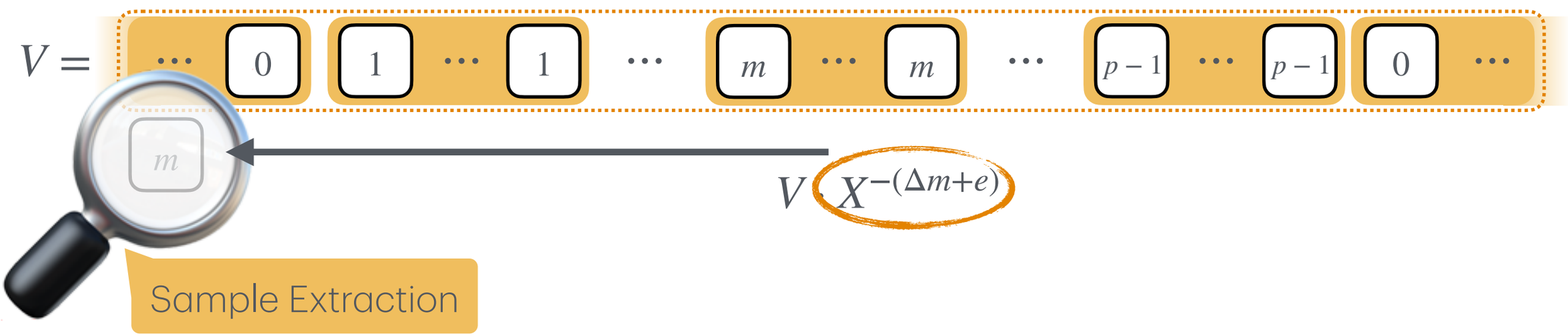
How it works (intuition)



Decryption

$\Delta m + e$

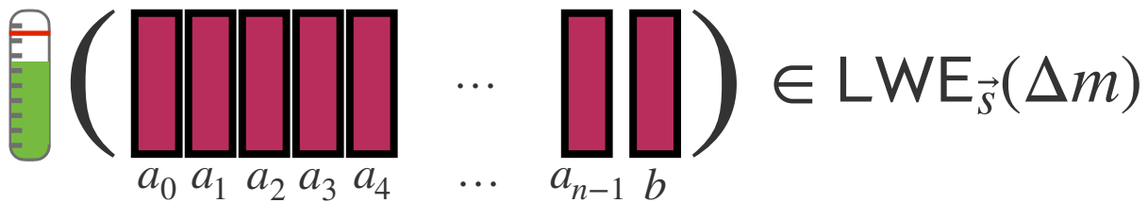
2 Rounding with a Look-Up Table (encoded in a polynomial)



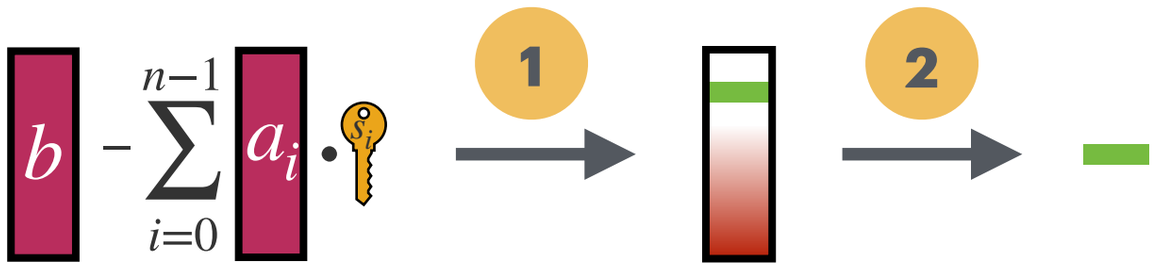
"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

TFHE Bootstrapping

How it works (intuition)

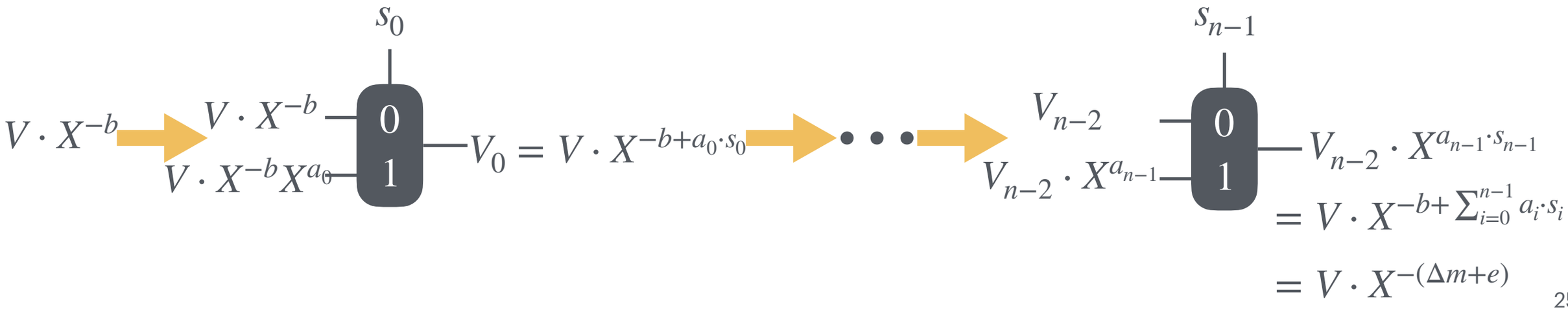


Decryption



0 or 1 (TFHE)

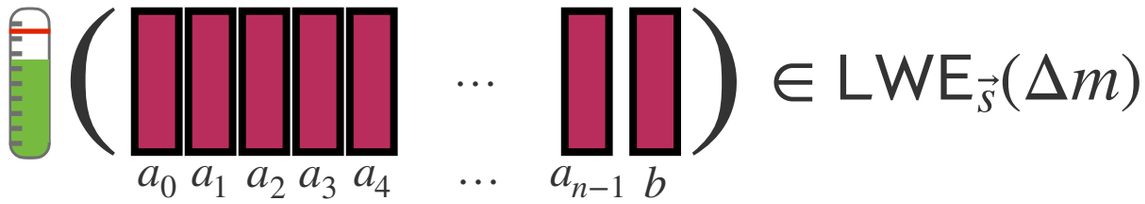
1 Compute $V \cdot X^{-(\Delta m+e)}$ → Observe that $-(\Delta m + e) = -b + \sum_{i=0}^{n-1} a_i \cdot s_i$

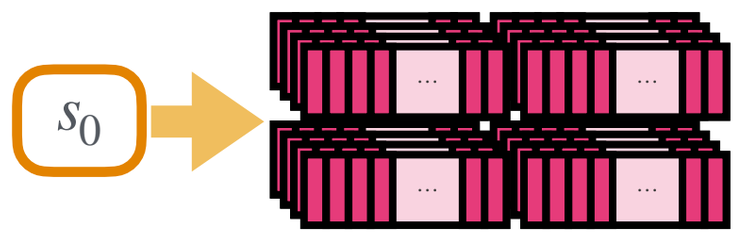


"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

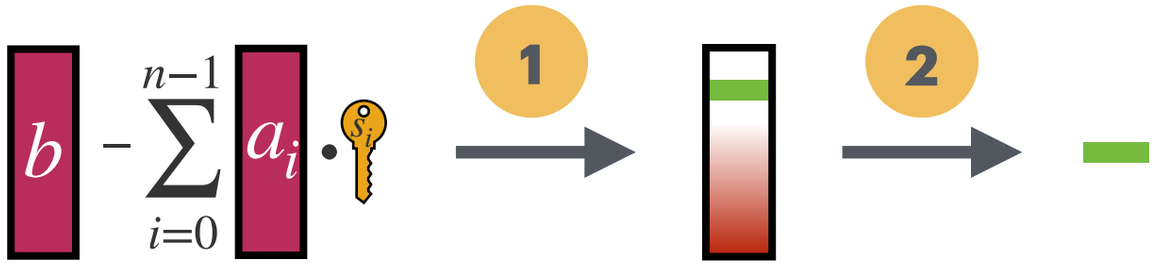
TFHE Bootstrapping

How it works (intuition)



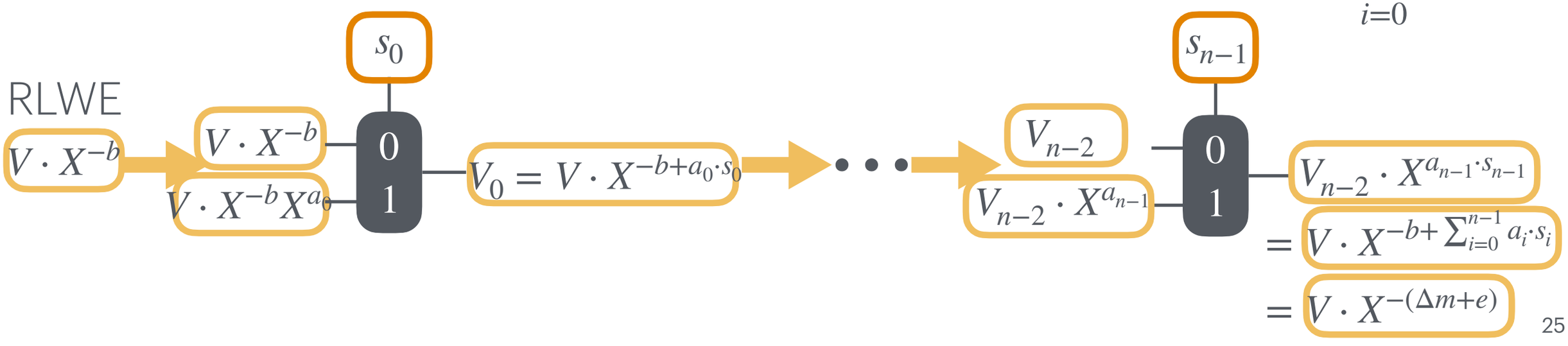


Decryption



0 or 1 (TFHE)

1 Compute $V \cdot X^{-(\Delta m+e)}$ → Observe that $-(\Delta m + e) = -b + \sum_{i=0}^{n-1} a_i \cdot s_i$



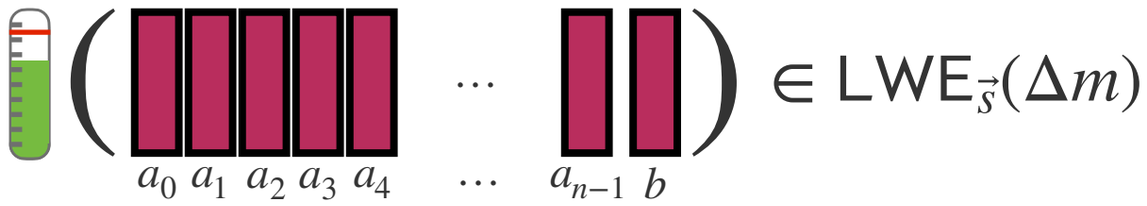
"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

TFHE Bootstrapping

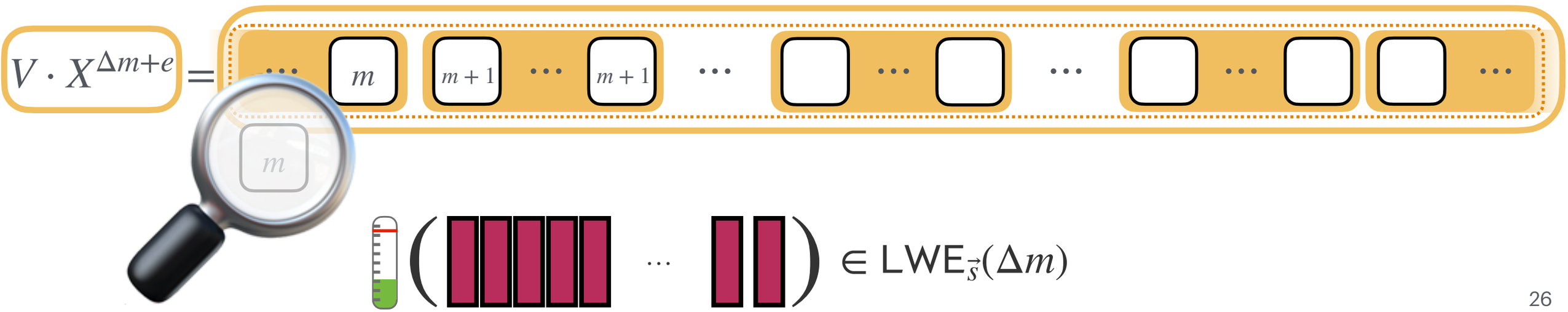
How it works (intuition)

A few details under the carpet:

- Scaling factor in V
- Initial modulus switching
- Negacyclic tables



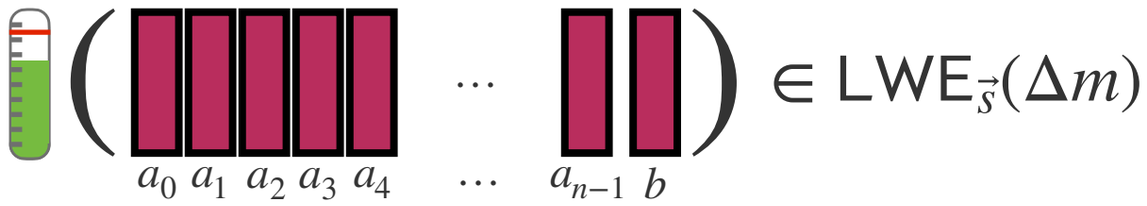
Decryption



"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

TFHE Bootstrapping

How it works (intuition)



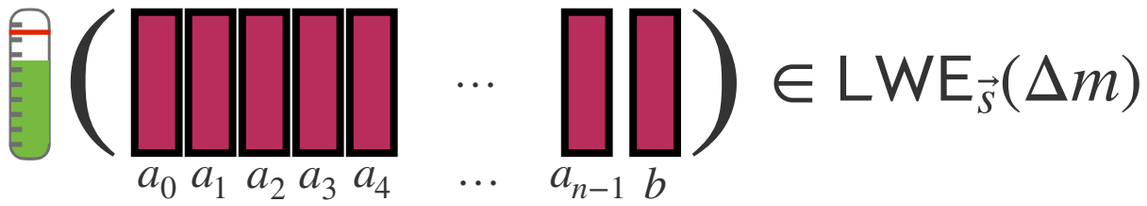
Decryption

2 Rounding with a **Look-Up Table** (encoded in a polynomial)

"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

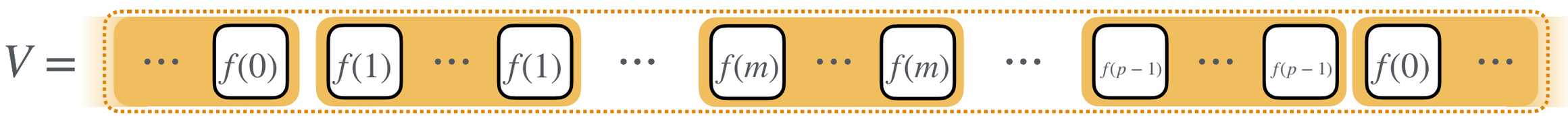
TFHE Bootstrapping

How it works (intuition)



Decryption

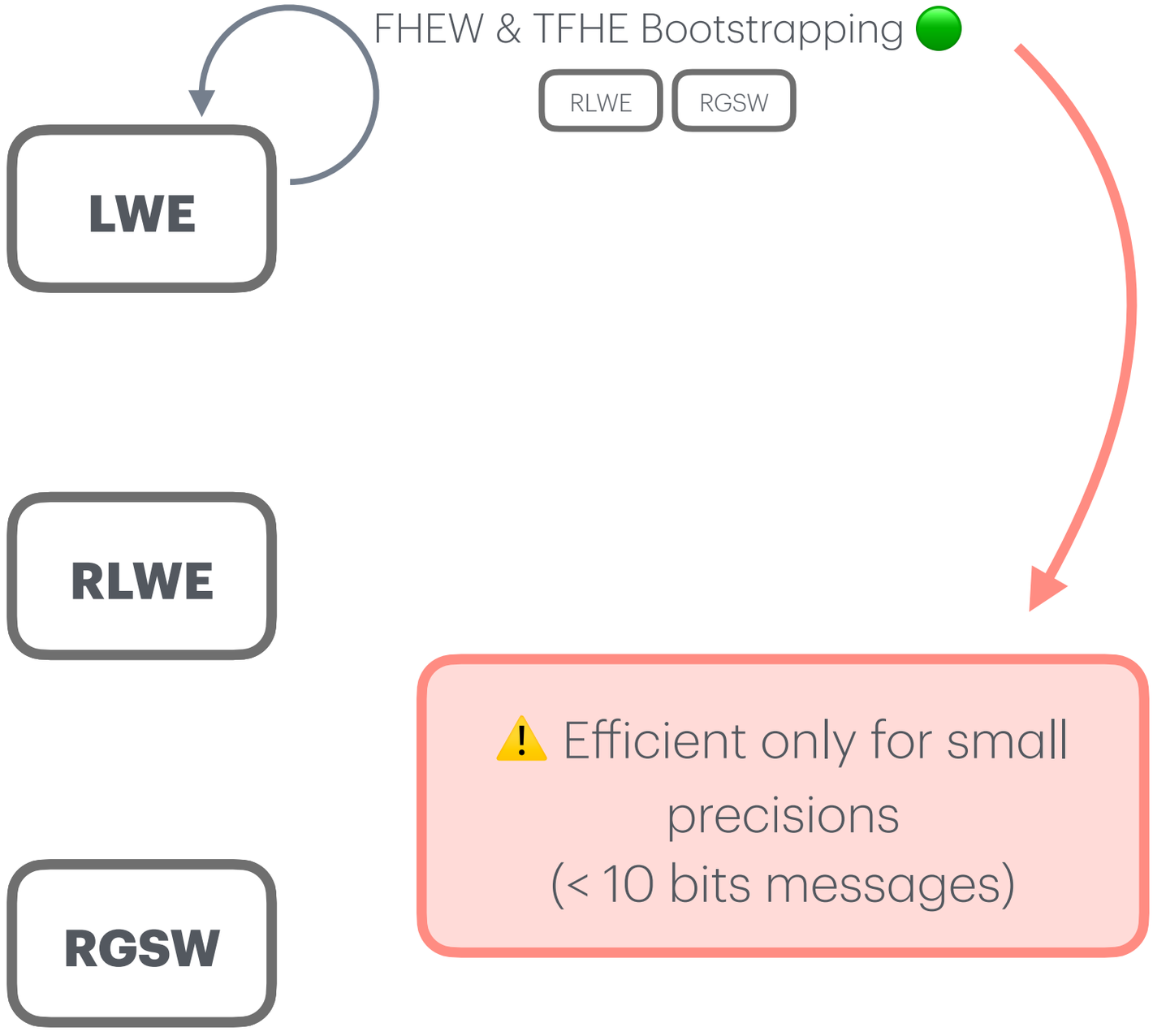
2 Rounding with a **Look-Up Table** (encoded in a polynomial)



"TFHE: Fast Fully Homomorphic Encryption over the Torus". I. Chillotti, N. Gama, M. Georgieva, M. Izabachène. Journal of Cryptology 2020.

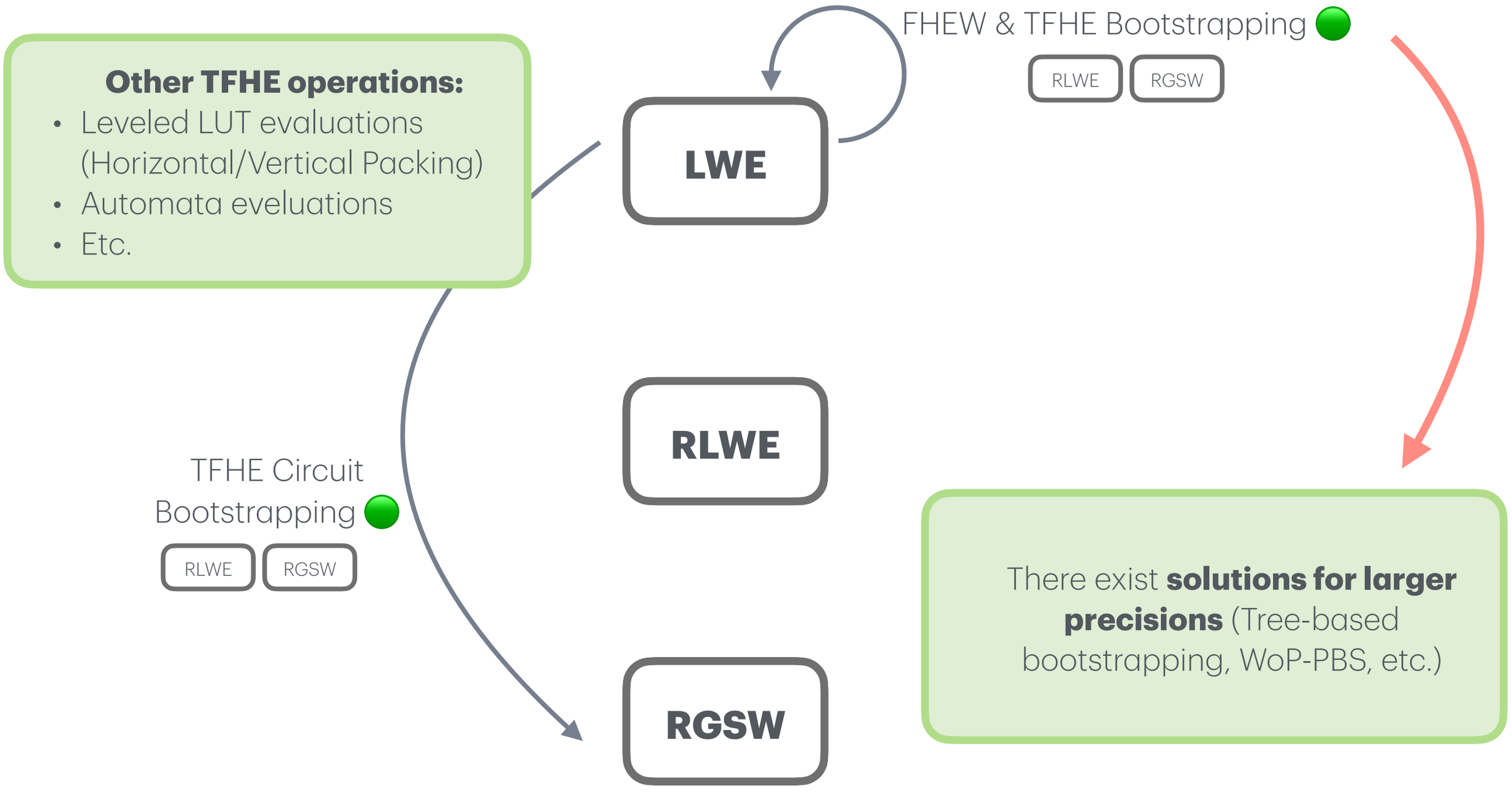
Homomorphic Properties

Bootstrappings



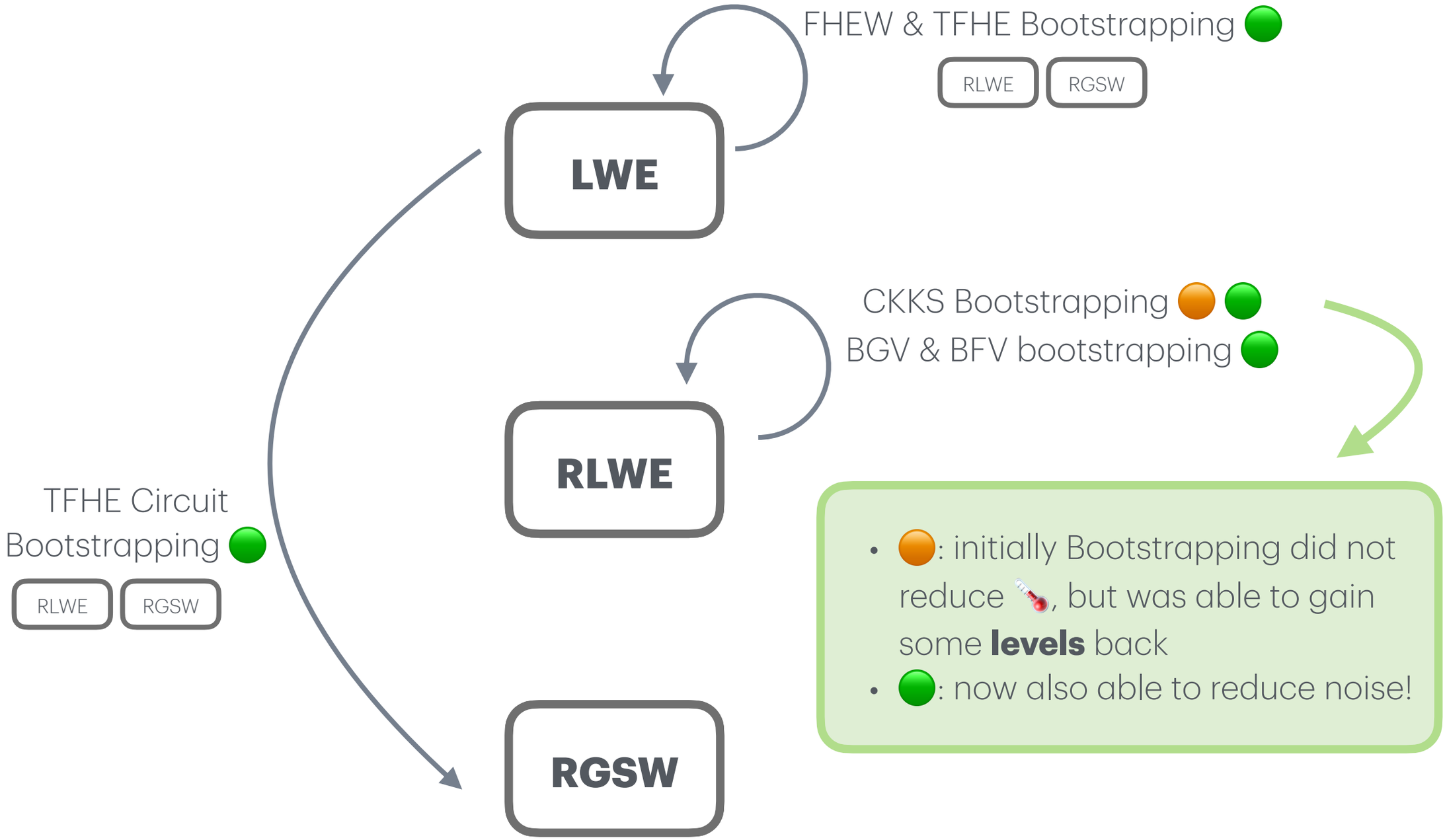
Homomorphic Properties

Bootstrappings



Homomorphic Properties

Bootstrappings



BGV based

BGV, BFV, CKKS/HEAAN

- **Ciphertexts:** RLWE
- **Support ciphertexts:** RLev (key switching)
- **Encoding:** slots and large precision — SIMD
- **Operations:**
  - Fast addition
  - Fast multiplication
  - Slow bootstrapping (reducing noise or getting back some levels) — New generalised Functional Bootstrap

GREAT FOR:

- Applications with multiple inputs
- Small depth circuit evaluations

High latency 🔴 high throughput 🟢

GSW based

GSW, DM/FHEW, TFHE/CGGI

- **Ciphertexts:** LWE, R/GLWE, R/GGSW
- **Support ciphertexts:** RLev (key switching)
- **Encoding:** single messages and small precision
- **Operations:**
  - Fast addition
  - Slow multiplication — Fast External Product
  - Fast bootstrapping (reducing noise and evaluating a LUT)

Large precision with composition of CT

GREAT FOR:

- Applications with single inputs
- Deep or unknown depth circuits

Low latency 🟢 low throughput 🔴

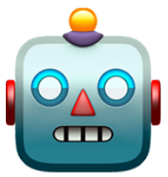
Scheme-switching approach

The best of both worlds combined!

"CHIMERA: combining ring-LWE-based fully homomorphic encryption schemes", C. Boura, N. Gama, M. Georgieva, D. Jetchev, J. Math. Cryptol. 2020

Applications

Real world applications



Machine Learning

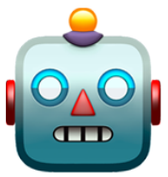
- Mostly Inference: even Deep NN, LLM
- Some results for training

Blockchain

Databases

Others

Real world applications



Machine Learning

- **Mostly Inference: even Deep NN, LLM**
- Some results for training

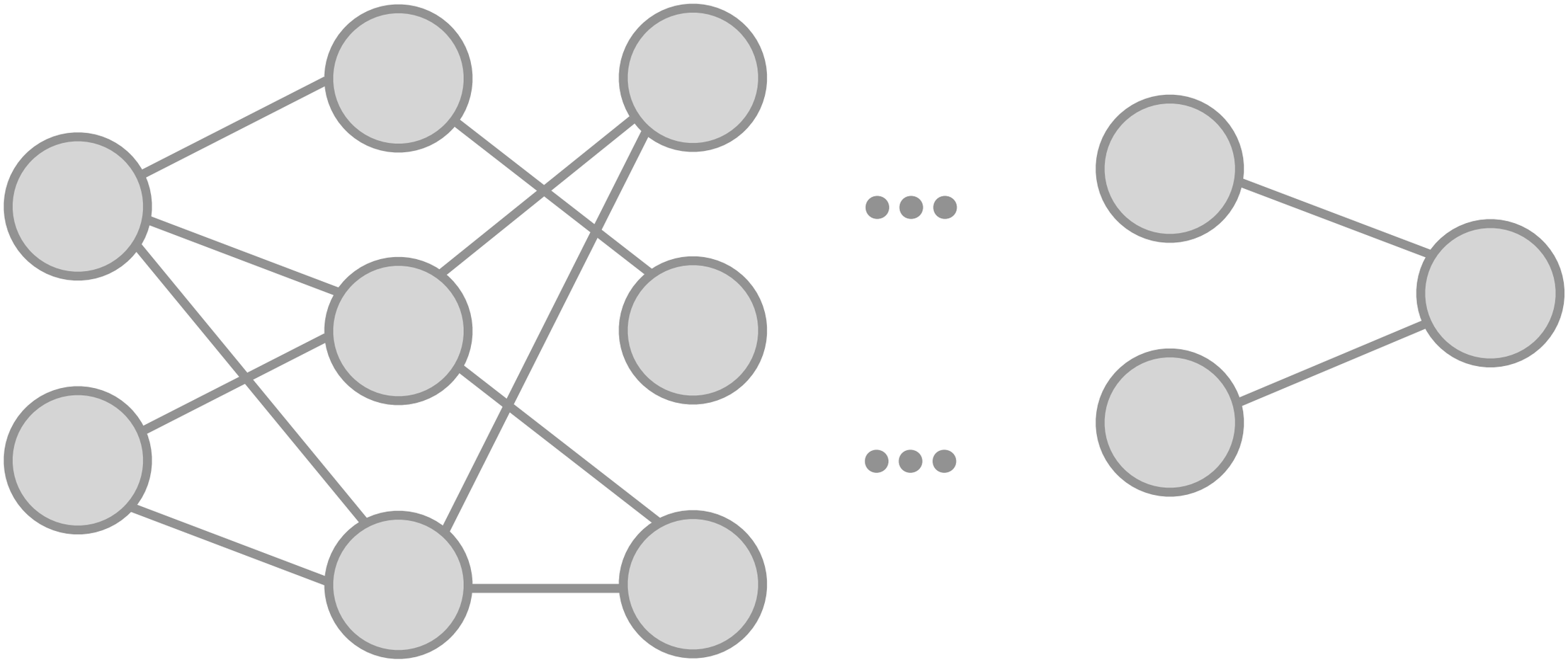
Blockchain

Databases

Others

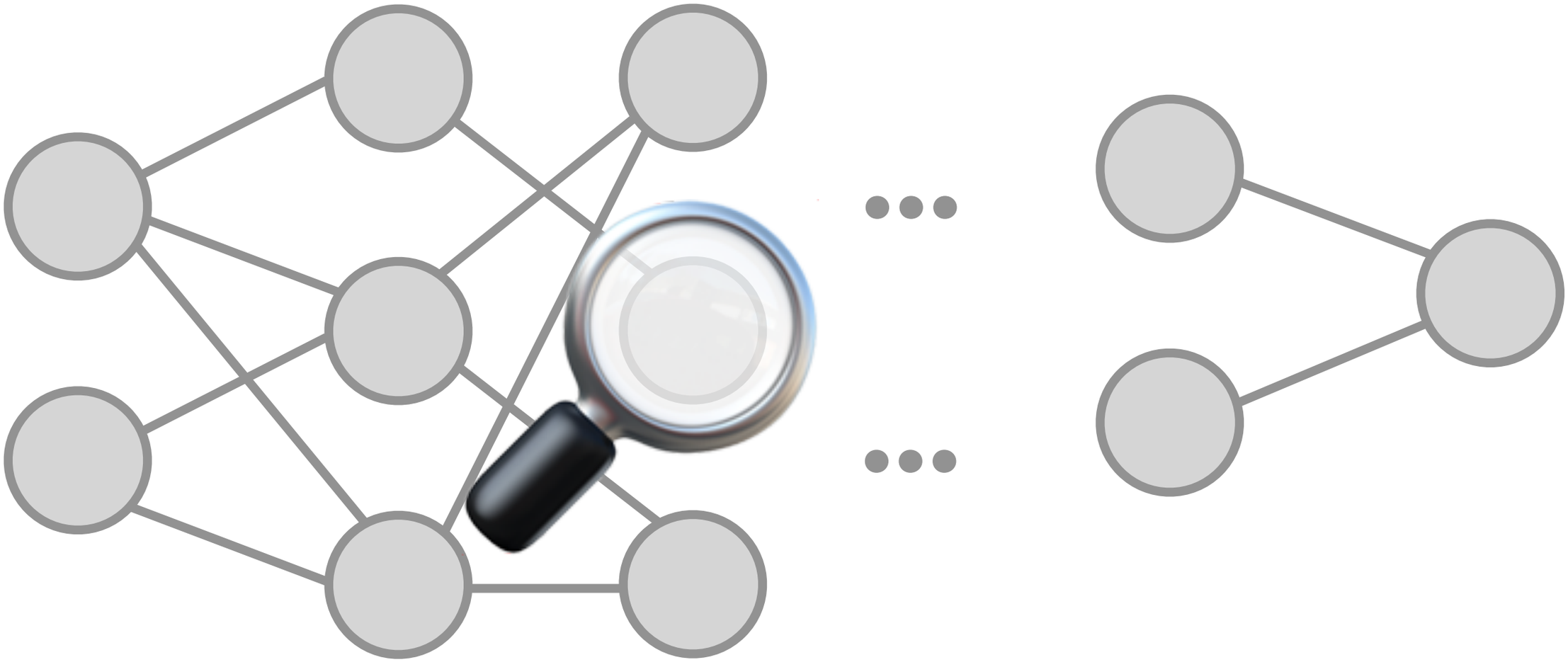
Inference of NN

Neural Networks



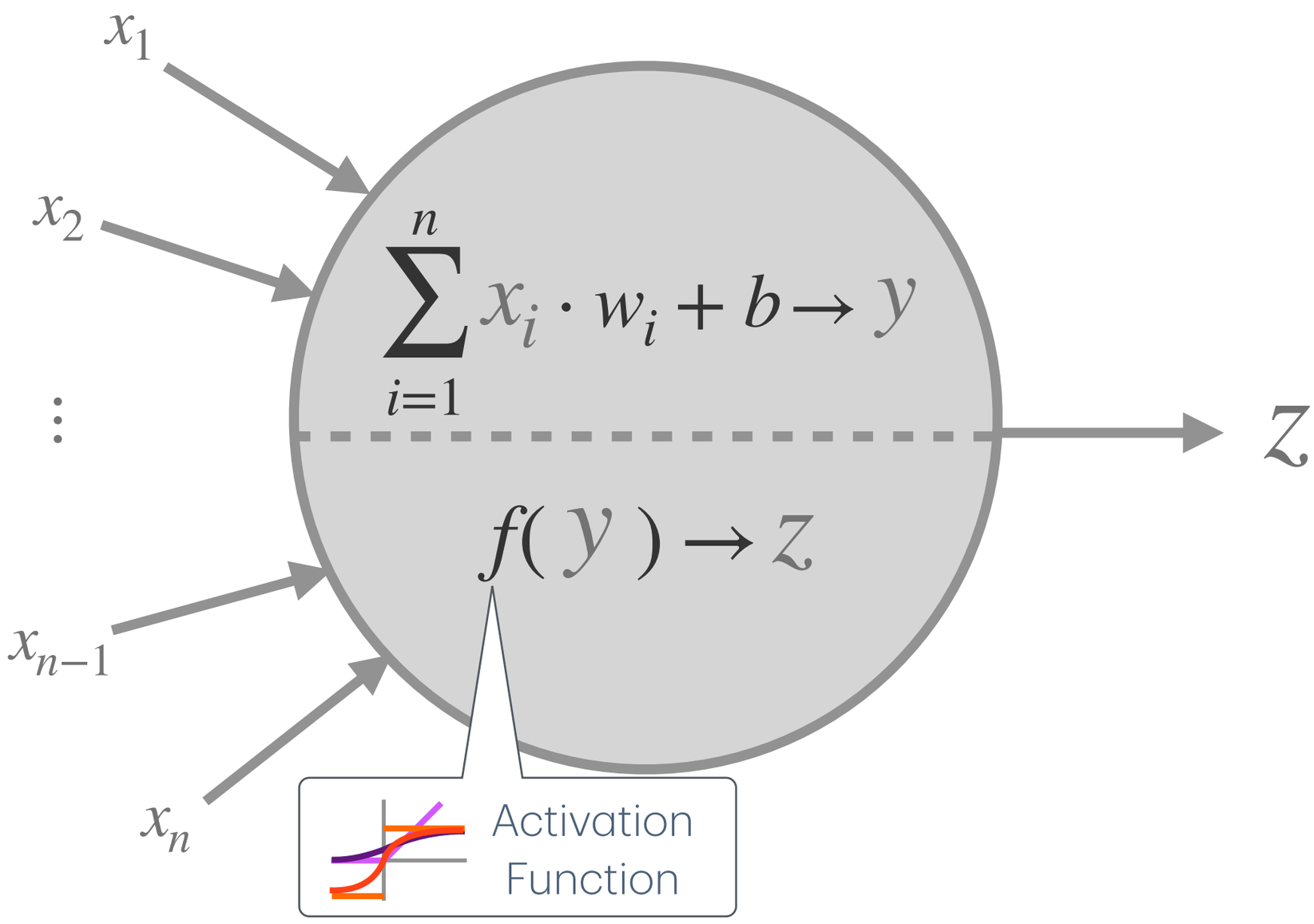
Inference of NN

Neural Networks



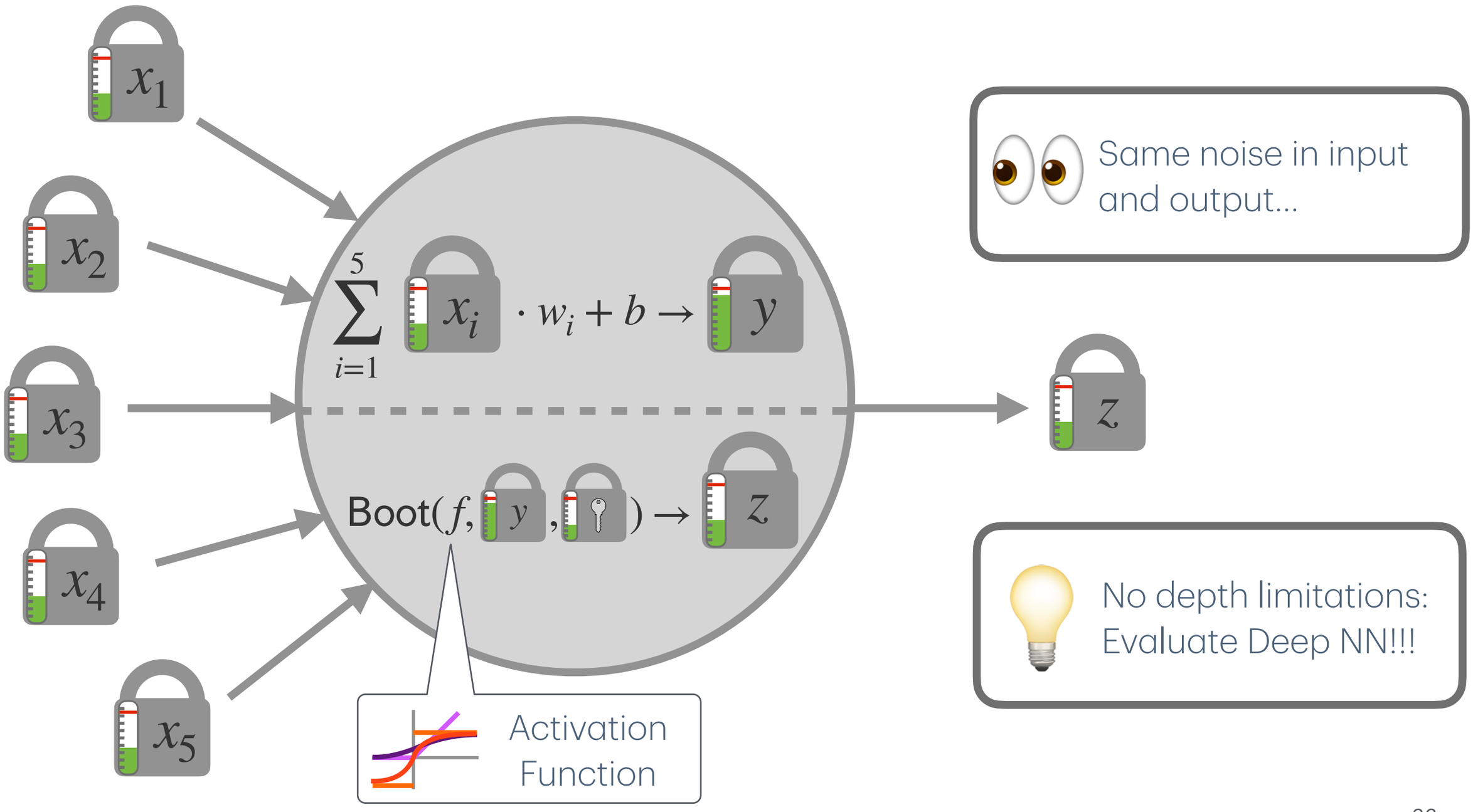
Inference of NN

One active neuron under the magnifying glass



Inference of NN using TFHE

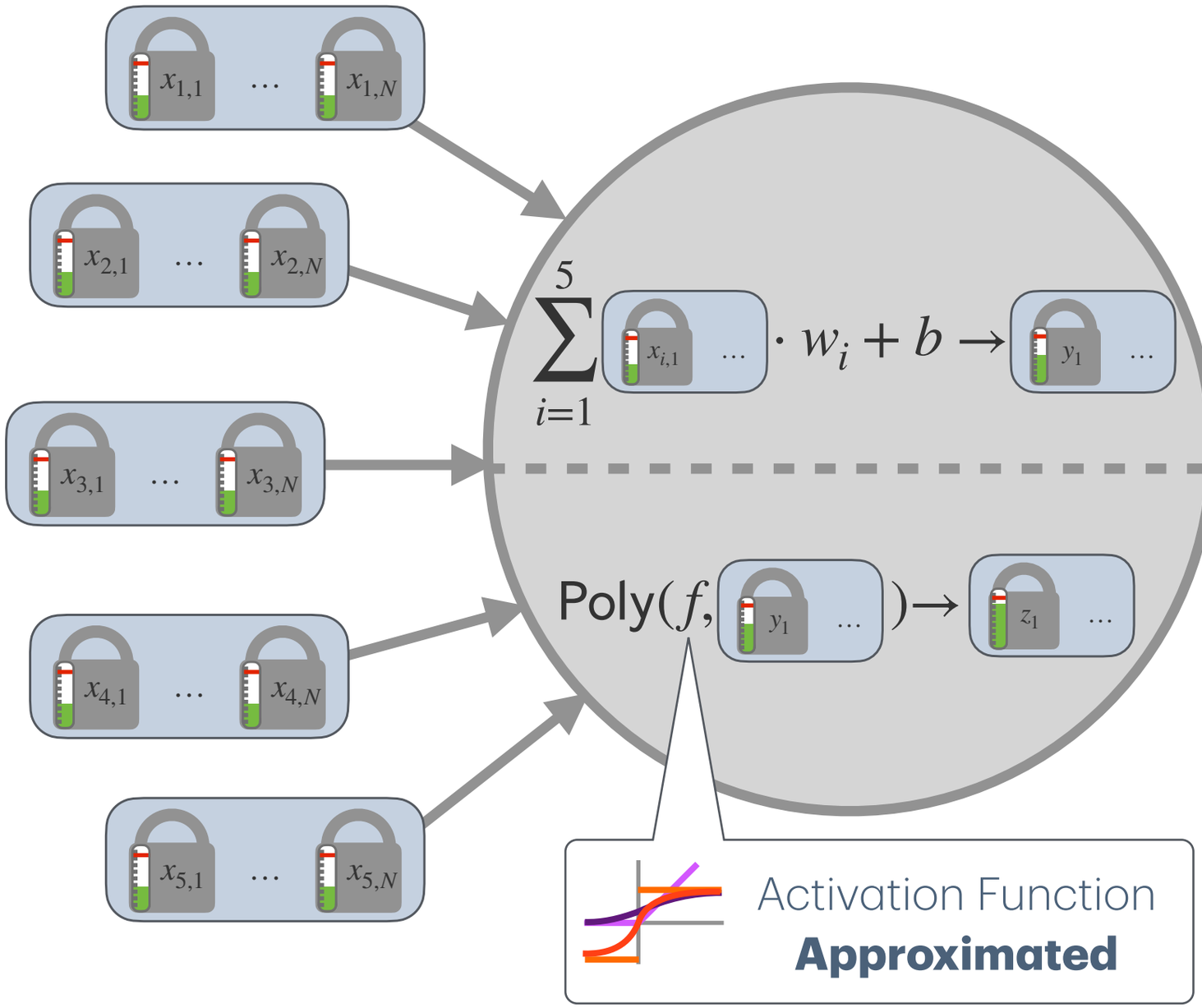
One active neuron under the magnifying glass

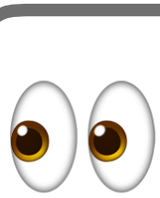


"Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks" I. Chillotti, M. Joye and P. Paillier, CSCML 2021

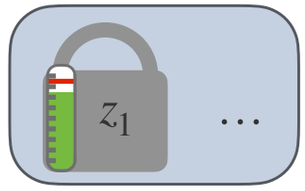
Inference of NN using CKKS

One active neuron under the magnifying glass





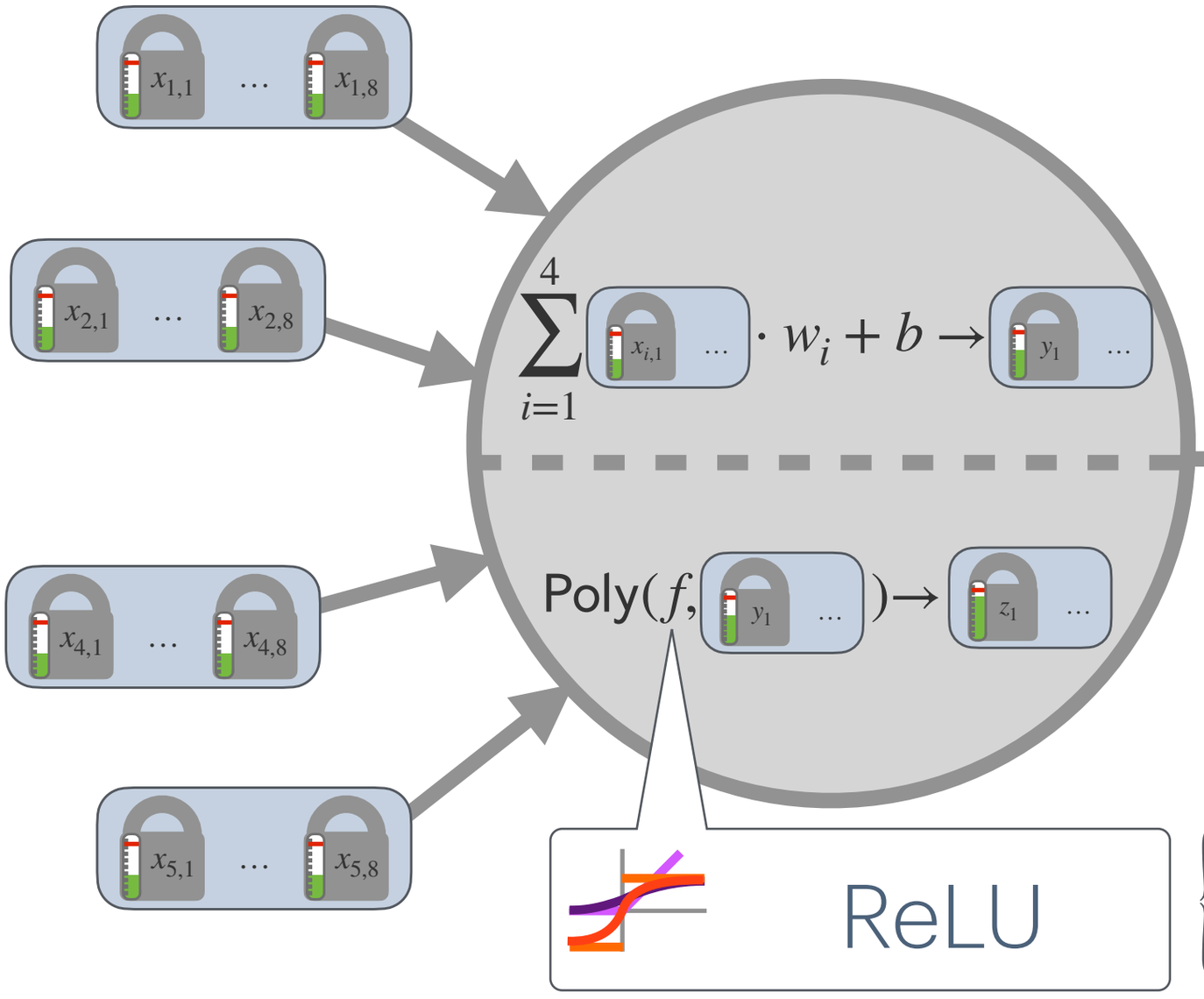
Noise grows 🔺, so bootstrapping at some point!



Fast leveled computations and several inputs evaluated at the same time!

Inference of NN using CKKS

Active neuron demo - DESILO FHE lib



Toy example:

- Pack 8 messages/ctxt
- 4 weighs, 1 bias
- ReLU activation function

DESILO FHE

https://fhe.desilo.dev/latest/

$$\text{ReLU}(y) = \text{Max}(0,y) = \frac{1}{2} \cdot (y + y \cdot \text{sign}(y))$$

Inference of NN using CKKS

Active neuron demo - DESILO FHE lib

Install

```
[1]
✓ 7s
!pip install desilofhe-cu124==1.5.1

Collecting desilofhe-cu124==1.5.1
  Downloading desilofhe_cu124-1.5.1-cp312-cp312-manylinux_2_34_x86_64.whl.metadata (4.4 kB)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from desilofhe-cu124==1.5.1) (2.0.2)
Downloading desilofhe_cu124-1.5.1-cp312-cp312-manylinux_2_34_x86_64.whl (1.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 26.4 MB/s eta 0:00:00
Installing collected packages: desilofhe-cu124
Successfully installed desilofhe-cu124-1.5.1
```

Inference of NN using CKKS

Active neuron demo - DESILO FHE lib

Define the Sign function

[2] ✓ 0s

```
def sign(x, relinearization_key):
    # Polynomial approximation p(x) = p_{7,2}(p_{7,1}(x))
    p71 = [
        3.60471572275560 * 10**-36,
        7.30445164958251,
        -5.05471704202722 * 10**-35,
        -3.46825871108659 * 10,
        1.16564665409095 * 10**-34,
        5.98596518298826 * 10,
        -6.54298492839531 * 10**-35,
        -3.18755225906466 * 10,
    ]
    p72 = [
        -9.46491402344260 * 10**-49,
        2.40085652217597,
        6.41744632725342 * 10**-48,
        -2.63125454261783,
        -7.25338564676814 * 10**-48,
        1.54912674773593,
        2.06916466421812 * 10**-48,
        -3.31172956504304 * 10**-1,
    ]

    # compute y = p_{7,1}(x)
    y = engine.evaluate_polynomial(x, p71, relinearization_key)

    # compute p_{7,2}(p_{7,1}(x))
    return engine.evaluate_polynomial(y, p72, relinearization_key)
```

Inference of NN using CKKS

Active neuron demo - DESILO FHE lib

Define the ReLU function

[3] ✓ 0s

```
def relu(x, relinearization_key):
    # Polynomial approximation p(x) = 0.5 * (x + x * sign(x))

    # Compute sign(x)
    sign_evaluation = sign(x, relinearization_key)

    # compute x * sign(x)
    multiplied = engine.multiply(sign_evaluation, x, relinearization_key)

    # compute x + x * sign(x)
    added = engine.add(multiplied, x)

    # finally compute 0.5 * (x + x * sign(x))
    result = engine.multiply(added, 0.5)
    return result
```

Inference of NN using CKKS

Active neuron demo - DESILO FHE lib

Define the weighted sum

[4] ✓ 0s

```
def inner_product_bias(encrypted_data, weight, bias):
    # initialize y = b
    inner_product = bias
    for encrypted_data, weight in zip(encrypted, weights):
        # compute the weighted sum y += x_i * w_i
        product = engine.multiply(encrypted_data, weight)
        inner_product = engine.add(inner_product, product)
    return inner_product
```

Inference of NN using CKKS

Active neuron demo - DESILO FHE lib

Compute an artificial neuron 🧠

[5]
✓ 1s

```python
from desilofhe import Engine
import math

# engine with more multiplicative levels and a parallel CPU implementation
engine = Engine(max_level=11, mode="gpu")

secret_key = engine.create_secret_key()
public_key = engine.create_public_key(secret_key)
relinearization_key = engine.create_relinearization_key(secret_key)
rotation_key = engine.create_rotation_key(secret_key)

# data stores enough data to evaluate 8 times our neuron in the SIMD fashion (i.e. at once)
# the first input is the first column [0.1, 0.9, 1.5, 0.8], the second is [0.2, 1.0, 0.3, 1.0] and so on.
data = [
    [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8],
    [0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 0.7, 1.6],
    [1.5, 0.3, 0.0, 0.7, 1.1, 1.3, 0.2, 0.8],
    [0.8, 1.0, 1.6, 1.2, 0.3, 0.7, 0.1, 1.1],
]

# encrypt
encrypted = [engine.encrypt(d, public_key) for d in data]

# inner-product between the input data and the weights plus the bias
weights = [-0.4, -1.2, 0.6, 1.0]
bias = 0.34
inner_product = inner_product_bias(encrypted, weights, bias)
# [~0.92 ~0.24 ~0.5 ~0.36 ~-0.46 ~-0.1 ~-0.56 ~-0.32]

# ReLU homomorphic evaluation
neuron_output = relu(inner_product, relinearization_key)

# decrypt and print the result
decrypted = engine.decrypt(neuron_output, secret_key)
print(decrypted[:8])  # [~0.92 ~0.24 ~0.5 ~0.36 ~0.0 ~0.0 ~0.0 ~0.0]
```

```
[ 9.16559206e-01  2.41556095e-01  4.98417598e-01  3.61381529e-01
 -3.03358356e-03  5.35617552e-04  3.91368452e-03 -1.89339383e-03]
```

New Directions

FHE is great… but it cannot be a standalone technology

Many PETs in the Privacy Zoo

Privacy Enhancing Technologies

FHE

Federated Learning

MPC

Anonymisation

Zero-Knowledge Proofs

Differential Privacy

TEE

Functional Encryption

Combining PETs

"L'unione fa la forza" (Unity is strength)

FHE + MPC: Computations with FHE (communication reduced) and threshold decryption

TEE + MPC: parties use TEE to secure their computations

ZKP + FHE: ZK proofs to prove validity of inputs or (parts of) the computations

ZKP + MPC: ZK proofs to create trusted setups for MPC

"Computing On Encrypted Data", N. Smart, IEEE S&P 2023

What's next?

- More research + HW accelerators
- Compilers & Optimisers
- Tools that are user-friendly and easy to deploy
- Security in real-world scenarios
- Combining with other PETs

Thank You!

Questions?

✉ ilaria.chillotti@desilo.ai