Verifiable Computation for Approximate Homomorphic Encryption Schemes

Ignacio Cascudo, Anamaria Costache, **Daniele Cozzo**, Dario Fiore,
Antonio Guimarães, Eduardo Soria-Vazquez

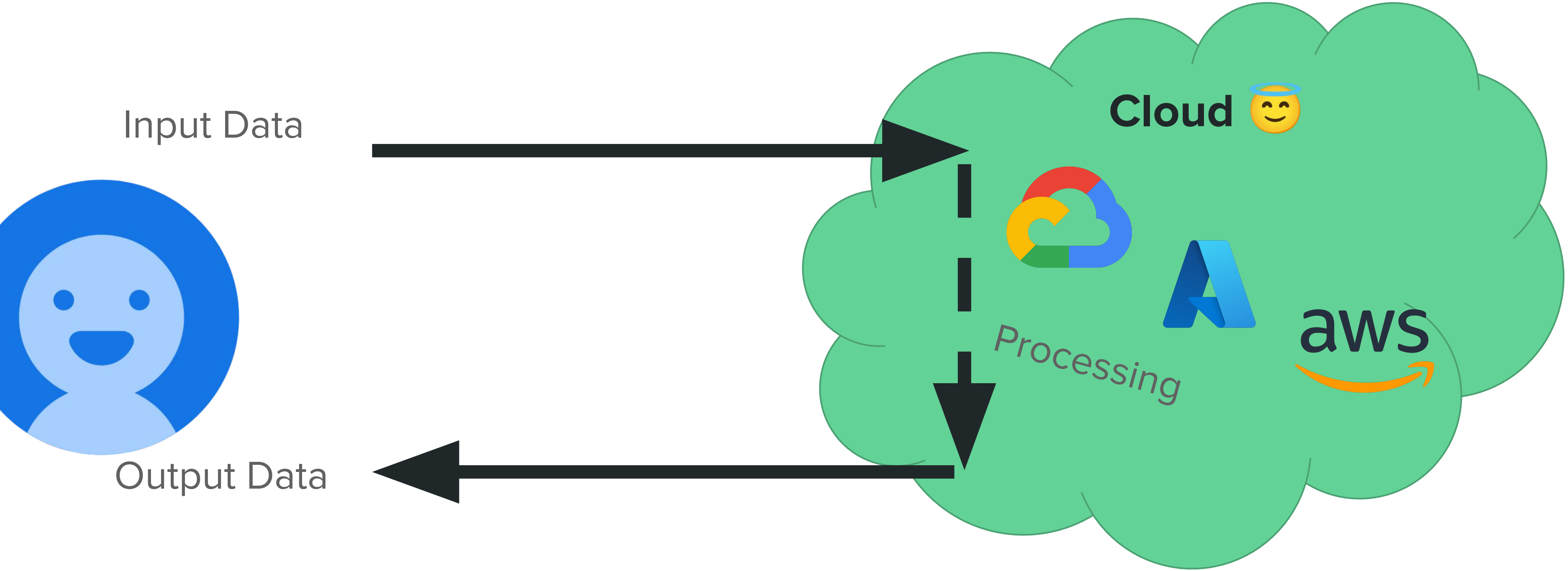
institute IMDEA software

NTNU Norwegian University of Science and Technology

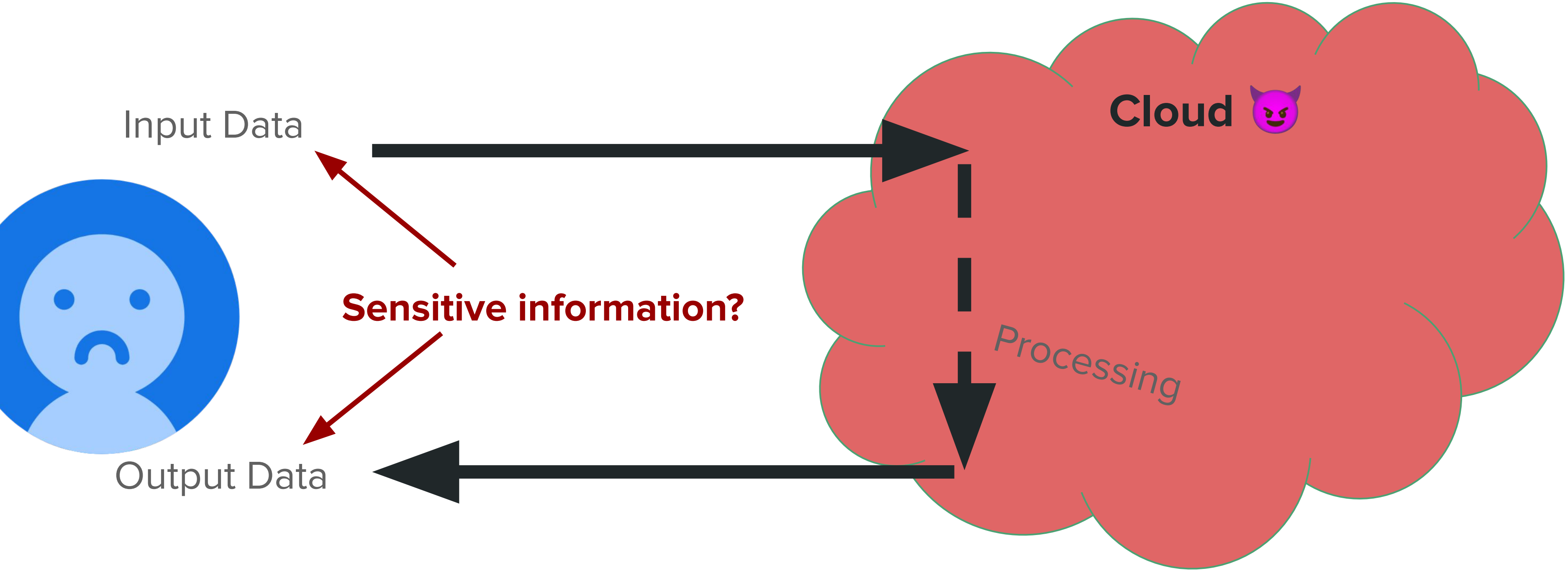
TII Technology Innovation Institute

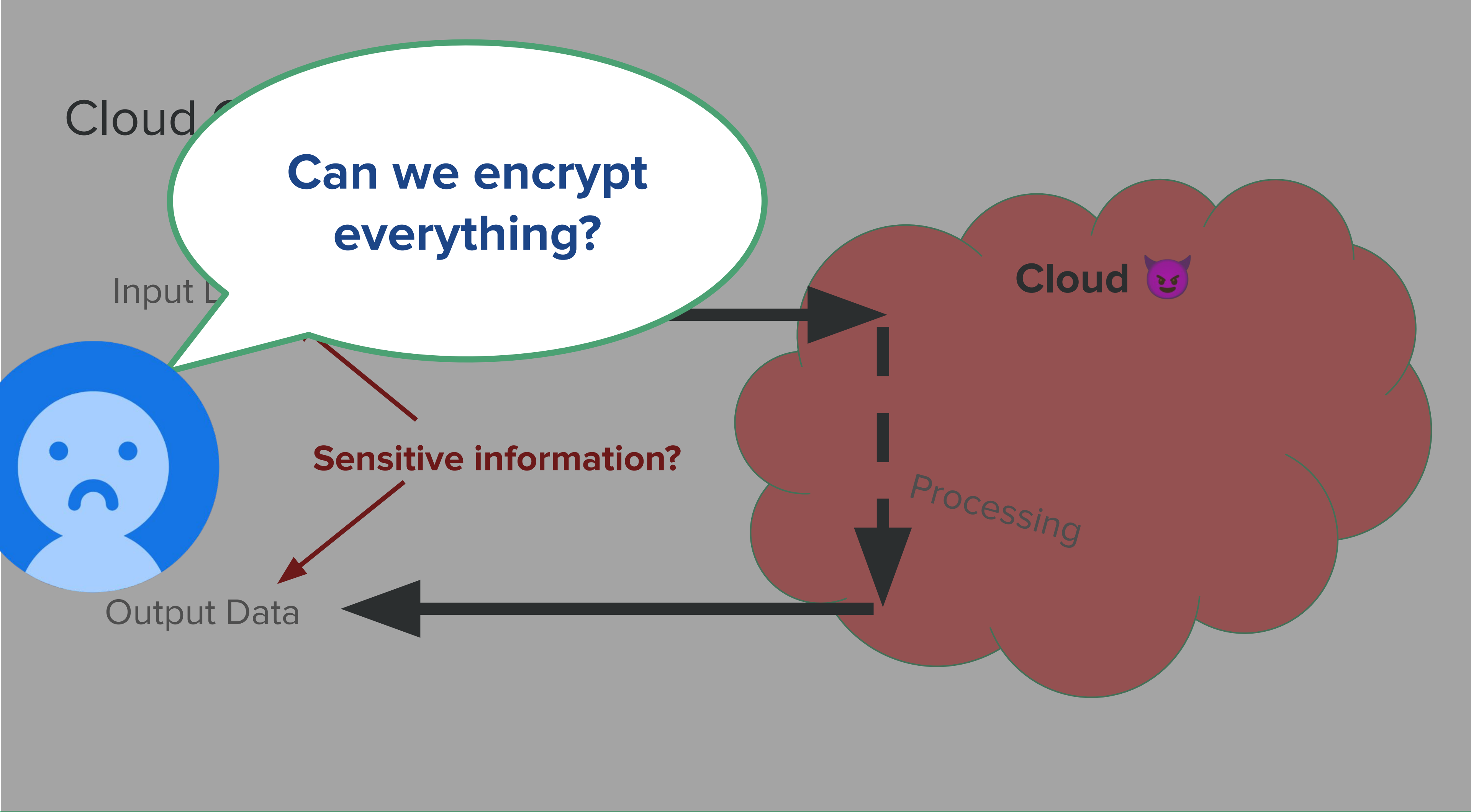
Context

Cloud Computing

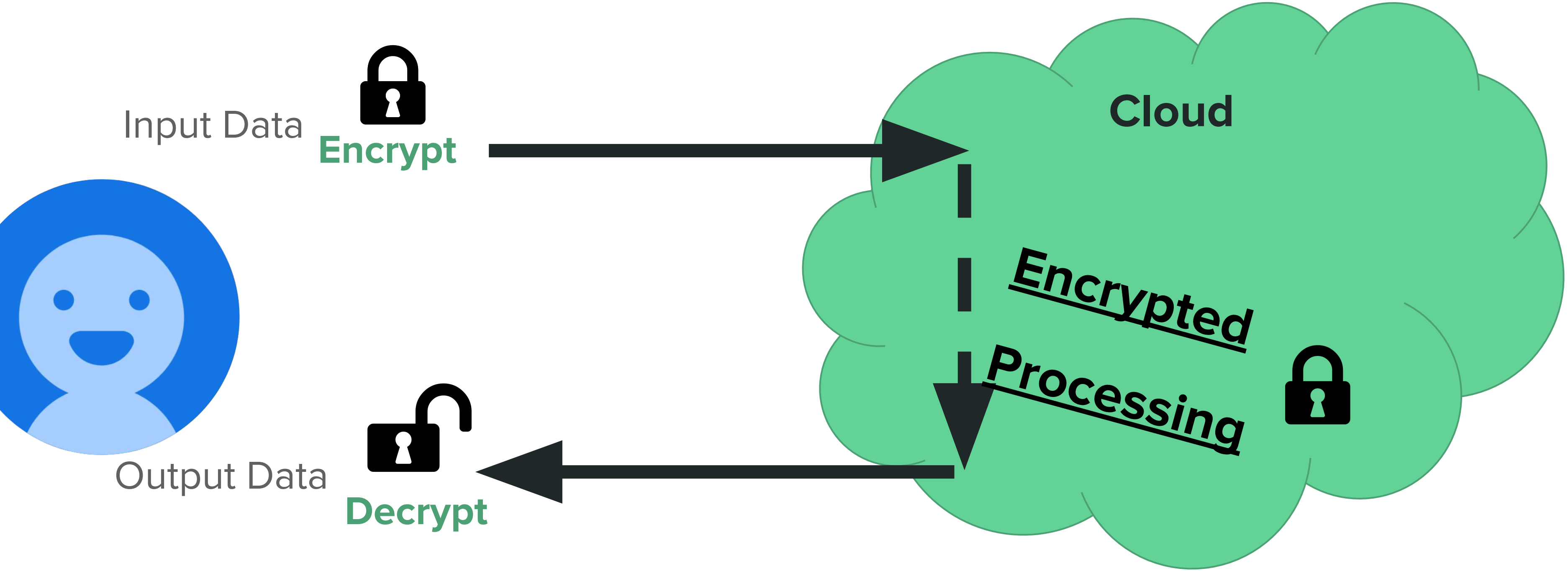


Cloud Computing



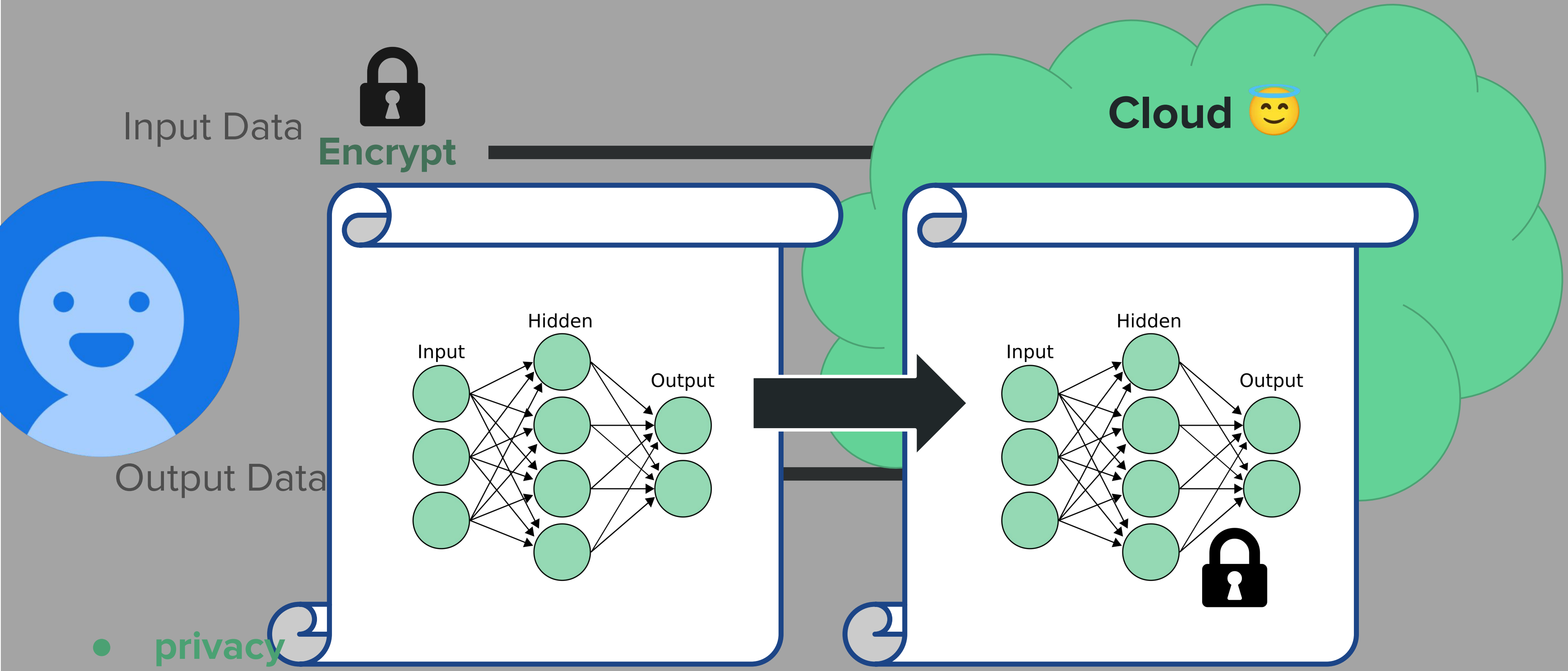

Cloud
Can we encrypt everything?
Input
Cloud 😈
Sensitive information?
Processing
Output Data

Homomorphic Encryption (HE)



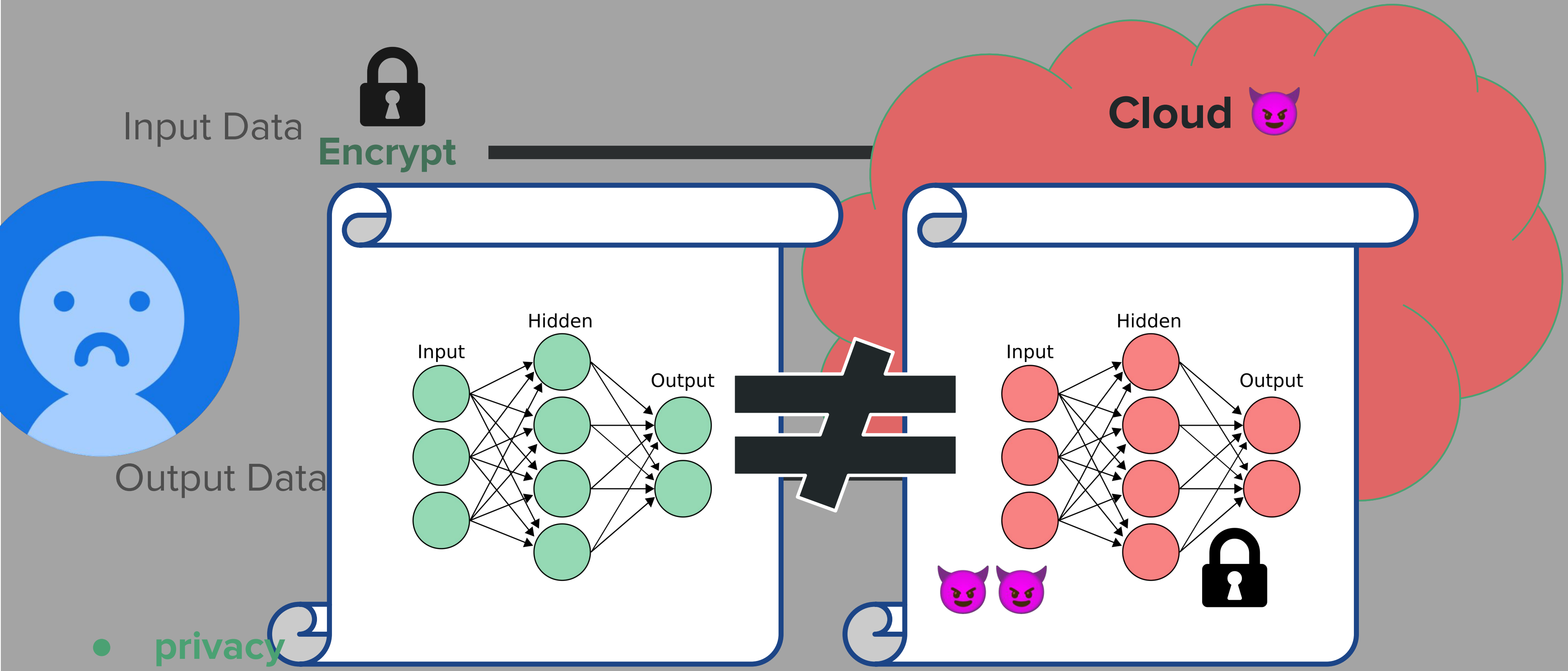
- **privacy**

Homomorphic Encryption (HE)

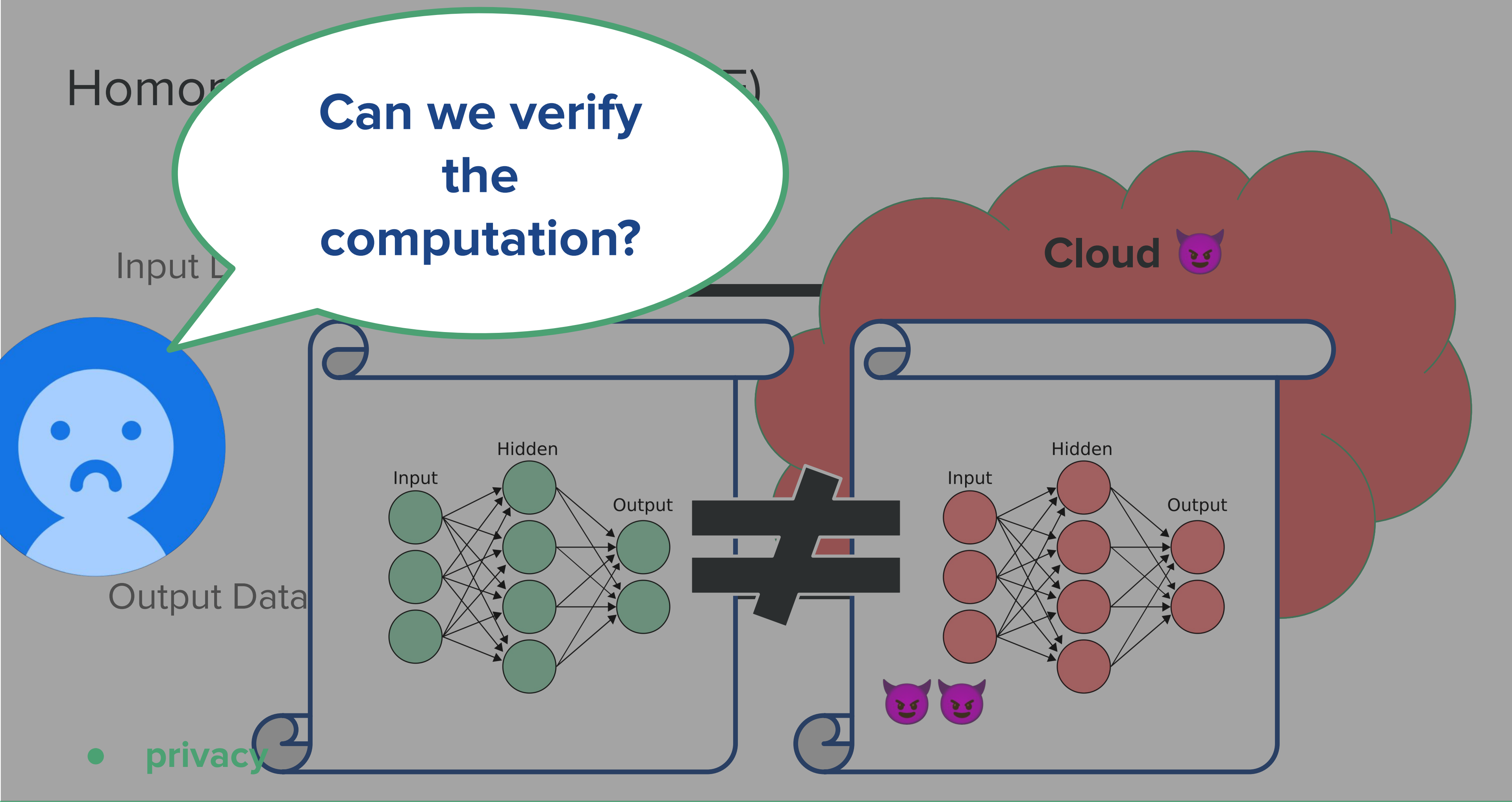


- privacy

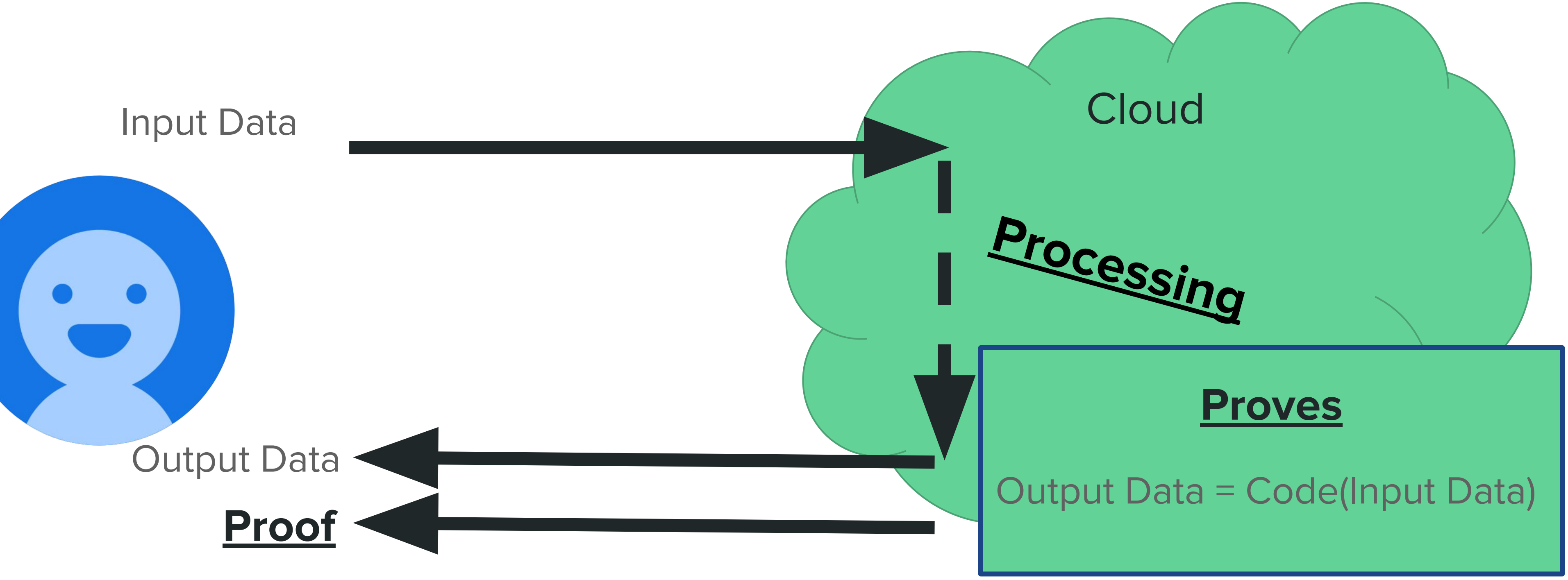
Homomorphic Encryption (HE)



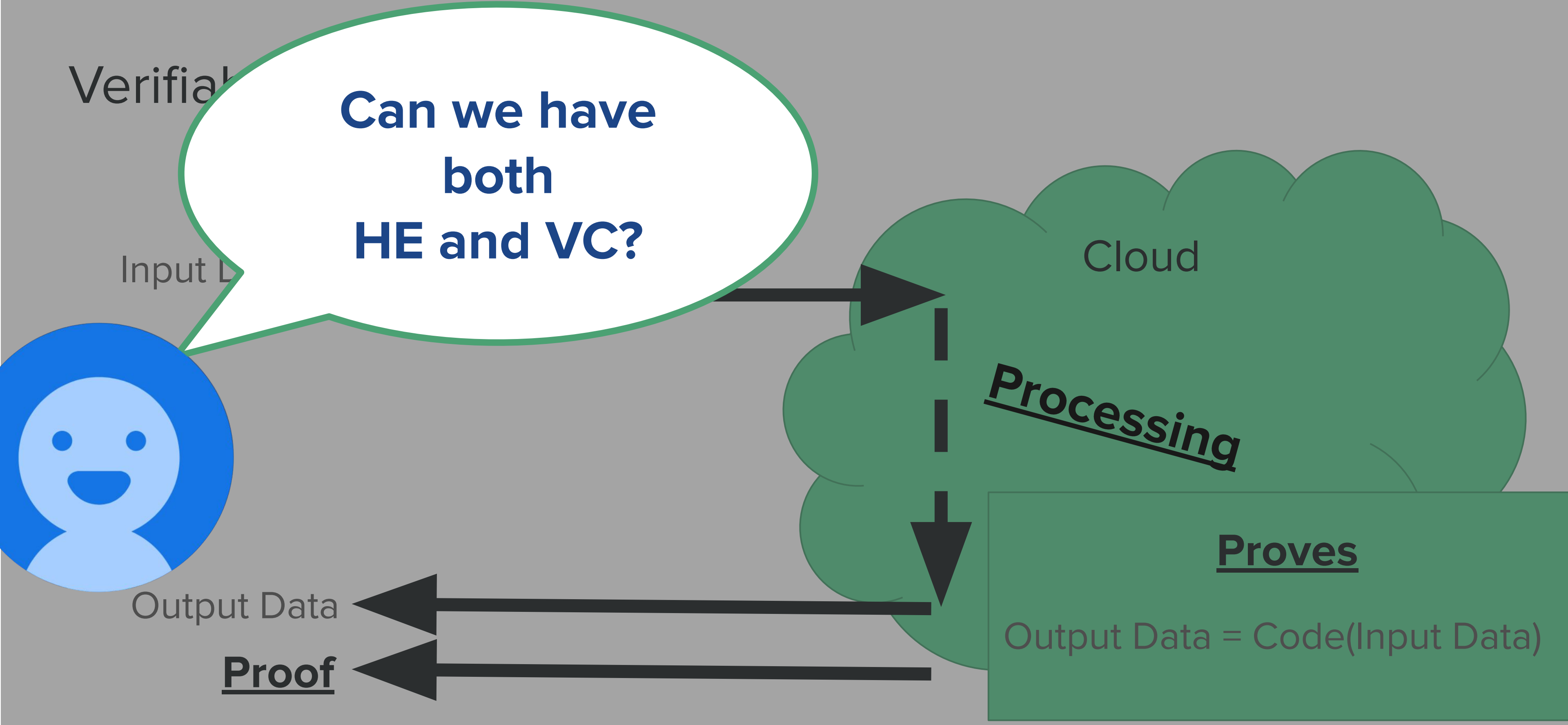
- privacy


Homor
E)
Input D
Output Data
privacy
Can we verify the computation?
Cloud
Input
Hidden
Output
Input
Hidden
Output

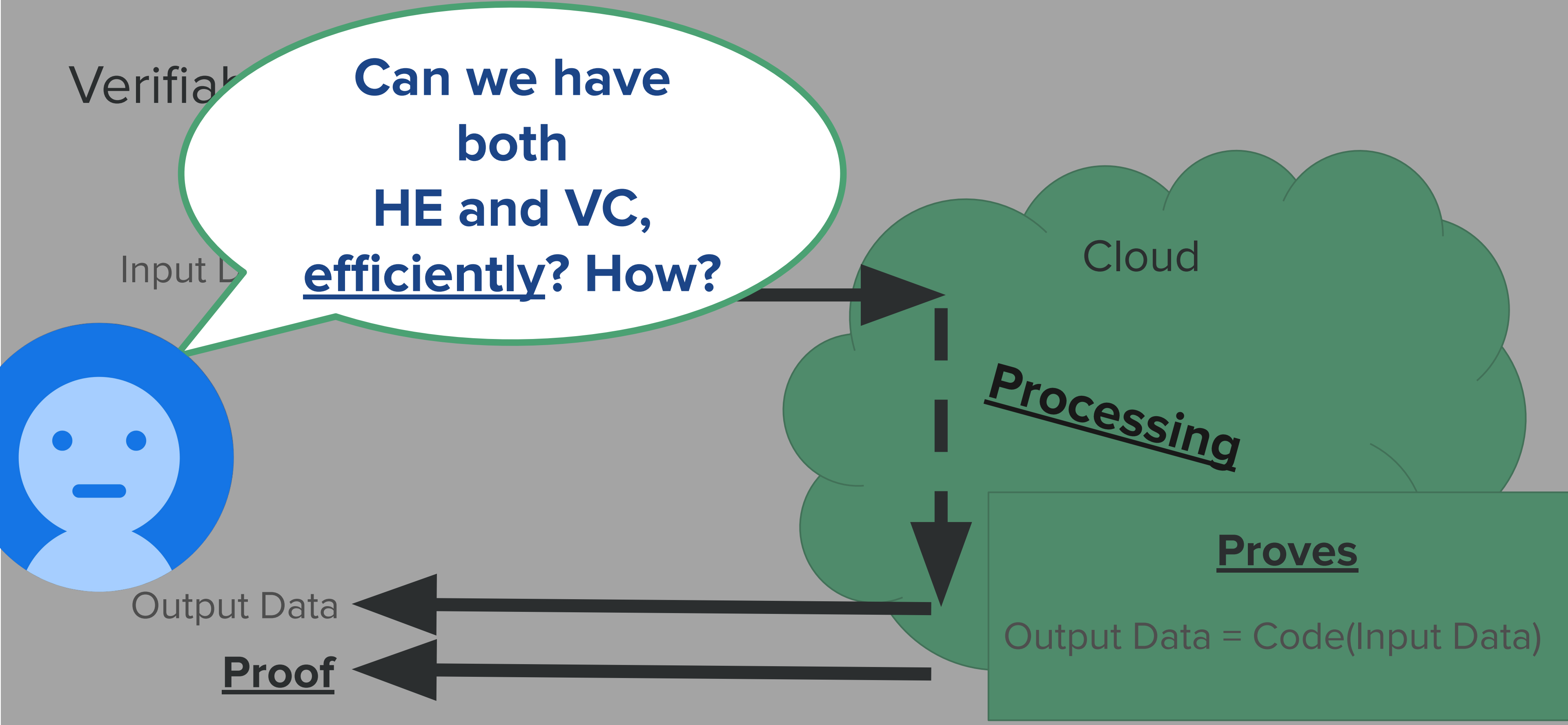
Verifiable computation (VC)



- **Integrity**
- **Succinctness**



- Privacy
- Integrity
- Succinctness



- Privacy
- Integrity
- Succinctness

vHE

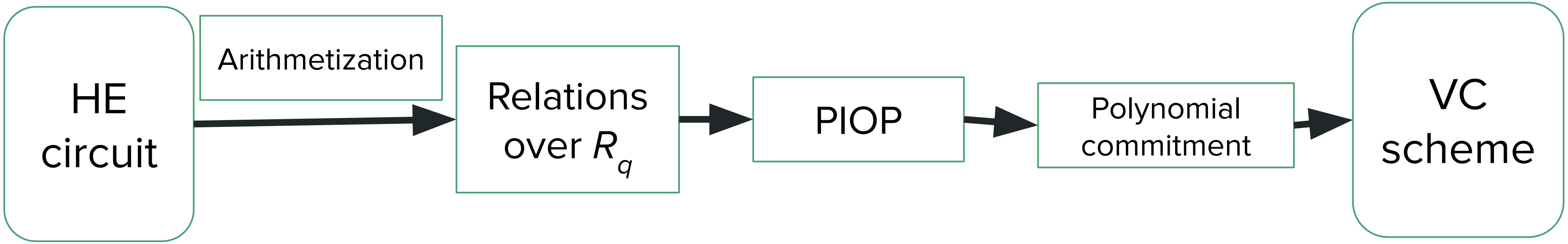
vHE

	Native R_q Arithmetic	Efficient Key Switching / Rescale	Efficient Bootstrapping	Public Verification	CKKS (approximate schemes)
Generic SNARK[1]	❌	❌	❌	✓	✓
Rinocchio[2]	✓	❌	❌	❌	✓
HE-IOPs[3]	✓	✓	✓	❌	❌
Our Work	✓	✓	?	✓	✓

[1] A. Viand, C. Knabenhans, and A. Hithnawi, "*Verifiable Fully Homomorphic Encryption*" arXiv:2301.07041
[2] C. Ganesh, A. Nitulescu, and E. Soria-Vazquez, "*Rinocchio: SNARKs for Ring Arithmetic*" Journal of Cryptology, 2023
[3] D. F. Aranha, A. Costache, A. Guimarães, and E. Soria-Vazquez, "*HELIOPOLIS: Verifiable Computation over Homomorphically Encrypted Data from Interactive Oracle Proofs is Practical*" ASIACRYPT 2024

Our contributions

- vHE for **CKKS**
- **Modular** solution



- **Proof-friendly CKKS**
- **AC satisfiability**
- **Range check**
- **Custom GKR**
- **Range proofs for polynomial rings**
- **Improved Brakedown**

Setting up the ring

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$q \approx 2^{300} \qquad N \approx 2^{14}$$

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$R_q$$

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$R_q$$

- Efficient HE computations
  - RNS

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$R_q$$

- Efficient HE computations
  - RNS
- Soundness
  - Large exceptional set

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$q = \prod_{i=1}^{L} p_i$$

$$R_q \cong \begin{matrix} R_{p_1} \\ R_{p_2} \\ R_{p_3} \end{matrix}$$

- Efficient HE computations
  - RNS
- Soundness
  - Large exceptional set

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$R_q \underset{q = \prod_{i=1}^{L} p_i}{\cong} R_{p_1} \times R_{p_2} \times R_{p_3}$$

$$R_{p_1} \underset{X^N + 1 = \prod_{i=1}^{k}(X^d - \zeta^{2i-1}) \mod p_1}{\cong} R_{11} \; R_{12} \; R_{13} \; R_{14}$$

$$R_{p_2} \underset{X^N + 1 = \prod_{i=1}^{k}(X^d - \zeta^{2i-1}) \mod p_2}{\cong} R_{21} \; R_{22} \; R_{23} \; R_{24}$$

$$R_{p_3} \underset{X^N + 1 = \prod_{i=1}^{k}(X^d - \zeta^{2i-1}) \mod p_3}{\cong} R_{31} \; R_{32} \; R_{33} \; R_{34}$$

- Efficient HE computations
  - RNS
- Soundness
  - Large exceptional set

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$R_q \cong R_{p_1} \times R_{p_2} \times R_{p_3}, \quad q = \prod_{i=1}^{L} p_i$$

$$X^N + 1 = \prod_{i=1}^{k} (X^d - \zeta^{2i-1}) \mod p_1 \qquad R_{p_1} \cong \mathbb{F}_{p_1^d} \times \mathbb{F}_{p_1^d} \times \mathbb{F}_{p_1^d} \times \mathbb{F}_{p_1^d}$$

$$X^N + 1 = \prod_{i=1}^{k} (X^d - \zeta^{2i-1}) \mod p_2 \qquad R_{p_2} \cong \mathbb{F}_{p_2^d} \times \mathbb{F}_{p_2^d} \times \mathbb{F}_{p_2^d} \times \mathbb{F}_{p_2^d}$$

$$X^N + 1 = \prod_{i=1}^{k} (X^d - \zeta^{2i-1}) \mod p_3 \qquad R_{p_3} \cong \mathbb{F}_{p_3^d} \times \mathbb{F}_{p_3^d} \times \mathbb{F}_{p_3^d} \times \mathbb{F}_{p_3^d}$$

- Efficient HE computations
  - RNS
- Soundness
  - Large exceptional set

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$q = \prod_{i=1}^{L} p_i$$

$$R_q \cong R_{p_1},\ R_{p_2},\ R_{p_3}$$

$$X^N + 1 = \prod_{i=1}^{k} (X^d - \zeta^{2i-1}) \mod p_1$$

$$\cong \quad \mathbb{F}_{p_1^d}\ \mathbb{F}_{p_1^d}\ \mathbb{F}_{p_1^d}\ \mathbb{F}_{p_1^d}$$

X^N … $\mathbb{F}_{p_2^d}$

X^N … $\mathbb{F}_{p_3^d}$

Size of our exceptional set!

- Efficient HE computations
  - RNS
- Soundness
  - Large exceptional set

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

$$R_q \quad q = \prod_{i=1}^{L} p_i \quad \cong \quad R_{p_1},\ R_{p_2},\ R_{p_3}$$

$$X^N + 1 = \prod_{i=1}^{k} (X^d - \zeta^{2i-1}) \mod p_1$$

$$\cong \quad \mathbb{F}_{p_1^d}\ \mathbb{F}_{p_1^d}\ \mathbb{F}_{p_1^d}\ \mathbb{F}_{p_1^d}$$

X^N

$\mathbb{F}_{p_2^d}$

X^N

$\mathbb{F}_{p_3^d}$

Size of our exceptional set!

Optimal performance/security:
$|p_i| = 49$, $d = 4$

- Efficient HE computations
  - RNS
- Soundness
  - Large exceptional set

The polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$

R_q

$q = \prod_{i=1}^{L}$

$\cong$

$\mathbb{F}_{p_1^d}$ $\mathbb{F}_{p_1^d}$ $\mathbb{F}_{p_1^d}$

$\mathbb{F}_{p_2^d}$

$\mathbb{F}_{p_3^d}$

al set!

curity:

- Efficient HE comp
  - RNS
- Soundness
  - Large exceptional set

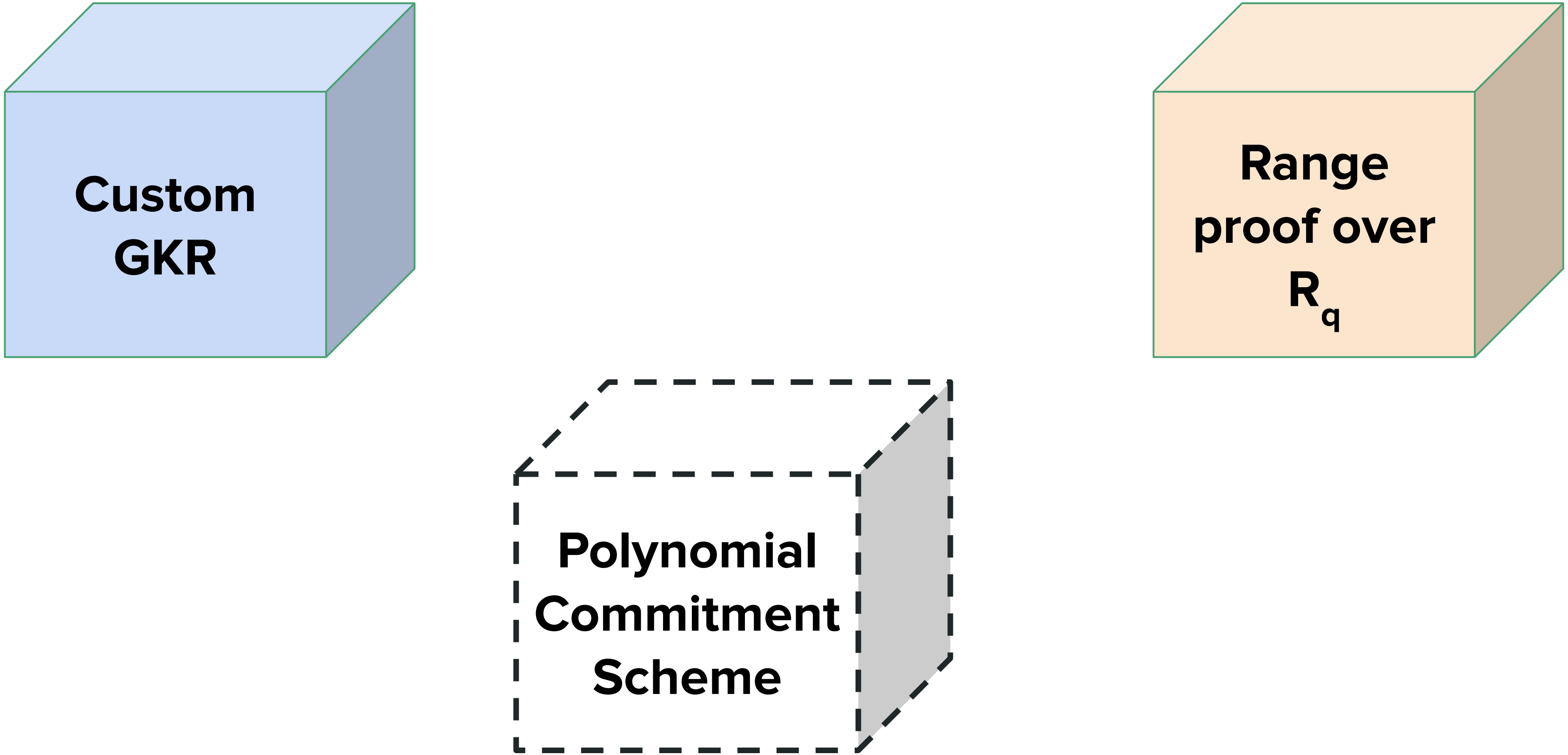
Efficient arithmetic for almost-fully-splitting rings:

- Incomplete NTTs[1]
- Cost:
  - d = 2 -> ~5%
  - d = 4 -> 20%

[1] V. Lyubashevsky and G. Seiler, "NTTRU: Truly Fast NTRU Using NTT," IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 180–201, May 2019, doi: 10.13154/tches.v2019.i3.180-201.

Proof-friendly CKKS

Proof components



CKKS

- An approximate scheme:

- RLWE ciphertext:

$$(a_0, a_1) \in R_q^2$$

- RNS representation (with e.g. 3 components):

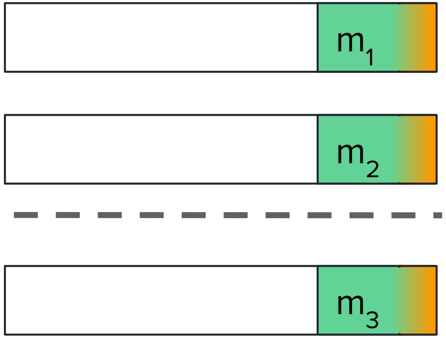
CKKS Addition

$a = (a_{01}, a_{02}, a_{03}) \; (a_{11}, a_{12}, a_{13})$

$+$

$b = (b_{01}, b_{02}, b_{03}) \; (b_{11}, b_{12}, b_{13})$

$c = (c_{01}, c_{02}, c_{03}) \; (c_{11}, c_{12}, c_{13})$



$m_3 = m_1 + m_2$

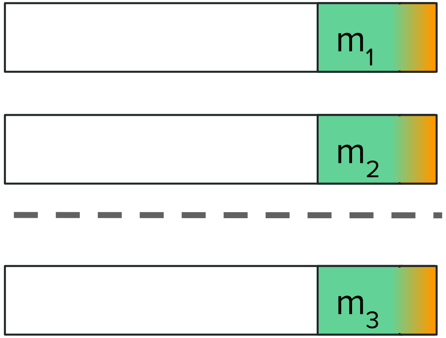
CKKS Addition

$a = [a_{01}\ a_{02}\ a_{03}]\ [a_{11}\ a_{12}\ a_{13}]$

$+$

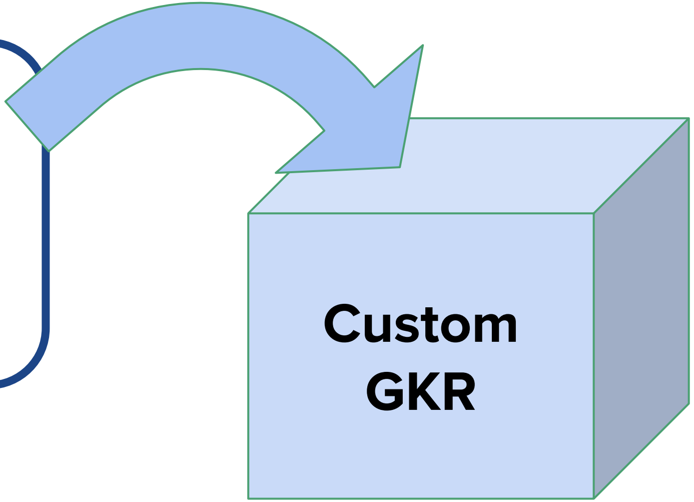
$b = [b_{01}\ b_{02}\ b_{03}]\ [b_{11}\ b_{12}\ b_{13}]$

$c = [c_{01}\ c_{02}\ c_{03}]\ [c_{11}\ c_{12}\ c_{13}]$



$m_3 = m_1 + m_2$

Pure arithmetic!



CKKS Multiplication *Level 1*

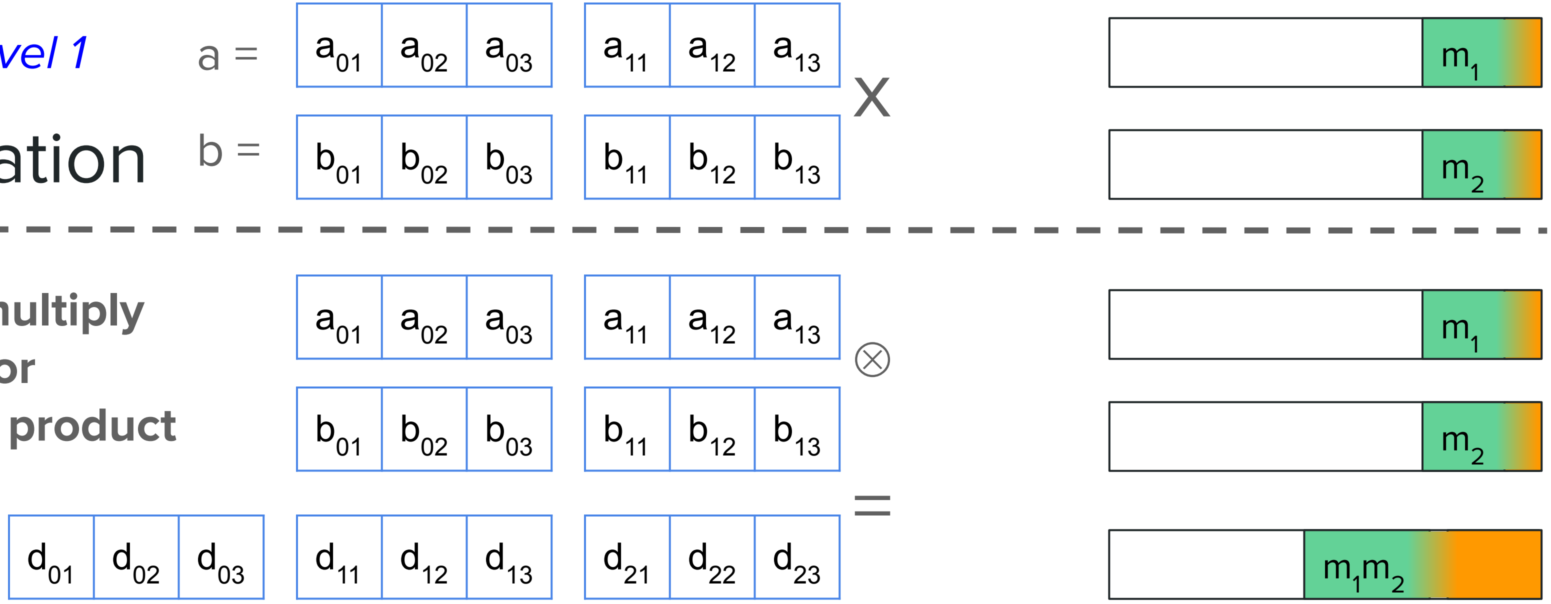
$a = [a_{01} \; a_{02} \; a_{03}] \; [a_{11} \; a_{12} \; a_{13}]$

x

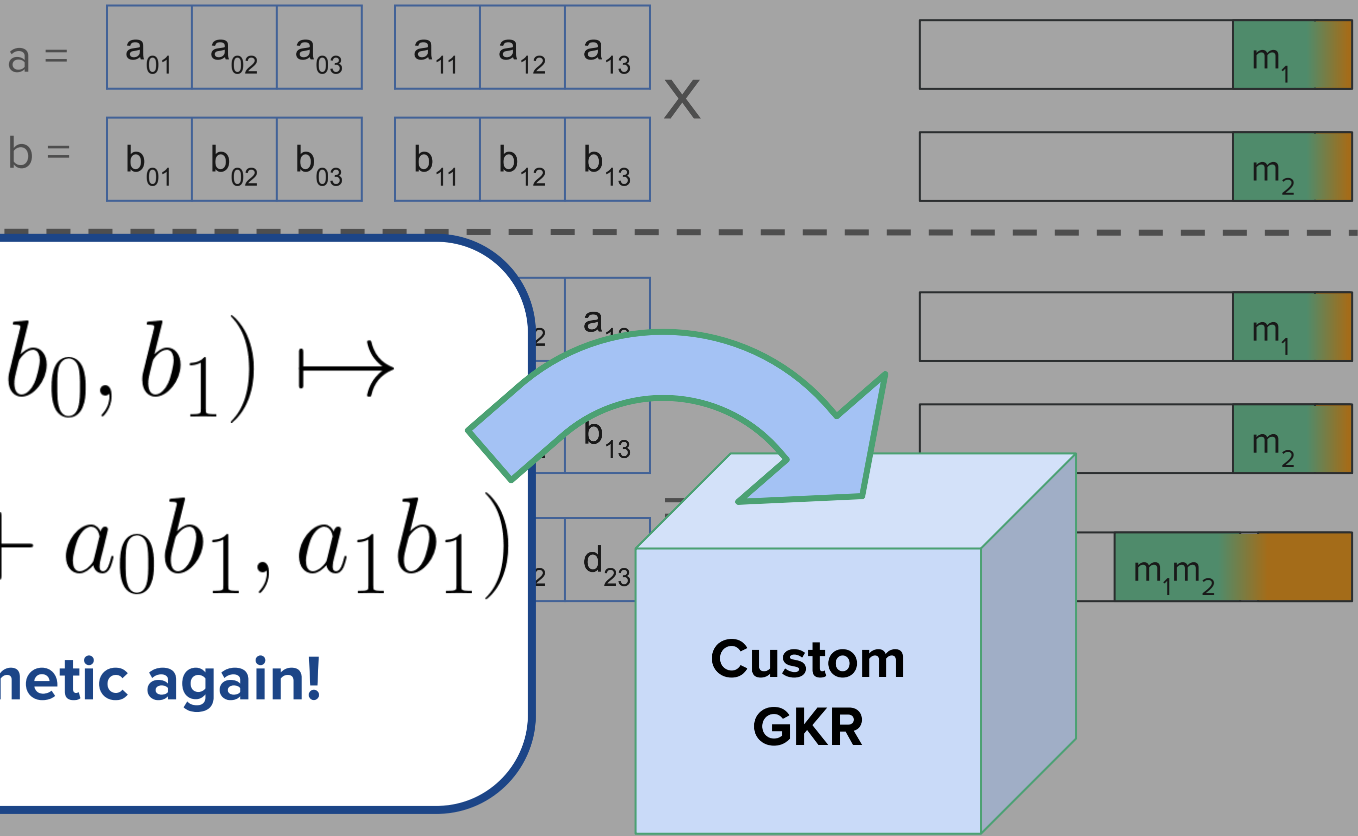
$b = [b_{01} \; b_{02} \; b_{03}] \; [b_{11} \; b_{12} \; b_{13}]$

CKKS *Level 1* Multiplication

Pre-multiply or Tensor product



CKKS *Level 1* Multiplication



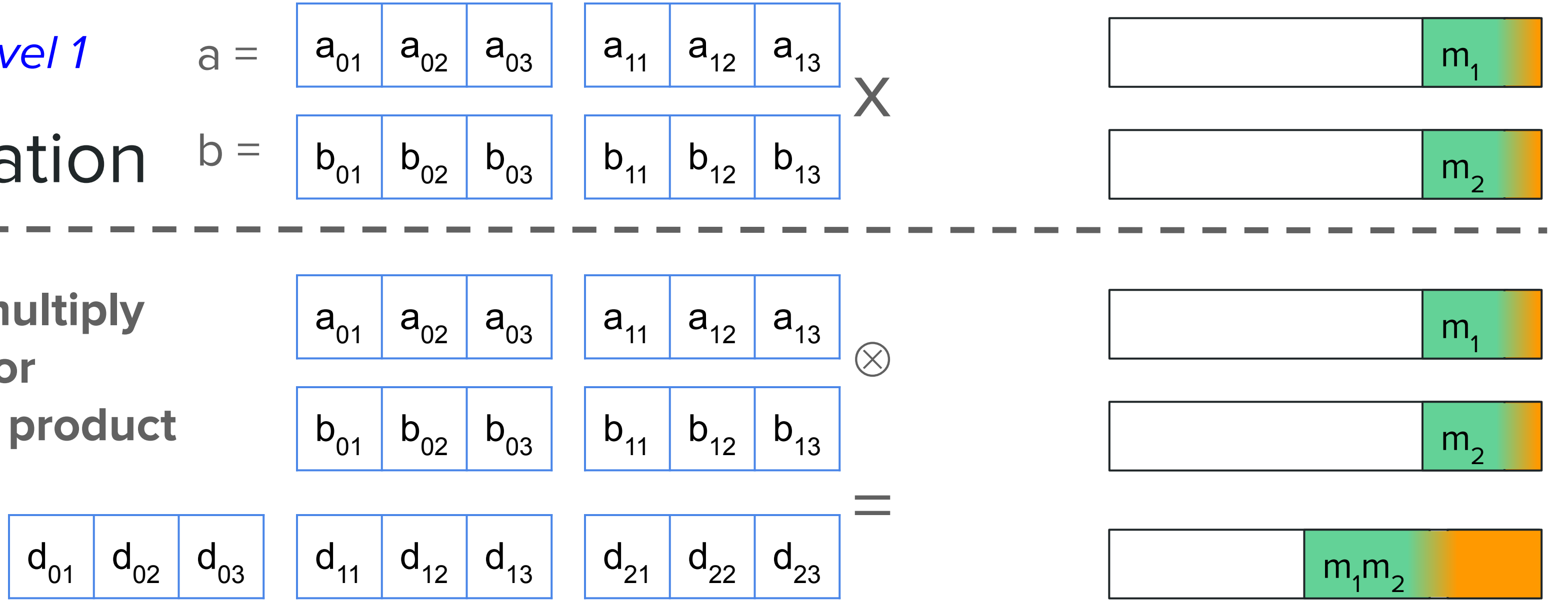
$$(a_0, a_1) \otimes (b_0, b_1) \mapsto$$
$$(a_0 b_0, a_1 b_0 + a_0 b_1, a_1 b_1)$$

Pure arithmetic again!

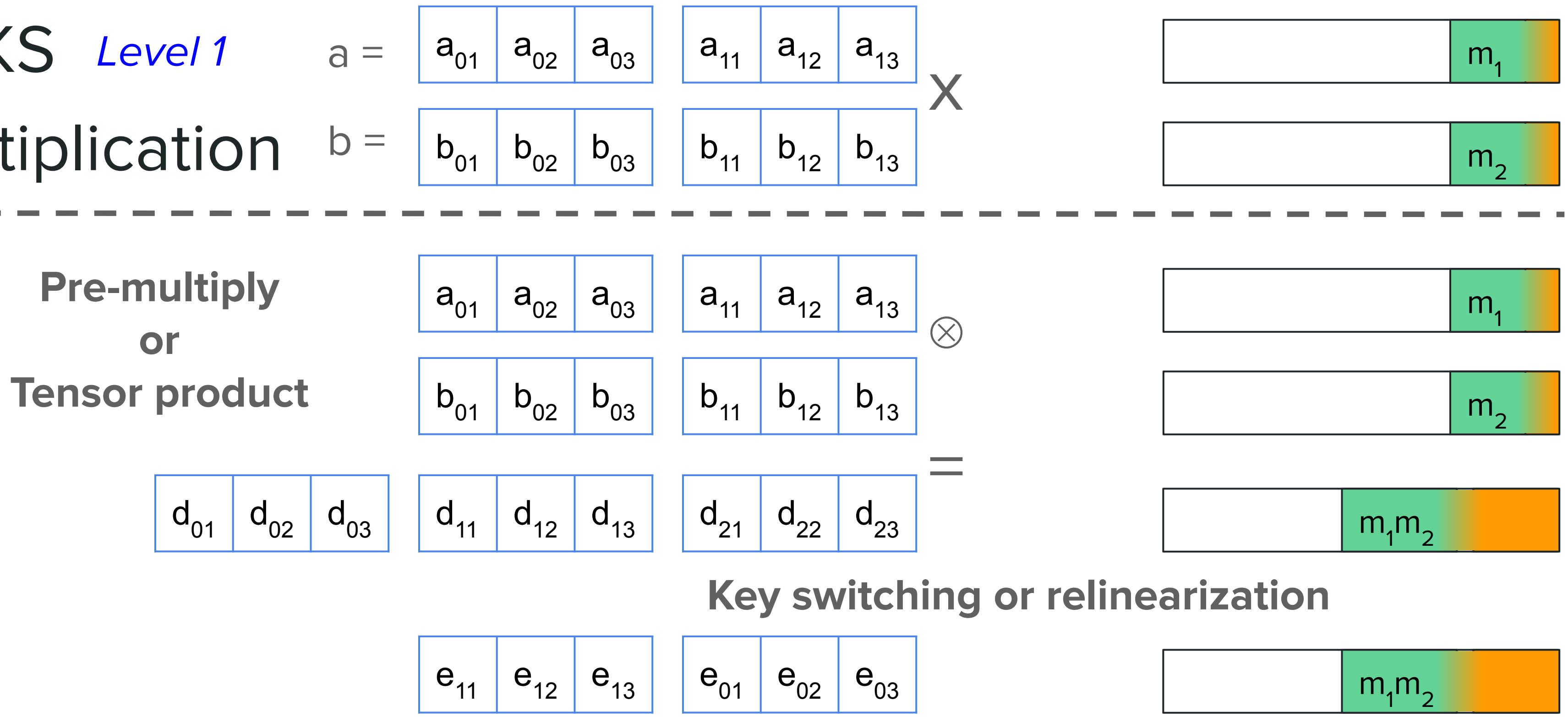
CKKS Multiplication

Level 1

Pre-multiply or Tensor product



CKKS *Level 1* Multiplication



CKKS *Level 1*

Multi

$a = $ | a_{01} | a_{02} | a_{03} | | a_{11} | a_{12} | a_{13} | × | m_1
b_{11} | b_{12} | b_{13} | | m_2

Key Switching:

d_0, d_1, d_2

$$e_0 = d_0 + \langle \text{evk}, CRT^{-1}(d_2) \rangle$$
$$e_1 = d_1 + \langle \text{evk}, CRT^{-1}(d_2) \rangle$$

e_0, e_1

a_{11}	a_{12}	a_{13}	m_1

$\otimes$

b_{11}	b_{12}	b_{13}	m_2

=

d_{21}	d_{22}	d_{23}	$m_1 m_2$

Key switching or relinearization

e_{01}	e_{02}	e_{03}	$m_1 m_2$

CKKS *Level 1*

Multi…

a = a_{01} a_{02} a_{03} | a_{11} a_{12} a_{13} — m_1

X

b_{11} b_{12} b_{13} — m_2

=

d_{21} d_{22} d_{23} — m_1m_2

Key switching or relinearization

e_{01} e_{02} e_{03} — m_1m_2

Key Switching:

d_0 d_1 d_2

$$e_0 = d_0 + \langle \text{evk}, CRT^{-1}(d_2) \rangle$$

$$e_1 = d_1 + \langle \text{evk}, CRT^{-1}(d_2) \rangle$$

e_0 e_1

Not algebraic!

We don't verify:

Inputs: d_0, d_1, d_2

$$e_0 = d_0 + \langle \text{evk}, CRT^{-1}(d_2)\rangle$$
$$e_1 = d_1 + \langle \text{evk}, CRT^{-1}(d_2)\rangle$$

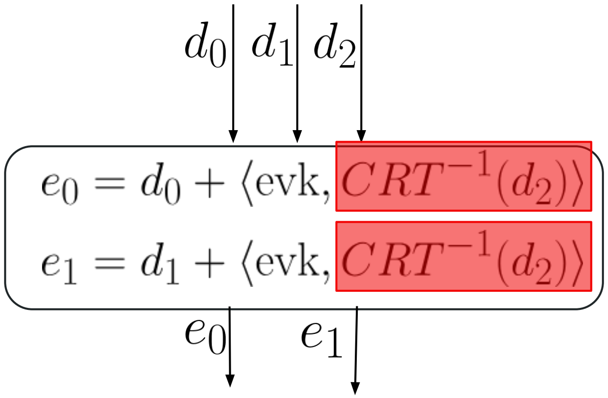
Outputs: e_0, e_1

Instead, we:

1. Ask the prover to provide $\mathbf{w} = CRT^{-1}(d_2)$
2. Check:
   a. $d_2 - CRT(\mathbf{w}) = 0$
   b. $\|\mathbf{w}[i]\| < p_i$
3. Compute and check:

$$e_0 = d_0 + \langle evk, \mathbf{w}\rangle$$
$$e_1 = d_1 + \langle evk, \mathbf{w}\rangle$$

We don't verify:



Instead, we:

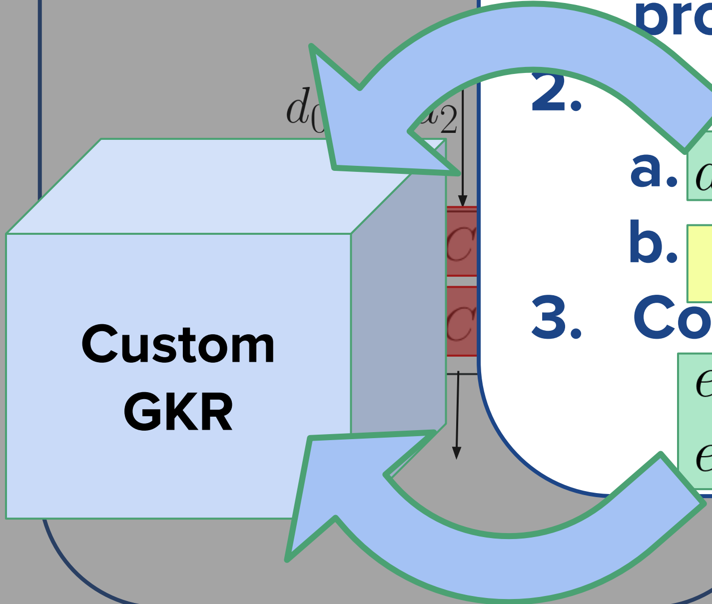
1. Ask the prover to provide $\mathbf{w} = CRT^{-1}(d_2)$
2. Check:
   a. $d_2 - CRT(\mathbf{w}) = 0$
   b. $\|\mathbf{w}[i]\| < p_i$
3. Compute and check:

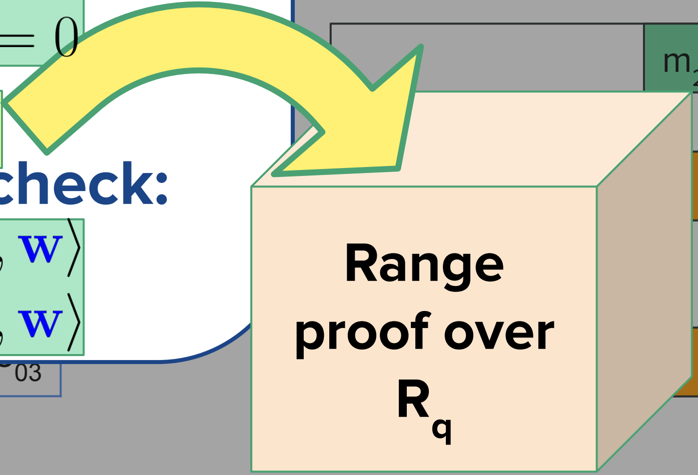
$$e_0 = d_0 + \langle evk, \mathbf{w} \rangle$$
$$e_1 = d_1 + \langle evk, \mathbf{w} \rangle$$

Instead, we:

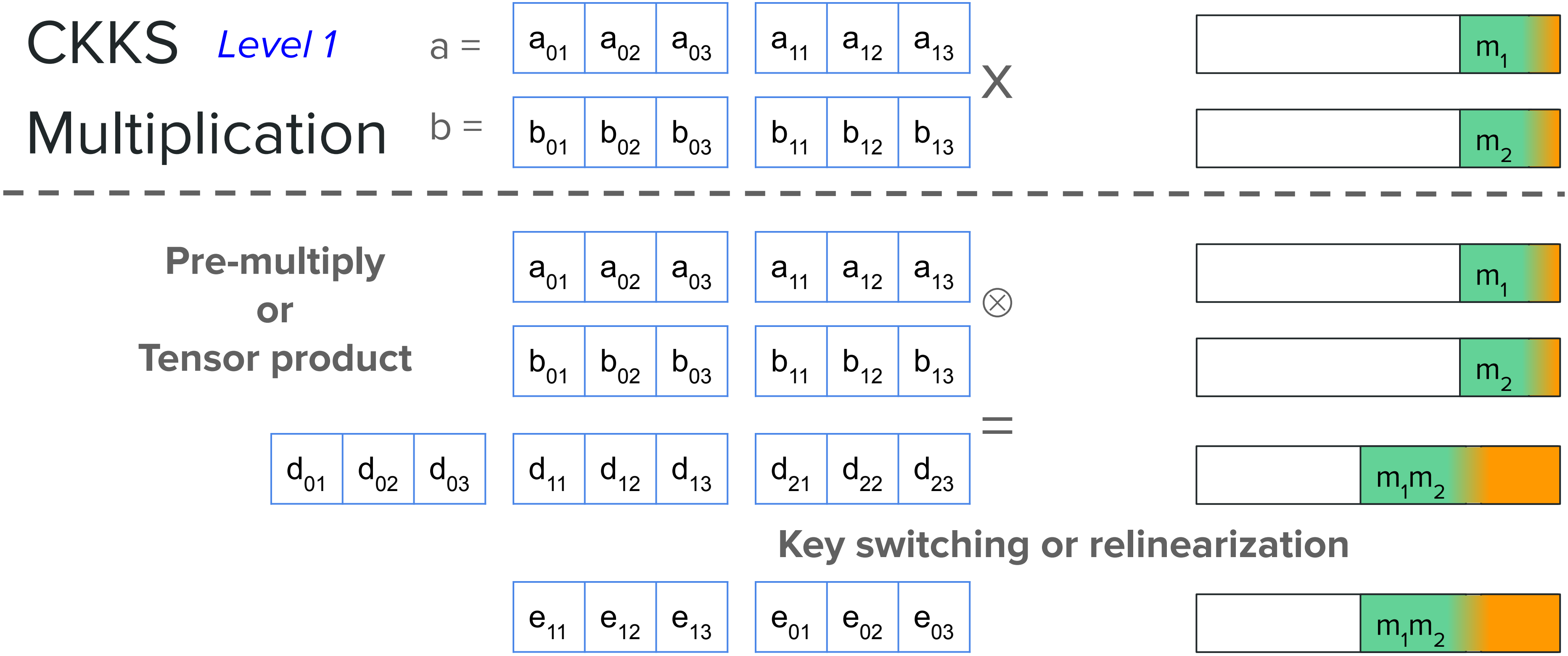
1. Ask the prover to provide **w** = $CRT^{-1}(d_2)$
2. Check:
   a. $d_2 - CRT(\mathbf{w}) = 0$
   b. $||\mathbf{w}[i]|| < p_i$
3. Compute and check:

$$e_0 = d_0 + \langle evk, \mathbf{w} \rangle$$
$$e_1 = d_1 + \langle evk, \mathbf{w} \rangle$$

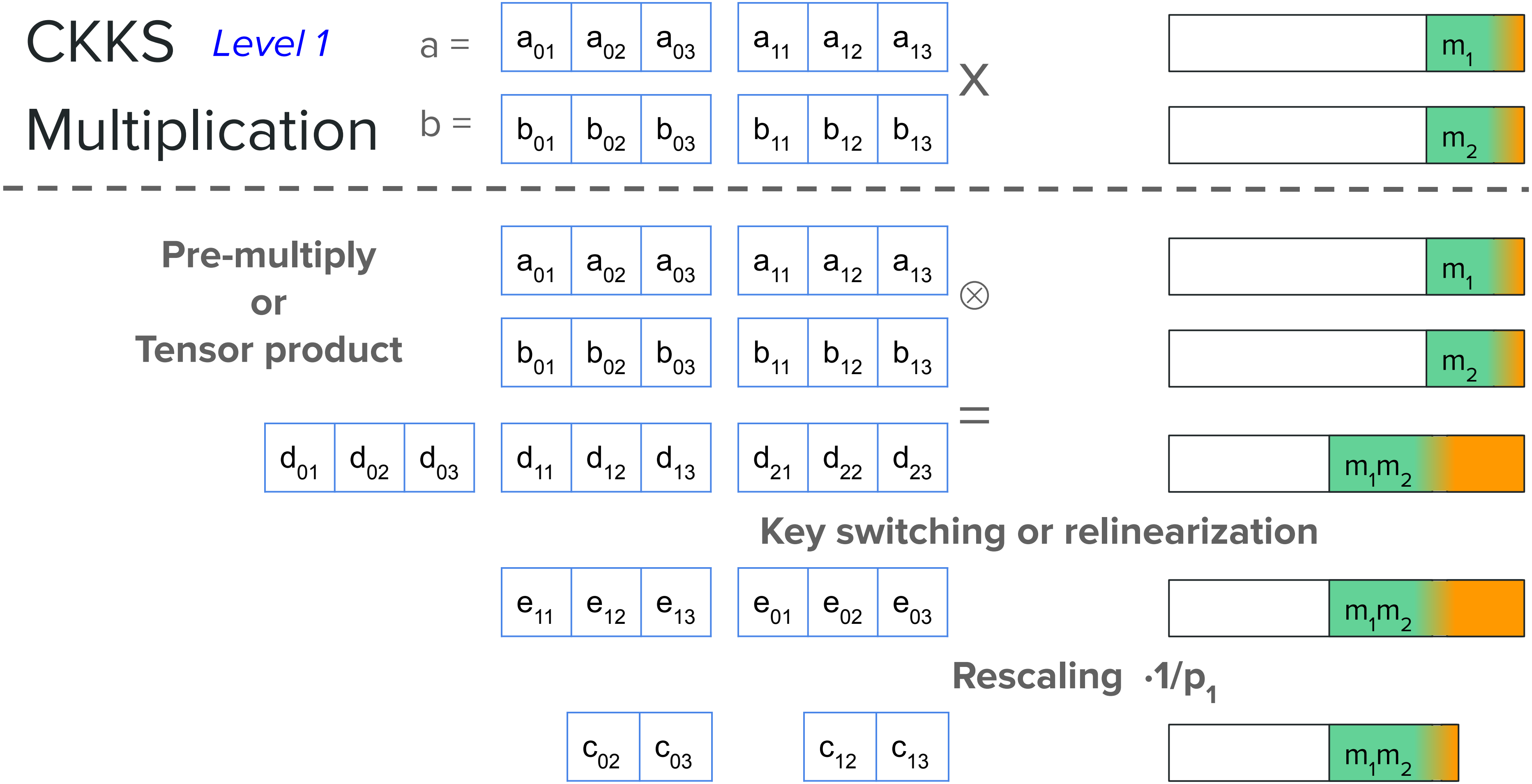




CKKS *Level 1* Multiplication



CKKS Multiplication *Level 1*



CKKS *Level 1*

Multi

$a =$ a_{01} a_{02} a_{03} a_{11} a_{12} a_{13} — m_1

X

b_{11} b_{12} b_{13} — m_2

Rescaling:

$$c_0 = (e_0 - [e_0]_{p_l}) \cdot p_l^{-1}$$
$$c_1 = (e_1 - [e_1]_{p_l}) \cdot p_l^{-1}$$

Not algebraic!

=

d_{21} d_{22} d_{23} — $m_1 m_2$

Key switching or relinearization

e_{01} e_{02} e_{03} — $m_1 m_2$

Rescaling $\cdot 1/p_1$

c_{02} c_{03} c_{12} c_{13} — $m_1 m_2$

CKKS *Level 1*

a = a_{01} a_{02} a_{03} a_{11} a_{12} a_{13} ... m_1

Rescaling:

$$c_0 = (e_0 - [e_0]_{p_l}) \cdot p_l^{-1}$$
$$c_1 = (e_1 - [e_1]_{p_l}) \cdot p_l^{-1}$$

Can be rewritten as **Euclidean division**

$$e_i = c_i \cdot p_l + [e_i]_{p_l}$$

Prover inputs $w_{\mathrm{quo},i}$ $w_{\mathrm{rmd},i}$

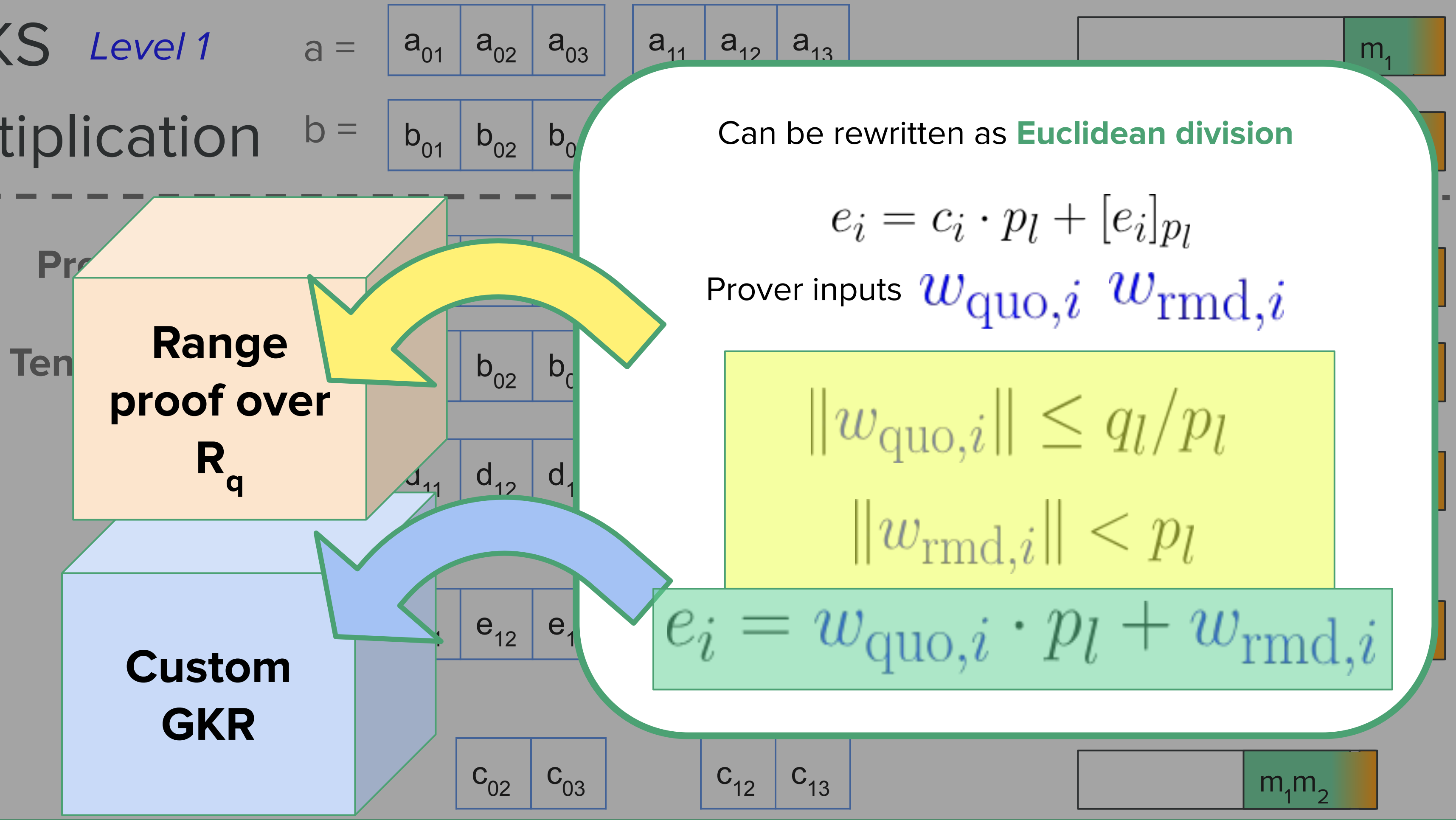
$$||w_{\mathrm{quo},i}|| \leq q_l / p_l$$

$$||w_{\mathrm{rmd},i}|| < p_l$$

$$e_i = w_{\mathrm{quo},i} \cdot p_l + w_{\mathrm{rmd},i}$$

c_{02} c_{03} c_{12} c_{13} $m_1 m_2$

CKKS *Level 1* Multiplication



Can be rewritten as **Euclidean division**

$$e_i = c_i \cdot p_l + [e_i]_{p_l}$$

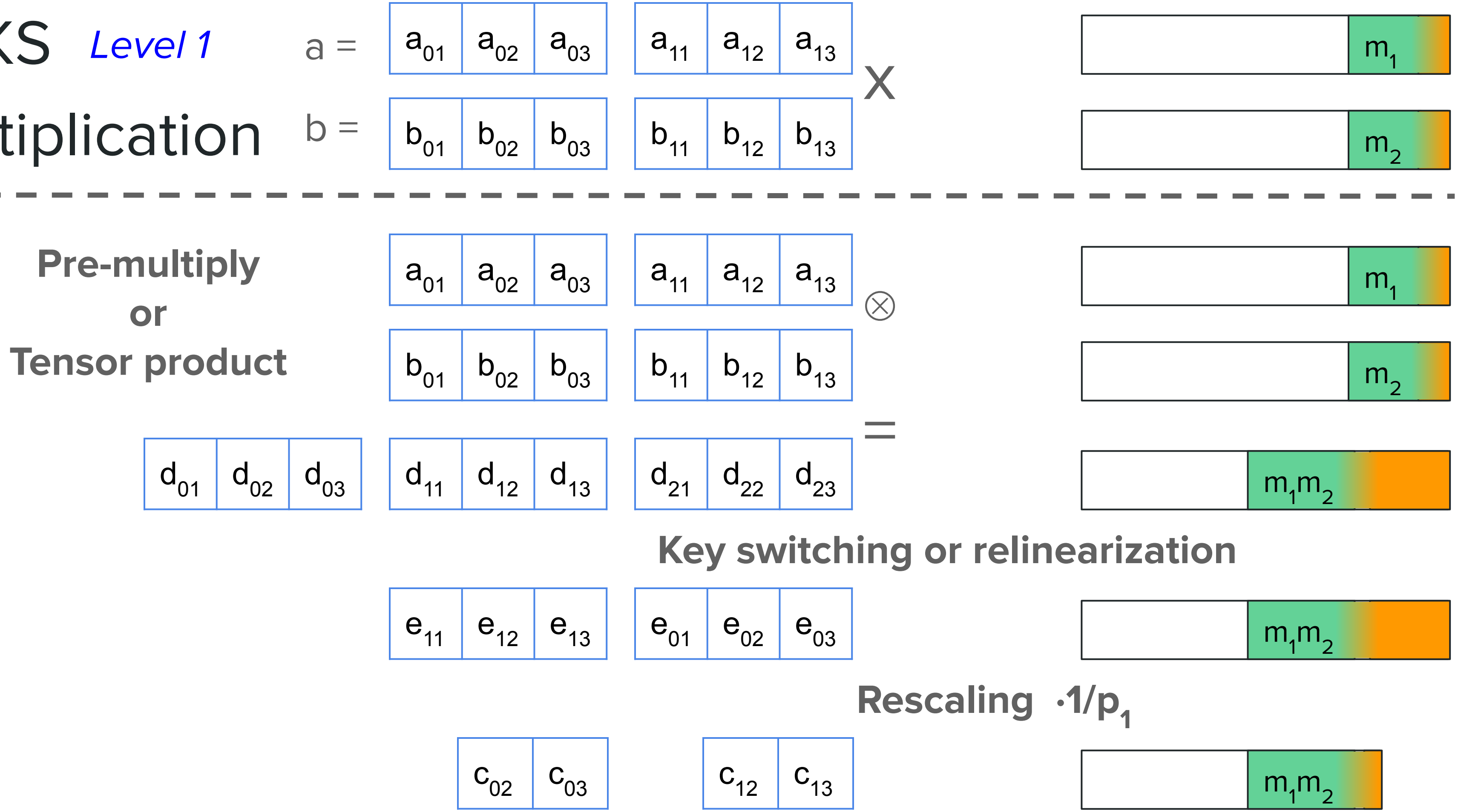
Prover inputs $w_{\text{quo},i}$ $w_{\text{rmd},i}$

$$\|w_{\text{quo},i}\| \le q_l / p_l$$

$$\|w_{\text{rmd},i}\| < p_l$$

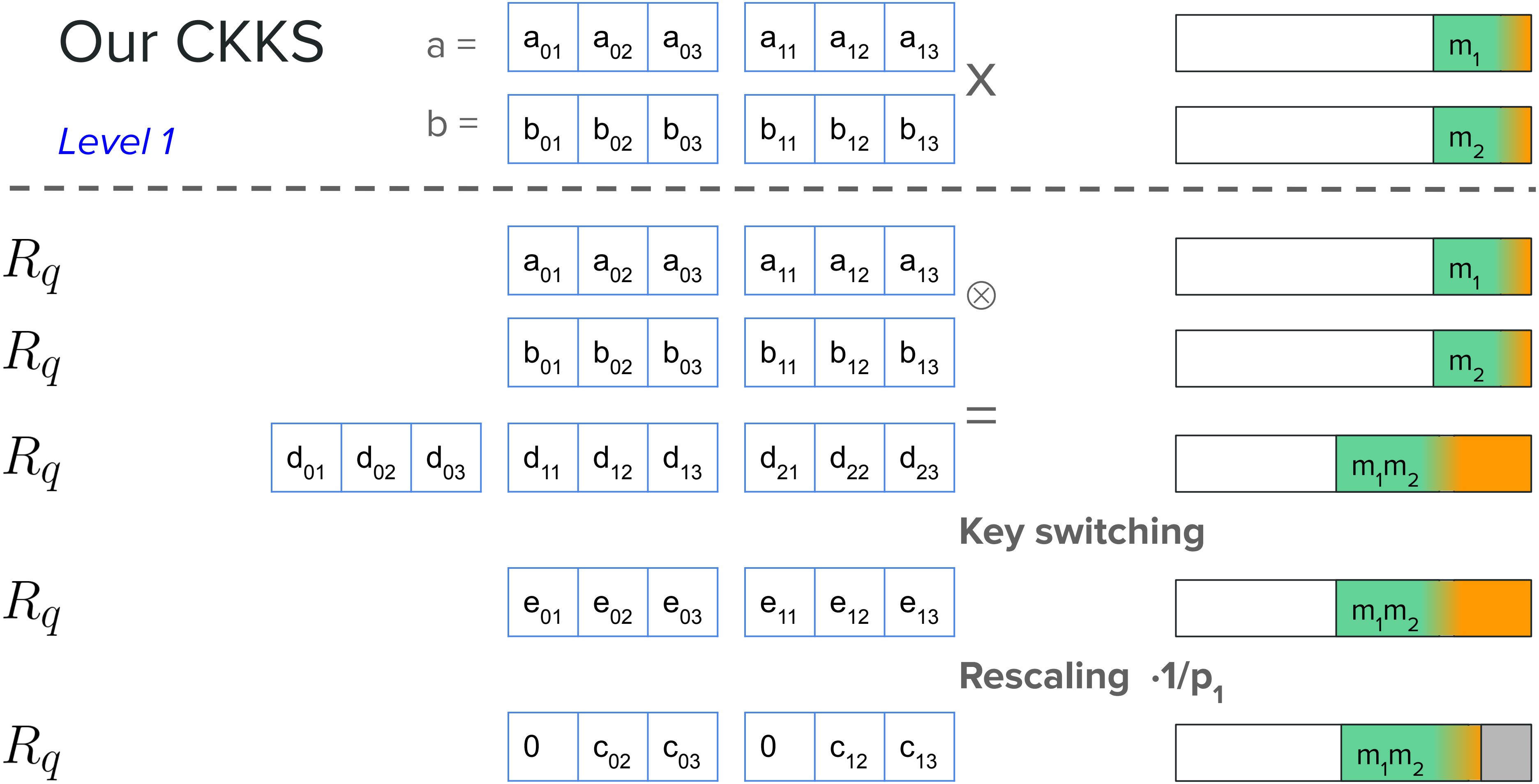
$$e_i = w_{\text{quo},i} \cdot p_l + w_{\text{rmd},i}$$

CKKS Multiplication *Level 1*



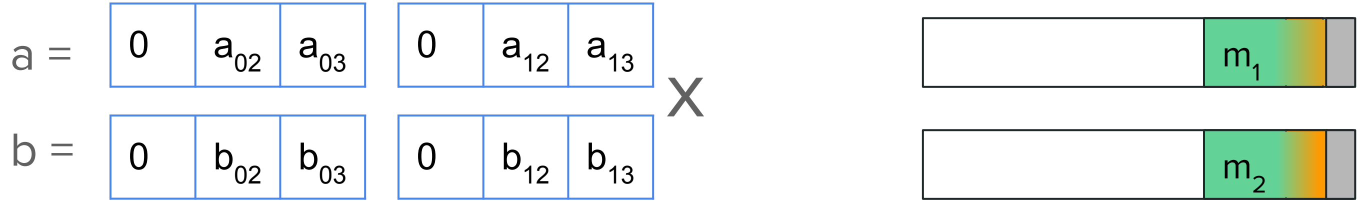
Our CKKS

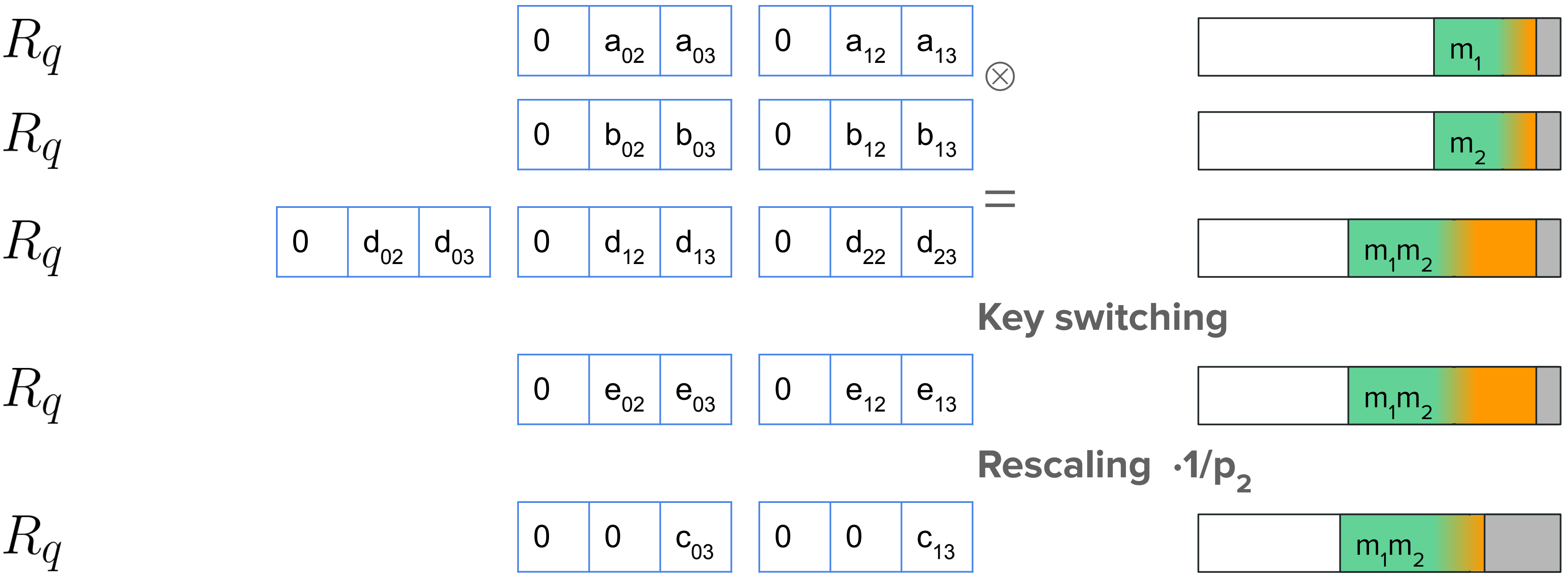
Level 1



Our CKKS

Level 2





Proof-friendly CKKS vs CKKS

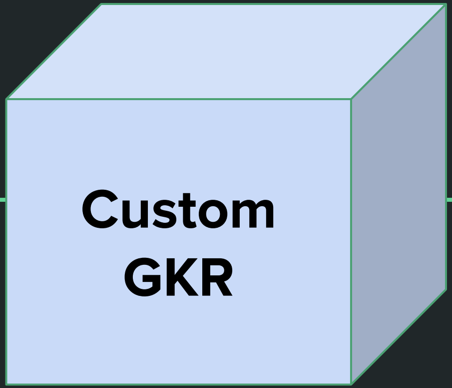
	Proof-friendly CKKS		CKKS
	d = 2	d = 4	HEXL
CKKS multiplication	7.394ms	8.457ms	7.197ms

N = 16384
#RNS components (L) = 6

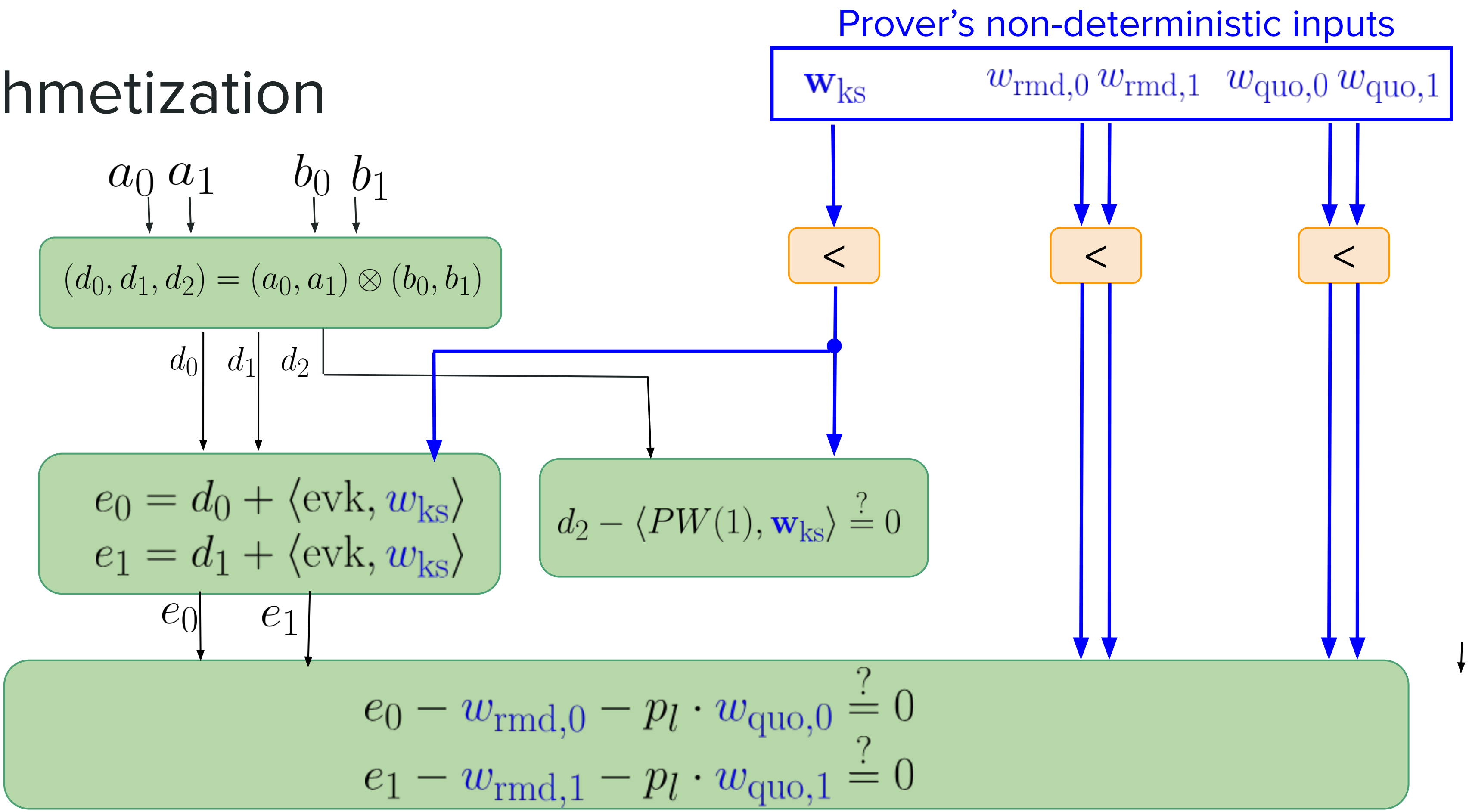
Proof-friendly CKKS in summary

- Carefully chosen ring setup
    - High **soundness** for proof system
    - **Efficiency** of computations
- Ring does not change
    - Proof system works on **same ring**
- Noise analysis
    - Easier to **prove bounds** on ciphertexts

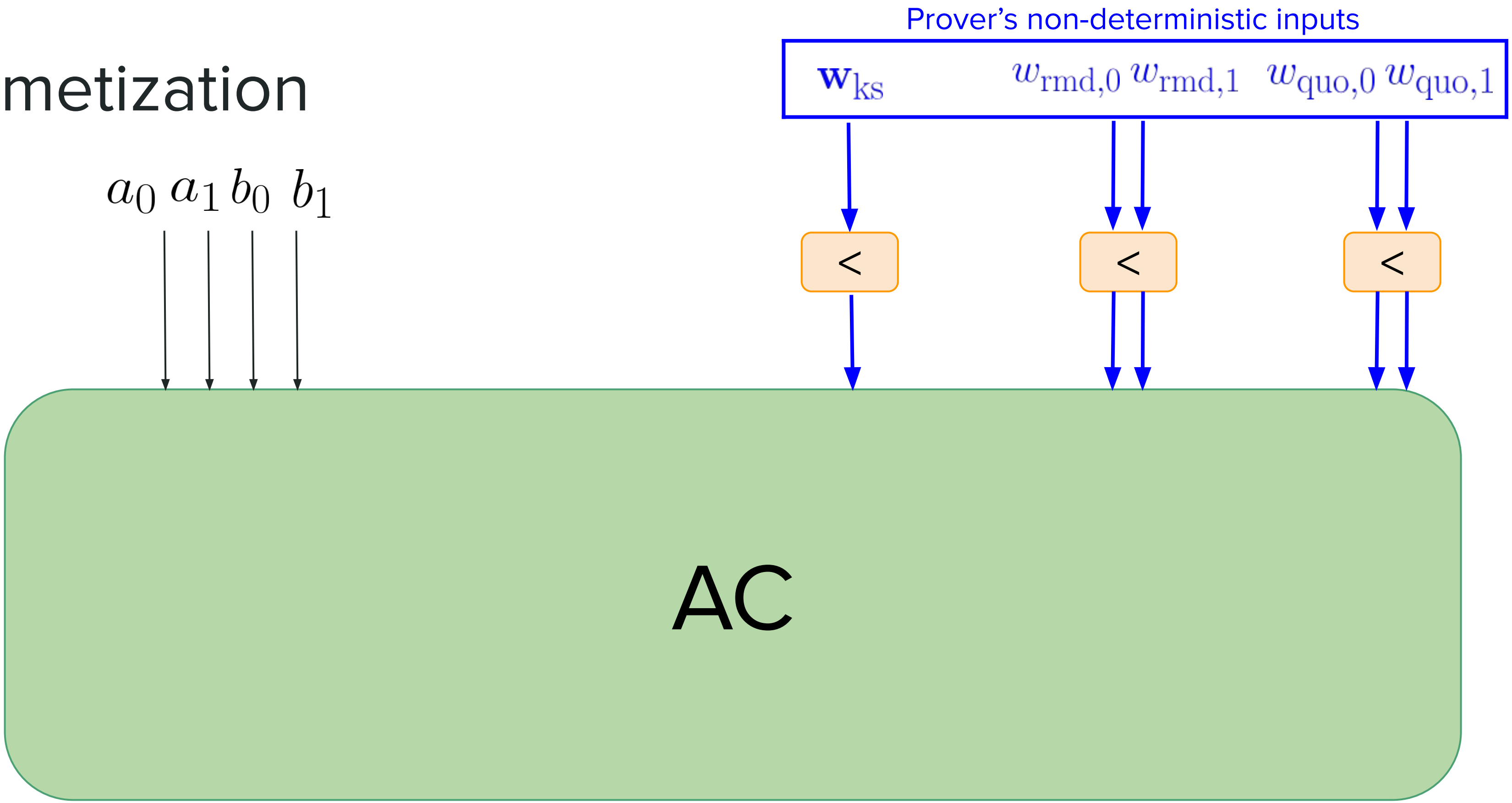
Proof of AC satisfiability



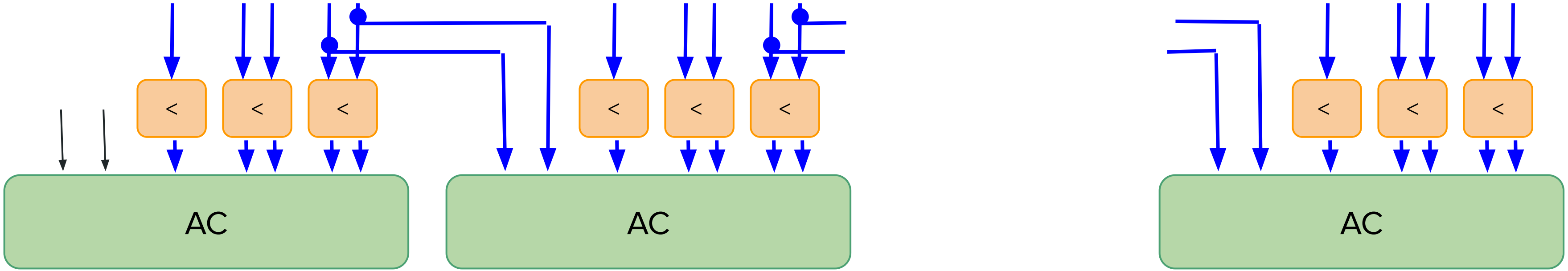
Arithmetization



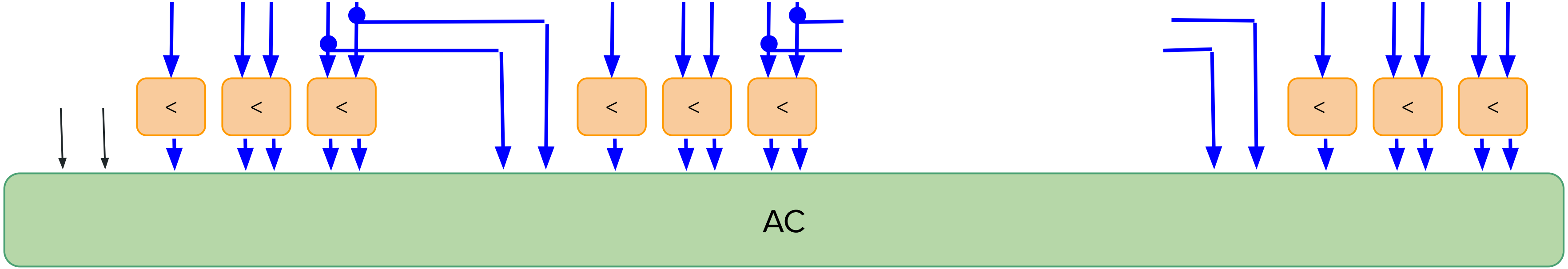
Arithmetization



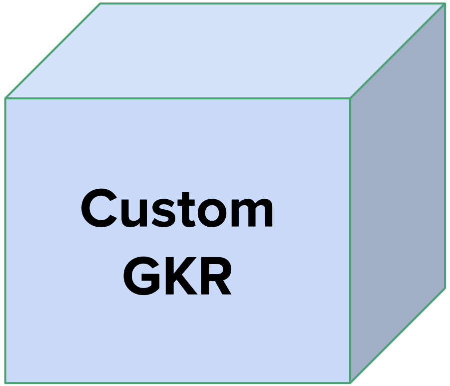
Flattening the circuit



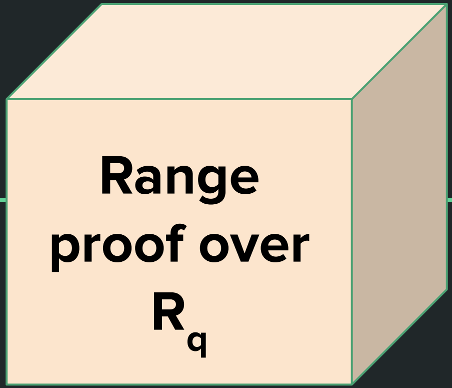
GKR-style proof system for AC



- **Custom gates** (bdcon, rescon, ...)
- Flattened system of relations => **constant depth 4**
- Not affected by recent FS attacks

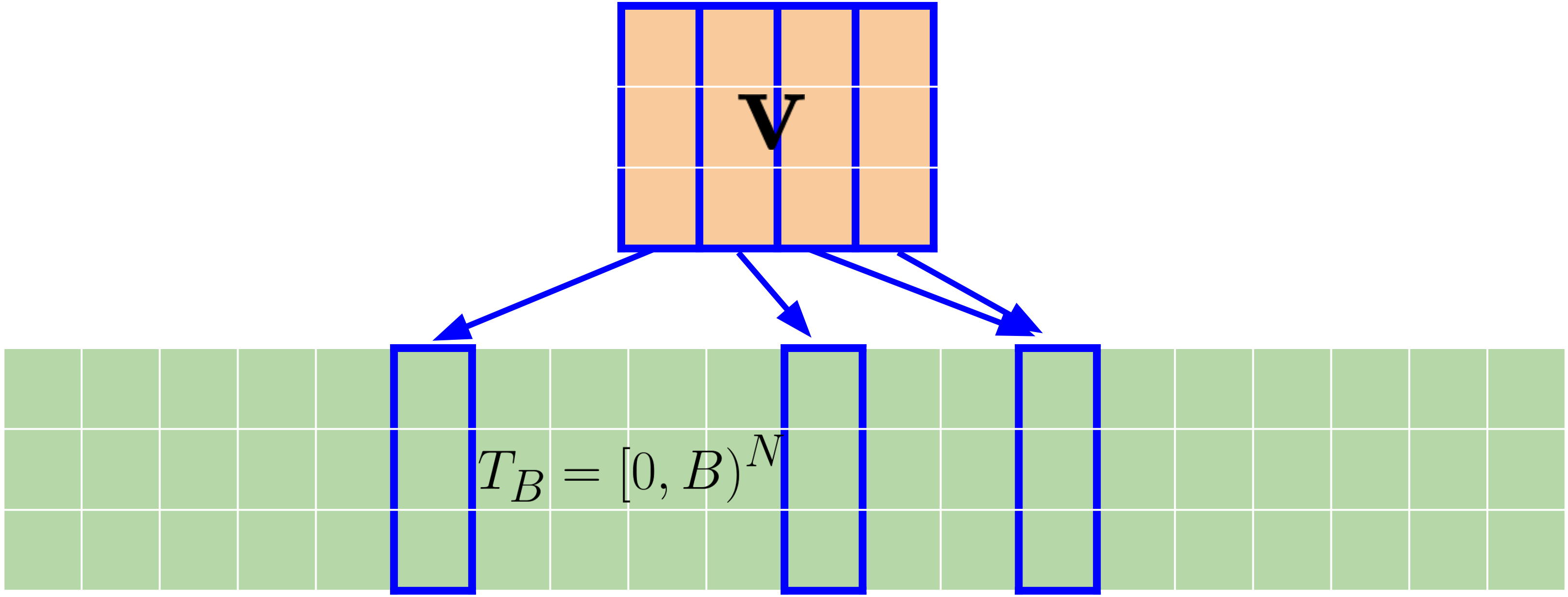


Range checks



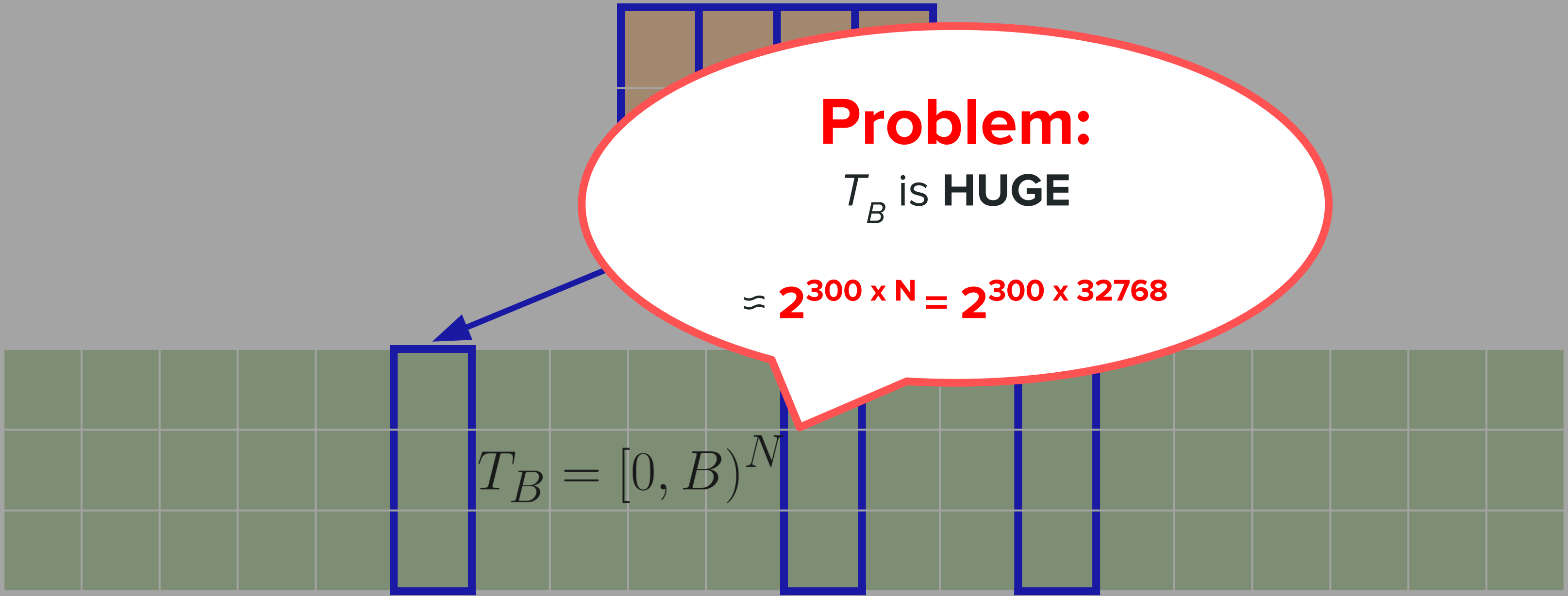
Proving ranges

- Prove that vector **v** of *m* elements in R_q has coeffs bounded by B (e.g B = q_l)
- Can be seen as a look-up argument



Proving ranges

- Prove that vector **v** of *m* elements in R_q has coeffs bounded by B (e.g B = q_l)
- Can be seen as a look-up argument



Proving ranges

n R_q has coeffs bounded by B (e.g B = q_l)

nt

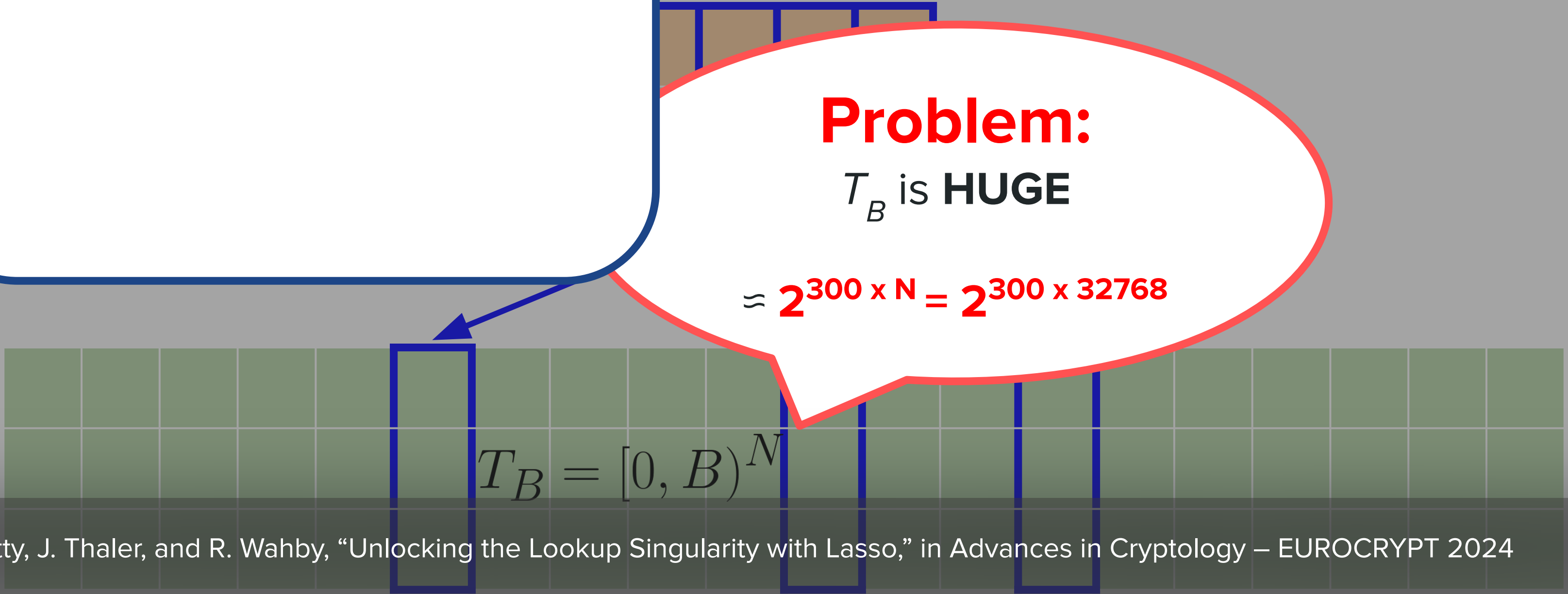
Solution for integers:

Decompose B (e.g. Lasso[1])

Problem:

T_B is **HUGE**

$\approx 2^{300 \times N} = 2^{300 \times 32768}$



$$T_B = [0, B)^N$$

[1] S. Setty, J. Thaler, and R. Wahby, “Unlocking the Lookup Singularity with Lasso,” in Advances in Cryptology – EUROCRYPT 2024

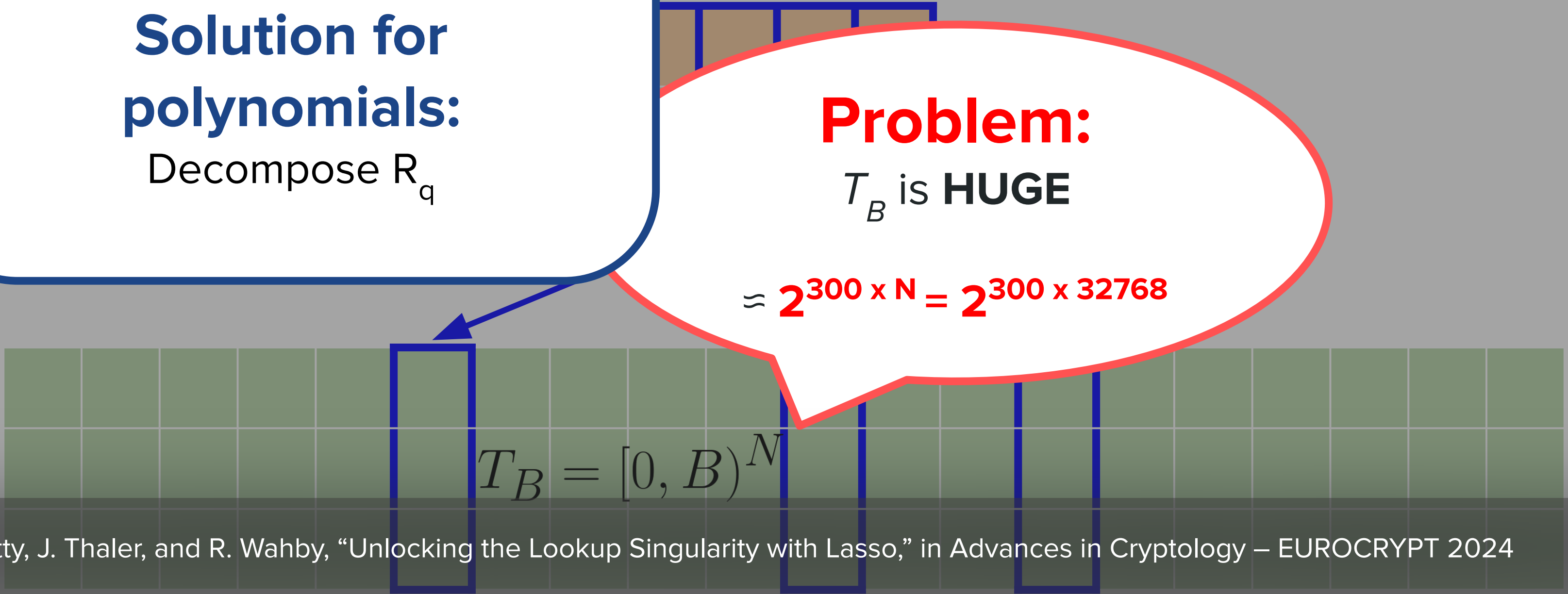
Proving ranges

Solution for integers:
Decompose B (e.g. Lasso[1])

Solution for polynomials:
Decompose R_q

...n R_q has coeffs bounded by B (e.g B = q_l)

...nt



[1] S. Setty, J. Thaler, and R. Wahby, “Unlocking the Lookup Singularity with Lasso,” in Advances in Cryptology – EUROCRYPT 2024

The polynomial commitment

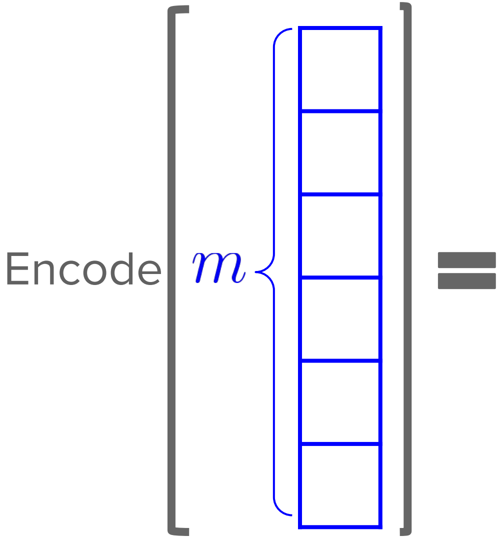
Polynomial Commitment

Need to commit to elements in $R_q[X_1, \ldots, X_\ell]$ where

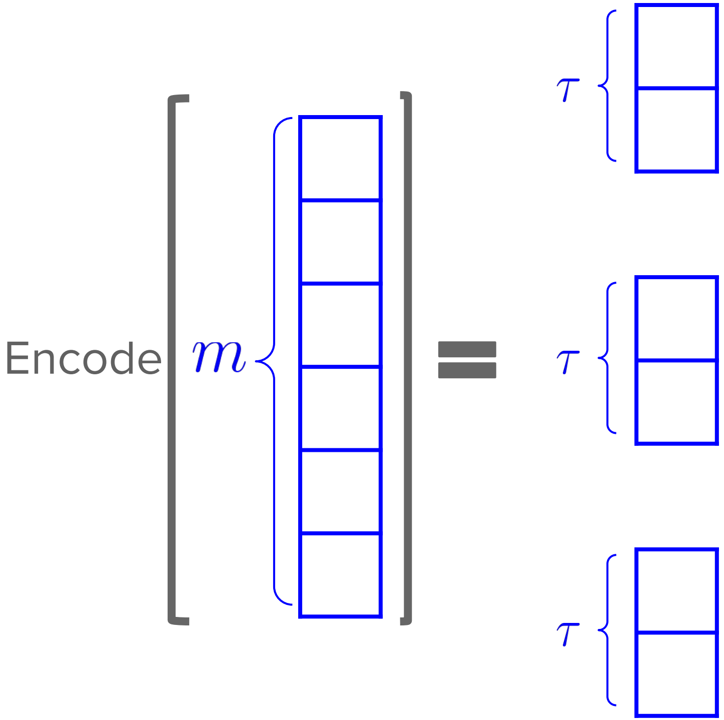
$$R_q \cong \mathbb{F}_{p_0^4} \times \cdots \times \mathbb{F}_{p_L^4}$$

- Reduce MV PC over R_q to MV PC over $\mathbb{F}_{p_i^4}$
- Small-ish fields => **Brakedown** (field-agnostic)
- $\mathbb{F}_{p_i^4}$ has $N/2$ roots of unity. **Can we use them?**

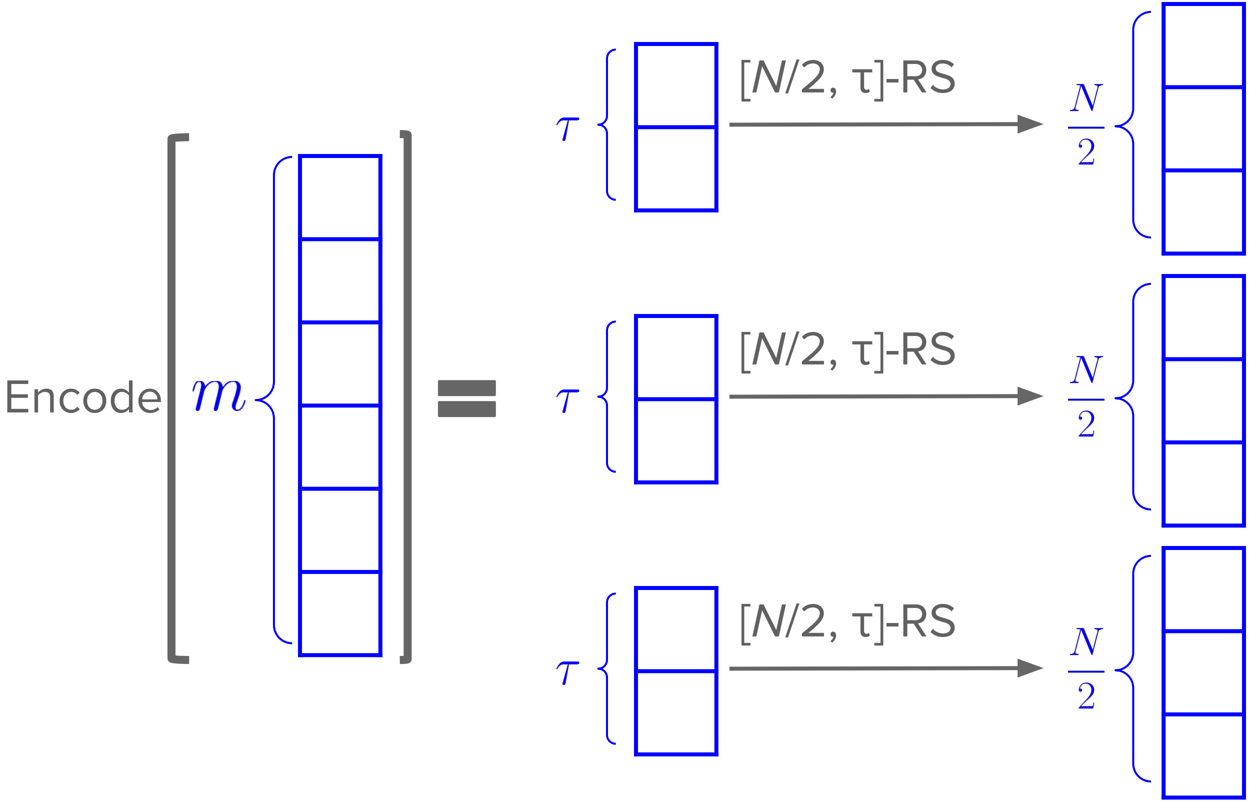
Piecewise Reed-Solomon code



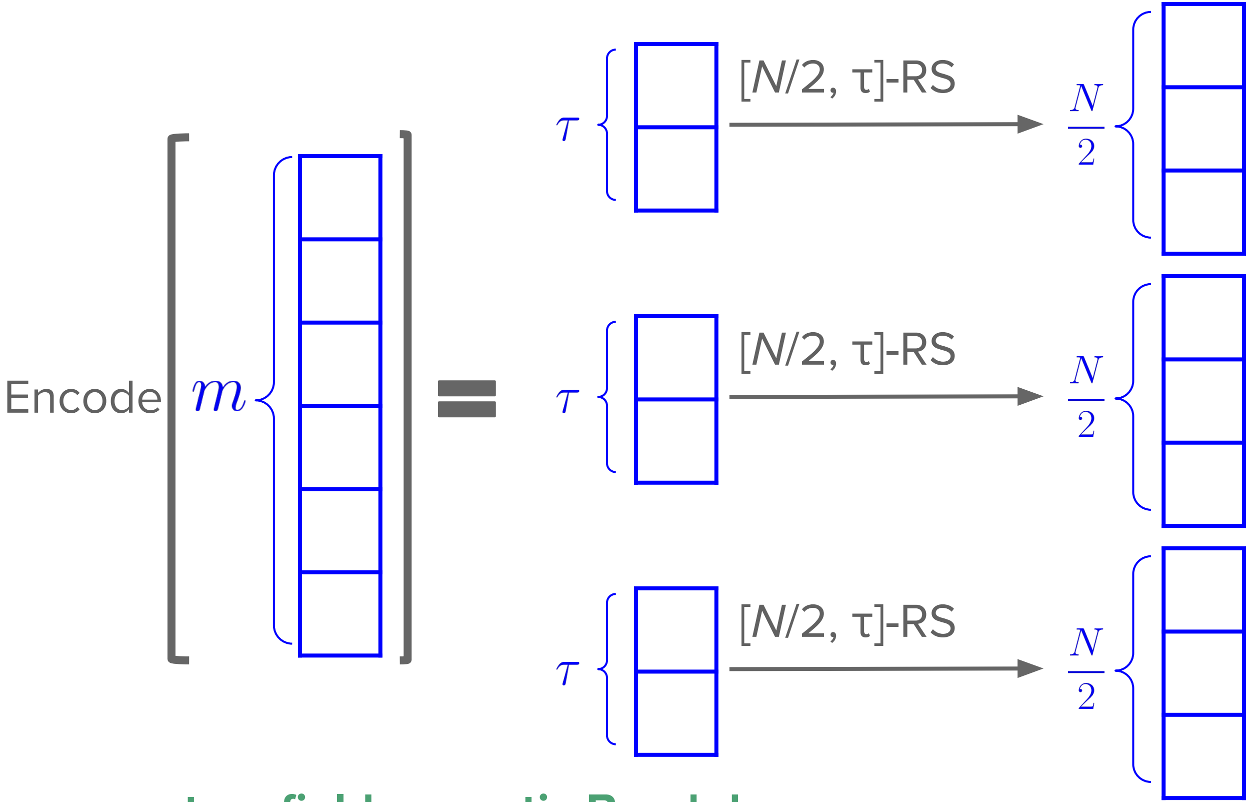
Piecewise Reed-Solomon code



Piecewise Reed-Solomon code



Piecewise Reed-Solomon code



x10 improvement on field-agnostic Breakdown

Conclusions

To summarize

- First practical VC for CKKS
  - Technique extend to FV/BGV
- Description of problem in a modular way (arithmetization)
  - AC satisfiability + range checks
- Design of proof-friendly CKKS
- Design of custom GKR to prove AC over rings
- Design of range proofs for polynomial rings
- Improved Brakedown for medium-sized fields
- Implemented all building blocks

https://eprint.iacr.org/2025/286

Thank you!

This work is supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (Grant agreement No. 101001283), partially supported by projects PRODIGY (TED2021-132464B- I00) and ESPADA (PID2022-142290OB-I00) funded by MCIN/AEI/10.13039/501100011033/. This work is part of the grant JDC2023-050791-I, funded by MCIN/AEI/10.13039/501100011033 and the ESF+. This work is also supported by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme in the scope of the CONFIDENTIAL6G project under Grant Agreement 101096435. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Images used in this presentation

- User faces: “Plump Interface Duotone Icons” by Streamline, Creative Commons Attribution 4.0 International, available at https://iconduck.com/sets/plump-interface-duotone-icons