Batch, Pack, and Prove

More Efficient Verifiable Computation for CKKS

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

institute IMDEA software

ÉCOLE POLYTECHNIQUE

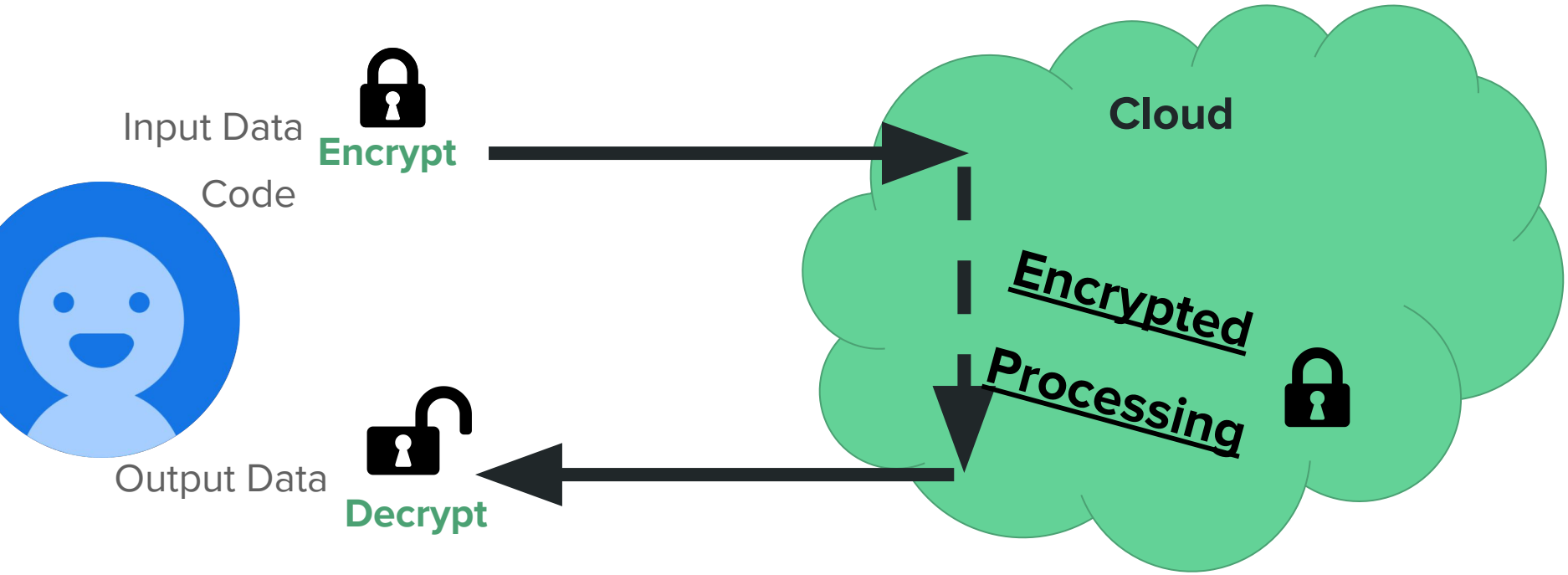
INSTITUT POLYTECHNIQUE DE PARIS

TII Technology Innovation Institute

Context

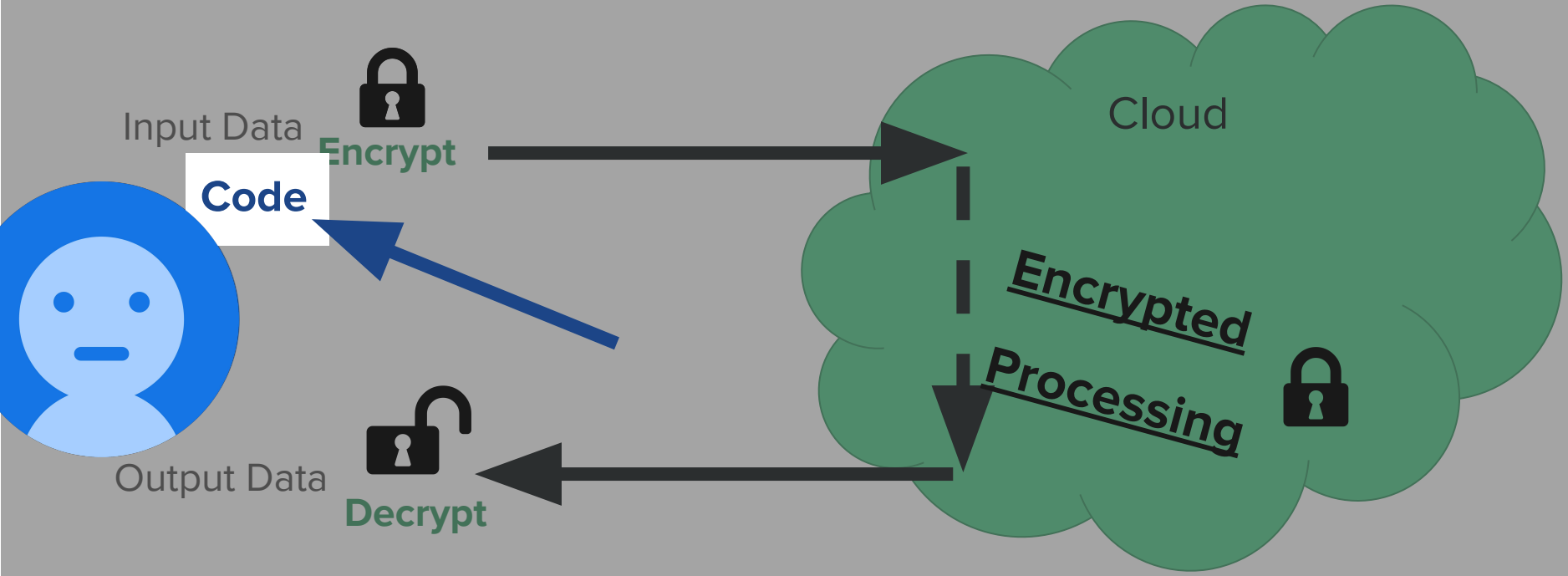
Verifiable FHE

Homomorphic Encryption (HE)



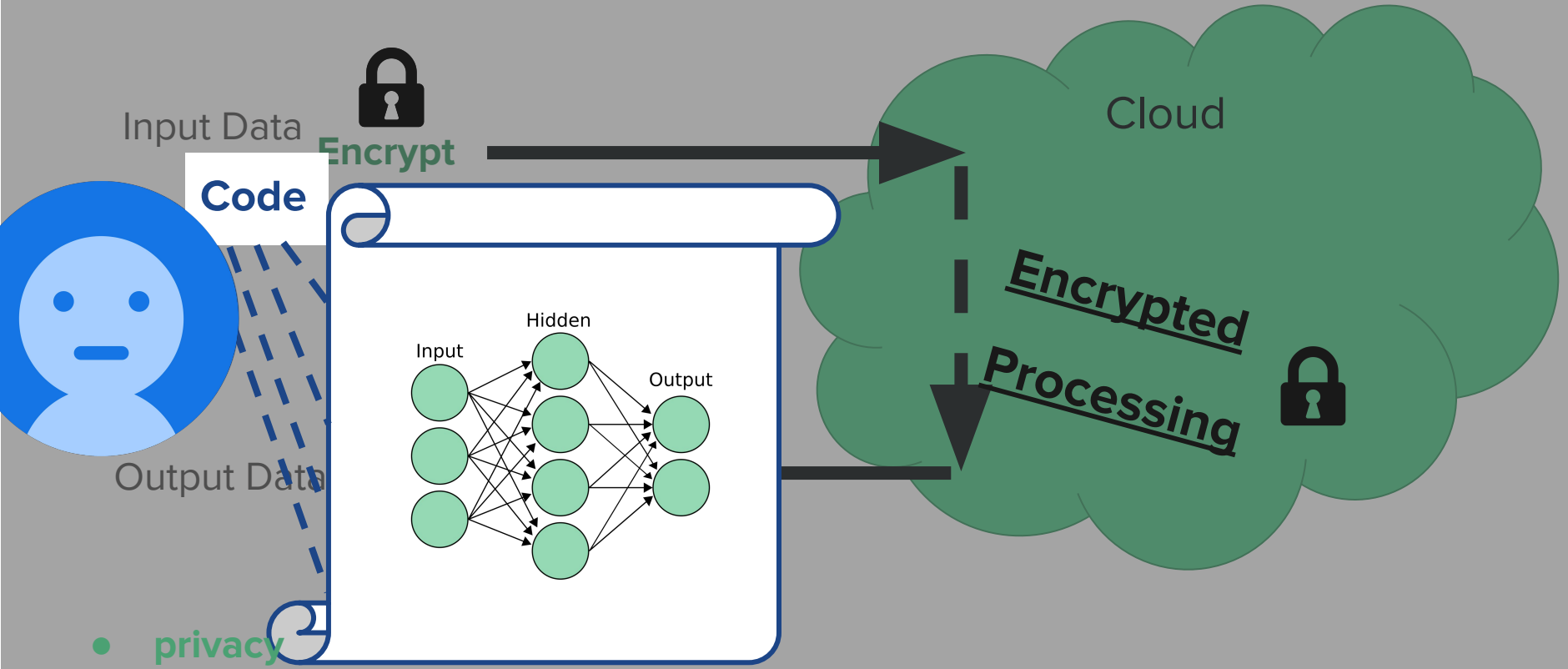
- **privacy**

Homomorphic Encryption (HE)



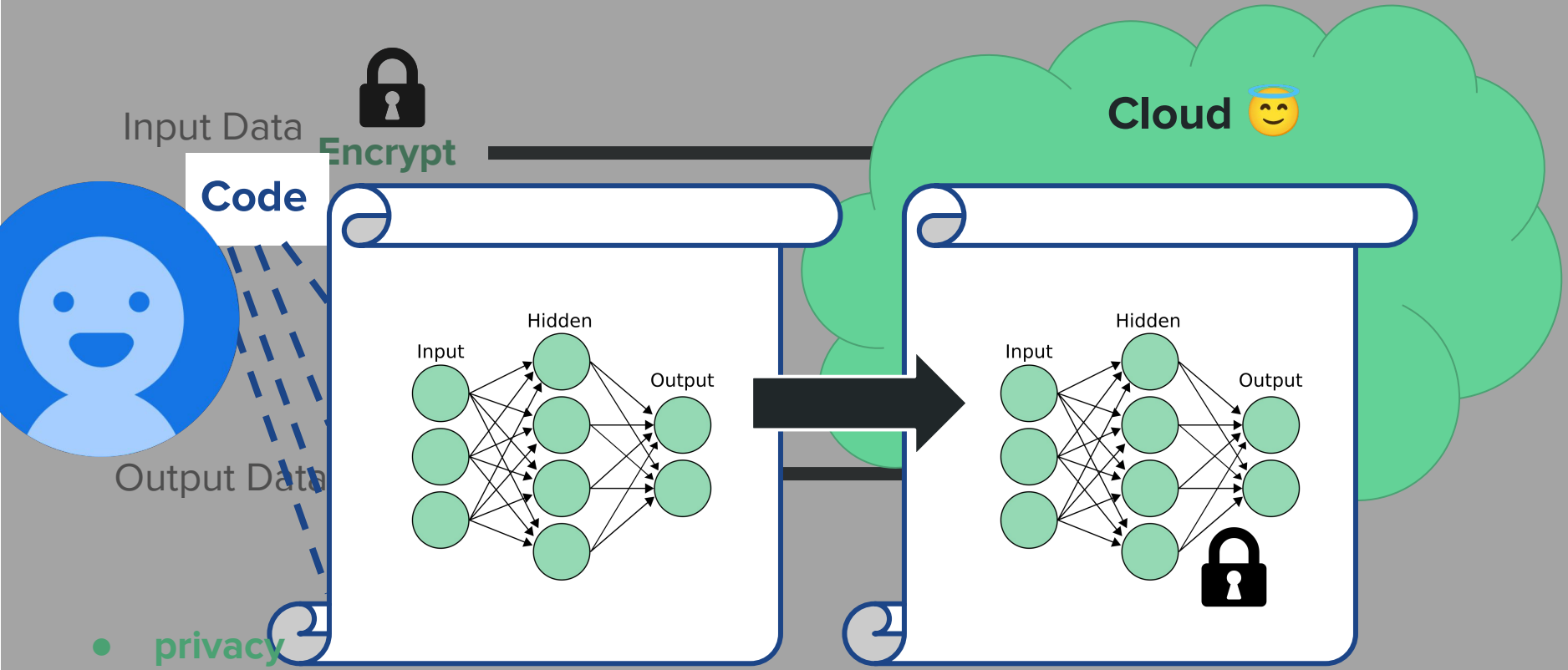
- privacy

Homomorphic Encryption (HE)



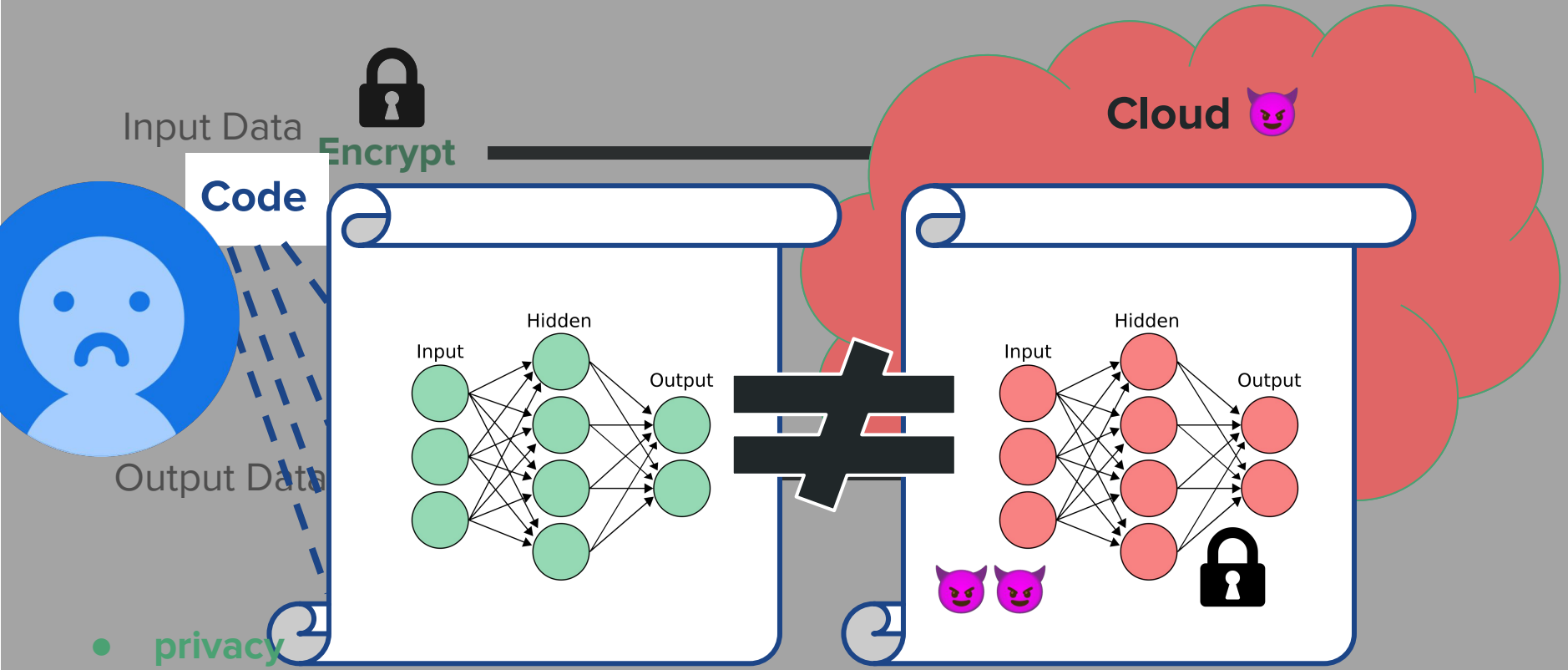
- privacy

Homomorphic Encryption (HE)



- privacy

Homomorphic Encryption (HE)



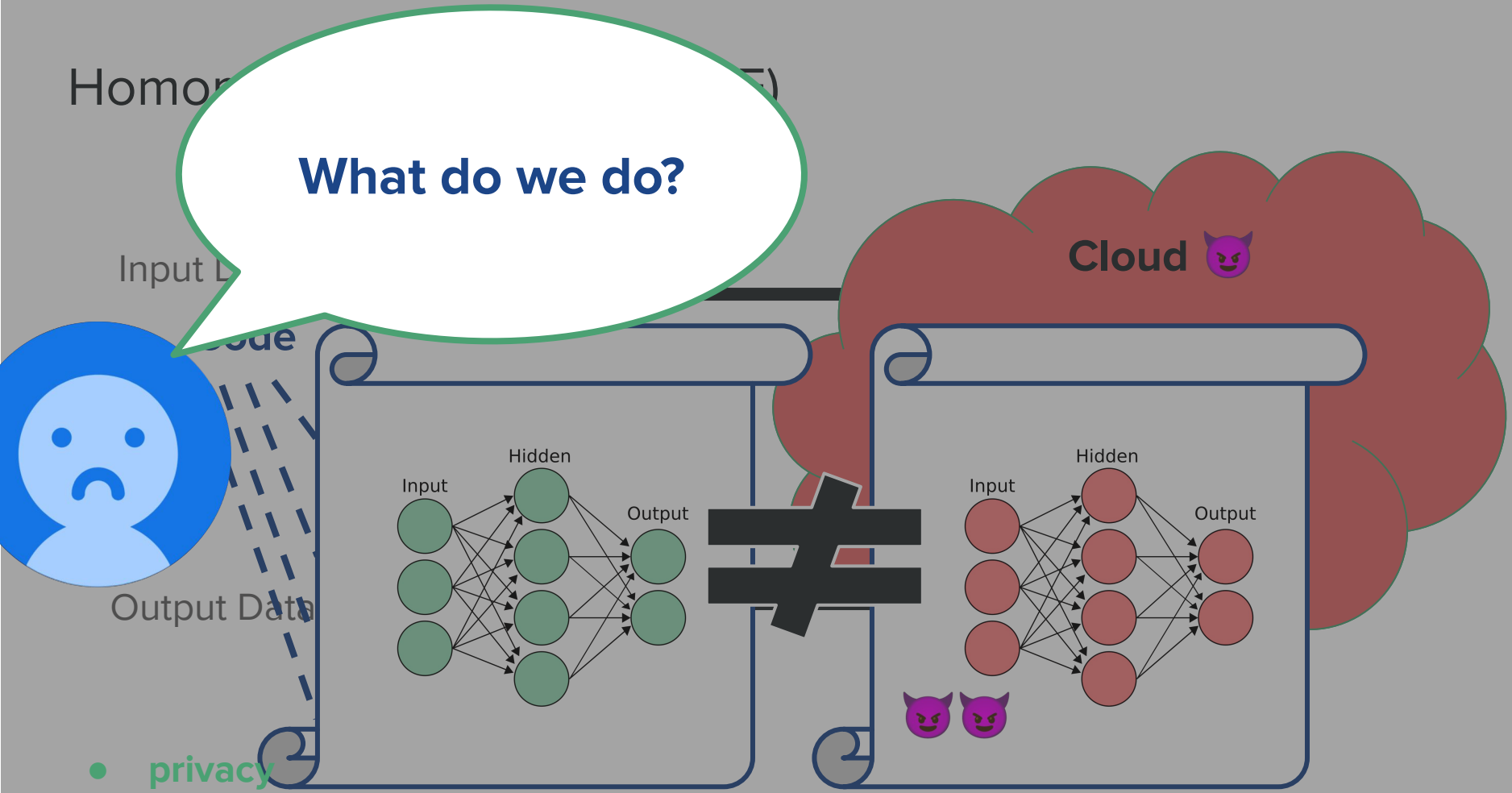
- privacy

Homor

Input D

ode

Output Data



- privacy

Homor

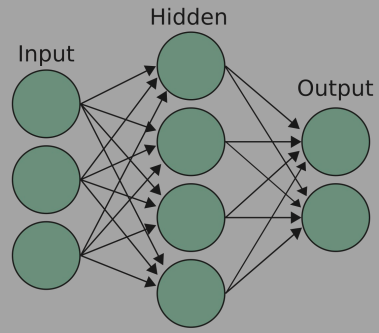
Input D

ode

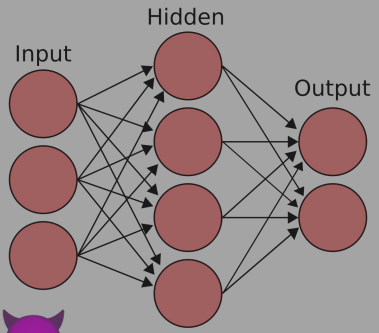
Output Data

Assume a semi-honest adversary!

Cloud 😈



≠



😈😈

- privacy

Homor

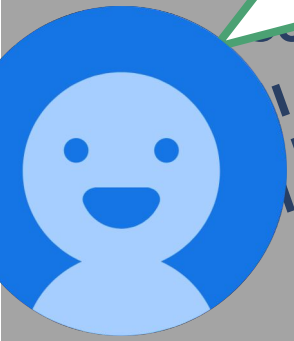
Input D

ode

Output Data

Assume a semi-honest adversary!

Everybody is happy... :)



- privacy

Homor

Assume a semi-honest adversary!

Input D

ode

Cloud 😈

Everybody is happy... :)

Input

Output

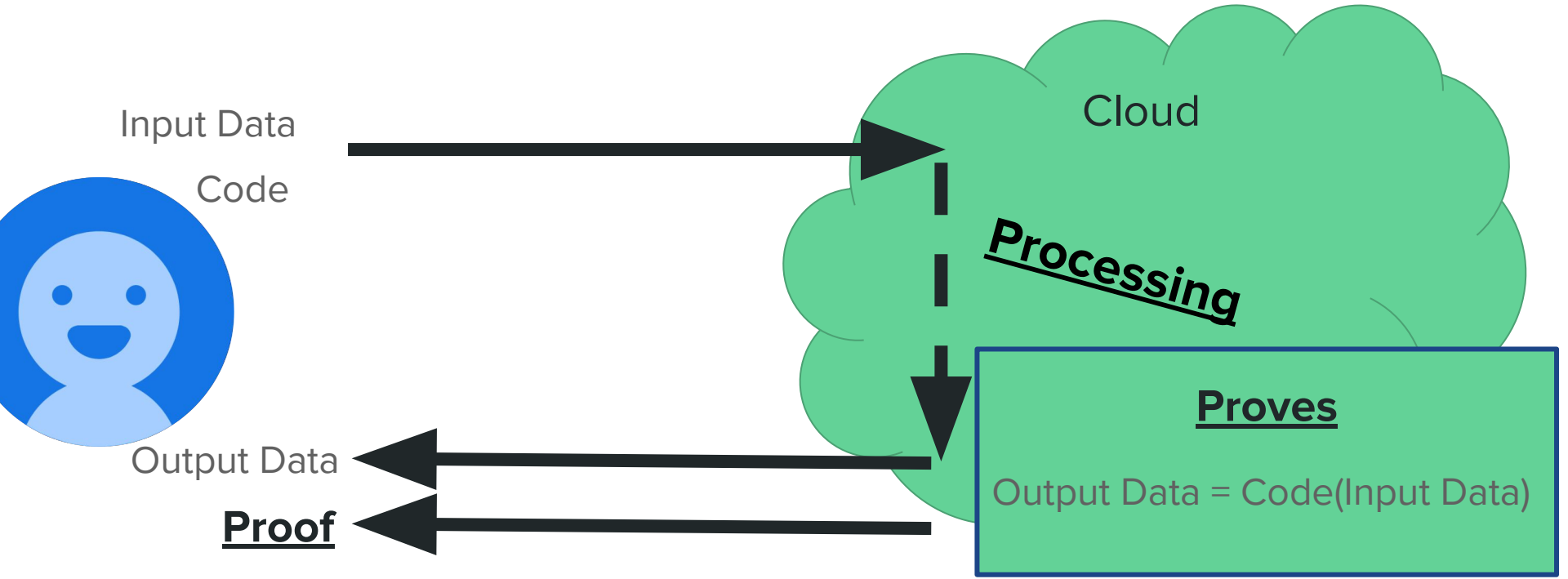
Output

Output Data

Including the malicious adversaries! :0

- privacy

Verifiable computation (VC)



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

Verifia

Input D

ode

Can we have both HE and VC? Efficiently?

Cloud

Processing

Proves

Output Data = Code(Input Data)

Output Data

Proof

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

VFHE and CCC^{+}25

VFHE

	Public Verification	Native R_q Arithmetic	Efficient Key Switching / Rescale	Efficient Bootstrapping	CKKS (approximate schemes)
Generic SNARK[1]	✓	✗	✗	✗	✓
Rinocchio[2]	✗	✓	✗	✗	✓
HE-IOPs[3]	!	✓	✓	✓	✗
CCC+25[4]	✓	✓	✓	?	✓

[1] A. Viand, C. Knabenhans, and A. Hithnawi, “Verifiable Fully Homomorphic Encryption,” Feb. 11, 2023, arXiv: 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,” in Advances in Cryptology – ASIACRYPT 2024
[4] I. Cascudo, A. Costache, D. Cozzo, D. Fiore, A. Guimarães, and E. Soria-Vazquez, “Verifiable Computation for Approximate Homomorphic Encryption Schemes,” in Advances in Cryptology – CRYPTO 2025

CCC+25 - VFHE blueprint

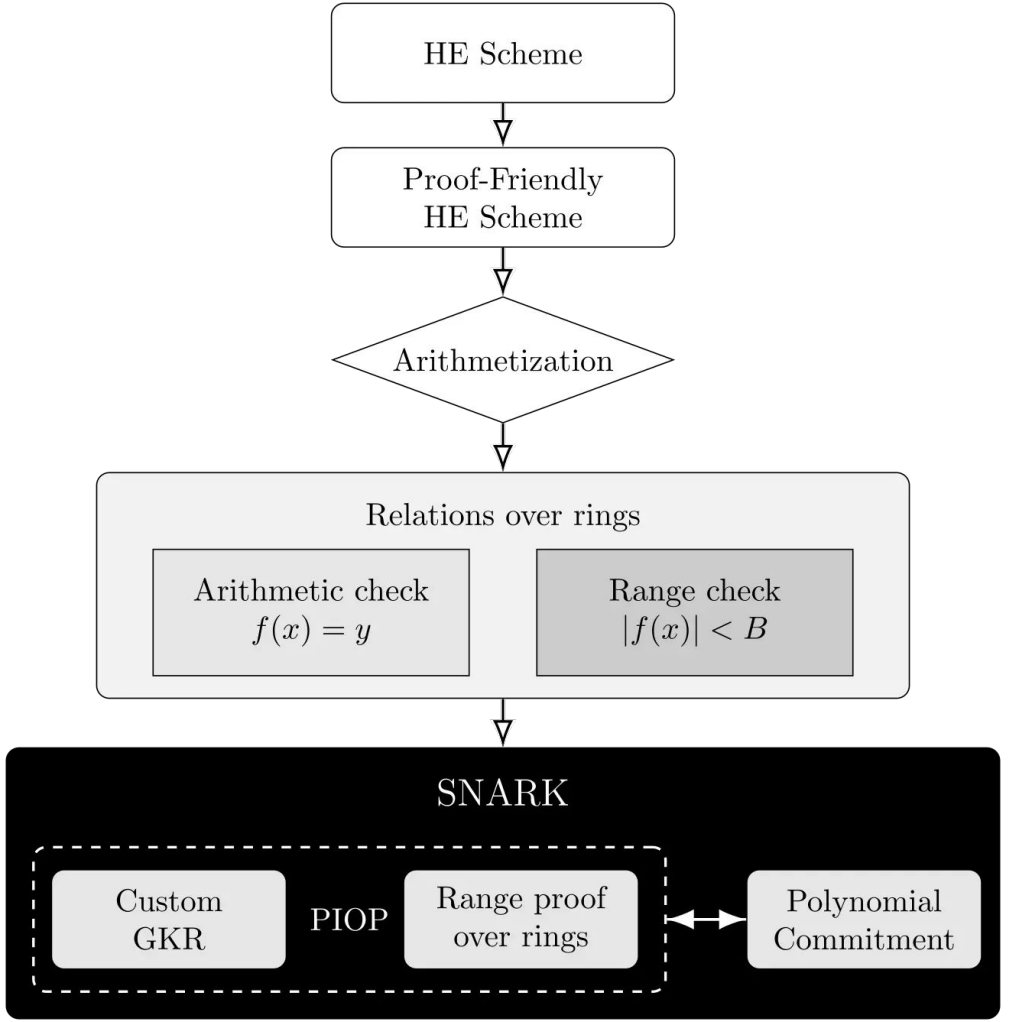
What we **did:**

- A blueprint for efficient VFHE and its instantiation to CKKS

Learn more:

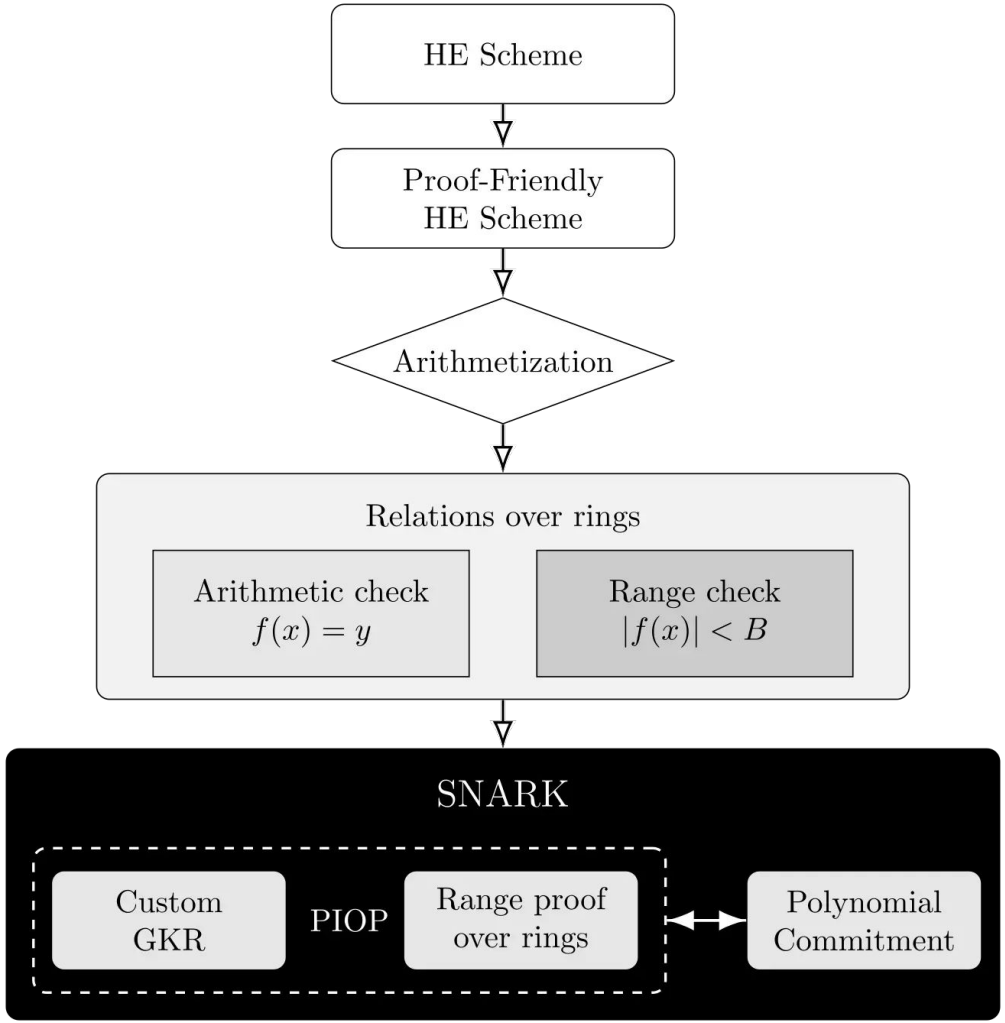
CKKS.org
blogpost:

FHE.org
2025:



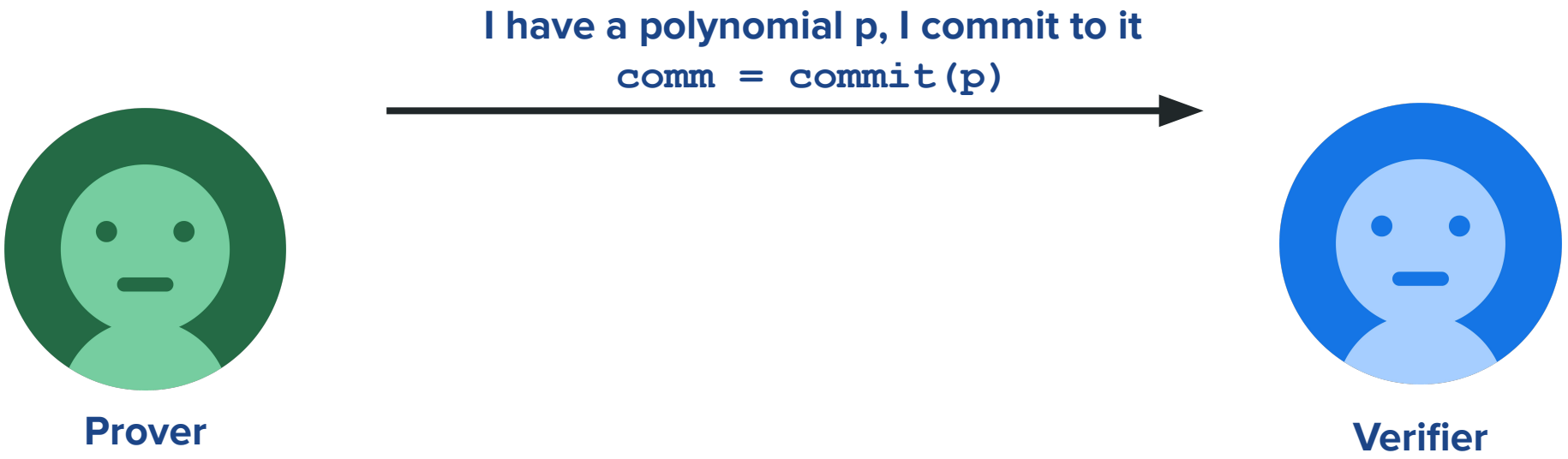
CCC+25 - VFHE blueprint

But what **didn't** we do**?**

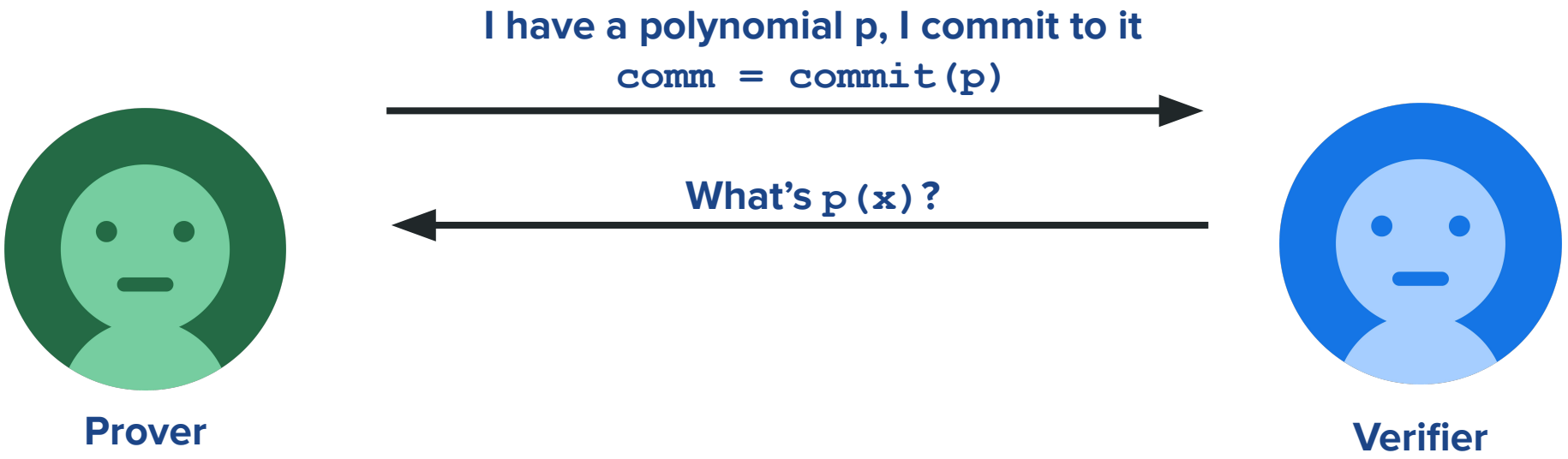


Preliminaries: Polynomial Commitments

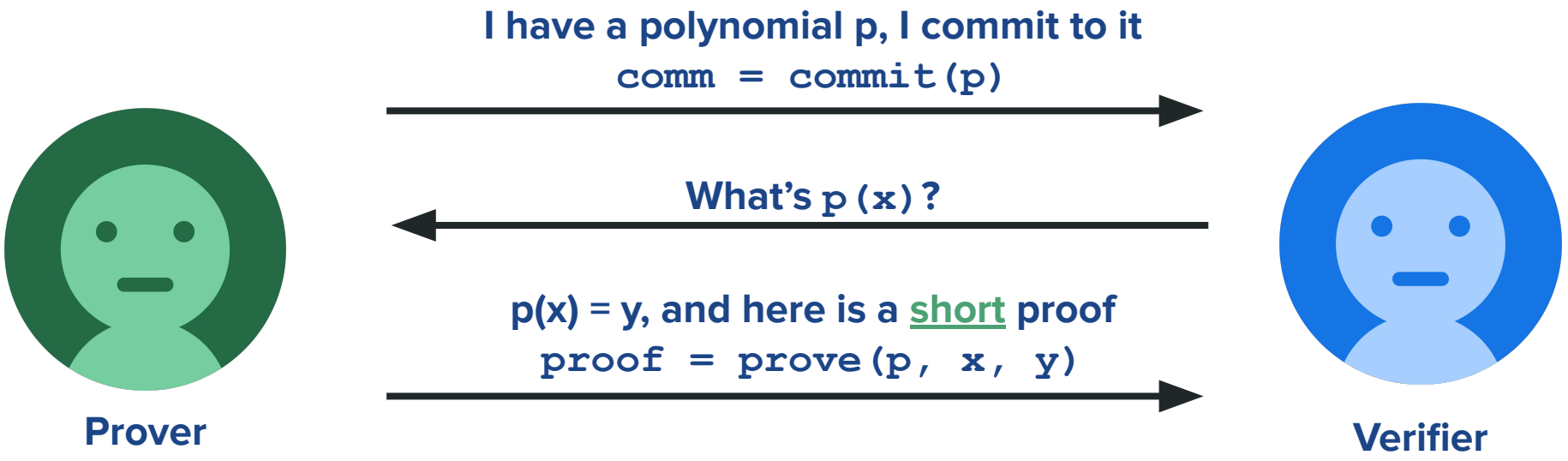
Polynomial commitments



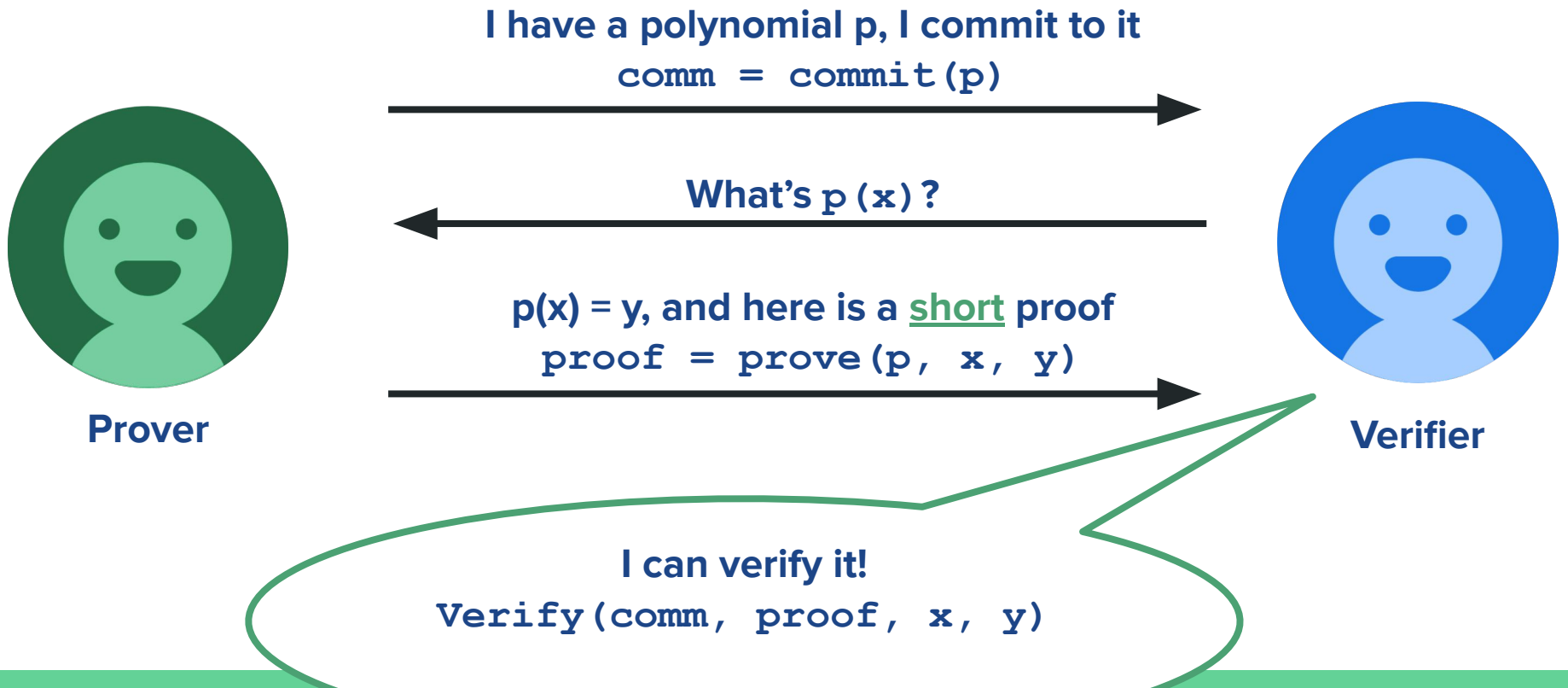
Polynomial commitments



Polynomial commitments



Polynomial commitments



Polynomial co

Important!

x, y, and p can be anything.

For example, in a PC for R_q:

$$p \in \mathcal{R}_q[X]$$

$$x, y \in \mathcal{R}_q$$

$$y = p(x)$$

Prover

Verifier

verify(comm, proof, x, y)

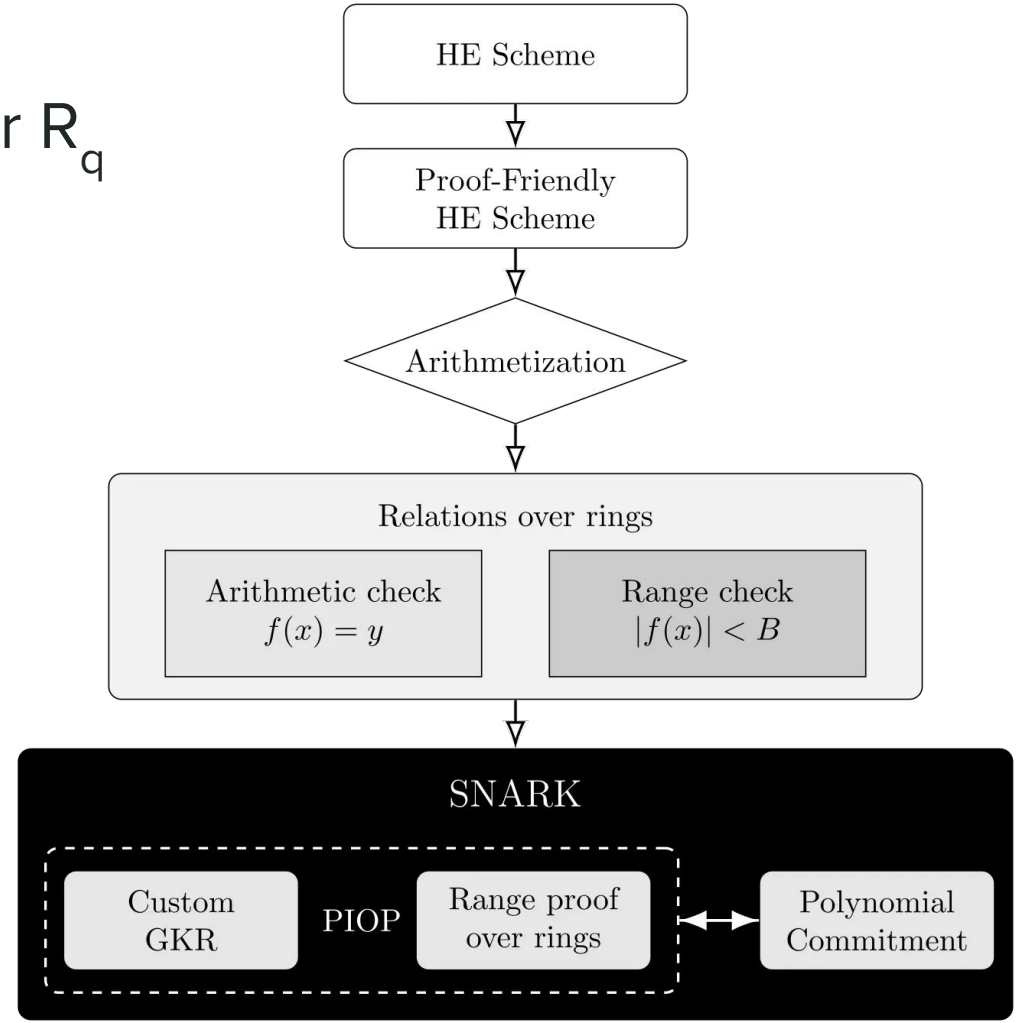
This work

This work

1. Oracle Packing
2. BatchFold
3. SparsePack

CCC+25 - A proof system for R_q

- Custom GKR for R_q
- Range checks in R_q
- Faster Brakedown PC over R_q

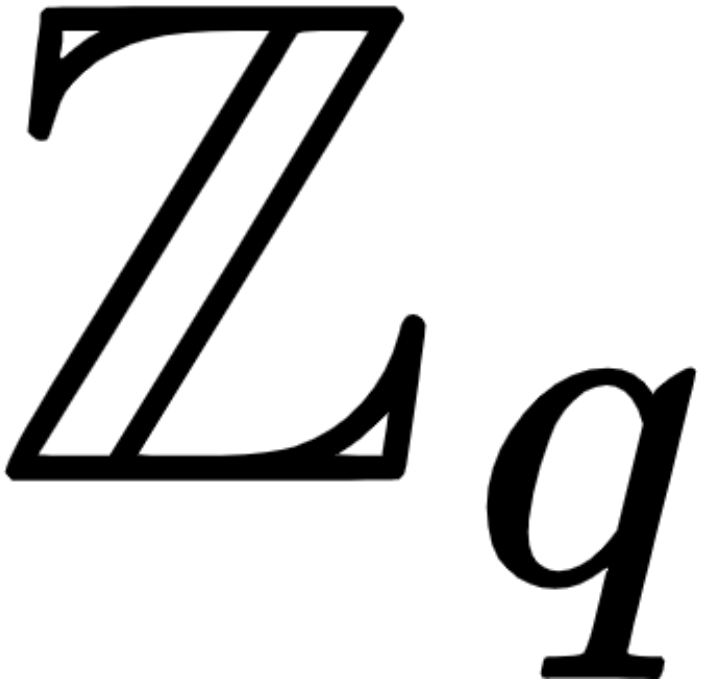


Not everything is about R_q

Not everything is about R_q

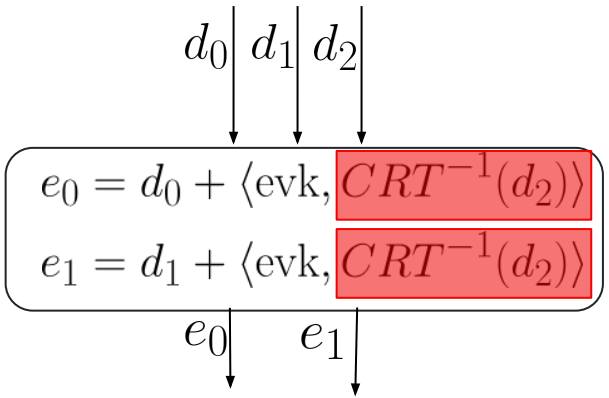
FHE also needs Z_q:

1. We scale by integers
2. We perform automorphisms (coefficient permutations)
3. We use **infinity norm** for bounding noise



CCC+25 - Range proofs for key switching

Key Switching



CCC+25 verification

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

$$e_0 = d_0 + \langle \text{evk}, w_{\text{ks}} \rangle$$
$$e_1 = d_1 + \langle \text{evk}, w_{\text{ks}} \rangle$$

CCC+25 - Range proofs

Every key switching produces several: $||\mathbf{w}_{\text{ks}}[i]|| < 2^k$

- Creates a big vector V with all $\mathbf{w}_{\text{ks}}$ (notice that $\mathbf{w}_{\text{ks}}$ is a polynomial R_q)
- Prove that all elements in V have **coefficients** bounded by 2^k using lookup arguments

CCC+25 - Range proofs

Every key switching produces several: $||\mathbf{w}_{\text{ks}}[i]|| < 2^k$

- Creates a big vector V with all $\mathbf{w}_{\text{ks}}$ (notice that $\mathbf{w}_{\text{ks}}$ is a polynomial R_q)
- Prove that all elements in V have **coefficients** bounded by 2^k using lookup arguments

CCC+25 - Range proofs

This proof is about $\mathbb{Z}_q$

Ev… $< 2^k$

- Creates a big vector V with … $_{ks}$ (notice that $\mathbf{w}_{ks}$ is a polynomial R_q)
- Prove that all elements in V have **coefficients** bounded by 2^k using lookup arguments

CCC+25 - Range proofs

This proof is about $\mathbb{Z}_q$

$< 2^k$

- Creates a big vector V with $\mathbf{w}_{ks}$ (notice that $\mathbf{w}_{ks}$ is a polynomial R_q)
- Prove that all elements in V have **coefficients** bounded by 2^k using lookup

CCC+25 - Lasso-style lookup arguments

1. **Decomposes R_q elements into Z_B for some small B**
2. **Proves that the decomposed values are in tables on size B**

CCC+25 - Range proofs

This proof is about $\mathbb{Z}_q$

$< 2^k$

- Creates a big vector V with $\mathbf{w}_{ks}$ (notice that $\mathbf{w}_{ks}$ is a polynomial R_q)
- Prove that all elements in V have **coefficients** bounded by 2^k using lookup

CCC+25 - Lasso-style lookup arguments

1. **Decomposes R_q elements into Z_B for some small B**
   - **It needs to commit to these decomposed elements**
2. **Proves that the decomposed values are in tables on size B**

CCC^{+}25 - Range proofs

1. Every key switching produces several: $||\mathbf{w}_{\mathrm{ks}}[i]|| < 2^{k}$
2. Every R_q element is decomposed in Z_B elements, which are committed to using a PC for R_q

CCC+25 - Range proofs

R_q has O(N) such elements

1. Every key switching produ… eral: $||\mathbf{w}_{\text{ks}}[i]|| < 2^k$
2. Every R_q element is decomposed in Z_B elements, which are committed to using a PC for R_q

Each element (input) in a PC for R_q has size at least O(N)

CCC+25 - Range proofs

R_q has O(N) such elements

1. Every key swit

2. Every R_q eleme e committed to using a PC for R_q

$O(N^2)$ total costs

Can't we use a PC for Z_q instead?

Each element (input) in a PC for R_q has size at least O(N)

CCC+25 - Range proofs

R_q has O(N) such elements

1. Every key swit

2. Every R_q eleme e committed to using a PC for R_q

$O(N^2)$ total costs

Can't we use a PC for Z_q instead?

Not yet! We still need R_q for the rest of the protocol

Virtual Oracles and Oracle Packing

The problem

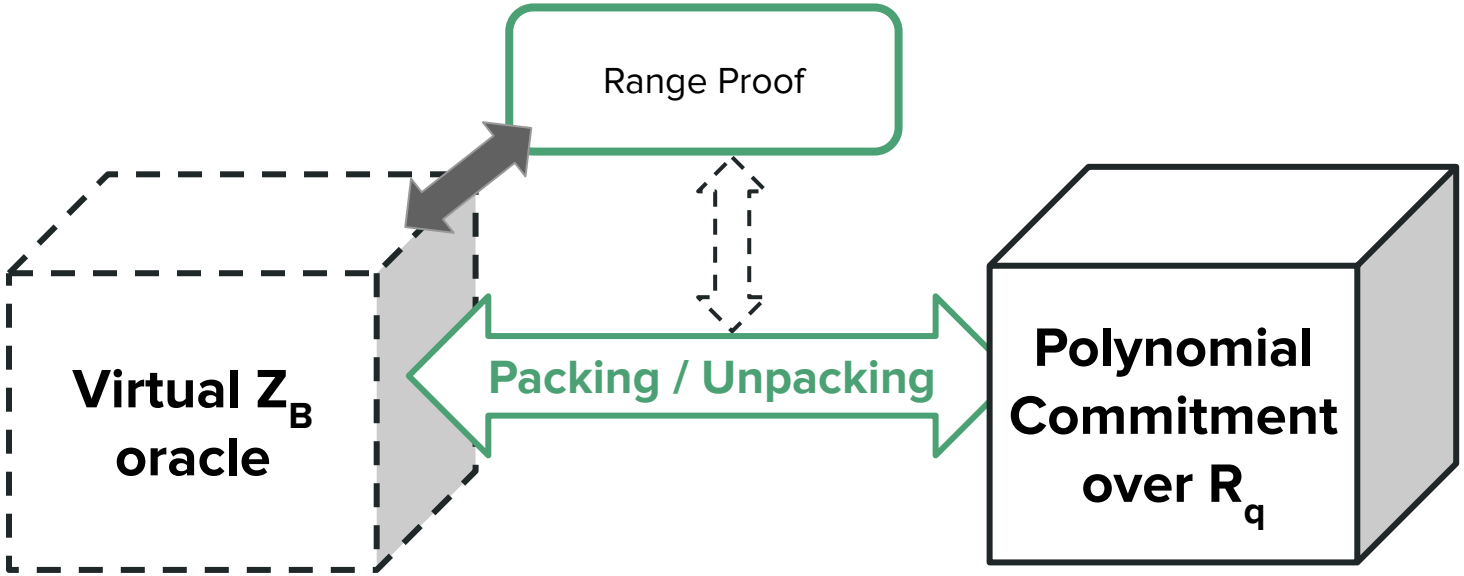
- We have <u>many</u> small Z_B elements we need to commit
- But we only have a polynomial commitment for R_q

Solution

- <u>Pack</u> many small Z_B elements in R_q
- Create virtual oracles for Z_B
- The rest of the protocol doesn't change!

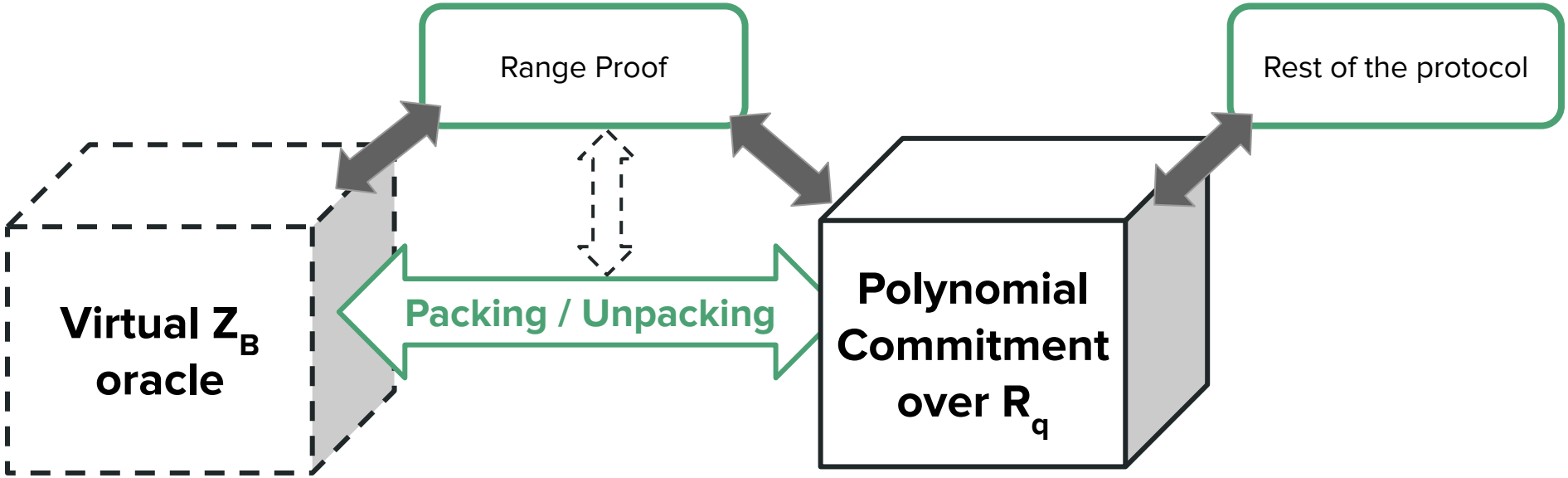
Solution

- Pack <u>many</u> small Z_B elements in R_q
- Create virtual oracles for Z_B
- The rest of the protocol doesn't change!



Solution

- Pack <u>many</u> small Z_B elements in R_q
- Create virtual oracles for Z_B
- The rest of the protocol doesn't change!



Solution

- Pack <u>many</u> small Z_B elements in R_q
- Create virtual oracles for Z_B
- The rest of the protocol doesn't change!

1. PC for range proofs are O(N) times faster
2. Beyond range proofs:

<u>We can use this whenever we need commitments to Z_q</u>

Rest of the protocol

Virtual Z_B oracle

Packing / Unpacking

Polynomial Commitment over R_q

Polynomial Commitment

Polynomial Commitments

We need a (multivariate) polynomial commitment scheme for our ring

- CCC^{+}25 uses **Brakedown** [1]
  - **Linear time prover**
  - **Square-root time verifier**

[1] A. Golovnev, J. Lee, S. Setty, J. Thaler, and R. S. Wahby, “Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS,” in Advances in Cryptology – CRYPTO 2023

Polynomial Commitments

We need a (multivariate) polynomial commitment scheme for our ring

- CCC^{+}25 uses **Brakedown** [1]
  - **Linear time prover**
  - **Square-root time verifier**
- Our commitments are O(N) smaller now, let's use **Basefold**[2]:
  - **Quasilinear prover**
  - **Polylog verifier**

[1] A. Golovnev, J. Lee, S. Setty, J. Thaler, and R. S. Wahby, "Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS," in Advances in Cryptology – CRYPTO 2023
[2] H. Zeilberger, B. Chen, and B. Fisch, "BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes," in Advances in Cryptology – CRYPTO 2024

Polynomial Commitments

We need a (multivariate) polynomial commitment scheme for our ring

- CCC^{+}25 uses **Brakedown** [1]
  - **Linear time prover**
  - **Square-root time verifier**
- Our commitments are O(N) smaller now, let's use **Basefold**[2]:
  - **Quasilinear prover**
  - **Polylog verifier**
- Can we do better?

[1] A. Golovnev, J. Lee, S. Setty, J. Thaler, and R. S. Wahby, "Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS," in Advances in Cryptology – CRYPTO 2023
[2] H. Zeilberger, B. Chen, and B. Fisch, "BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes," in Advances in Cryptology – CRYPTO 2024

Basefold → BatchFold

Oracle access to polynomial evaluations

$$p \in \mathcal{R}_q[X]$$

$$x, y \in \mathcal{R}_q$$

Prove: $y = p(x)$

Oracle access to polynomial evaluations

$$p_i \in \mathcal{R}_q[X] \qquad x_i, y_i \in \mathcal{R}_q$$

$$\texttt{Prove:}\ y_i = p_i(x_i)$$

Oracle access to polynomial evaluations

$$p_i \in \mathcal{R}_q[X] \qquad x_i, y_i \in \mathcal{R}_q$$

$$\textbf{for } i \leftarrow 0 \textbf{ to } O(\log_2 |C|) \textbf{ do:}$$

`Prove:` $y_i = p_i(x_i)$

Oracle access to polynomial evaluations

$$p_i \in \mathcal{R}_q[X] \qquad x_i, y_i \in \mathcal{R}_q$$

Let's batch them!

for $i \leftarrow$ [illegible] $\mathcal{O}(\log_2 |\mathcal{O}|)$ **do:**

Prove: $y_i = p_i(x_i)$

Original Basefold[1]

- Works over **Fields**
- For ***M*** **polynomials of size *n*:**

$$O(M \log^2 n)$$

[1] H. Zeilberger, B. Chen, and B. Fisch, “BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes,” in Advances in Cryptology – CRYPTO 2024

Our Batched Ring Basefold

- Works over our **proof-friendly CKKS Ring**
- For M polynomials of size n:

$$O(M \log n + \log^2 n)$$

Original Basefold[1]

- Works

- For *M* polynomials of size *n*:

$$O(M \log^2 n)$$

Our Batched Ring Basefold

- roof-friendly

- For M poly ials of size n:

$$O(M \log n + \underline{\log^2 n})$$

Big “constant” factor here!

[1] H. Zeilberger, B. Chen, and B. Fisch, “BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes,” in Advances in Cryptology – CRYPTO 2024

Original Basefold[1]

- Works

- For *M* po

$$O(M \log^2 n)$$

Our Batched Ring Basefold

- roof-friendly

- ials of size n:

$$O(M \log n + \log^2 n)$$

Big “constant” factor here!

Can we go further?

[1] H. Zeilberger, B. Chen, and B. Fisch, “BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes,” in Advances in Cryptology – CRYPTO 2024

Remember the **proof-friendly ring**

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

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

R_{p_1}

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

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

R_{p_2}

$$X^N + 1 = \prod_{i=1}^{k} (X^d - \zeta^{2i-1}) \mod p_2 \cong$$

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

R_{p_3}

$$X^N + 1 = \prod_{i=1}^{k} (X^d - \zeta^{2i-1}) \mod p_3 \cong$$

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

$$d = 1 \longrightarrow \mathbb{F}_{p_1} \qquad d = 2 \longrightarrow \mathbb{F}_{p_1^2}$$

Remember the Ring

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

- **Fully-splitting NTT-friendly ring:**

$$d = 1 \longrightarrow \mathbb{F}_{p_1}$$

- **Almost-Fully-splitting Almost-NTT-friendly ring:**

$$d = 2 \longrightarrow \mathbb{F}_{p_1^2}$$

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

$$R_q \cong \prod_{i=1}^{L} R_{p_i} \quad \text{where } q = \prod_{i=1}^{L} p_i$$

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

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

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

- Efficient arithmetic with incomplete NTTs!
- Soundness
  - Proportional to $|p_i^d|$

1. **These fields are big enough for soundness**
2. **For each prime p_i, they are isomorphic to each other.**

$$q[X]/(X^N+1)$$

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

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

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

q

R_{p_3}

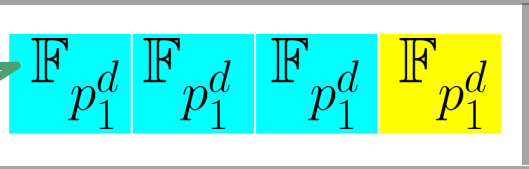
- Efficient arithmetic with incomplete NTTs!
- Soundness
  - Proportional to $|p_i^d|$

1. **These fields are big enough for soundness**
2. **For each prime p_i, they are isomorphic to each other.**

$$q[X]/(X^N+1)$$

$$\mod p_1$$

$$\mod p_2$$



Let's batch them!

- Efficient arithn
- Soundness
  - Proportional to $|p_i^d|$

Original Basefold[1]

- Works over **Fields** (adapted to rings as in CCC+25)
- For *M* polynomials of size *n*:

$$O(NLM \log^2 n)$$

[1] H. Zeilberger, B. Chen, and B. Fisch, "BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes," in Advances in Cryptology – CRYPTO 2024

BatchFold

- Works over $\mathbb{F}_{p^d}$
- Supports our **proof-friendly CKKS Ring** (by isomorphism)
- For M polynomials of size n:

$$O(NLM \log n + L \log^2 n)$$

N = Ring Degree

L = Number of RNS components

Origina

BatchFold

Big “constant” factor here!

- Works
  rings as in CCC+25)
- For *M* polynomials of size *n*:

$$O(NLM \log^2 n)$$

- ur proof-friendly
  CKKS
  y isomorphism)
- For M poly ls of size n:

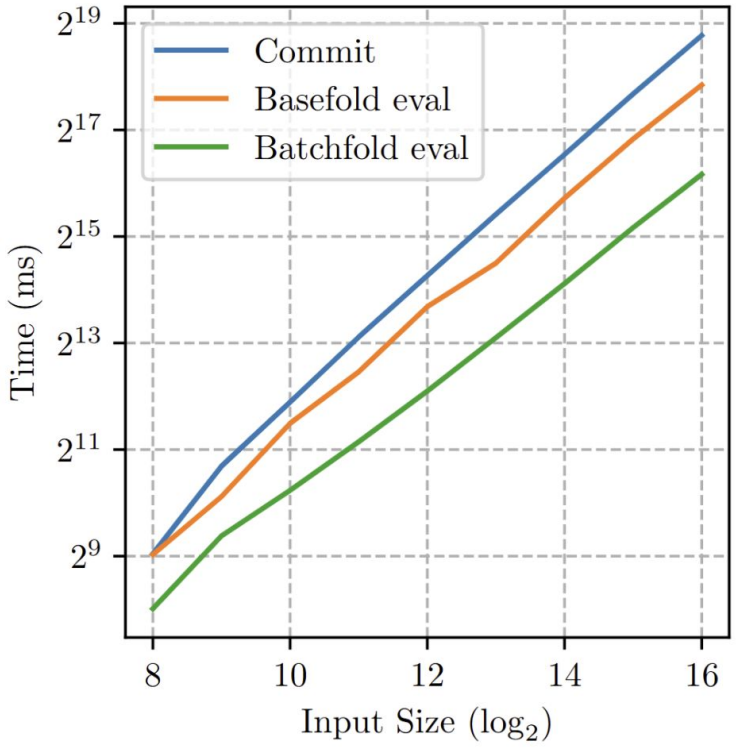
$$O(NLM \log n + \underline{L \log^2 n})$$

N = Ring Degree

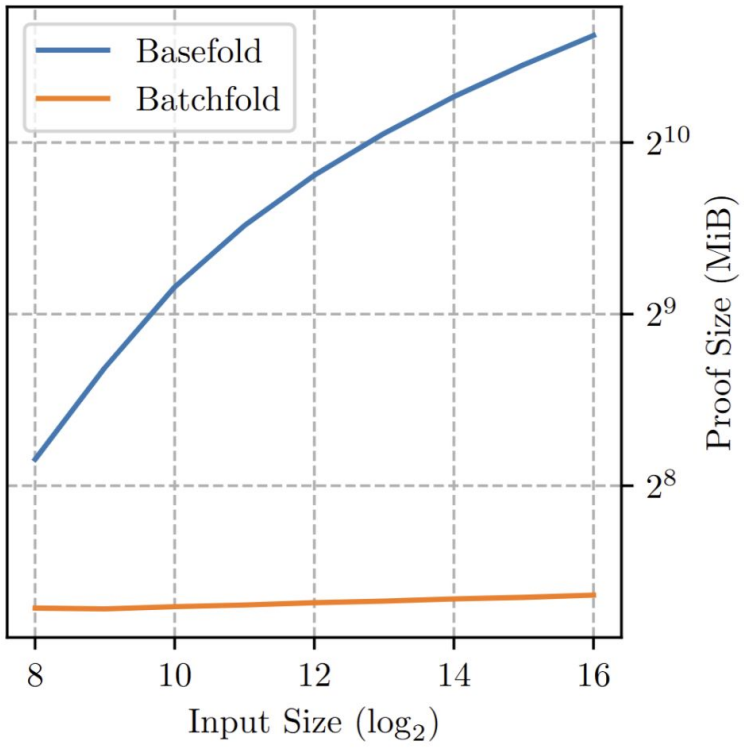
L = Number of RNS components

[1] H. Zeilberger, B. Chen, and B. Fisch, “BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes,” in Advances in Cryptology – CRYPTO 2024

BatchFold



(a) Execution time

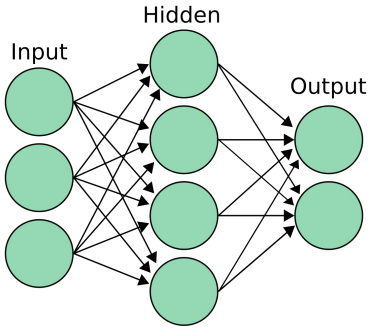


(b) Proof size

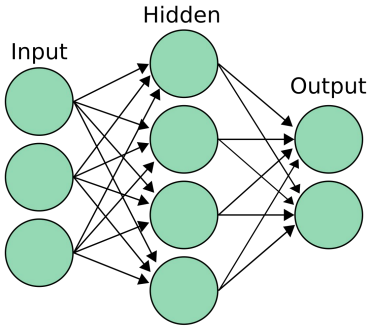
VFHE with GKR in practice

How do we prove a circuit?

```
Prove( HE(                    ) )
```



How do we prove a circuit?



NN Circuit
(using Keras)

Pure python
code
(* , + , -)

CKKS

HE evaluation
using CKKS

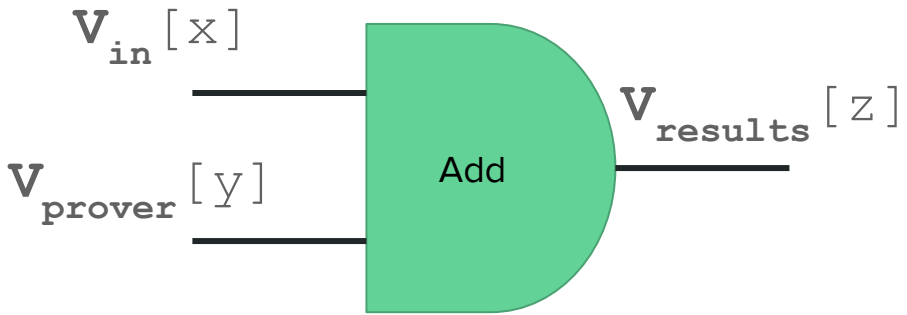
GKR

Custom GKR
circuit
(to prove HE
evaluation)

How do we **verify** this circuit?

GKR circuit example:

- Wires
  - V_{in}
  - V_{prover}
  - $V_{results}$
- Gates
  - Add



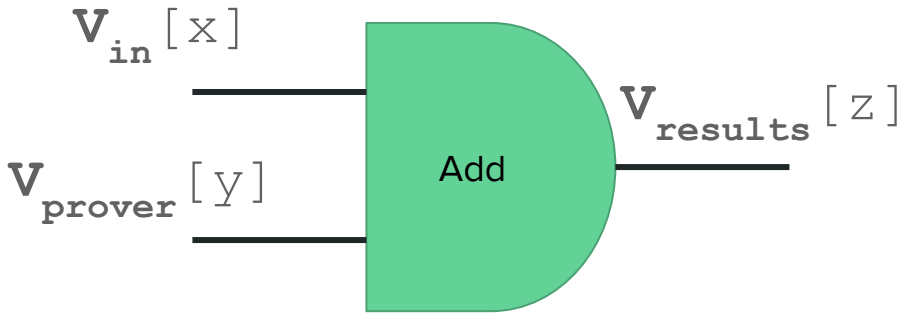
How do we **verify** this circuit?

Representing it as polynomials

Wires

1. Create a polynomial
   $p_{Vin} = 0$

2. **for** x = 0 **to** size(V_{in}):
   $p_{Vin}(x) = V_{in}[x]$



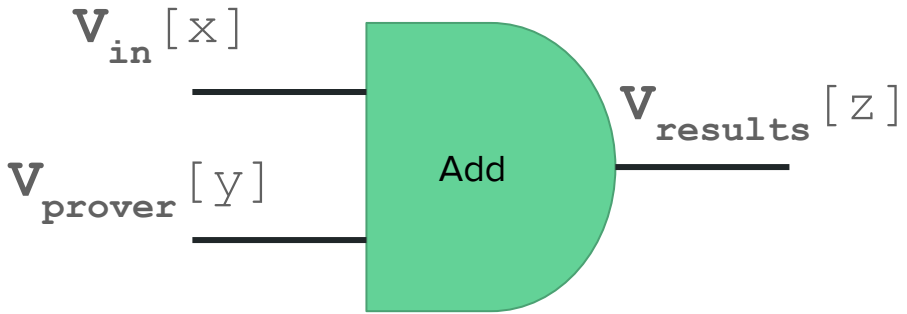
How do we **verify** this circuit?

Representing it as polynomials

Gates

1. Create a polynomial
   $p_{Add} = 0$

2. **for all** x,y,z:
   If Add(x,y,z) exists:
   $p_{add}(x|y|z) = 1$
   else:
   $p_{add}(x|y|z) = 0$



The verifier needs oracle access to these polynomials

Wires

1. Create a polynomial
   $p_{Vin} = 0$

2. **for** x = 0 **to** size(V_{in}):
   $p_{Vin}(x) = V_{in}[x]$

Gates

1. Create a polynomial
   $p_{Add} = 0$

2. **for all** x,y,z:
   If Add(x,y,z) exists:
   $p_{add}(x|y|z) = 1$
   else:
   $p_{add}(x|y|z) = 0$

The verifier needs oracle access to these polynomials

We can use a polynomial commitment for this one

Wires

1. Create a p[illegible]al
   $p_{Vin} = 0$

2. **for** x <= 0 **to** size(V_{in}):
   $p_{Vin}(x) = V_{in}[x]$

1. Create a polynomial
   $p_{Add} = 0$

2. **for all** x,y,z:
   If Add(x,y,z) exists:
   $p_{add}(x|y|z) = 1$
   else:
   $p_{add}(x|y|z) = 0$

The verifier needs oracle access to these polynomials

But this one is too big!

Wires

1. Create a polynomial
   $p_{Vin} = 0$

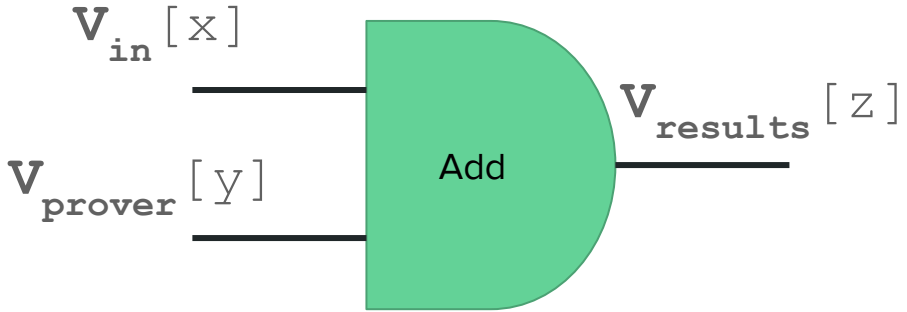
2. **for** x <- 0 **to** size(V_{in}):
   $p_{Vin}(x) = V_{in}[x]$

1. Create a polynomial
   $p = 0$

2. **for** all x,y,z:
   If Add(x,y,z) exists:
   $p_{add}(x|y|z) = 1$
   else:
   $p_{add}(x|y|z) = 0$

Solutions

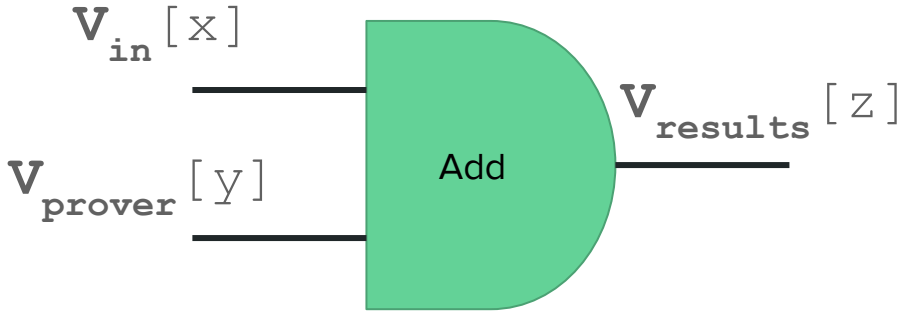
1. Log-space description
2. Sparse polynomial commitments



Solutions

1. Log-space description
2. Sparse polynomial commitments

We do both!



Improved Arithmetization

Gates

$$\texttt{rescon}_1(\mathbf{z},\mathbf{x}) = \begin{cases} 1, & \text{if wire } \mathbf{x} \text{ holds the } \mathbf{z}\text{-th } c' \text{ value.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\texttt{rescon}_3(\mathbf{z},\mathbf{x}) = \begin{cases} -1, & \text{if wire } \mathbf{x} \text{ holds the } \mathbf{z}\text{-th remainder.} \\ -\mathsf{p}_l, & \text{if } \mathbf{x} \text{ belongs to the } l\text{-th HE layer and holds the } \mathbf{z}\text{-th quotient.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\texttt{bdcon}_2(\mathbf{z},\mathbf{x}) = \begin{cases} 1, & \text{if wire } \mathbf{x} \text{ holds the } \mathbf{z}\text{-th } d_2 \text{ value.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\texttt{bdcon}_3(\mathbf{z},\mathbf{x}) = \begin{cases} -\mathsf{PW}_{\omega_l}(1)[i], & \text{if } \mathbf{x} \text{ belongs to the } l\text{-th HE layer and holds} \\ & \text{the } i\text{-th decomposition of the } \mathbf{z}\text{-th } d_2 \text{ value.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\texttt{evk}(\mathbf{z},\mathbf{x}) = \begin{cases} \mathsf{evk}_{l,b}[i], & \text{if wire } \mathbf{x} \text{ belongs to the } l\text{-th HE layer and} \\ & \text{holds the } i\text{-th decomposition of the } d_2 \text{ value} \\ & \text{corresponding to the } \mathbf{z}\text{-th } c'[b], \text{ where } b \in \{0,1\}. \\ 0, & \text{otherwise.} \end{cases}$$

$$\texttt{Xmult}(\mathbf{z},\mathbf{x},\mathbf{y}) = \begin{cases} 1, & \text{if } V(\mathbf{x}) \cdot V(\mathbf{y}) \text{ is added to } V_2(\mathbf{z}). \\ 0, & \text{otherwise.} \end{cases} \tag{25}$$

Prescriptive wires

	LSPs		Middle positions		MSPs	
	Bits	Meaning	Bits	Meaning	Bits	Meaning
$\tilde{V}_0$	$\mathtt{T}_0, \mathtt{T}_1$	Gate type	$\mathtt{Z}$	Gate number (for that type)	-	-
$\tilde{V}_1$	$\mathtt{Z}_{[\delta]}$	HE level	$\mathtt{Z}_\delta$	key-switch output type: c'_0 vs c'_1	$\mathtt{Z}_{(\delta,\mu)}$	wire on level
$\tilde{V}_{2,d_{01}}$	$\mathtt{Z}_{[\delta]}$	HE level	$\mathtt{Z}_\delta$	pre-mult output type: d_0 vs d_1	$\mathtt{Z}_{(\delta,\mu)}$	wire on level
$\tilde{V}_{2,d_2}$	$\mathtt{Z}_{[\delta]}$	HE level	$\mathtt{Z}_{[\delta,\mu-1)}$	wire carrying pre-mult output d_2	-	-
$\tilde{V}_2$	$\mathtt{Z}_0$	d_{01} vs d_2	As $\tilde{V}_{2,d_{01}}$ or $\tilde{V}_{2,d_2}$			
$\tilde{V}_{3,BD}$	$\mathtt{Z}_{[\delta]}$	HE level	$\mathtt{Z}_{[\delta,\mu-1)}$	wire carrying pre-mult output d_2	$\mathtt{X}_{[\delta]}$	specific p_i
$\tilde{V}_{3,quot}$	$\mathtt{Y}_{[\delta]}$	HE level	$\mathtt{Y}_\delta$	mult output type: c''_0 vs c''_1	$\mathtt{Y}_{(\delta,\mu)}$	wire on level
$\tilde{V}_{3,rc_3}$	$\mathtt{Z}_{[\delta]}$	HE level	$\mathtt{Z}_{[\delta,\mu)}$	$\mathtt{Z}_\delta$: c''_0 vs c''_1 | $\mathtt{Z}_{(\delta,\mu)}$: wire on level	$\mathtt{X}_0$	rmd vs quot

Table 1: Variable information for the different intermediate-values MLEs. LSPs = Least significant positions, MSPs = Most significant positions.

SparsePack

Problem: Existing **Sparse** Polynomial Commitments (e.g. Spark) are not efficient enough for R_q

Solution:

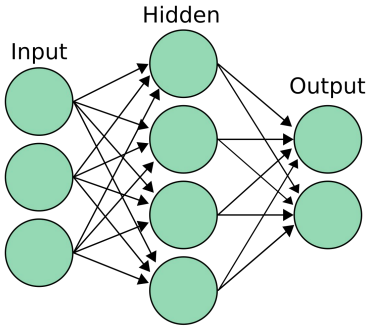
Oracle Packing + **BatchFold** → **SparsePack**

Results

Applications

We consider 3 small Neural Network circuits

- Small NNs for binary classification:
  - 30 inputs → 2 outputs
  - 2 hidden layers each
- Activation: $f(x) = x^2$



Model	Ctxt Multiplications	Circuit size	Dense circuit size
(128, 64)	192	2^{37}	2^{15}
(64, 16)	80	2^{33}	2^{13}
(32, 16)	28	2^{30}	2^{12}

Spark vs SparsePack

Table 2: Performance of our SparsePack and comparison with Spark. Notice that our solution does not require execution-time commitments.

Model (n_1, n_2)	Spark					SparsePack			
	`SumCheck`		`Comm.`	`Eval`		`SumCheck`		`Eval`	
	Size	Time	Time	Size	Time	Size	Time	Size	Time
$(128, 64)$	867.9 MiB	5m54s	20m57s	3.8 GiB	14m42s	54.3 MiB	25m58s	162.7 MiB	1m
$(64, 16)$	666.0 MiB	1m31s	20m57s	3.8 GiB	3m31s	48.8 MiB	6m40s	159.5 MiB	22s
$(32, 16)$	575.1 MiB	45s	20m57s	3.7 GiB	1m45s	36.1 MiB	1m20s	159.5 MiB	7s

Range proofs

Single-threaded execution:

Table 3: Timing performance of our range proofs

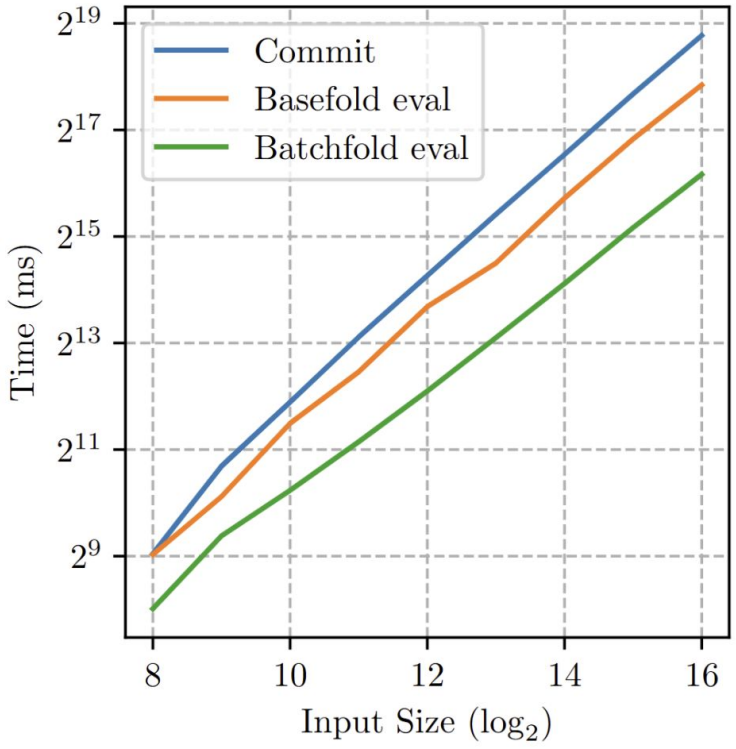
Model (n_1, n_2)	SumCheck	Com	Eval	Total
$(128, 64)$	12s	9m27s	5m8s	14m47s
$(64, 16)$	7s	4m10s	2m33s	6m49s
$(32, 16)$	6s	1m59s	1m27s	3m32s

CCC+25: 25s per multiplication in **48** threads

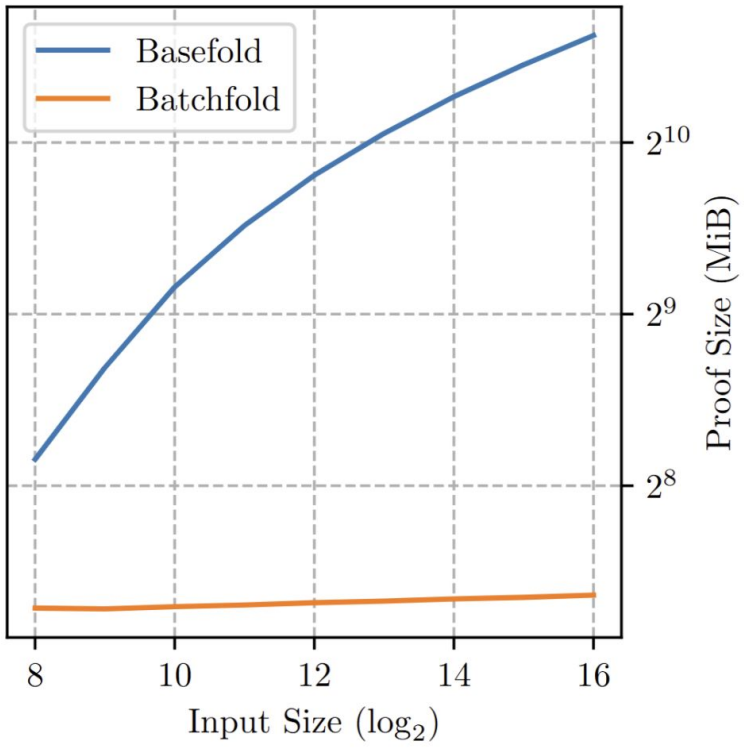
Thank you!

This work is supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's 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/; by grant JDC2023-050791-I, funded by MCIN/AEI/10.13039/501100011033 and the ESF+; by grant JDC2024-055789-I, funded by MCIN/AEI/10.13039/501100011033 and the ESF+; and by grant CEX2024001471-M funded by MICIU/AEI/10.13039/50110001103.

BatchFold



(a) Execution time



(b) Proof size

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