Garuda and Pari: Faster and Smaller SNARKs via Equifficient Polynomial Commitments

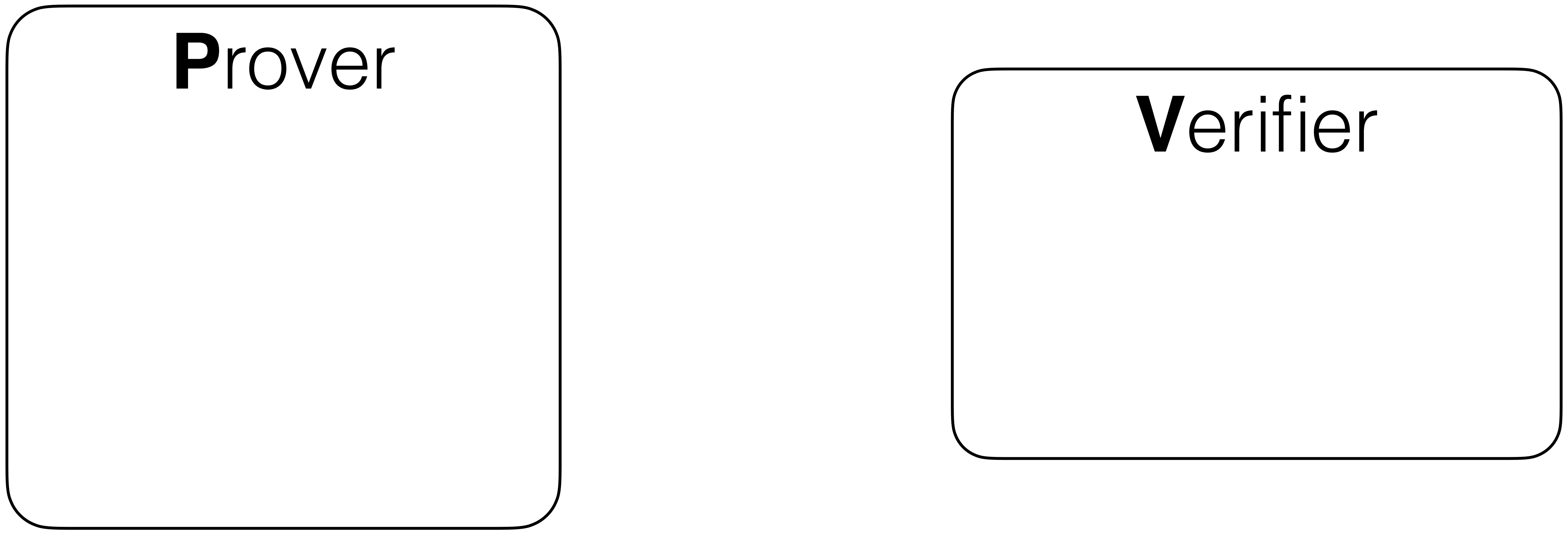
Michel Dellepere
Ava Labs

Pratyush Mishra
UPenn

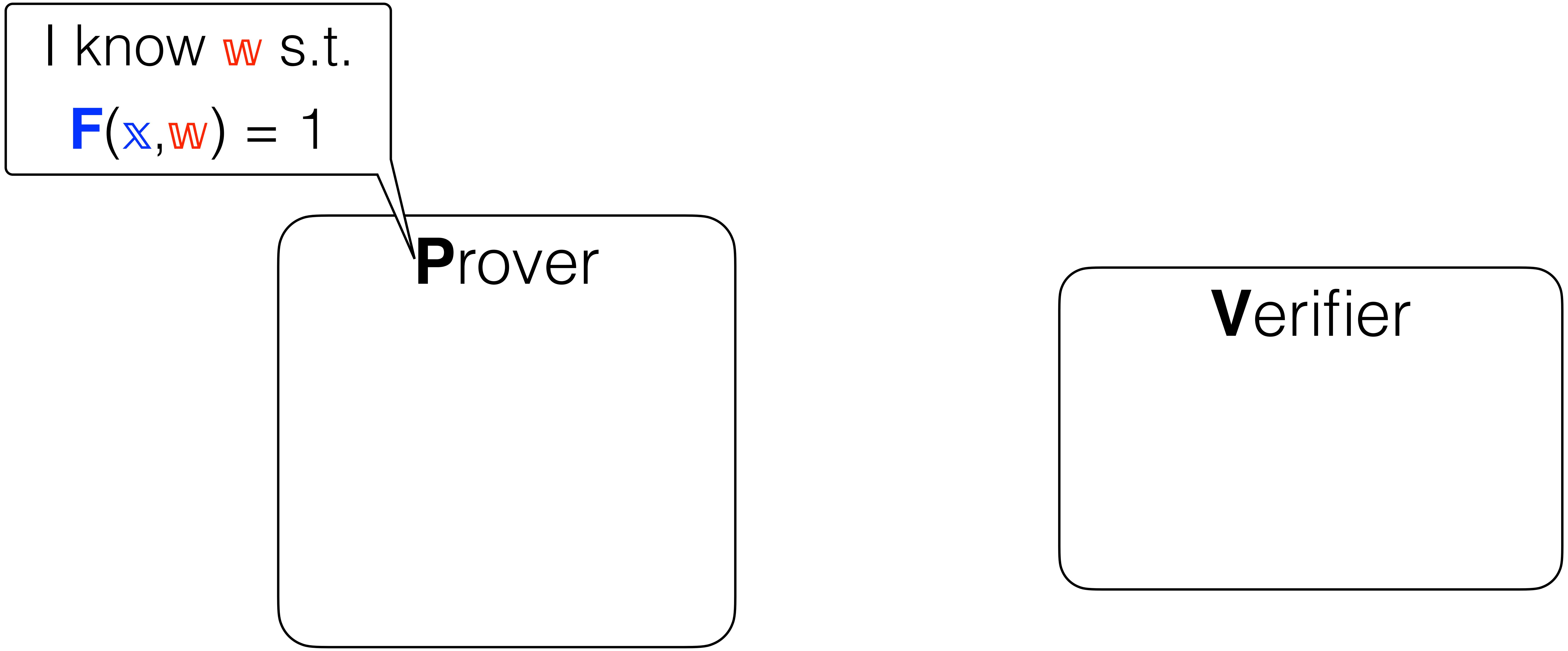
Alireza Shirzad
UPenn

Succinct Non-Interactive Argument of Knowledge (SNARK)

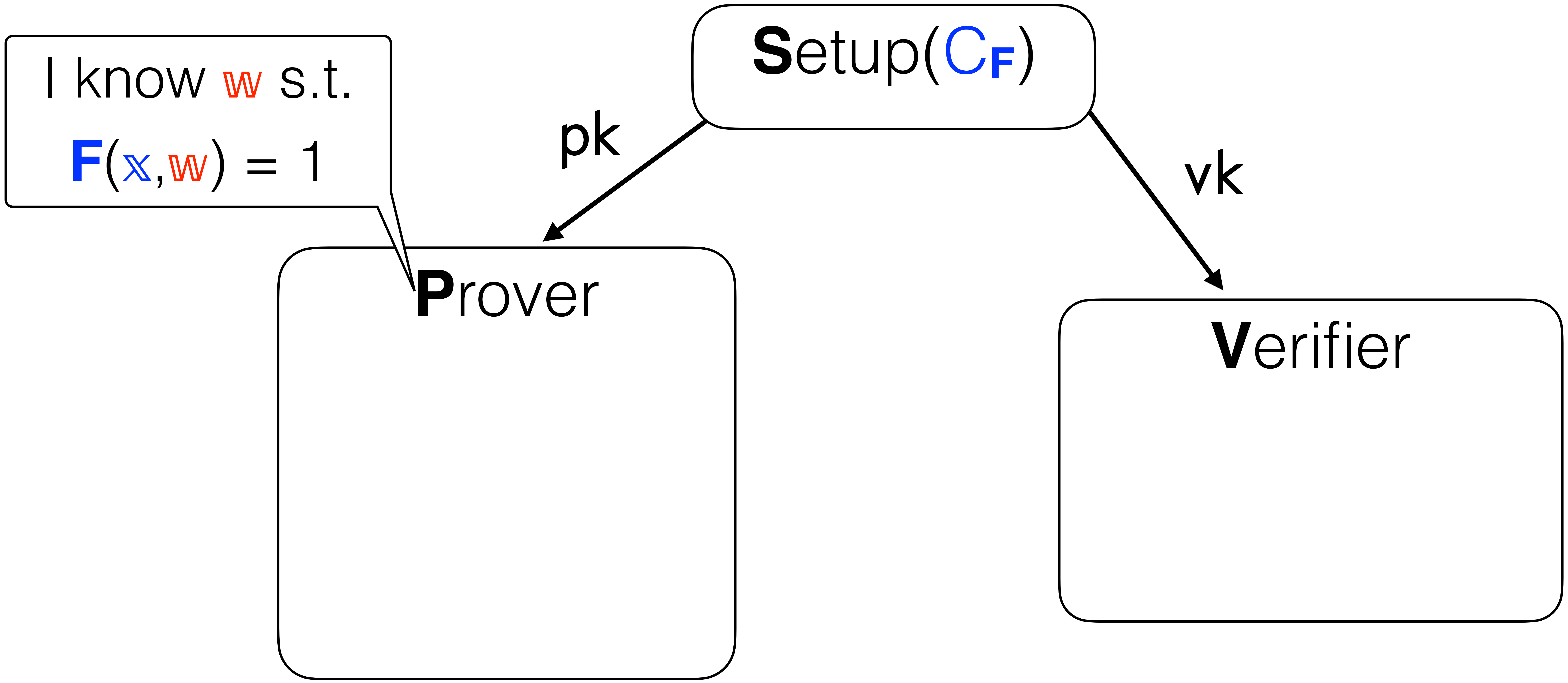
Succinct Non-Interactive Argument of Knowledge (SNARK)



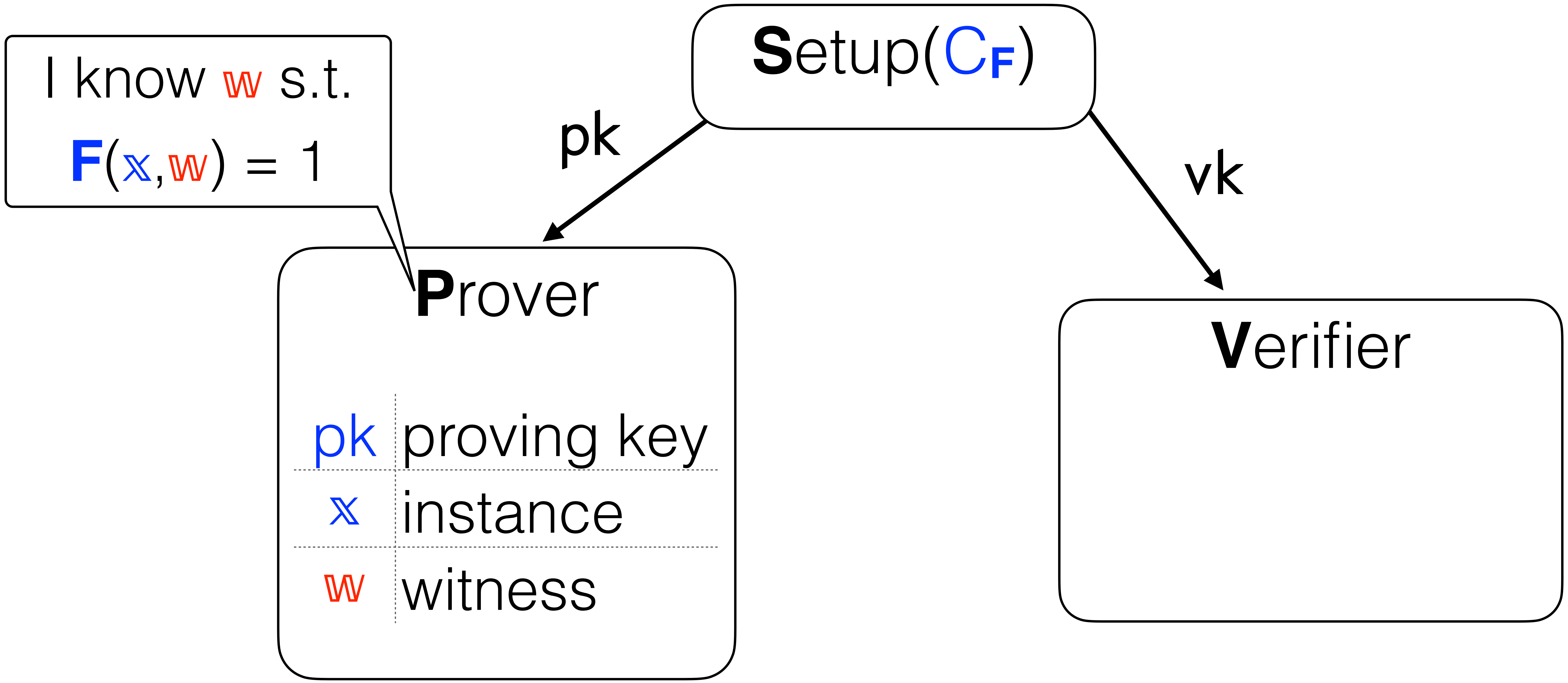
Succinct Non-Interactive Argument of Knowledge (SNARK)



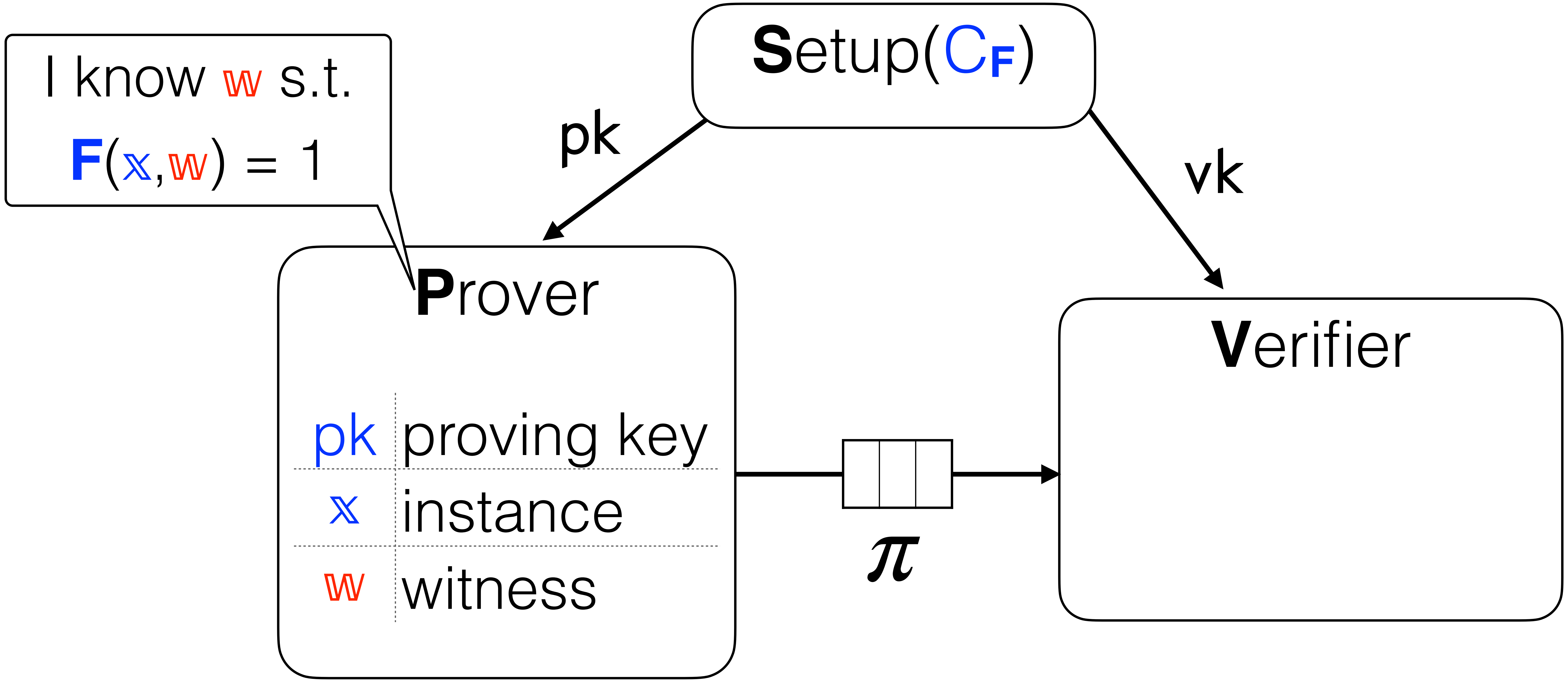
Succinct Non-Interactive Argument of Knowledge (SNARK)



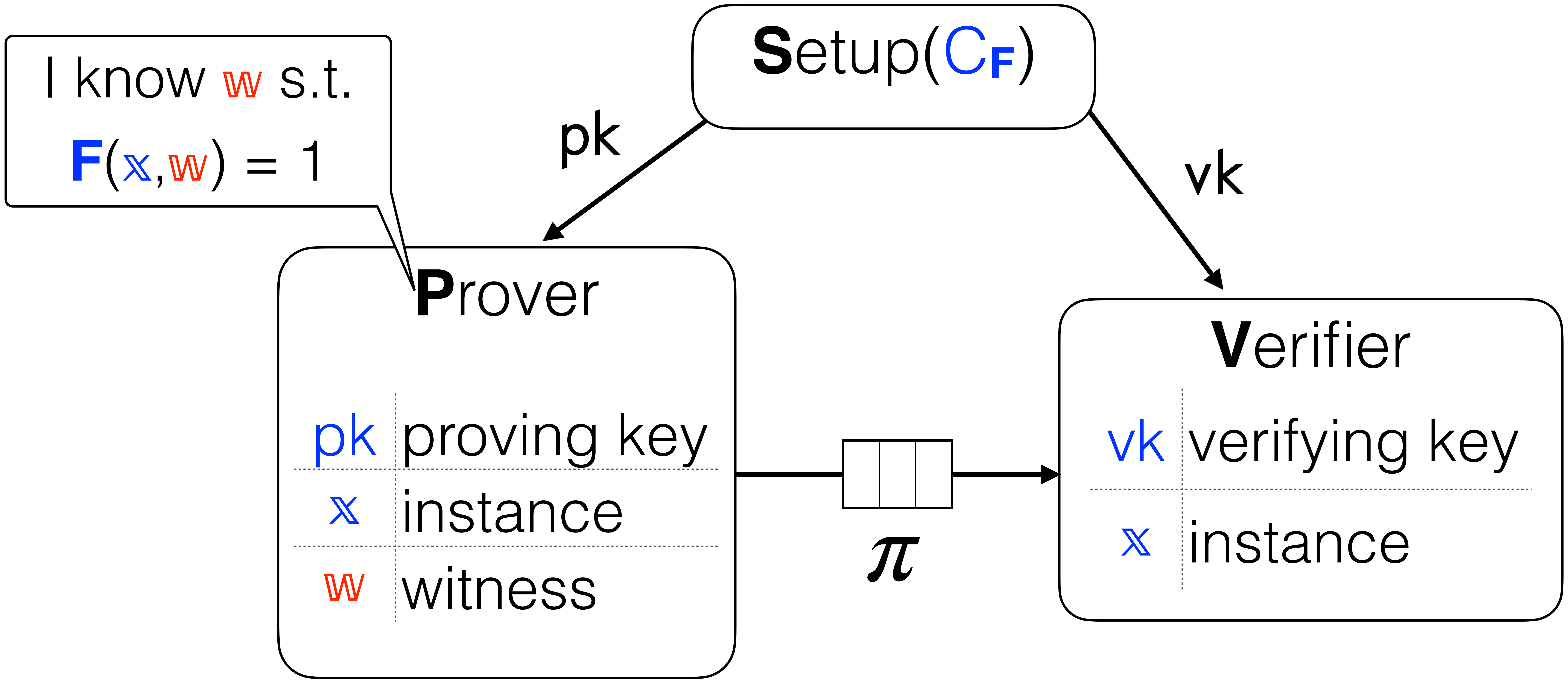
Succinct Non-Interactive Argument of Knowledge (SNARK)



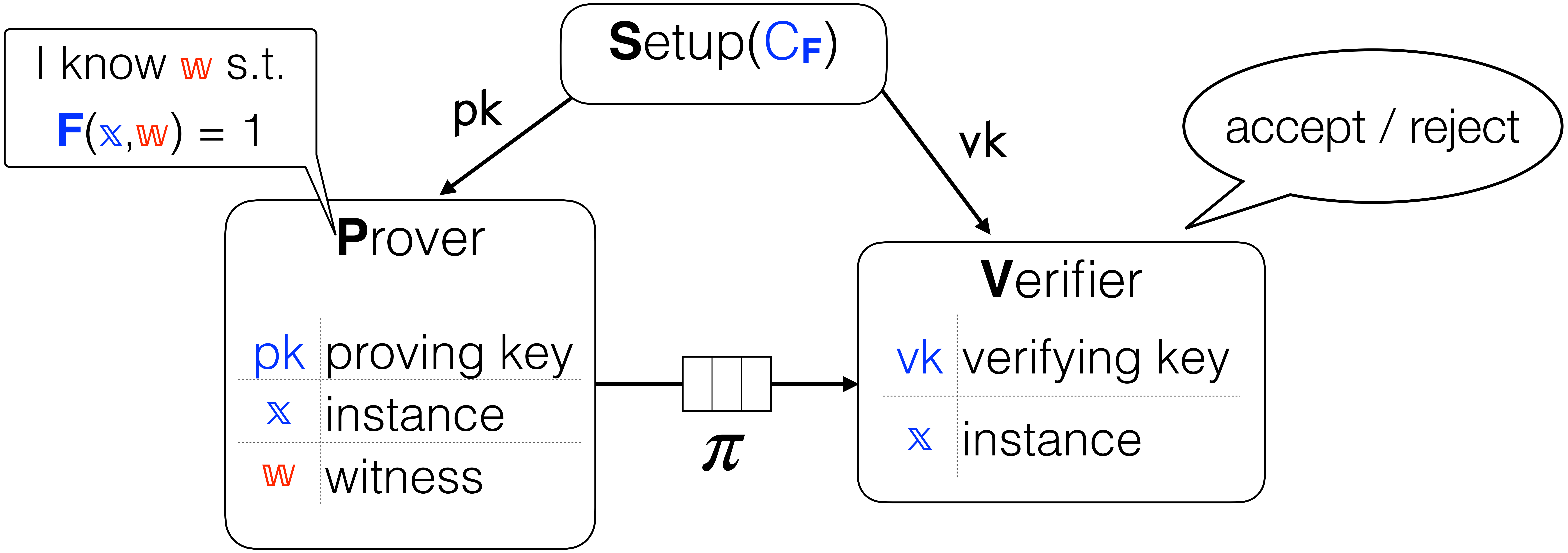
Succinct Non-Interactive Argument of Knowledge (SNARK)



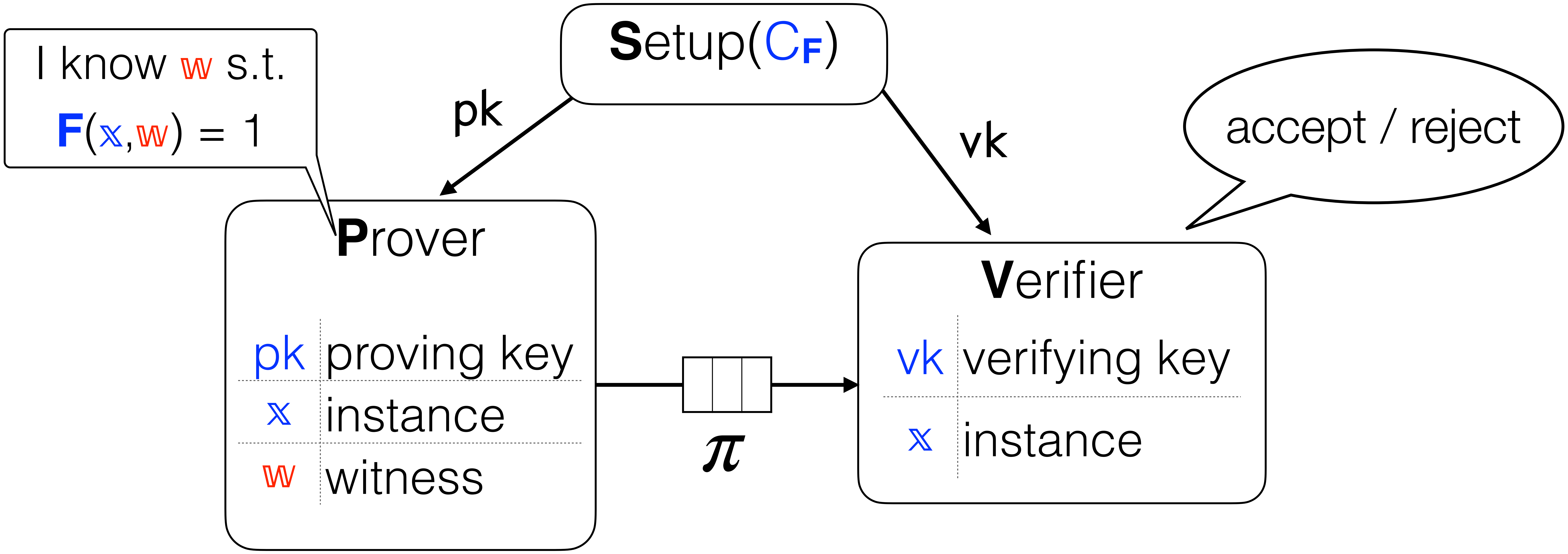
Succinct Non-Interactive Argument of Knowledge (SNARK)



Succinct Non-Interactive Argument of Knowledge (SNARK)

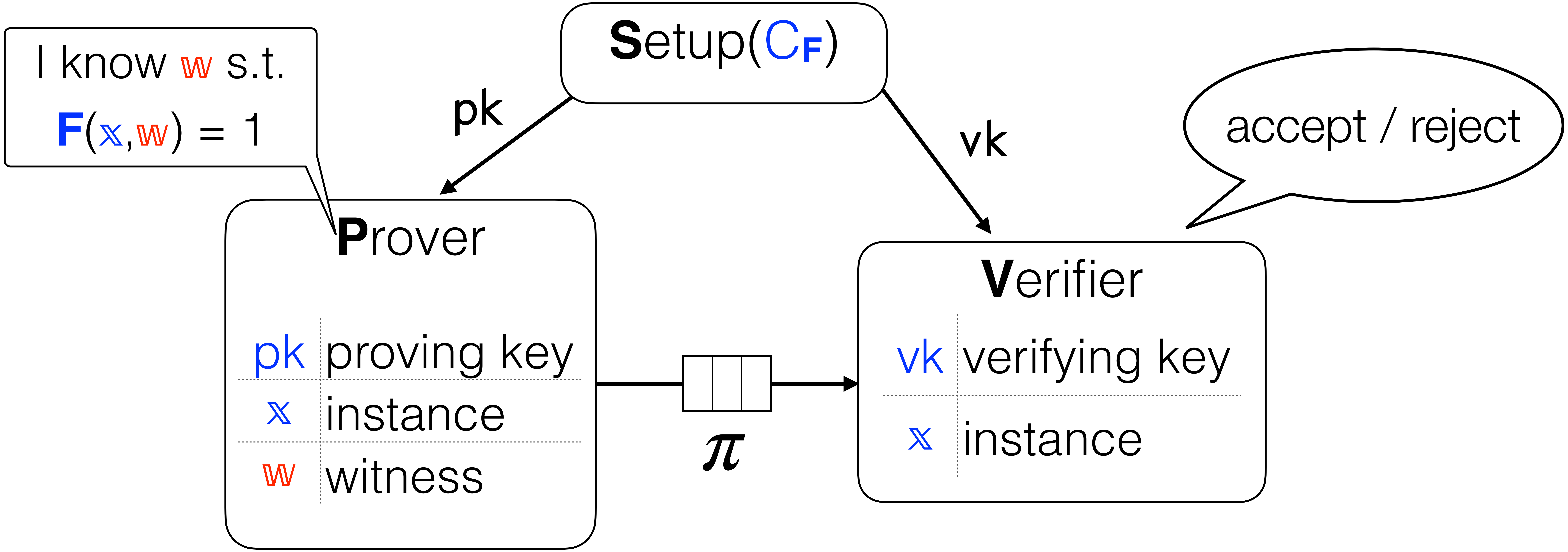


Succinct Non-Interactive Argument of Knowledge (SNARK)



Completeness: If **P** knows valid w, then **V** accepts the proof π

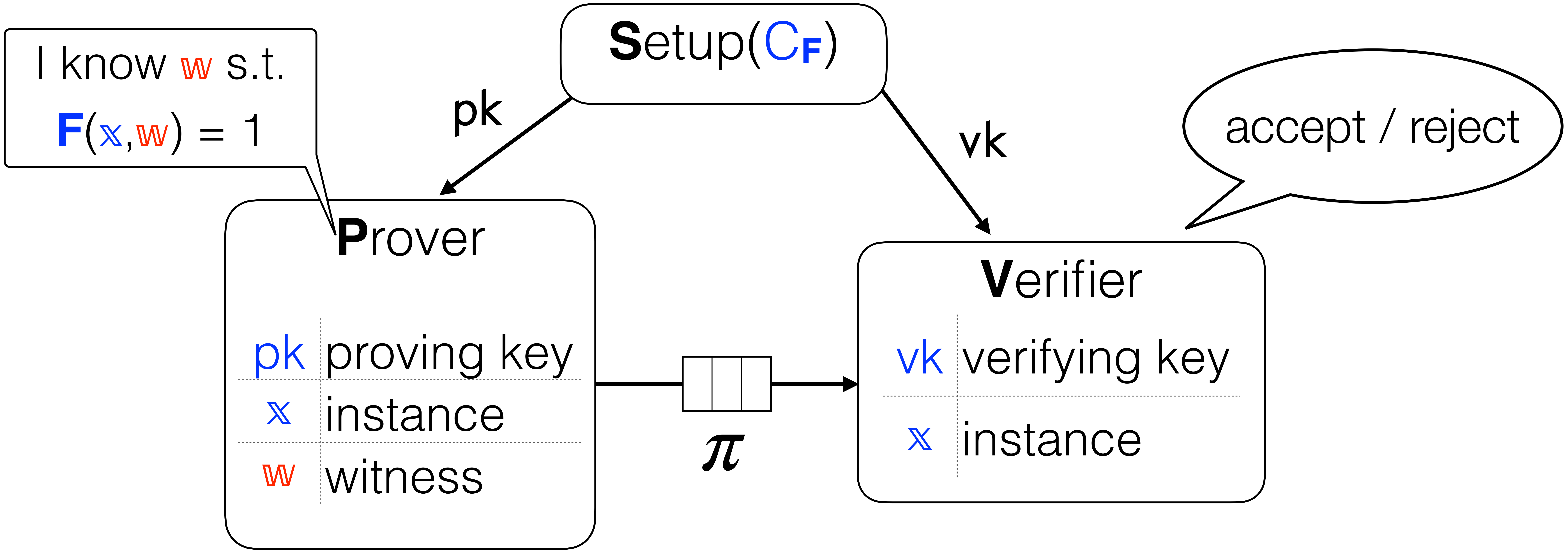
Succinct Non-Interactive Argument of Knowledge (SNARK)



Completeness: If **P** knows valid $\mathbb{w}$, then **V** accepts the proof π

Knowledge Soundness: If **P** does not know a valid witness $\mathbb{w}$, then **V** rejects π

Succinct Non-Interactive Argument of Knowledge (SNARK)



Completeness: If **P** knows valid w, then **V** accepts the proof π

Knowledge Soundness: If **P** does not know a valid witness w, then **V** rejects π

Succinctness: Size of proof π and verifier running time are much smaller than running time of **F**

Q1: How small can the proof π be?

Q1: How small can the proof π be?

For blockchains, smaller is better!

Q1: How small can the proof π be?

For blockchains, smaller is better!

<u>Groth16 lower bound</u>
Pairing-based SNARKs in GGM contain at least 2 group elements:
$|\pi| \geq 1\mathbb{G}_1 + 1\mathbb{G}_2$

144 bytes

$|\pi|$
(on BLS12-381)

Q1: How small can the proof π be?

For blockchains, smaller is better!

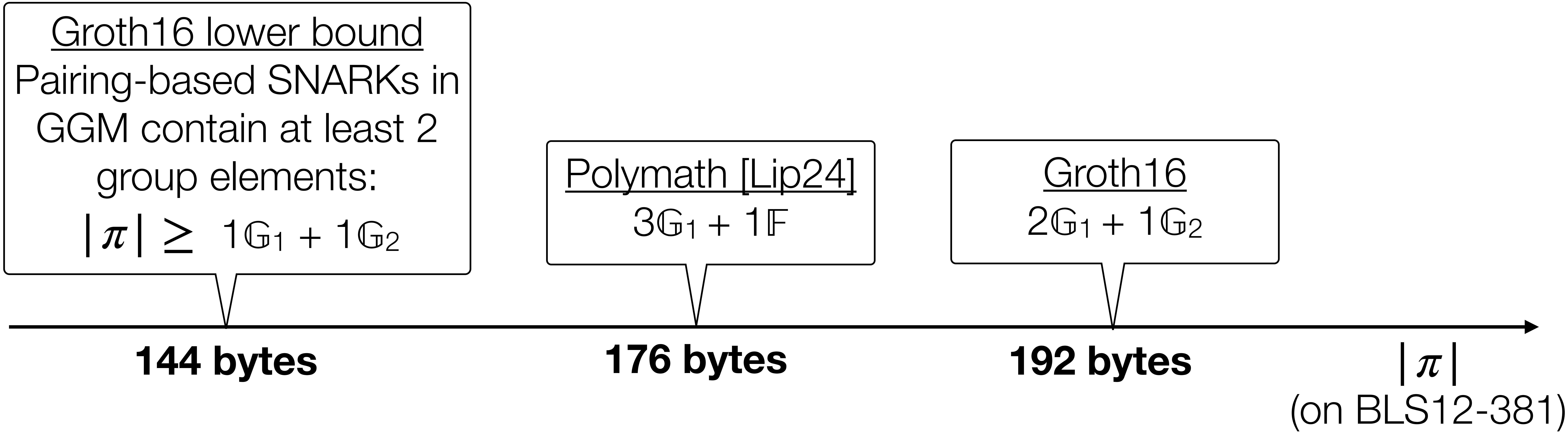
<u>Groth16 lower bound</u>
Pairing-based SNARKs in GGM contain at least 2 group elements:
$|\pi| \geq 1\mathbb{G}_1 + 1\mathbb{G}_2$

<u>Groth16</u>
$2\mathbb{G}_1 + 1\mathbb{G}_2$

144 bytes **192 bytes** $|\pi|$
(on BLS12-381)

Q1: How small can the proof π be?

For blockchains, smaller is better!



Q1: How small can the proof π be?

For blockchains, smaller is better!

<u>Groth16 lower bound</u>
Pairing-based SNARKs in GGM contain at least 2 group elements:
$|\pi| \geq 1\mathbb{G}_1 + 1\mathbb{G}_2$

144 bytes

<u>Polymath [Lip24]</u>
$3\mathbb{G}_1 + 1\mathbb{F}$

176 bytes

<u>Groth16</u>
$2\mathbb{G}_1 + 1\mathbb{G}_2$

192 bytes

$|\pi|$
(on BLS12-381)

Can we go lower than 176 bytes?

Q2: How fast can we prove?

Q2: How fast can we prove?

Proving has a large overhead (~1000x) over native computation

How to reduce this cost?

Q2: How fast can we prove?

Proving has a large overhead (~1000x) over native computation

How to reduce this cost?

Q2: How fast can we prove?

Proving has a large overhead (~1000x) over native computation

How to reduce this cost?

Approach 1: Free addition gates

Q2: How fast can we prove?

Proving has a large overhead (~1000x) over native computation

How to reduce this cost?

Approach 1: Free addition gates

- Only pay cryptographic (e.g., MSM) costs for multiplication gates
- Achieved by circuit-specific SNARKs [GGPR13, BCTV14, Groth16]

Q2: How fast can we prove?

Proving has a large overhead (~1000x) over native computation

How to reduce this cost?

Approach 1: Free addition gates

- Only pay cryptographic (e.g., MSM) costs for multiplication gates
- Achieved by circuit-specific SNARKs [GGPR13, BCTV14, Groth16]

Approach 2: Custom gates

Q2: How fast can we prove?

Proving has a large overhead (~1000x) over native computation

How to reduce this cost?

Approach 1: Free addition gates

- Only pay cryptographic (e.g., MSM) costs for multiplication gates
- Achieved by circuit-specific SNARKs [GGPR13, BCTV14, Groth16]

Approach 2: Custom gates

- Specialized gates for particular computations (e.g., EC addition, Poseidon S-box)
- Proposed recently for TurboPlonk [GW19], used widely [RISC0,Plonky3,CBBZ23,STW23]

Q2: How fast can we prove?

Proving has a large overhead (~1000x) over native computation

How to reduce this cost?

Approach 1: Free addition gates

- Only pay cryptographic (e.g., MSM) costs for multiplication gates
- Achieved by circuit-specific SNARKs [GGPR13, BCTV14, Groth16]

Approach 2: Custom gates

- Specialized gates for particular computations (e.g., EC addition, Poseidon S-box)
- Proposed recently for TurboPlonk [GW19], used widely [RISC0,Plonky3,CBBZ23,STW23]

Unfortunately, no existing SNARK supports both!

Can we fix this?

Our Contributions

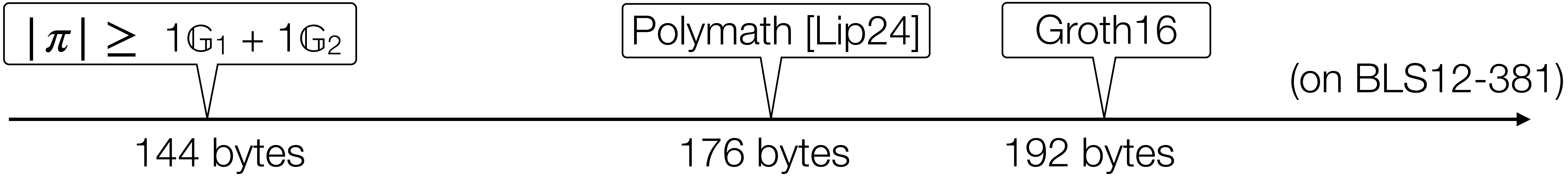
Garuda and Pari

Garuda and Pari

Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$

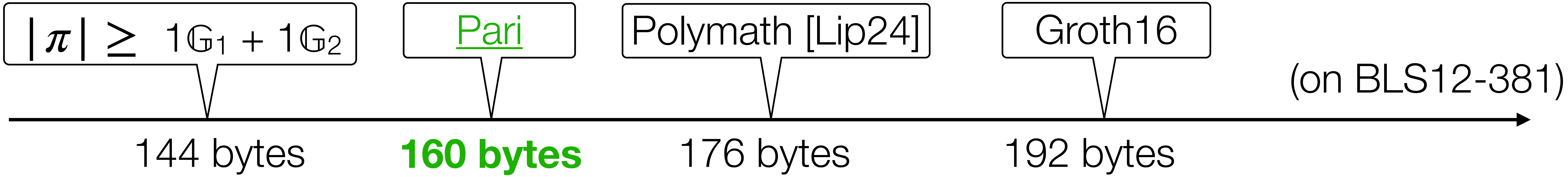
Garuda and Pari

Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$



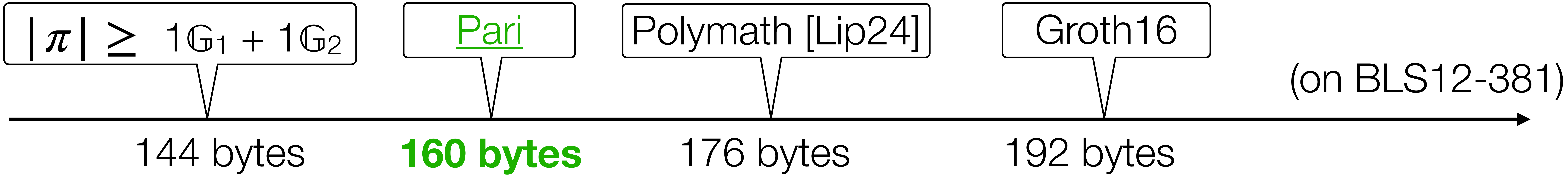
Garuda and Pari

Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$



Garuda and Pari

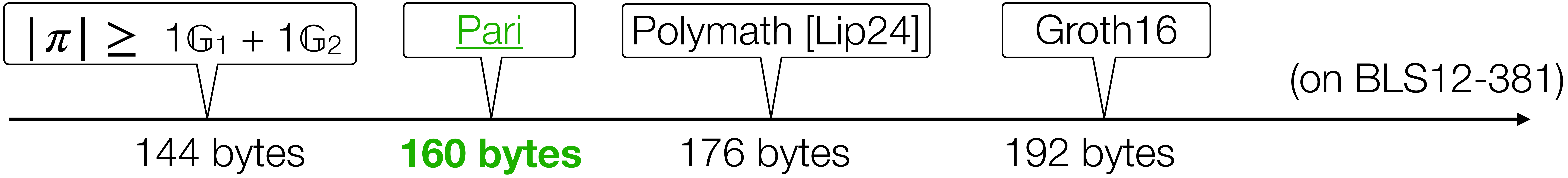
Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$



Garuda: The first SNARK with free linear gates that support custom gates

Garuda and Pari

Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$



Garuda: The first SNARK with free linear gates that support custom gates

> 3x faster than Groth16 (free linear gates)
> 2x faster than HyperPlonk (custom gates)

Garuda and Pari

Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$

$\|\pi\| \geq 1\mathbb{G}_1 + 1\mathbb{G}_2$	Pari	Polymath [Lip24]	Groth16
144 bytes	**160 bytes**	176 bytes	192 bytes

(on BLS12-381)

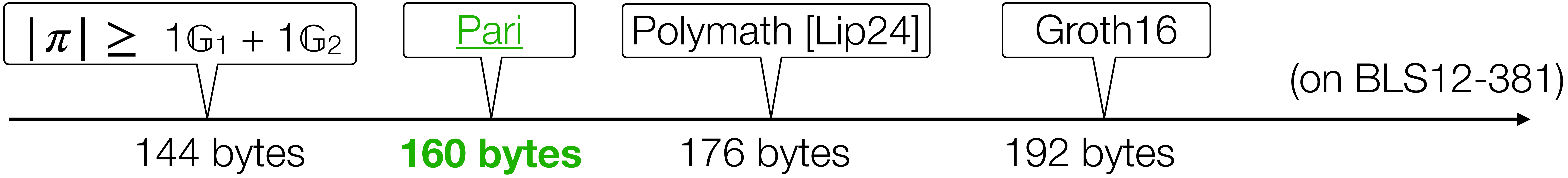
Garuda: The first SNARK with free linear gates that support custom gates

> 3x faster than Groth16 (free linear gates)
> 2x faster than HyperPlonk (custom gates)

Both in ROM + AGM

Garuda and Pari

Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$



Garuda: The first SNARK with free linear gates that support custom gates

> 3x faster than Groth16 (free linear gates)
> 2x faster than HyperPlonk (custom gates)

Both in ROM + AGM

Both Circuit-Specific

Garuda and Pari

Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$

$|\pi| \geq 1\mathbb{G}_1 + 1\mathbb{G}_2$ — 144 bytes

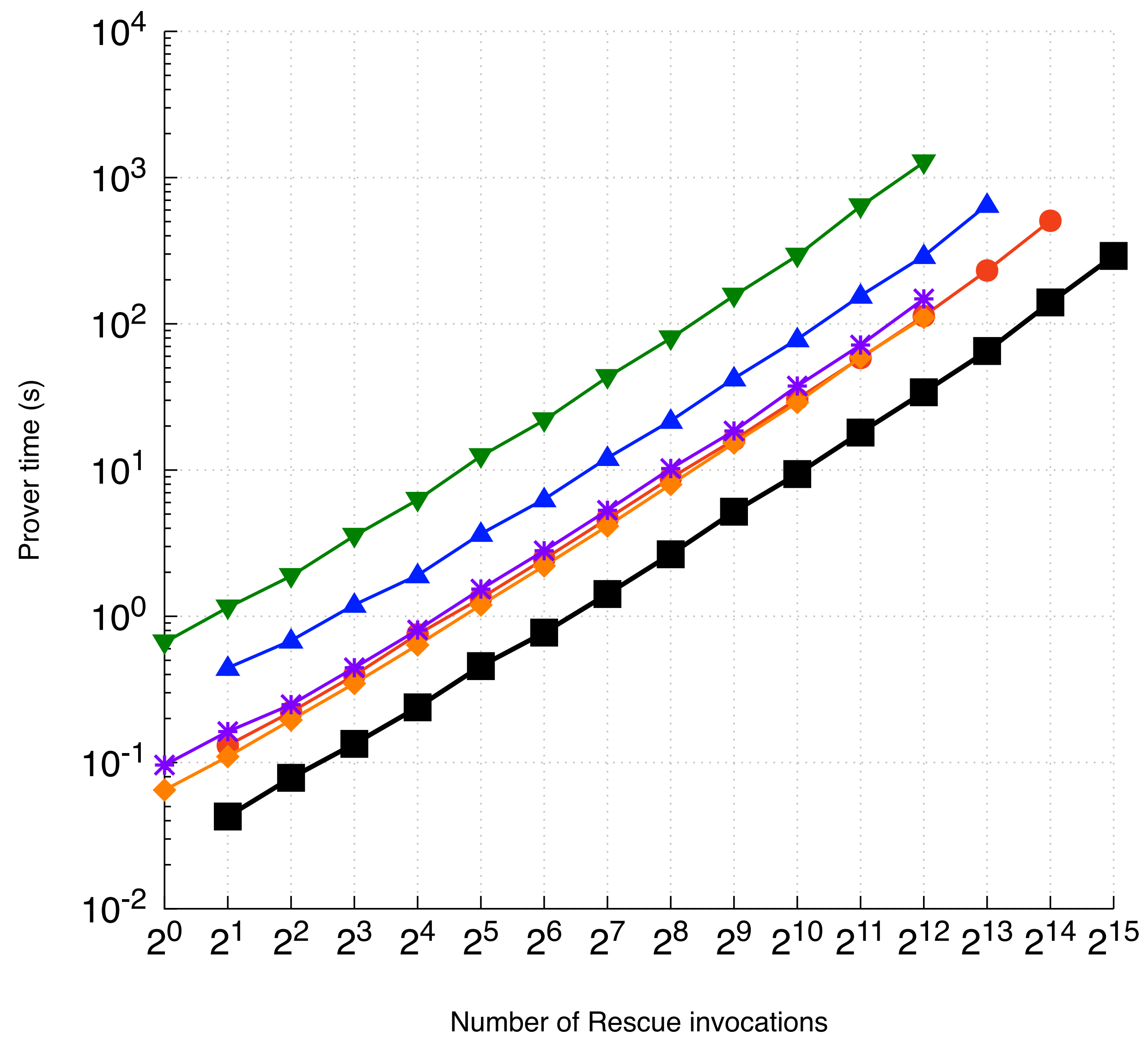
Pari — **160 bytes**

Polymath [Lip24] — 176 bytes

Groth16 — 192 bytes

(on BLS12-381)

Garuda: The first SNARK with free linear gates that support custom gates



> 3x faster than Groth16 (free linear gates)

> 2x faster than HyperPlonk (custom gates)

Both in ROM + AGM

Both Circuit-Specific

Garuda and Pari

Pari: The smallest known SNARK with proof size $|\pi| = 2\mathbb{G}_1 + 2\mathbb{F}$

$|\pi| \geq 1\mathbb{G}_1 + 1\mathbb{G}_2$ — 144 bytes

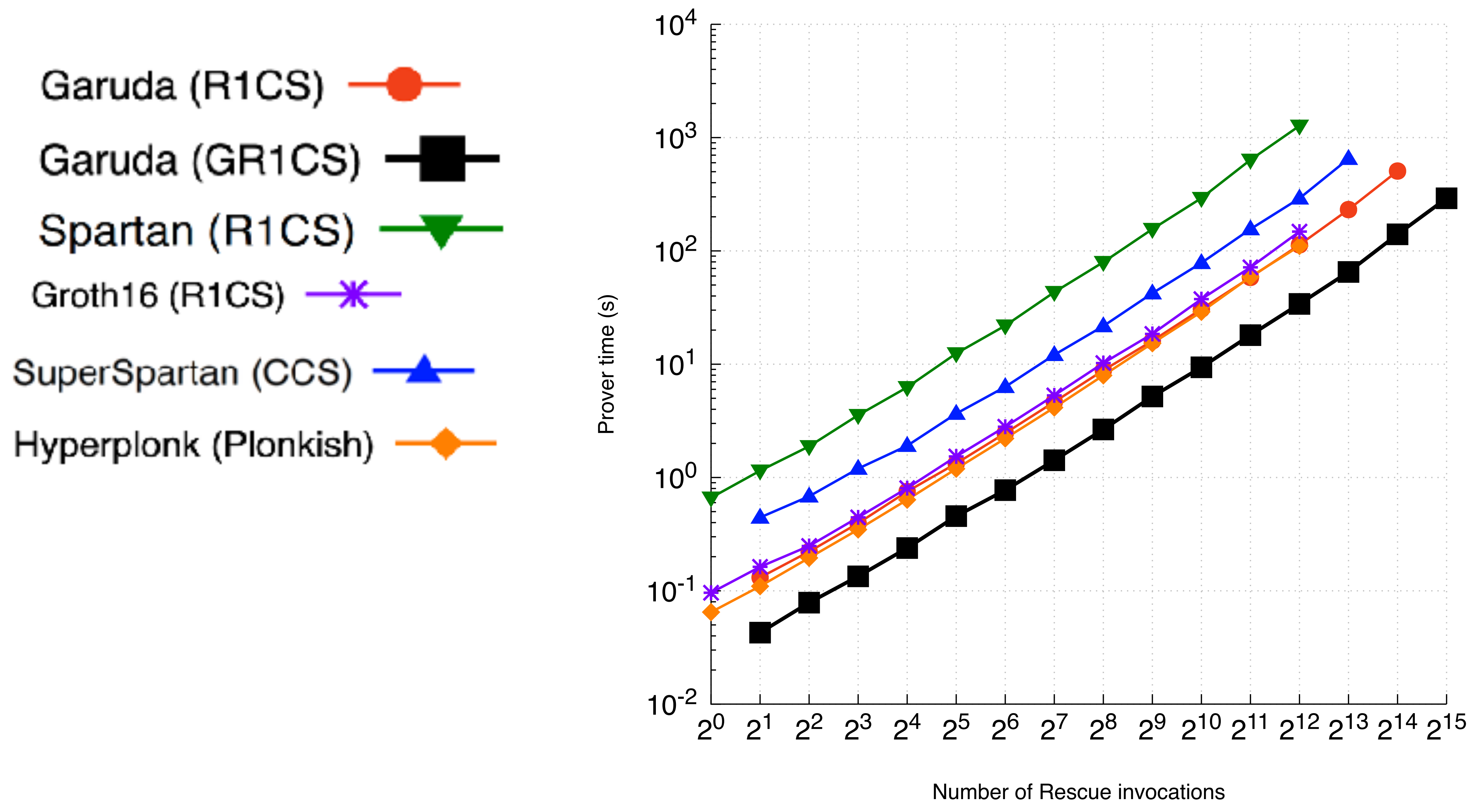
Pari — **160 bytes**

Polymath [Lip24] — 176 bytes

Groth16 — 192 bytes

(on BLS12-381)

Garuda: The first SNARK with free linear gates that support custom gates



> 3x faster than Groth16 (free linear gates)

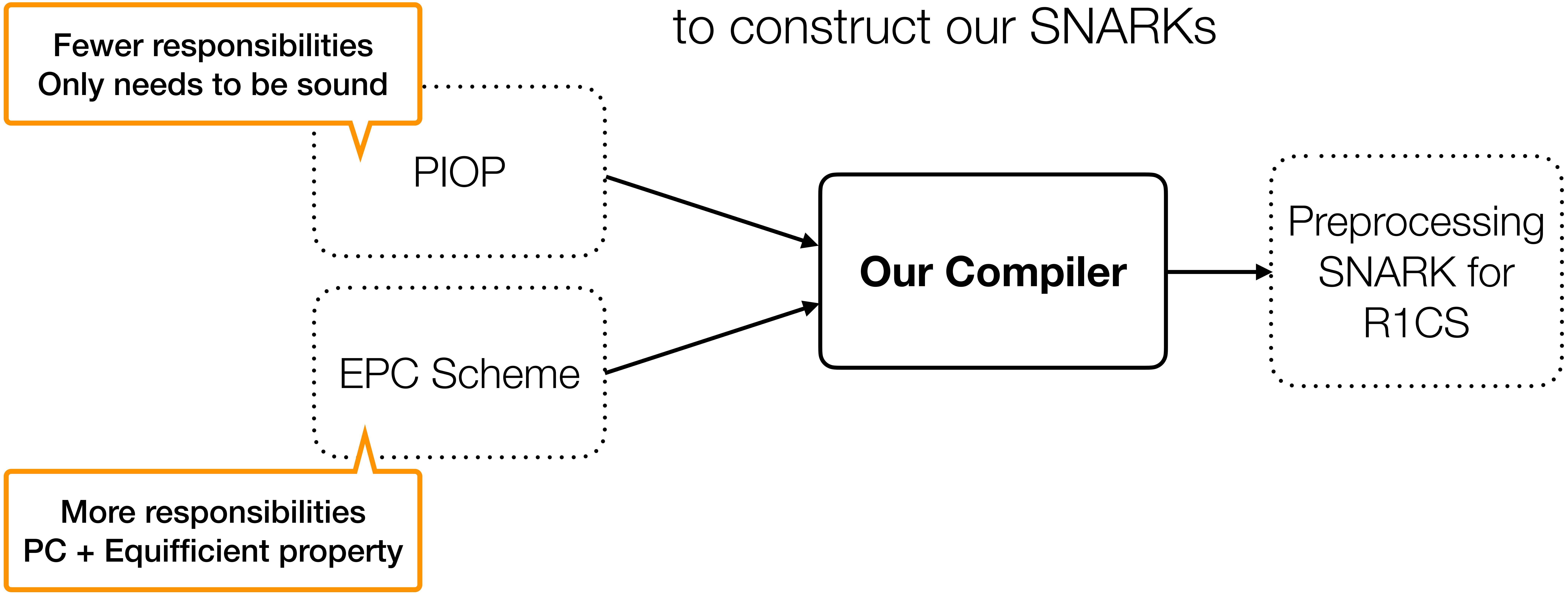
> 2x faster than HyperPlonk (custom gates)

Both in ROM + AGM

Both Circuit-Specific

New Methodology

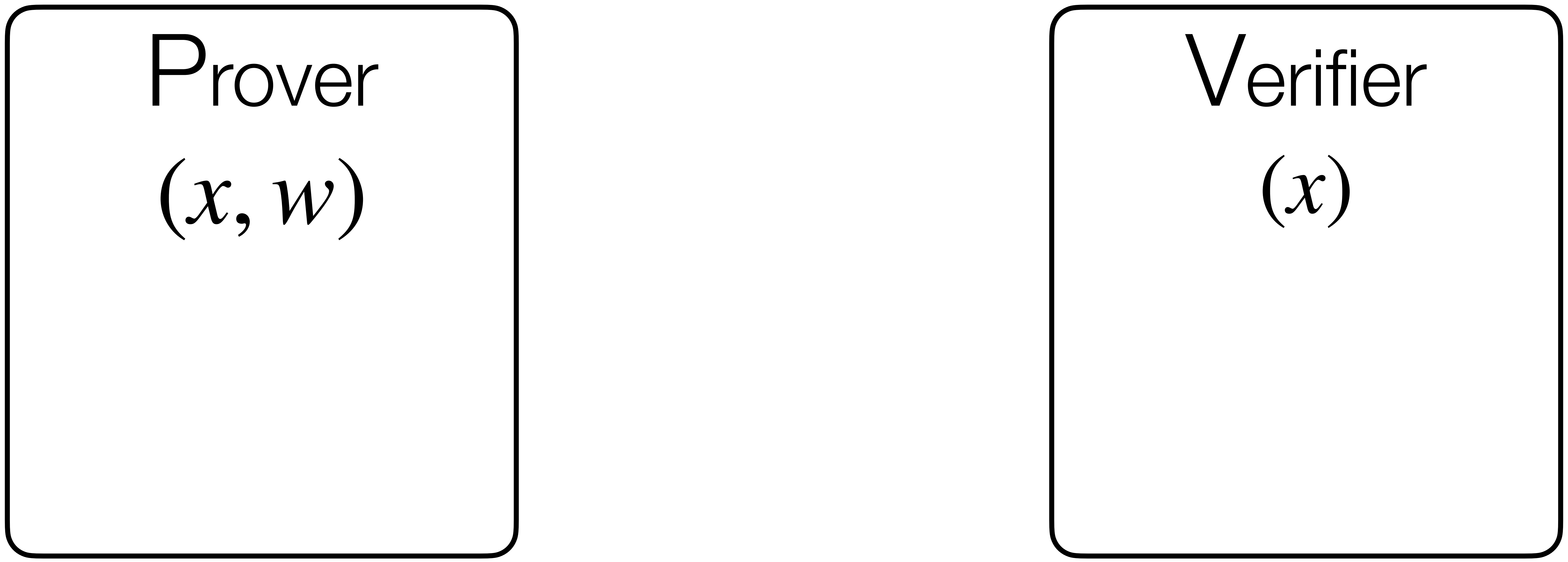
We adapt existing SNARK methodologies [CHMMVW20, BFS20]
to construct our SNARKs



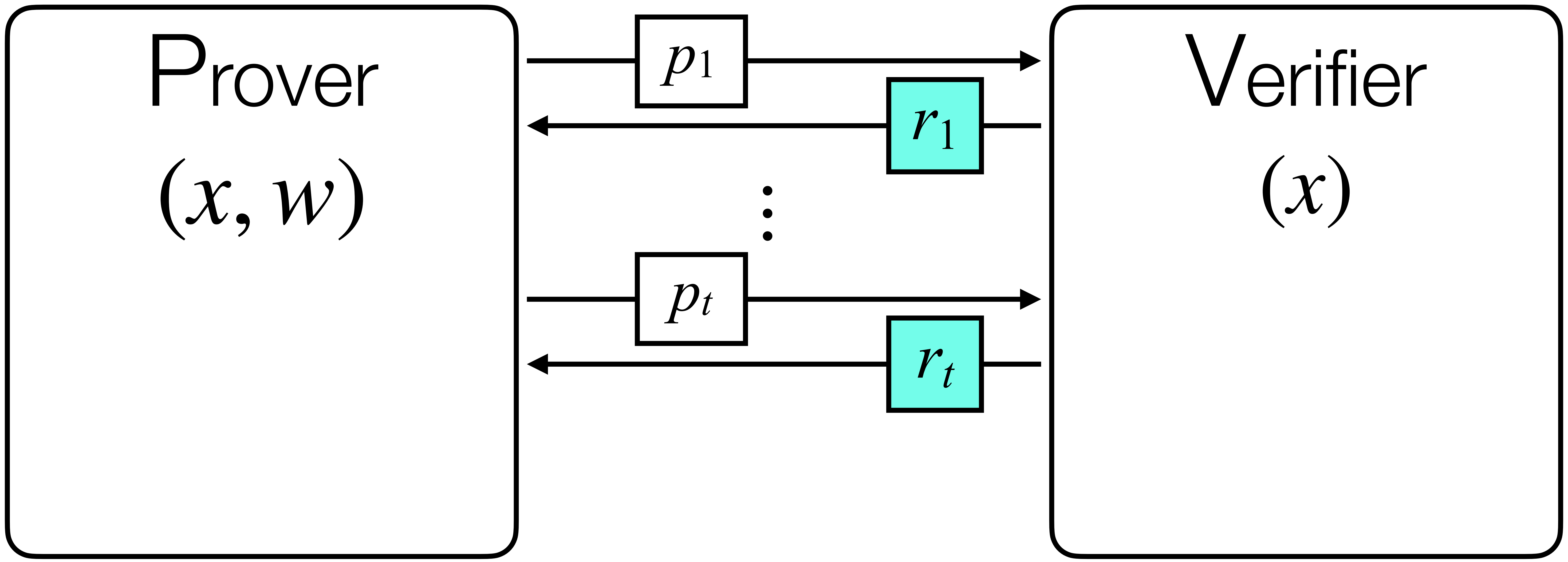
Background

Background: Polynomial IOPs

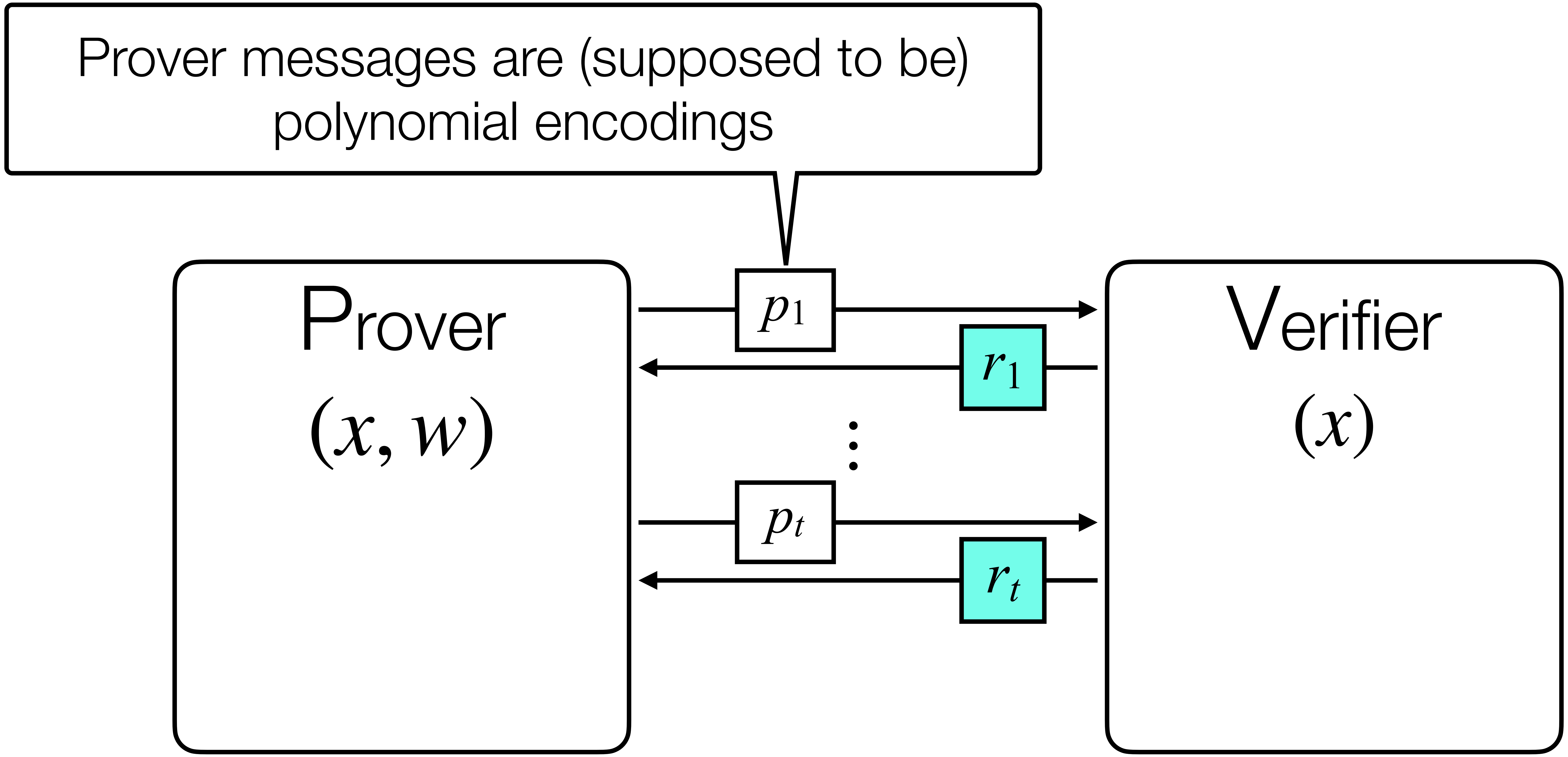
Background: Polynomial IOPs



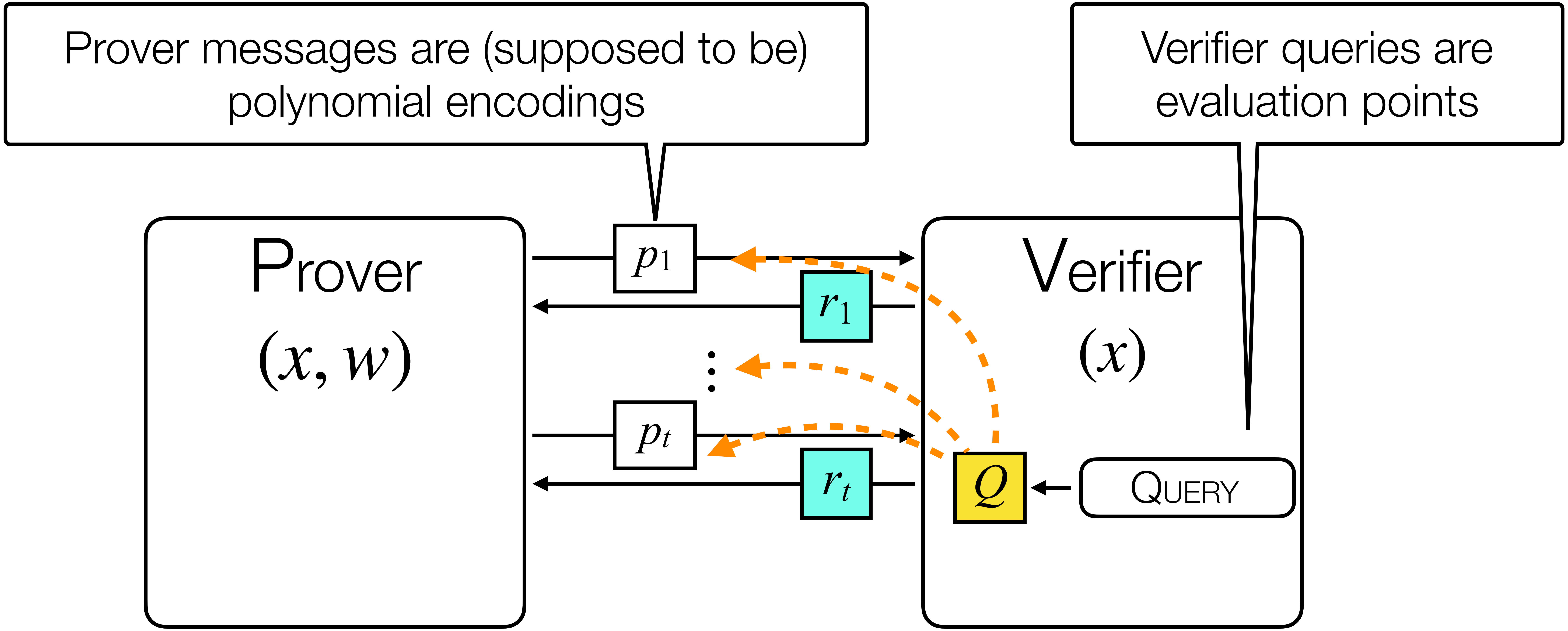
Background: Polynomial IOPs



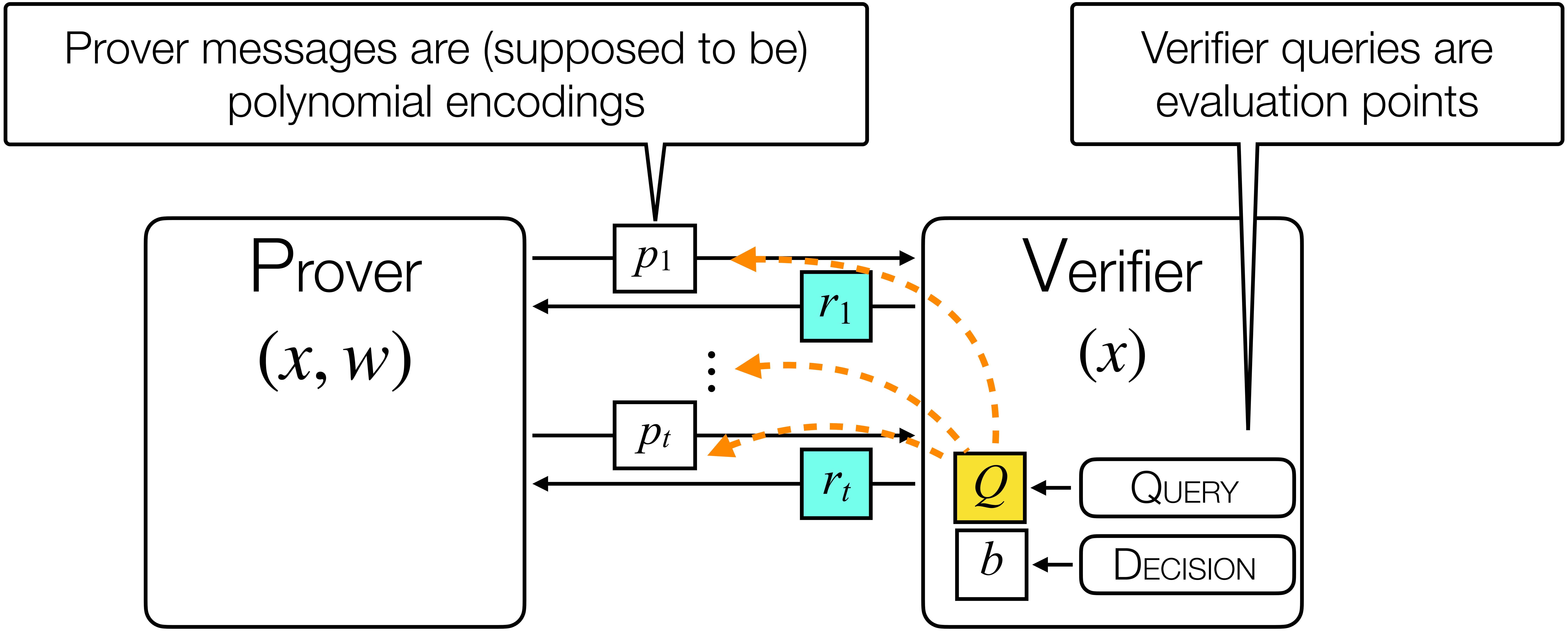
Background: Polynomial IOPs



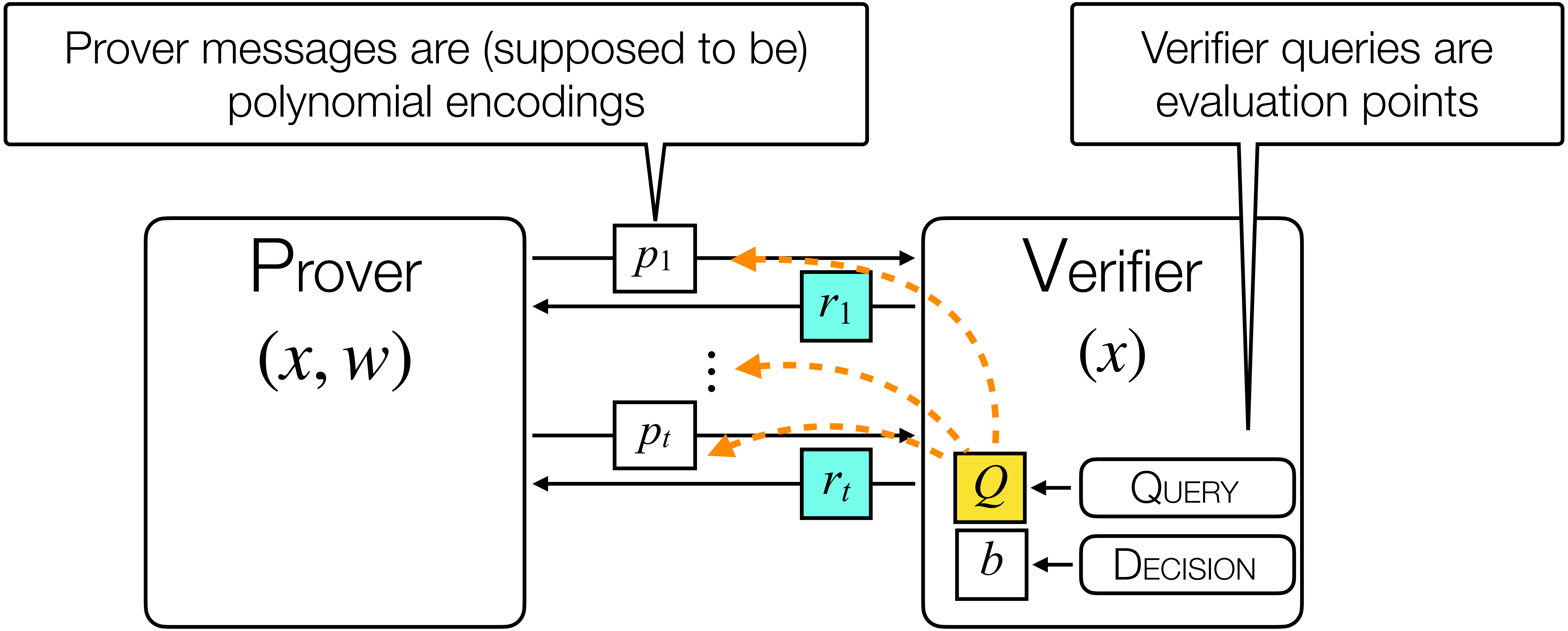
Background: Polynomial IOPs



Background: Polynomial IOPs

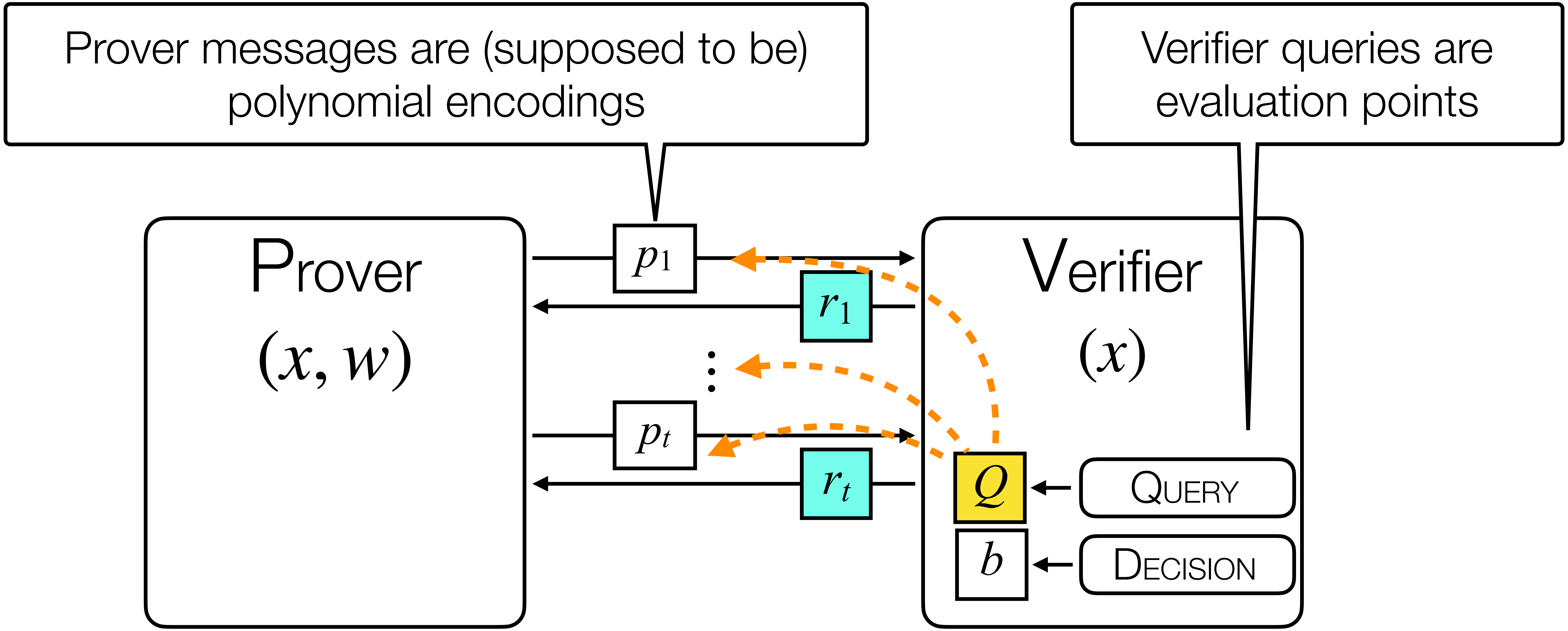


Background: Polynomial IOPs



- **Completeness:** If $F(x, w) = 1$, then $\mathscr{V}$ accepts

Background: Polynomial IOPs



- **Completeness:** If $F(x, w) = 1$, then $\mathscr{V}$ accepts
- **Soundness:** If $F(x, w) \neq 1$, then $\mathscr{V}$ rejects

Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation

Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation

Maximum degree n → SETUP →

Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation

Maximum degree n → SETUP → Committer key ck, Verifier key vk

Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation

Maximum degree n → SETUP → Committer key ck, Verifier key vk

SENDER

Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation

Maximum degree n → SETUP → Committer key ck, Verifier key vk

SENDER

RECEIVER

Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation

Maximum degree n ⟶ SETUP ⟶ Committer key ck, Verifier key vk

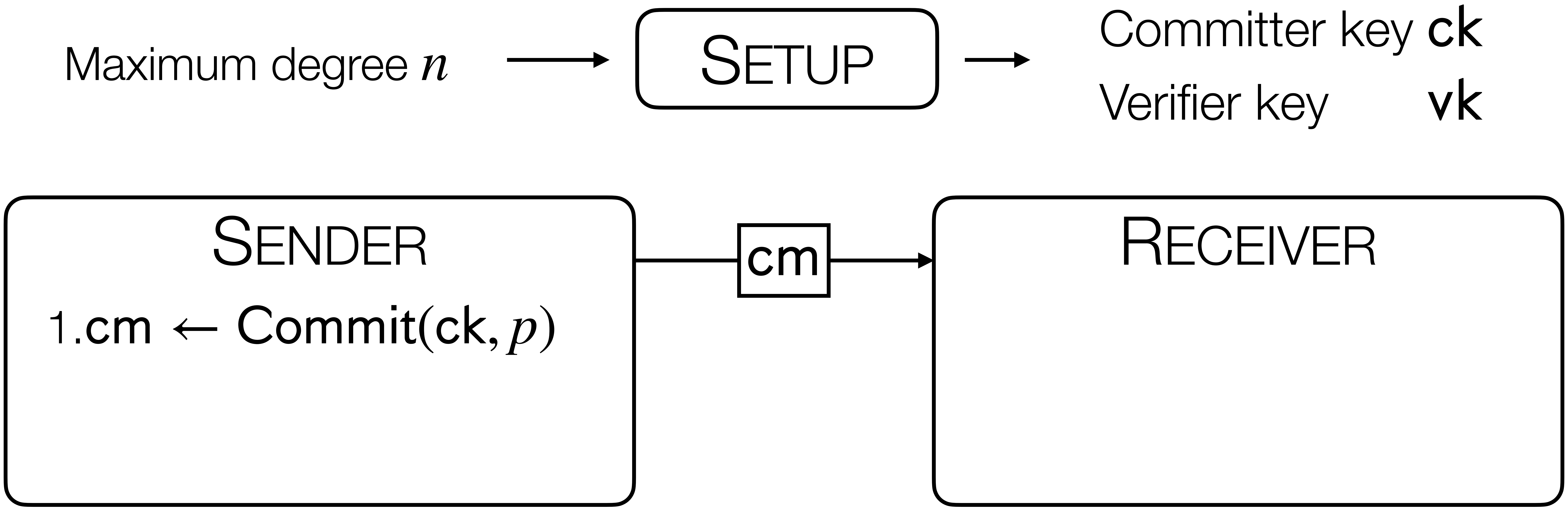
SENDER

1. $\mathsf{cm} \leftarrow \mathsf{Commit}(\mathsf{ck}, p)$

RECEIVER

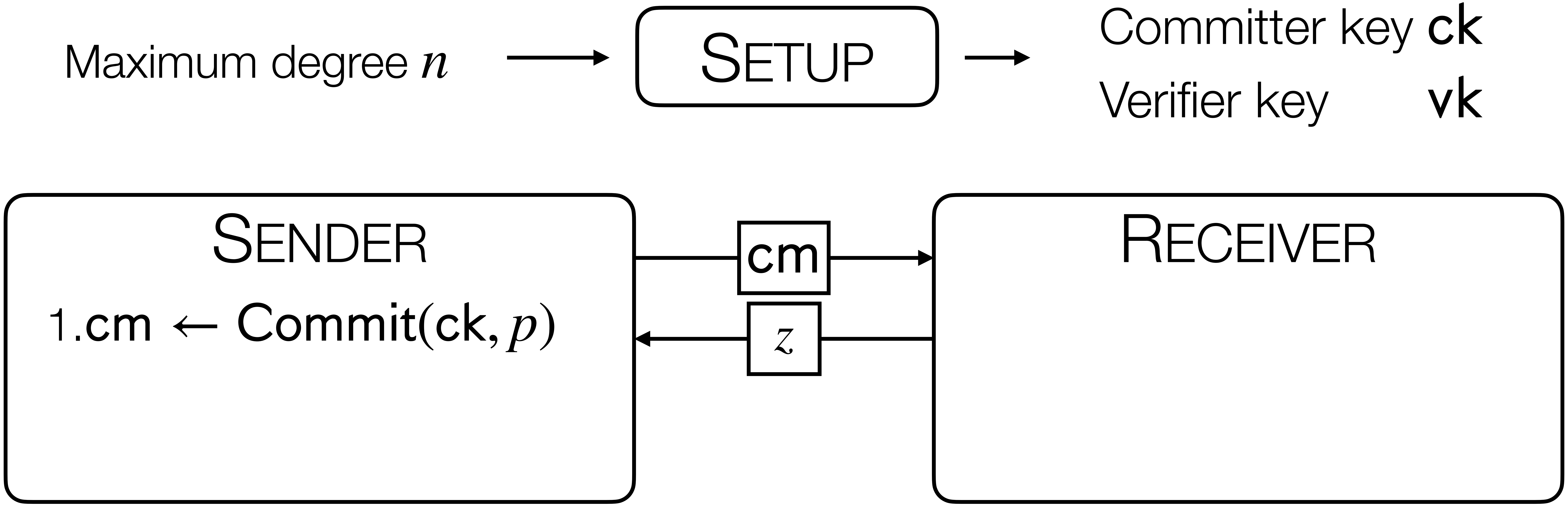
Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation



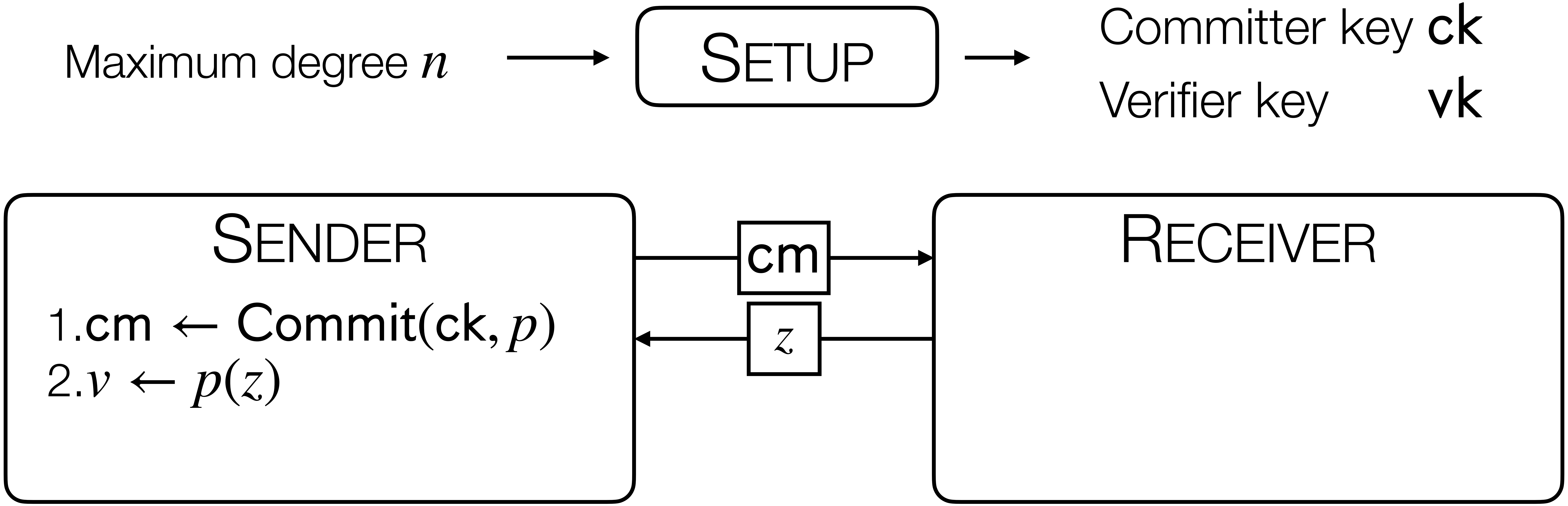
Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation



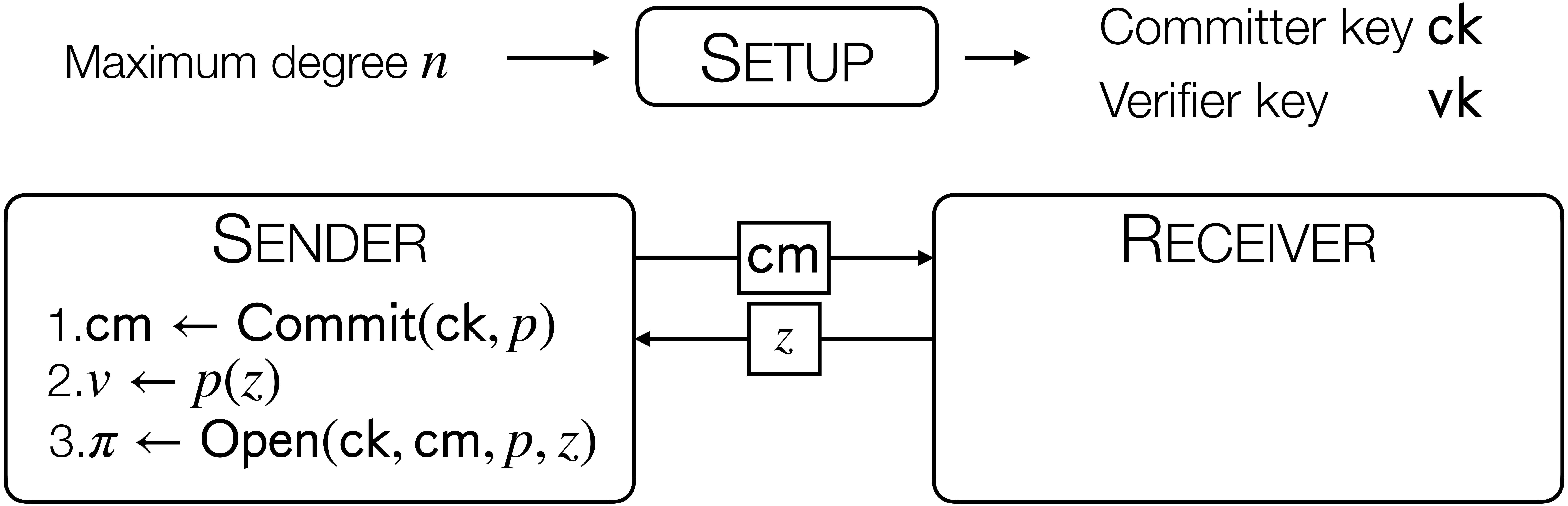
Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation



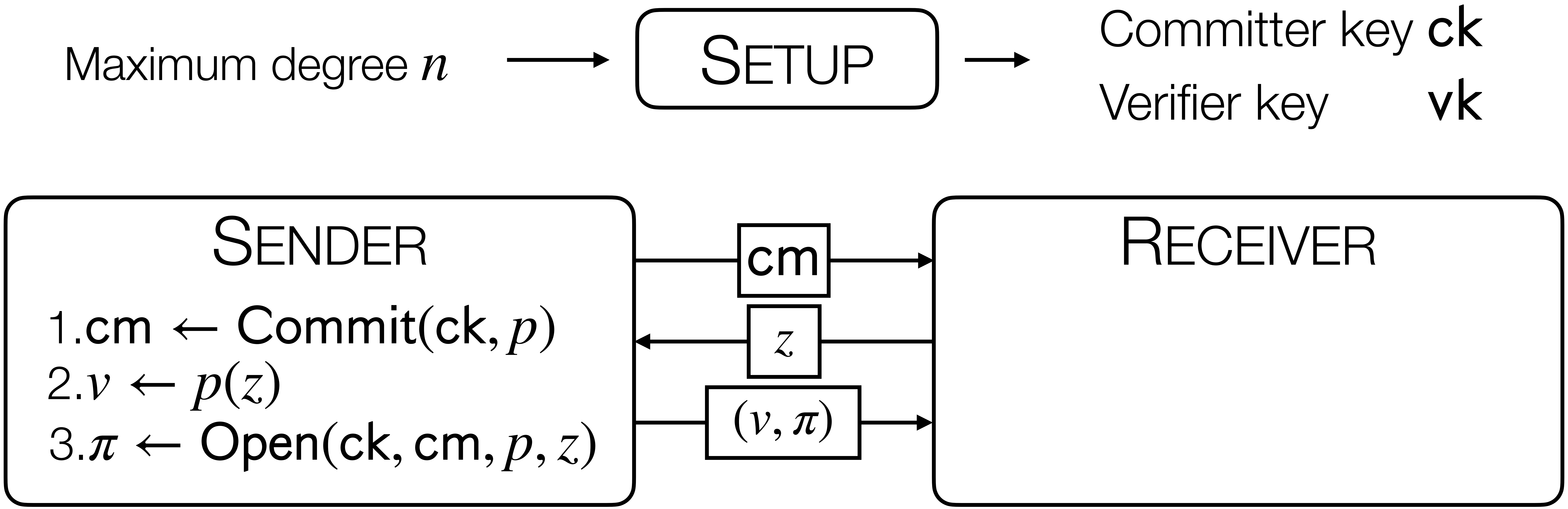
Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation



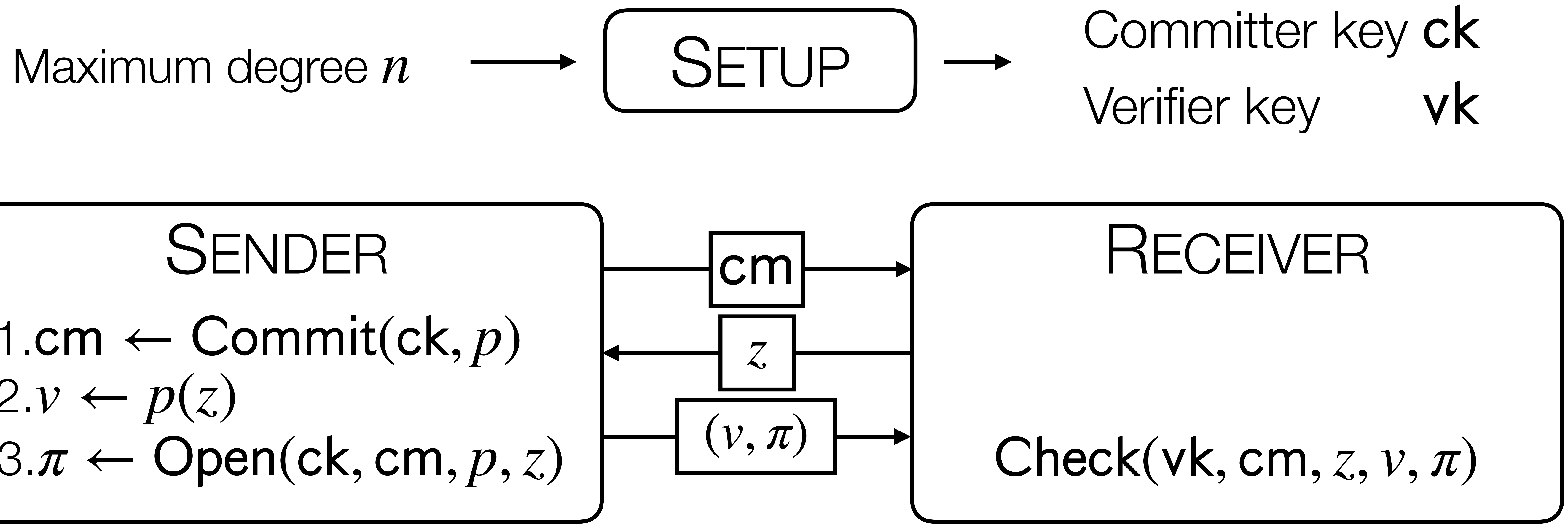
Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation



Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation



Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation

Maximum degree n → SETUP → Committer key ck, Verifier key vk

SENDER

1. $\mathsf{cm} \leftarrow \mathsf{Commit}(\mathsf{ck}, p)$
2. $v \leftarrow p(z)$
3. $\pi \leftarrow \mathsf{Open}(\mathsf{ck}, \mathsf{cm}, p, z)$

Sender → Receiver: cm

Receiver → Sender: z

Sender → Receiver: (v, π)

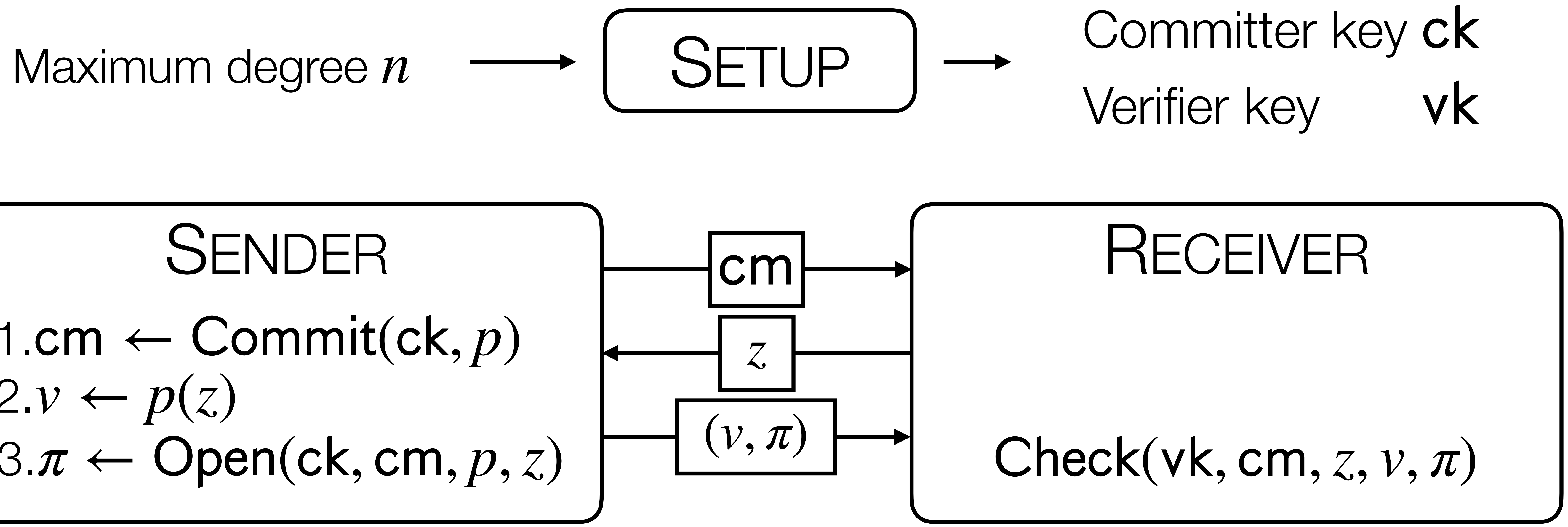
RECEIVER

$\mathsf{Check}(\mathsf{vk}, \mathsf{cm}, z, v, \pi)$

- **Completeness:** Whenever $p(z) = v$, the Receiver accepts.

Background: PC schemes

Commit to a polynomial, and later on prove its correct evaluation



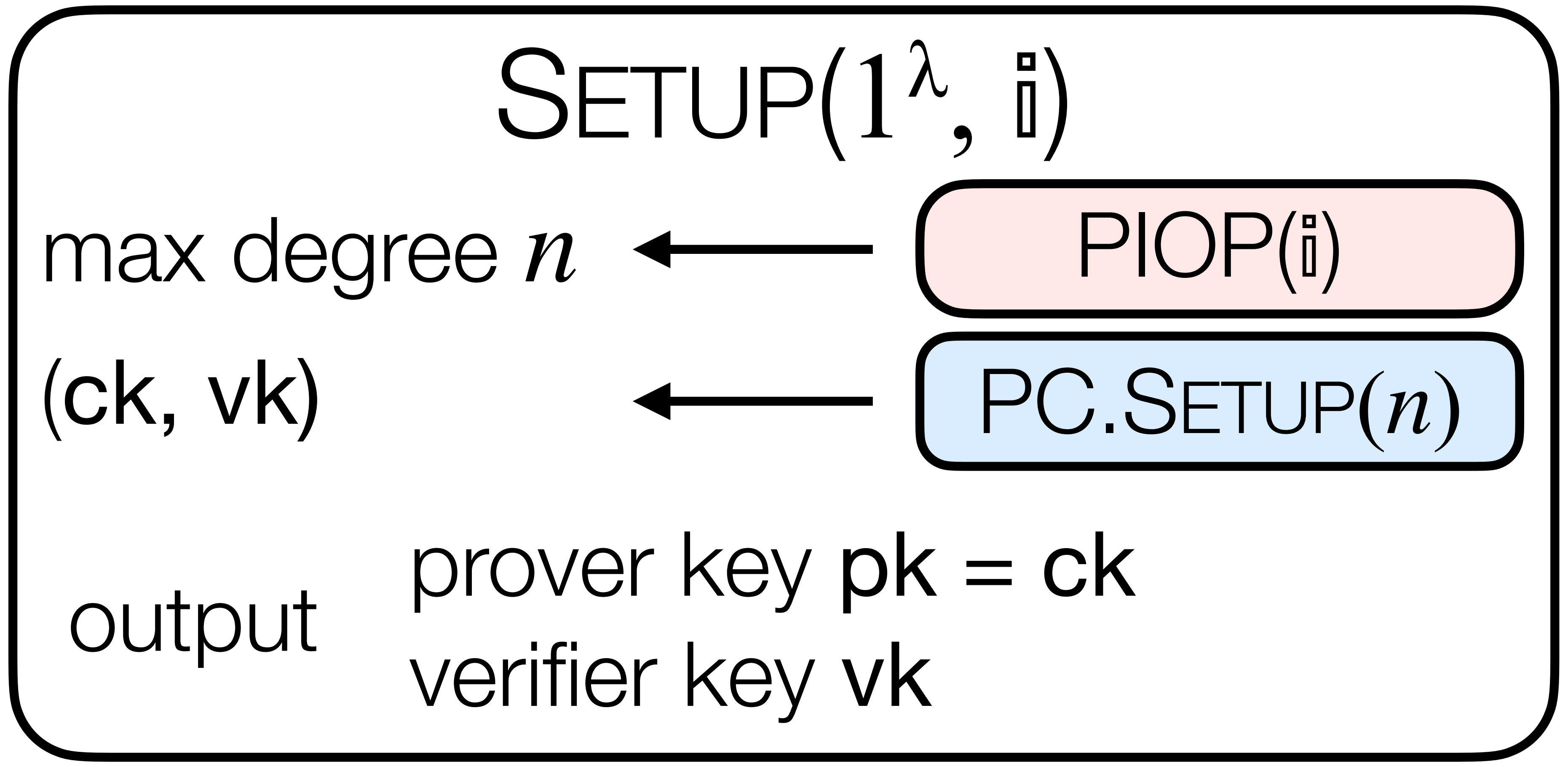
- **Completeness:** Whenever $p(z) = v$, the Receiver accepts.
- **Extractability:** Whenever the Receiver accepts, the Sender's commitment **cm** "contains" a polynomial p satisfying $p(z) = v$.

PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]

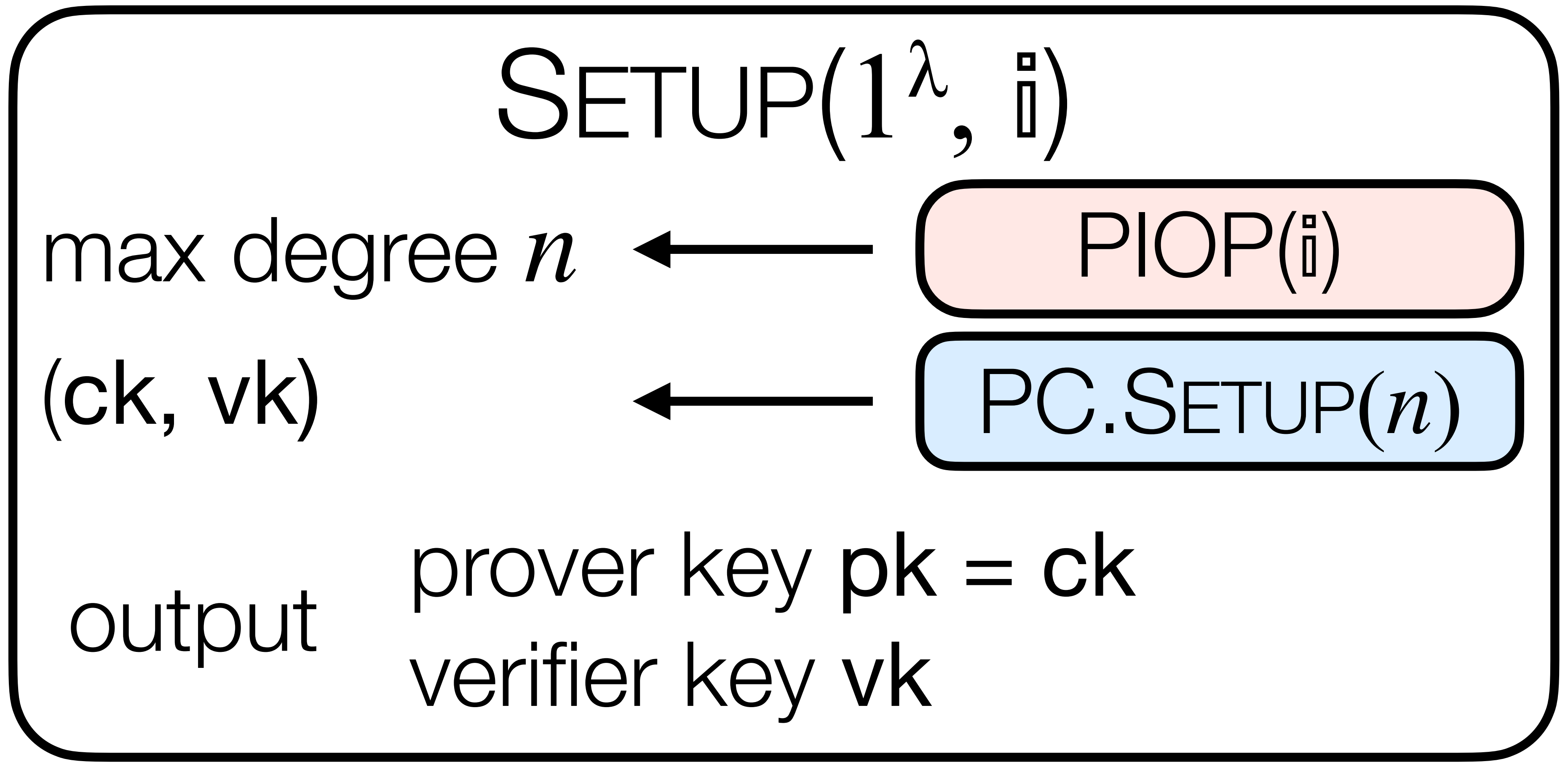
PIOPs + PC Schemes → SNARK

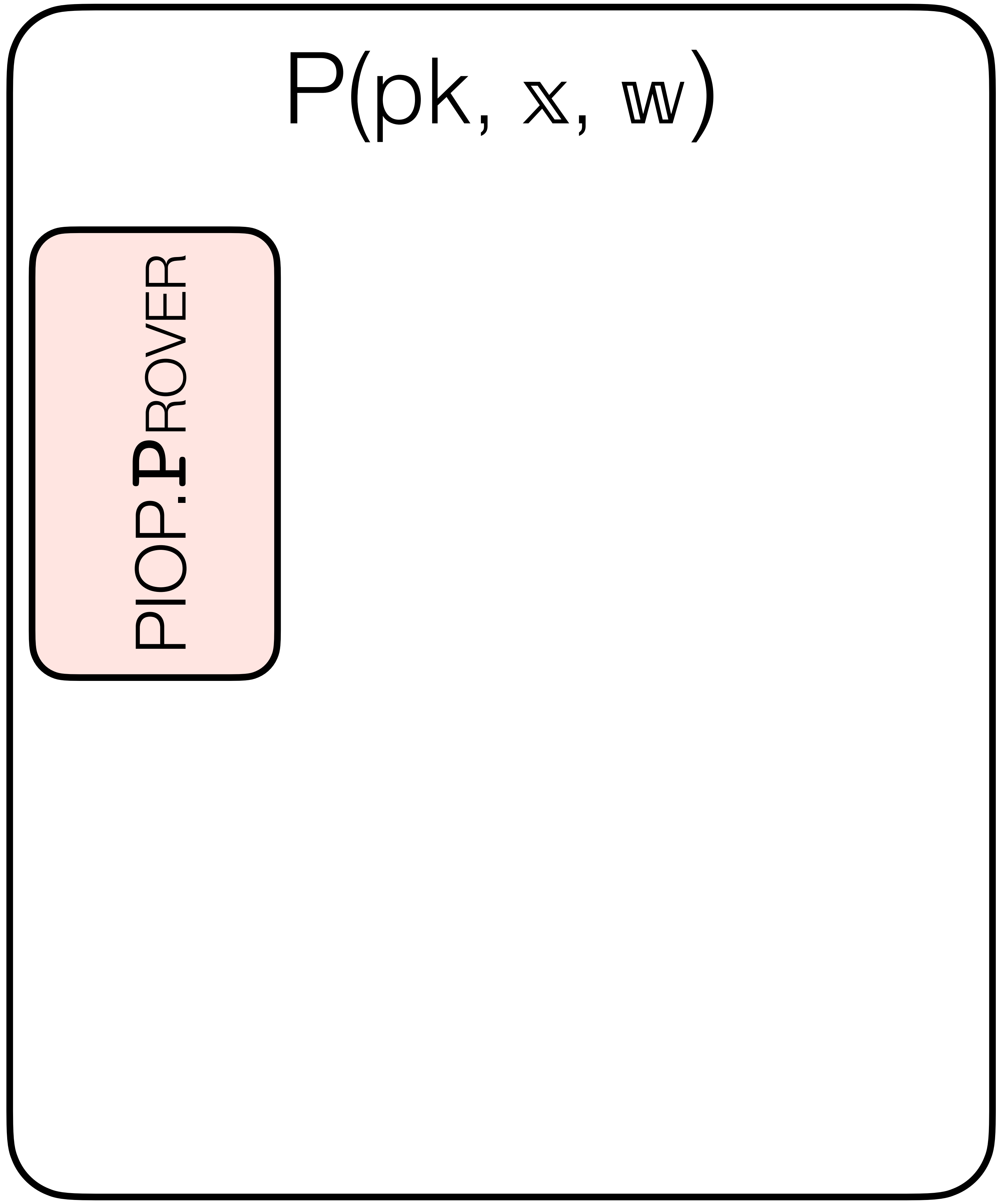
[CHMMVW20, BFS20]



PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]





PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]

PIOPs + PC Schemes → SNARK

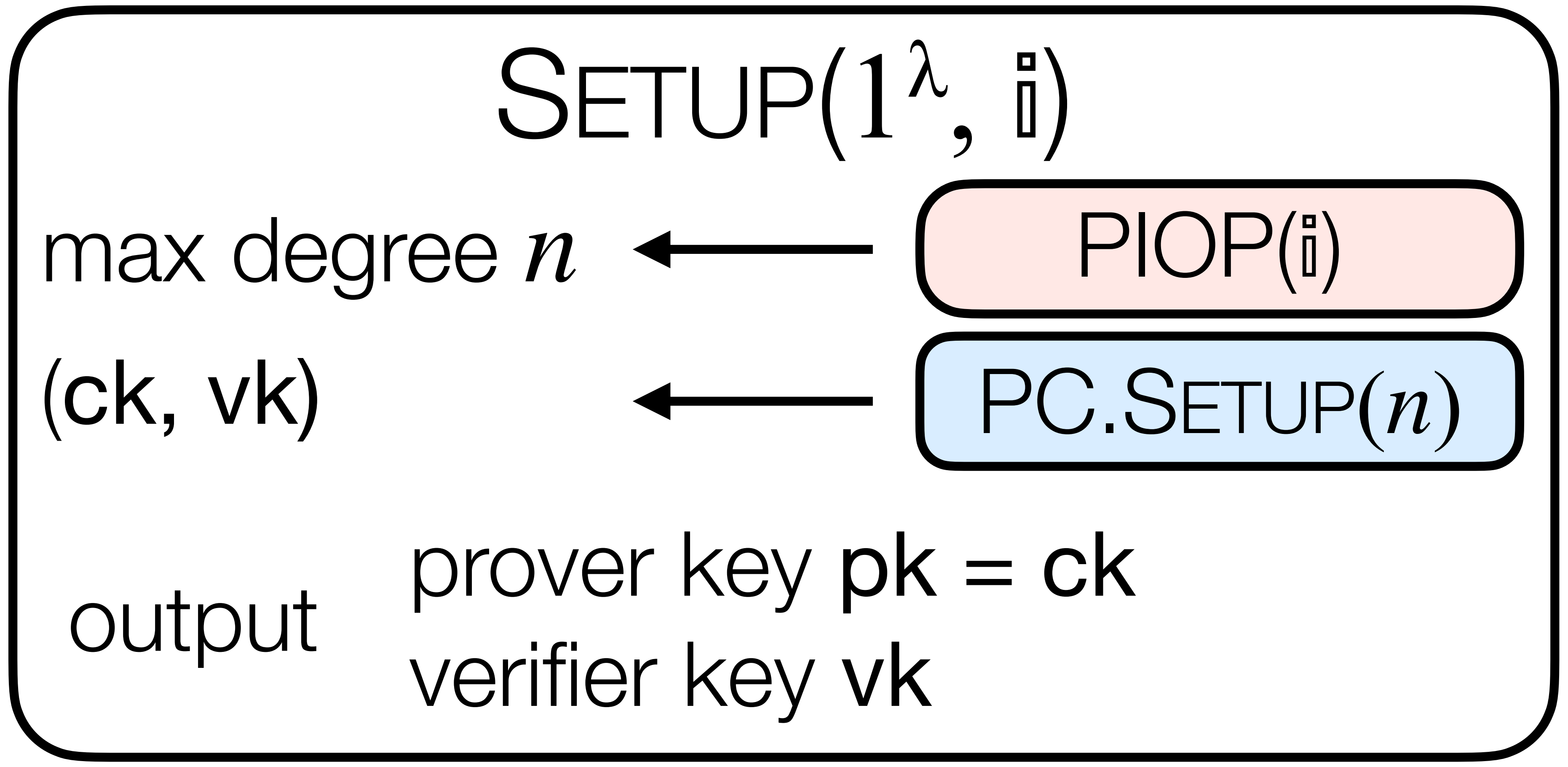
[CHMMVW20, BFS20]

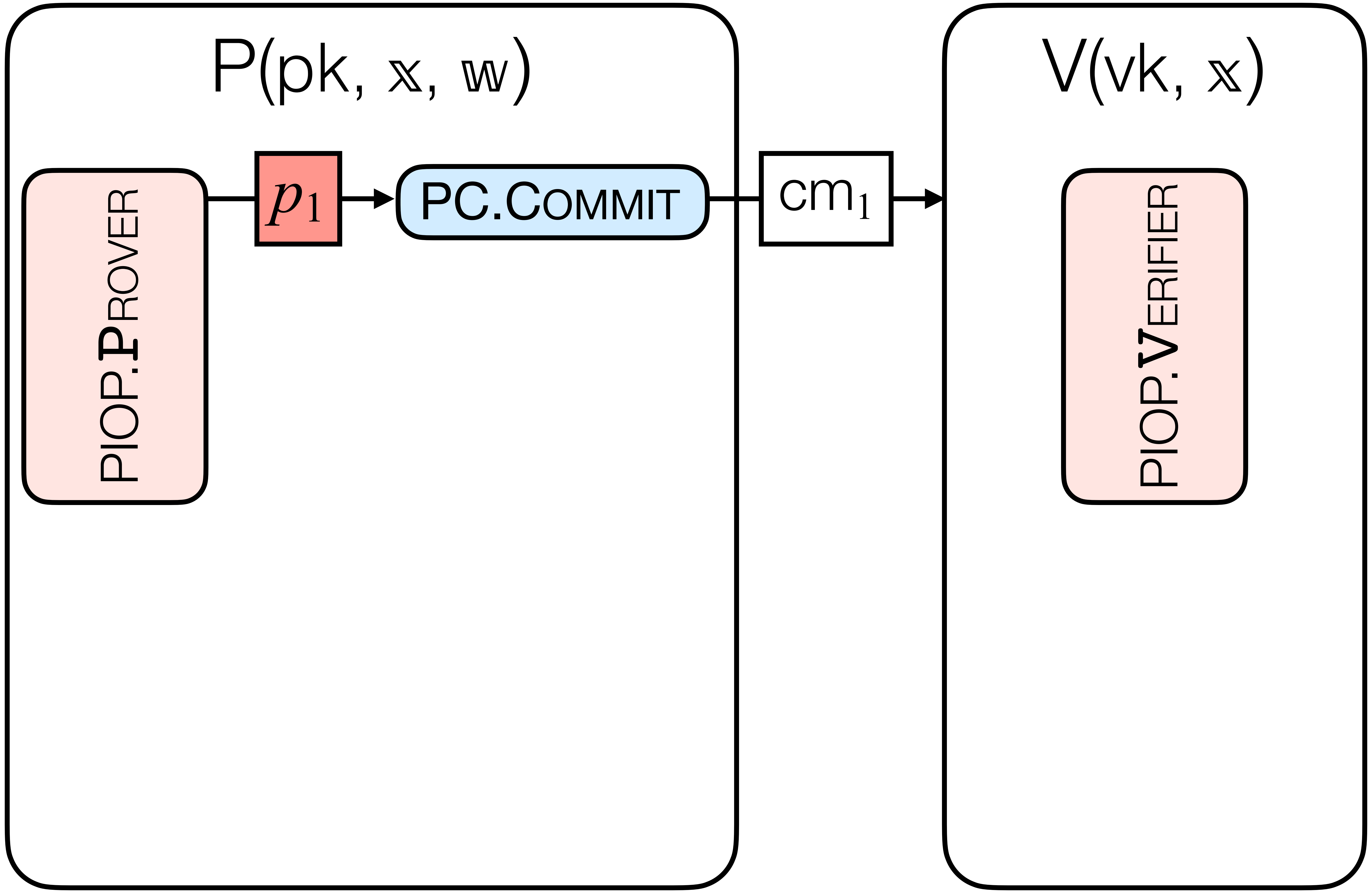
PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]

PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]



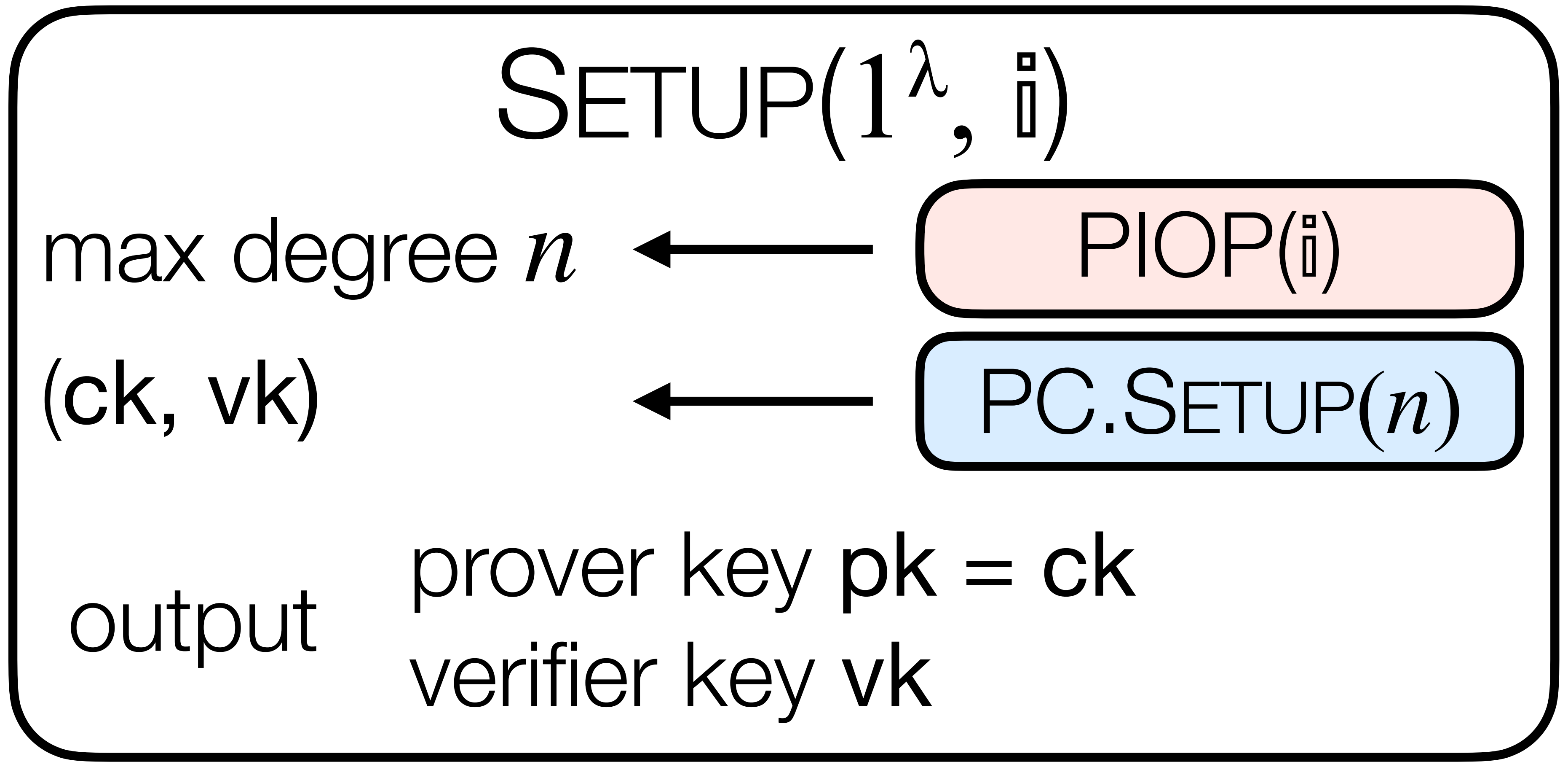


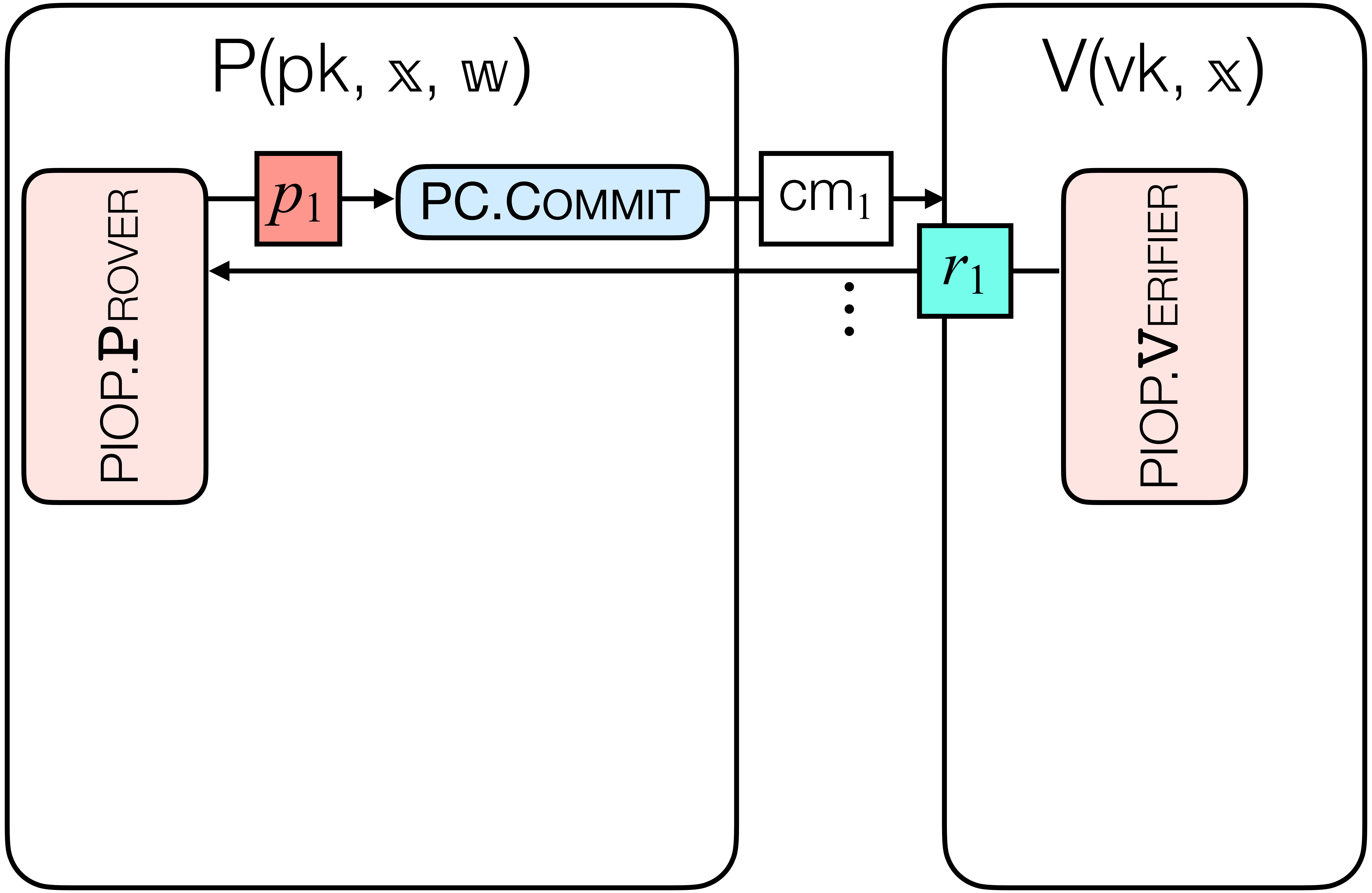
PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]

PIOPs + PC Schemes → SNARK

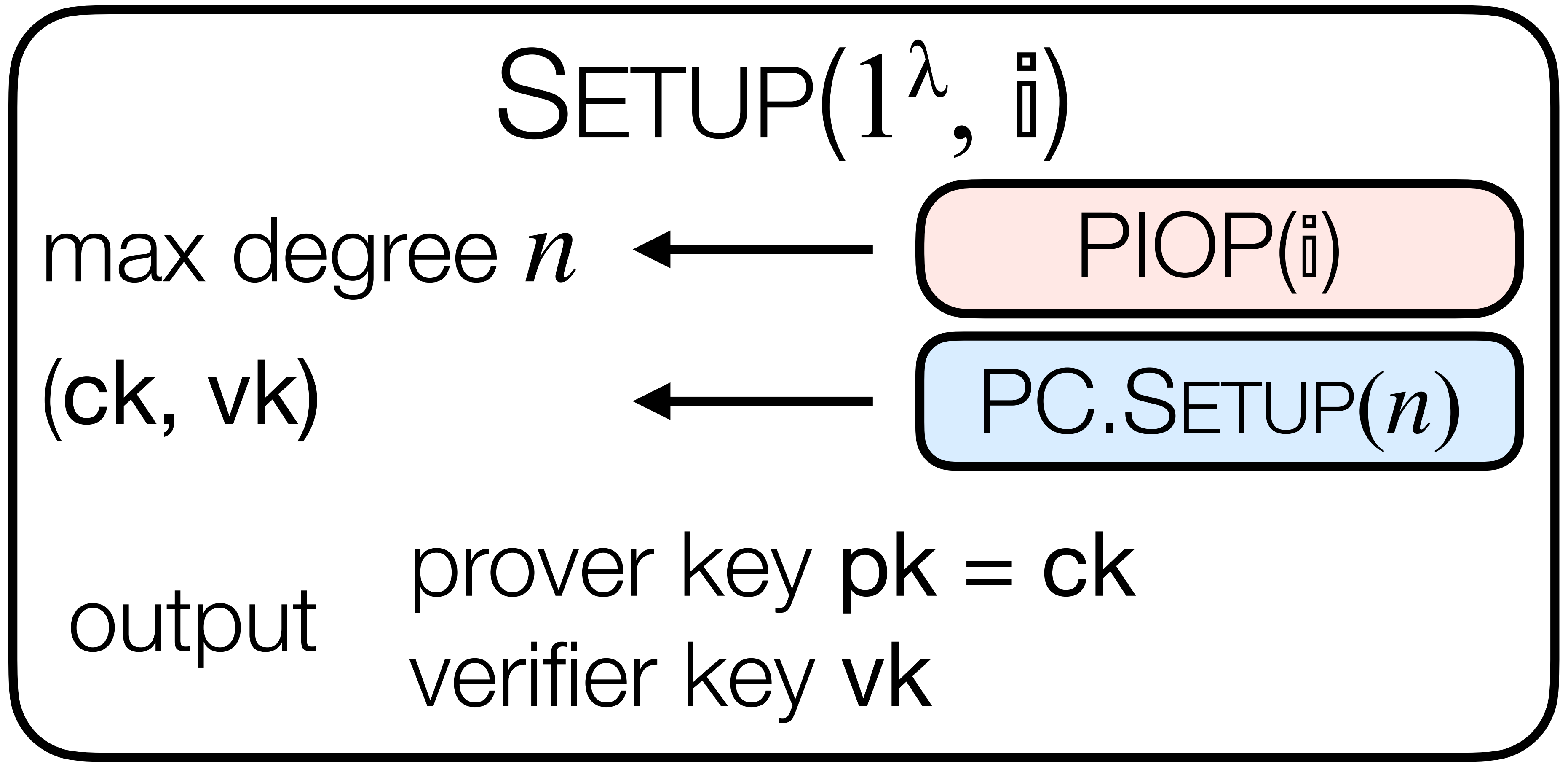
[CHMMVW20, BFS20]

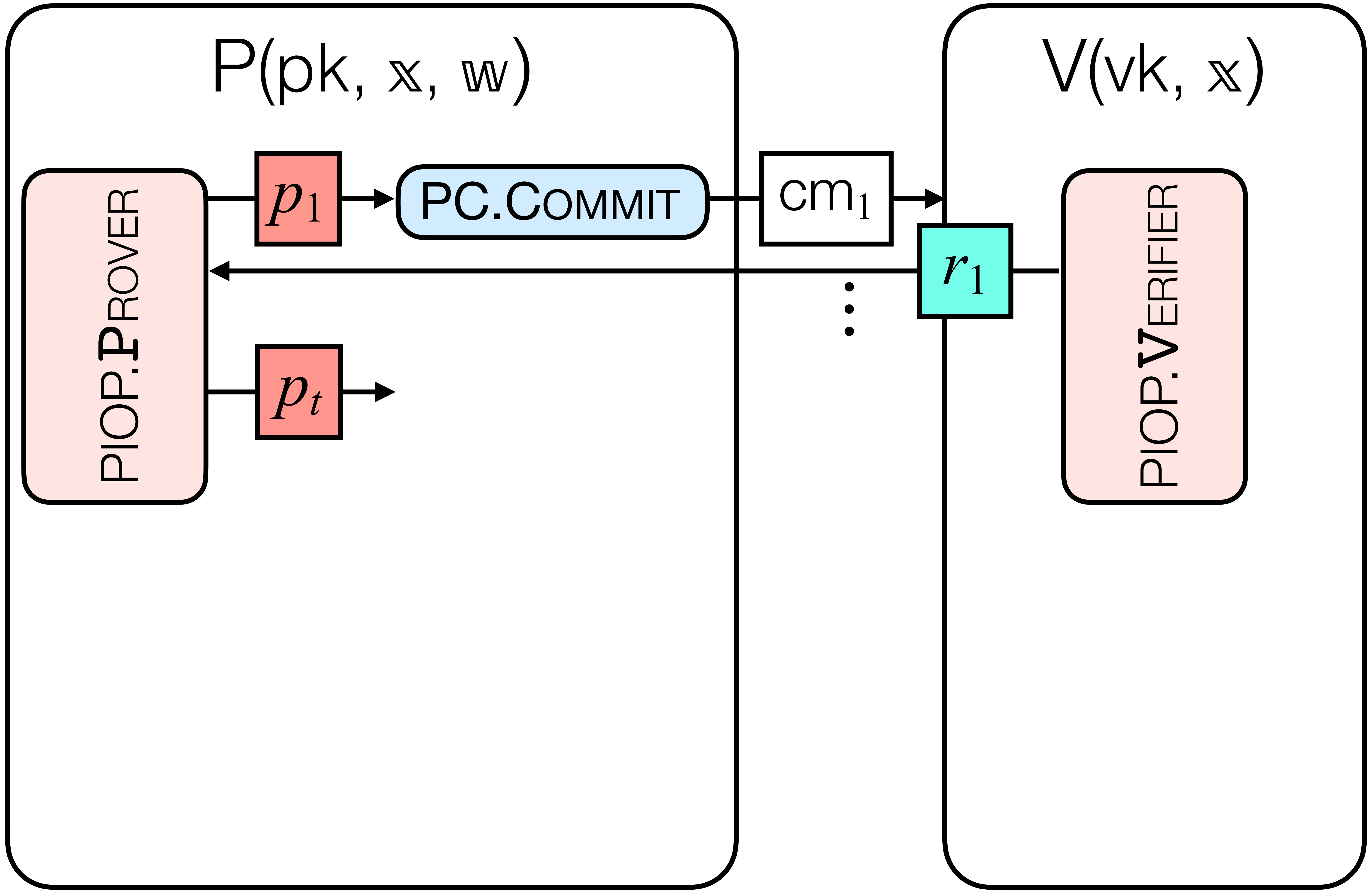




PIOPs + PC Schemes → SNARK

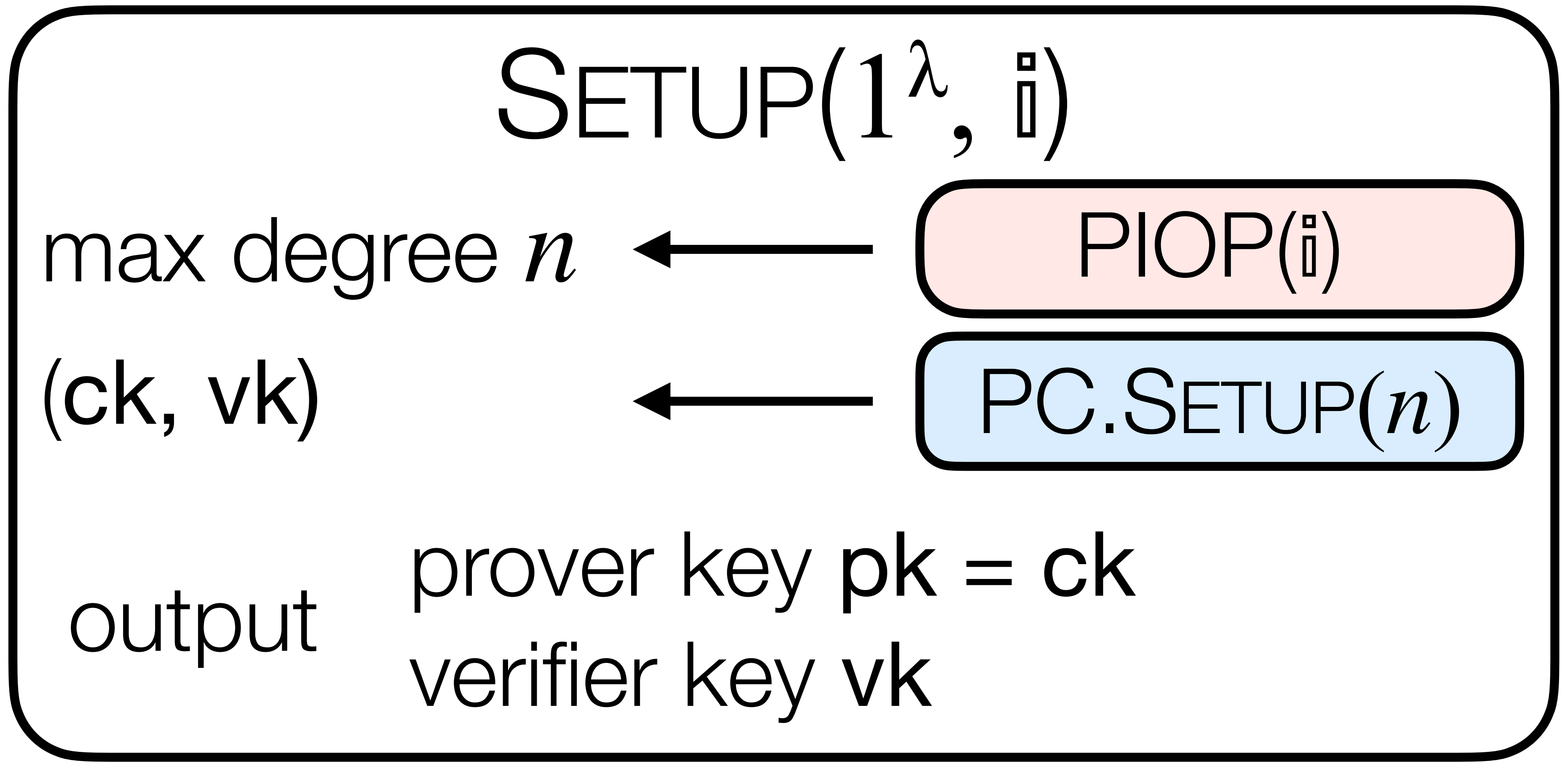
[CHMMVW20, BFS20]

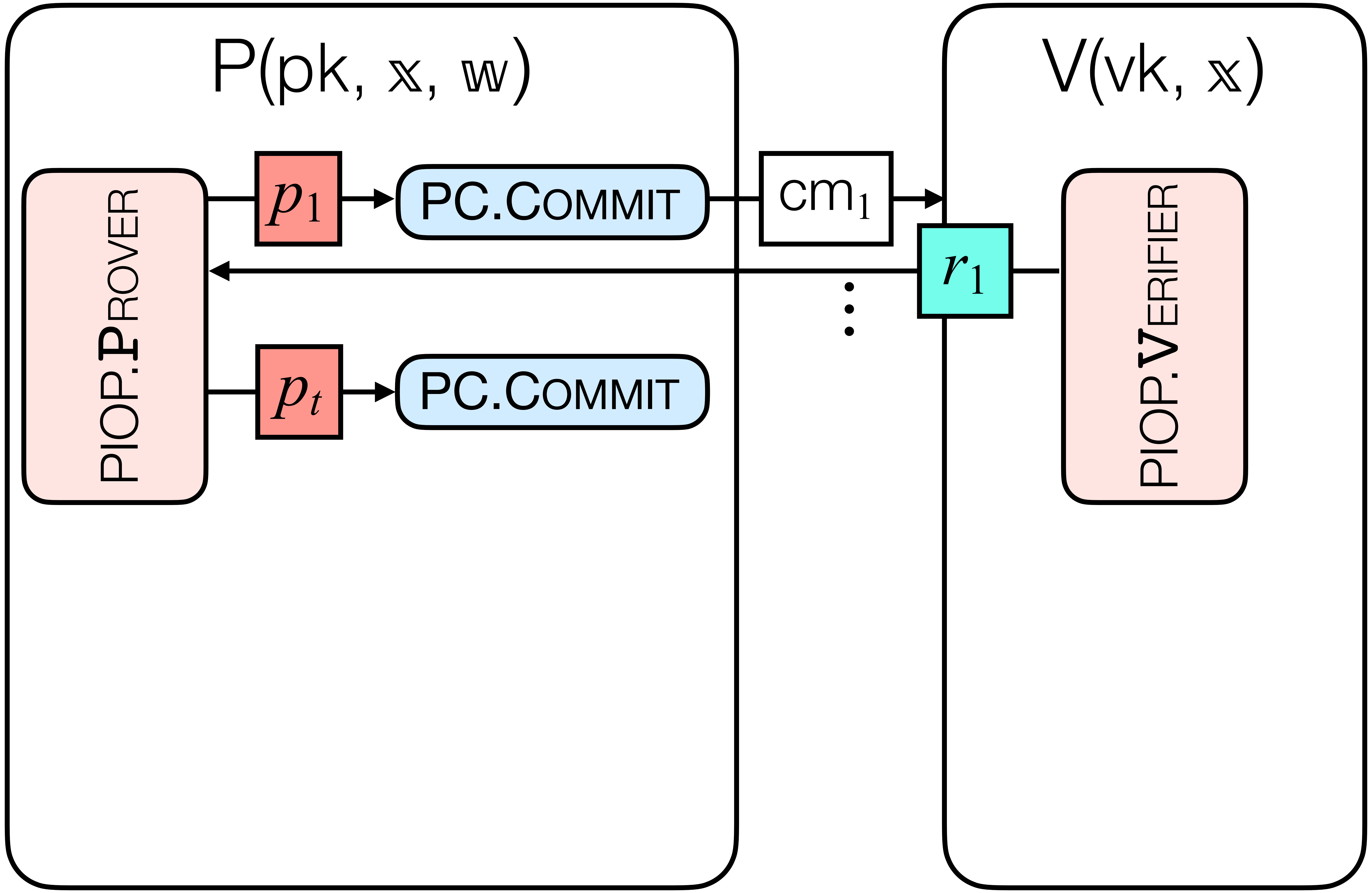




PIOPs + PC Schemes → SNARK

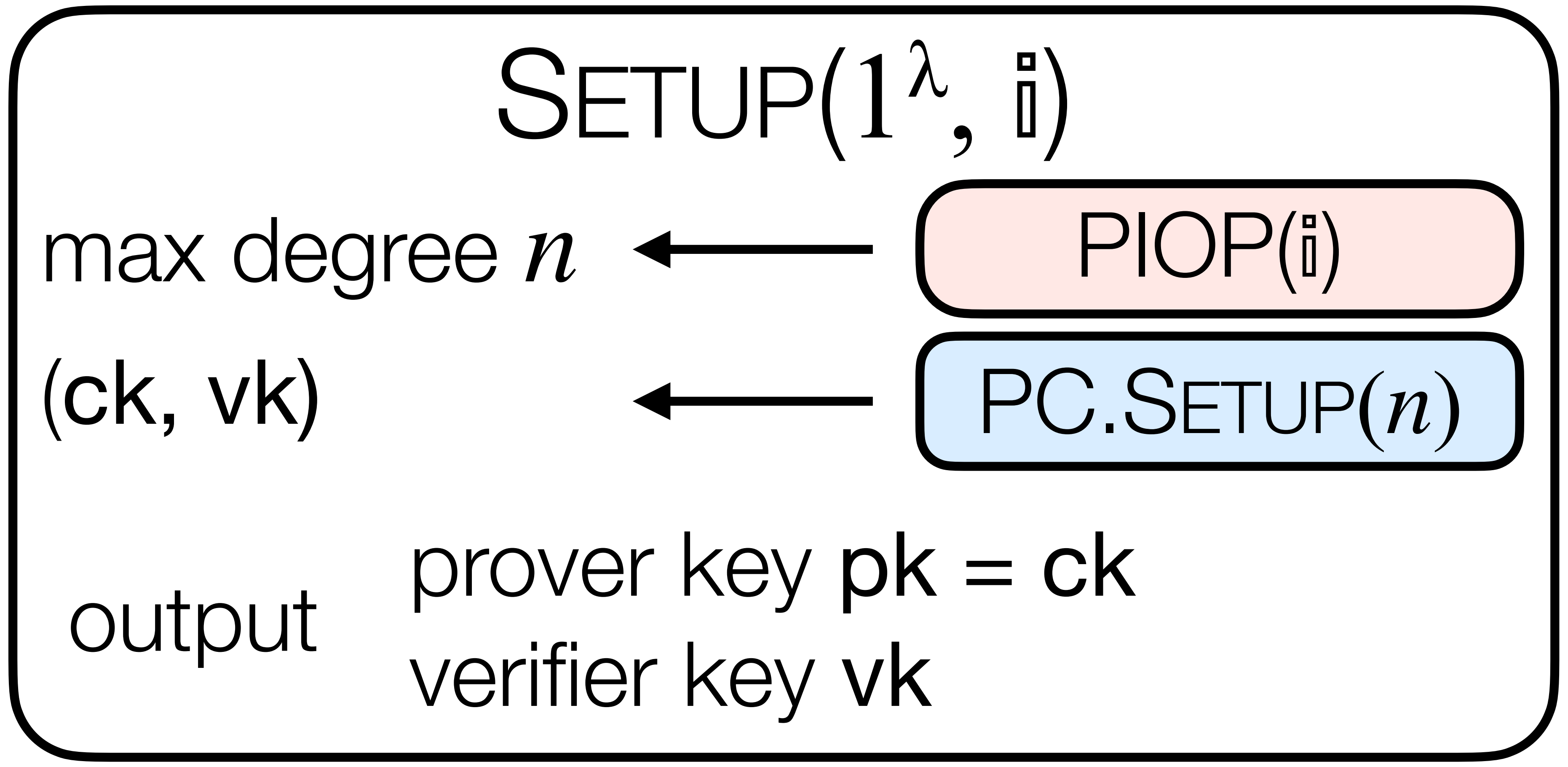
[CHMMVW20, BFS20]

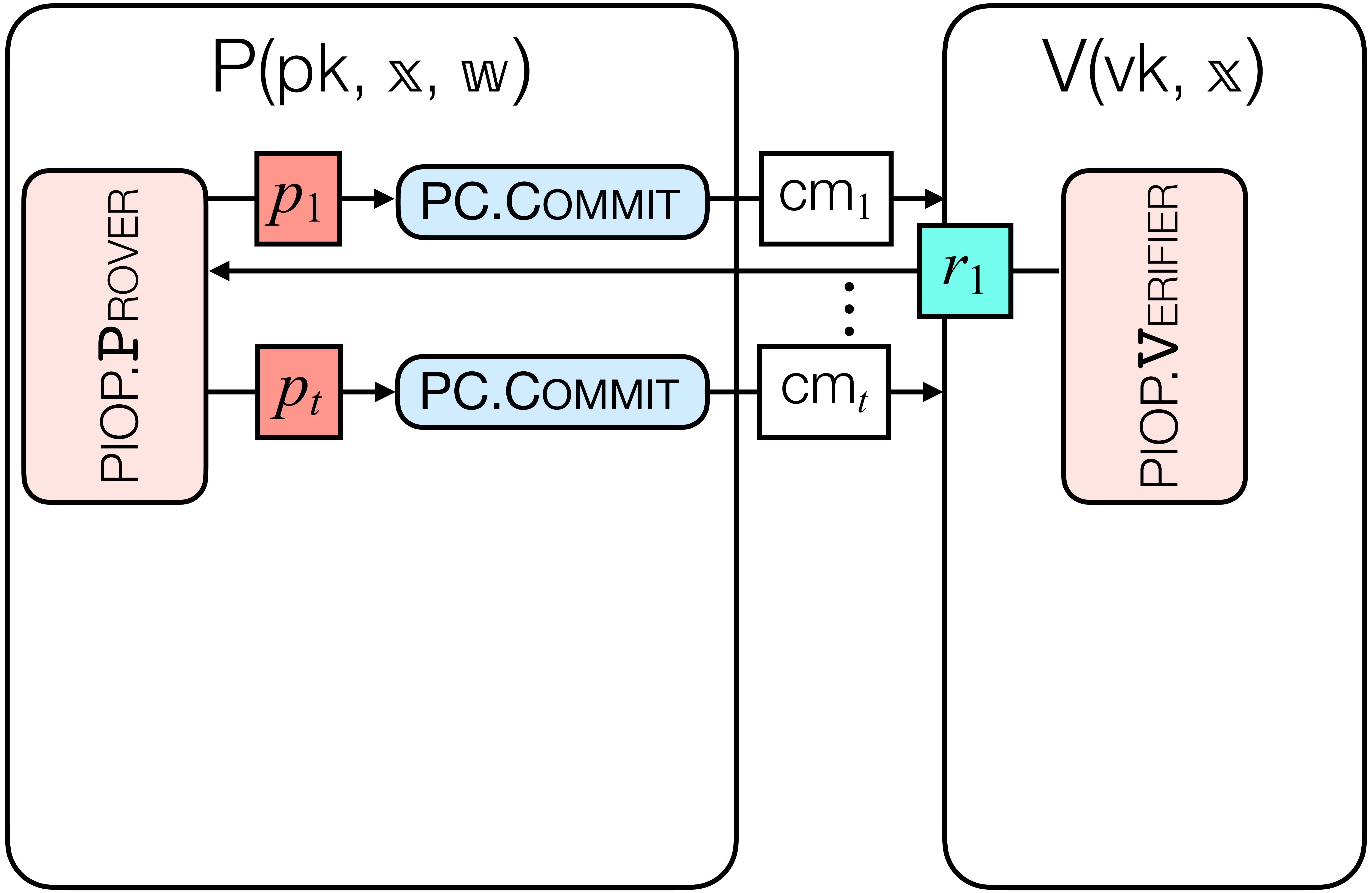




PIOPs + PC Schemes → SNARK

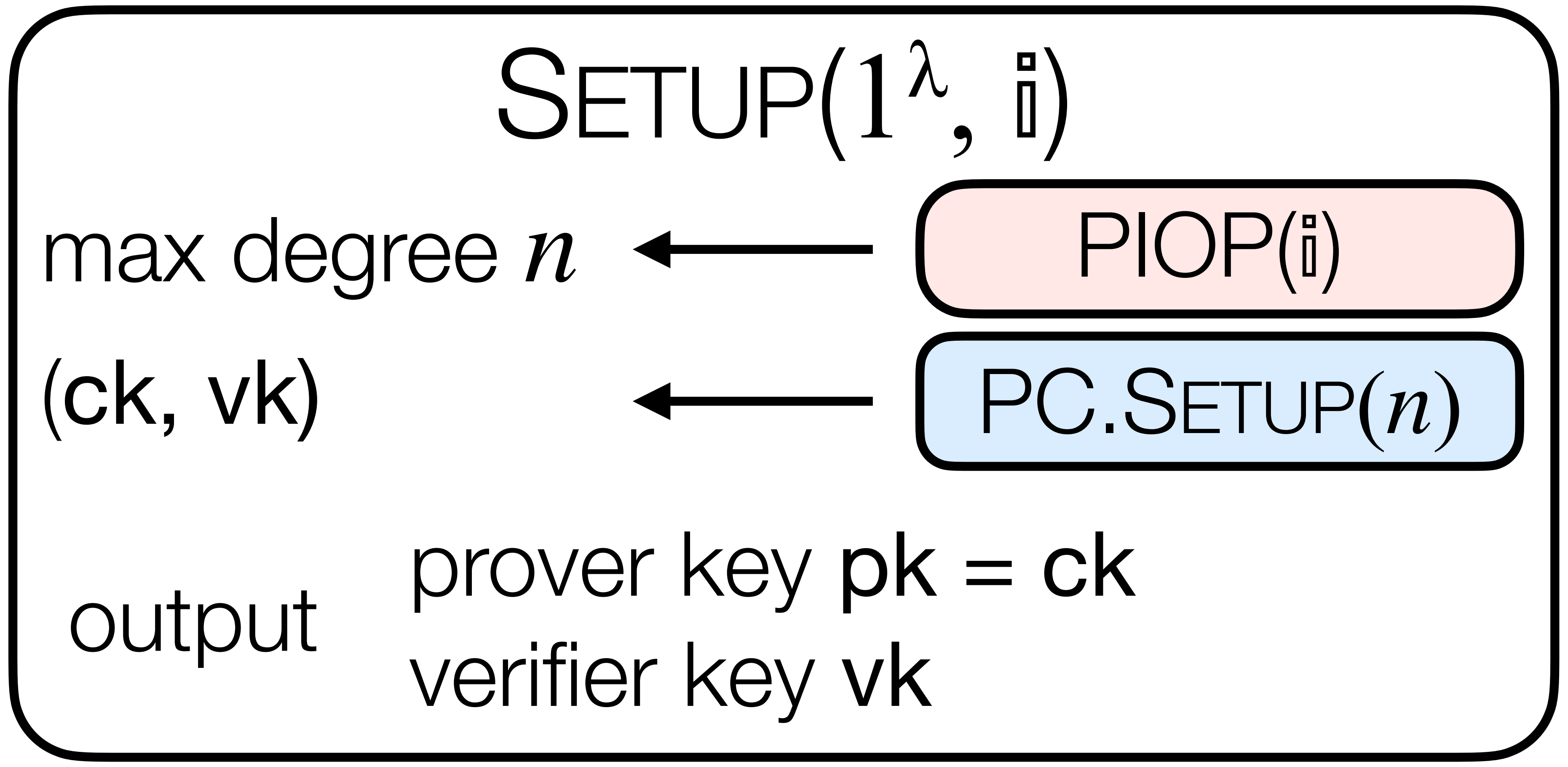
[CHMMVW20, BFS20]

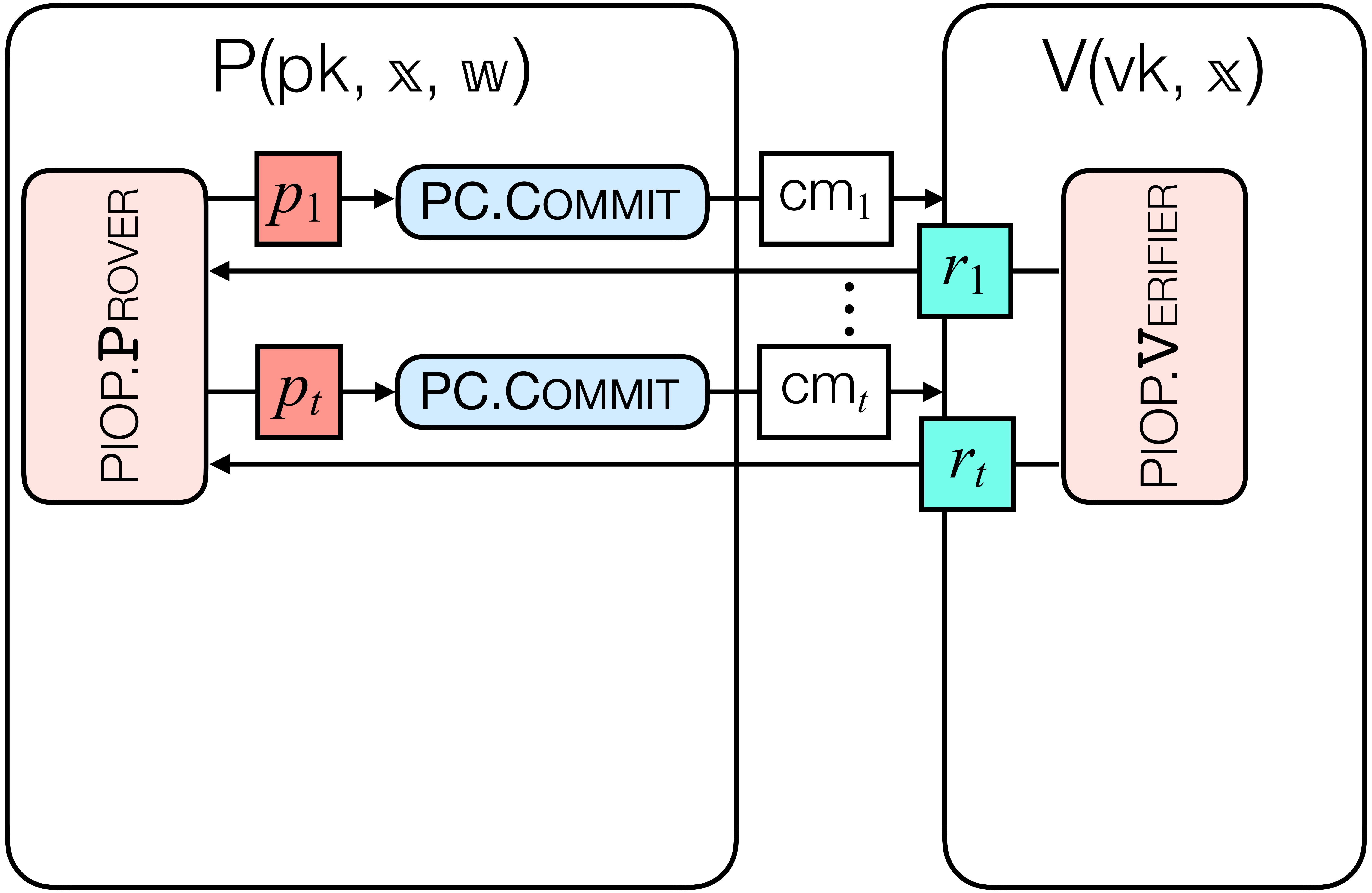




PIOPs + PC Schemes → SNARK

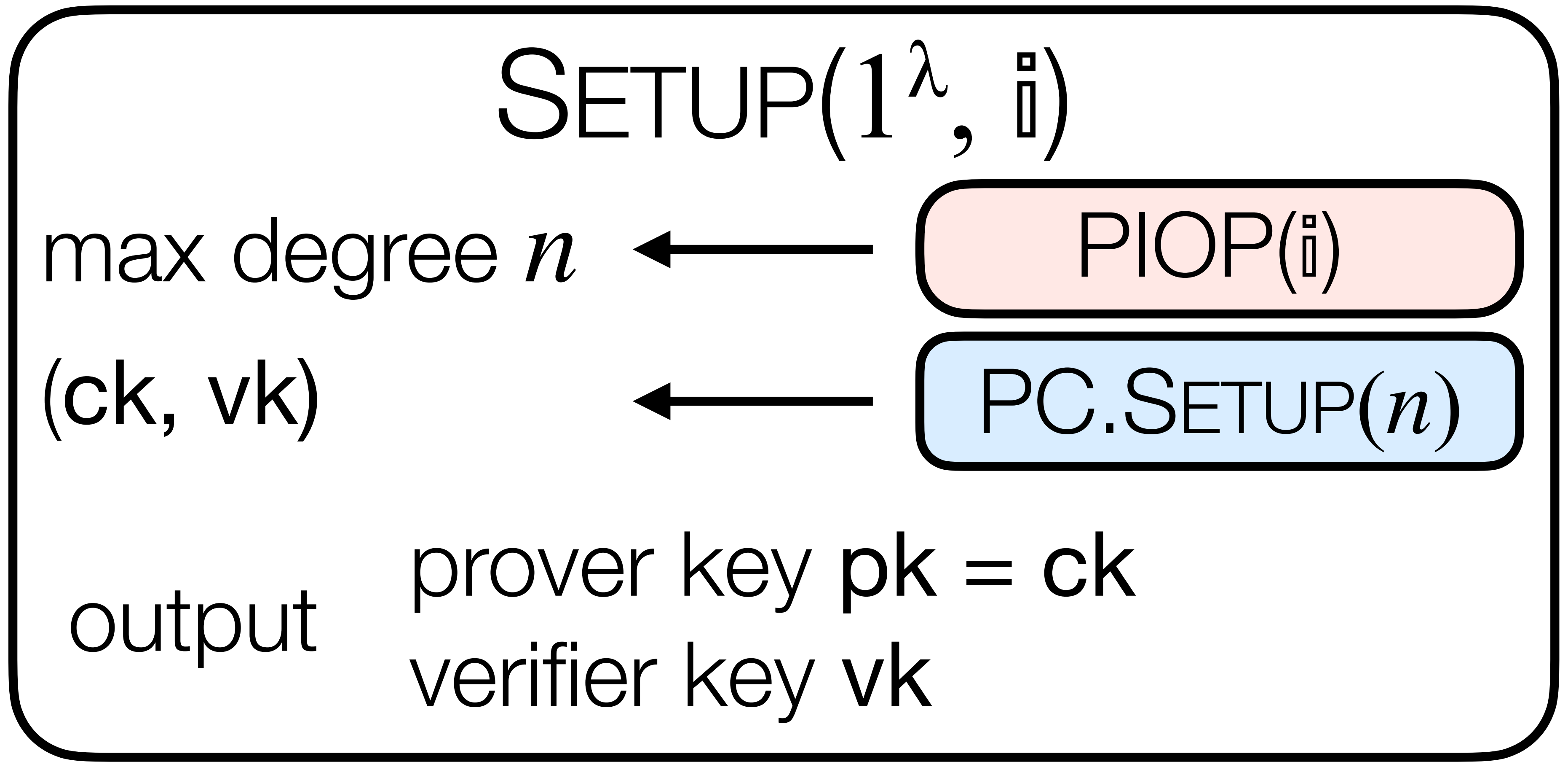
[CHMMVW20, BFS20]

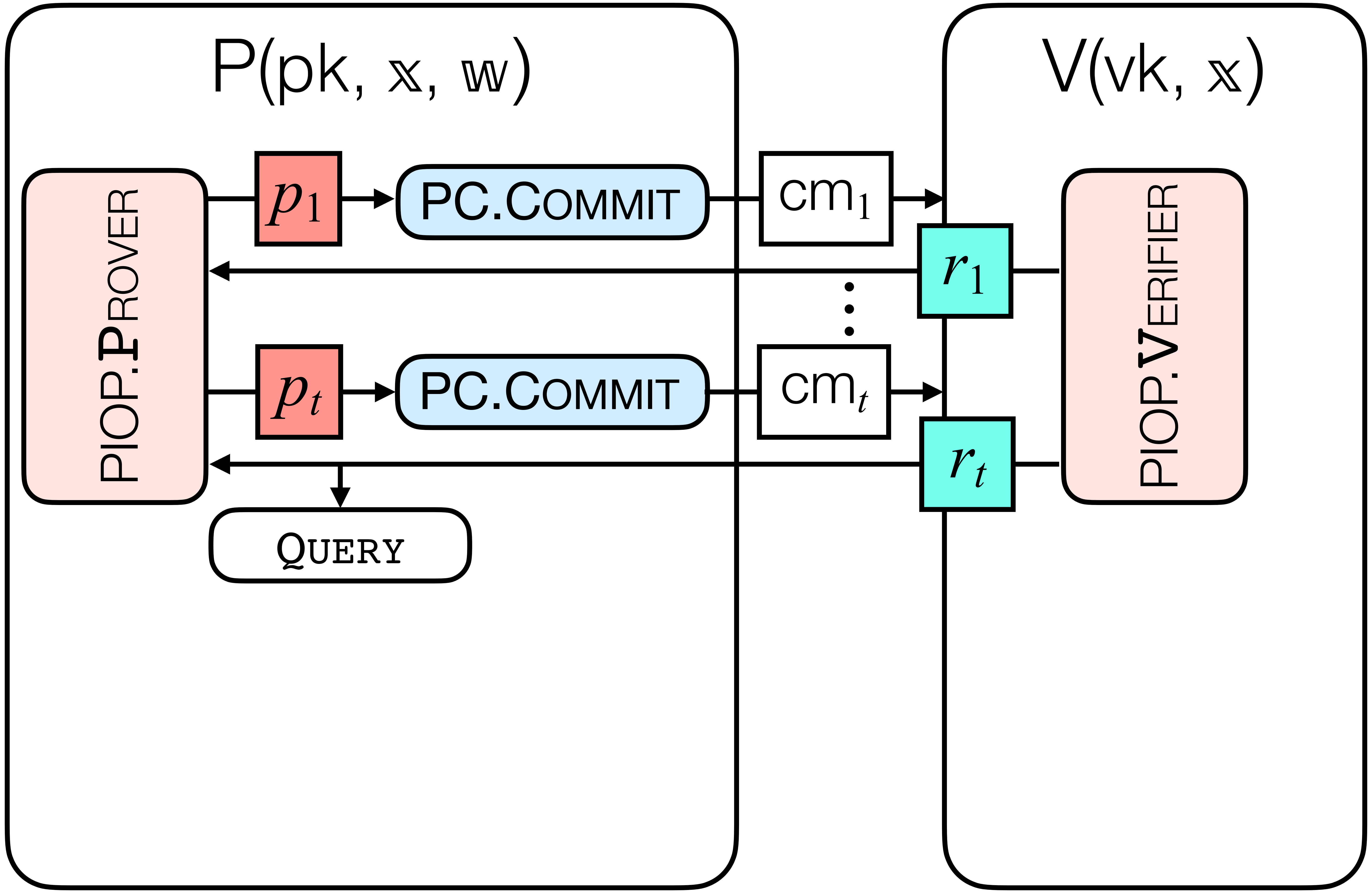




PIOPs + PC Schemes → SNARK

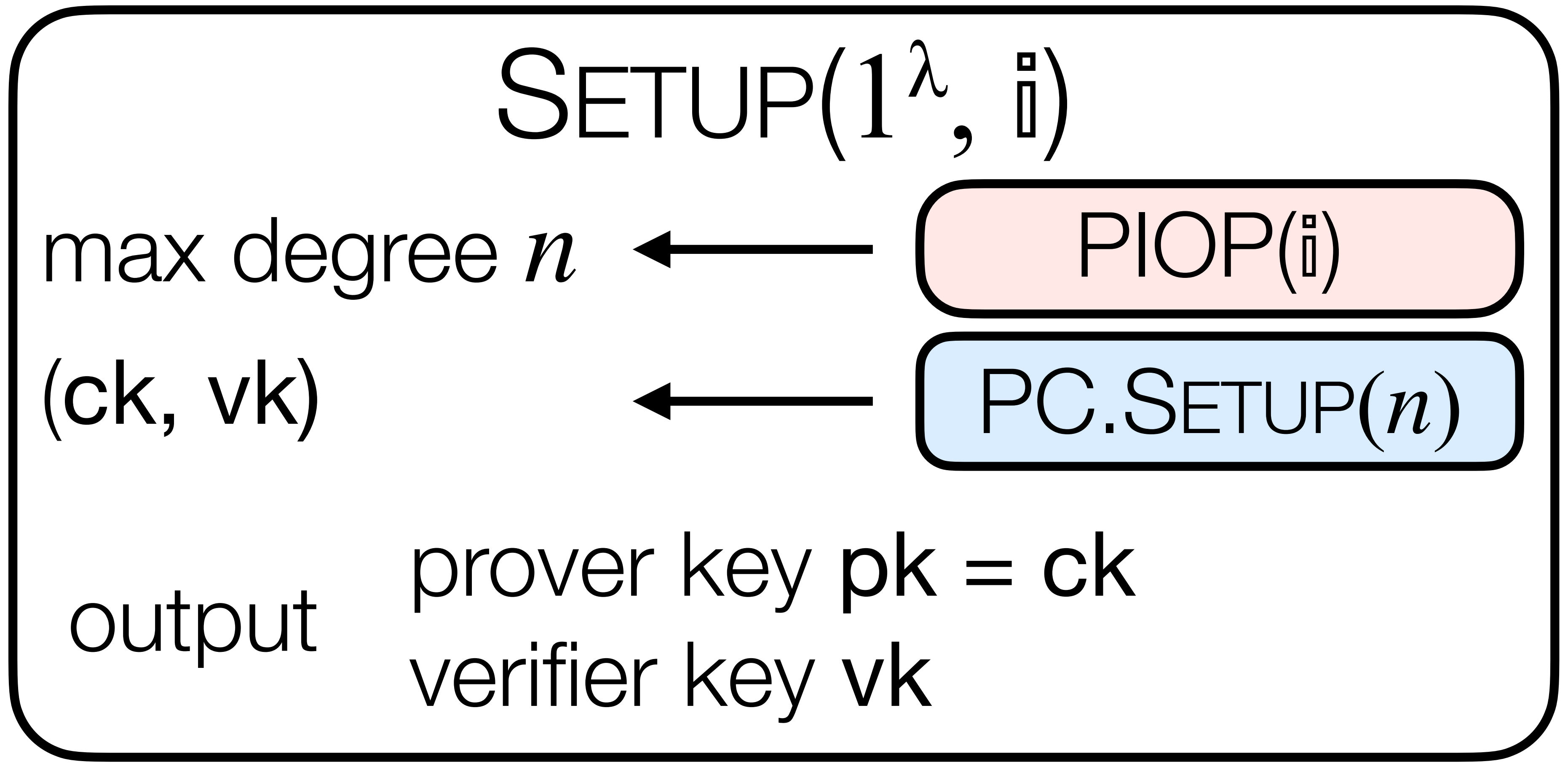
[CHMMVW20, BFS20]

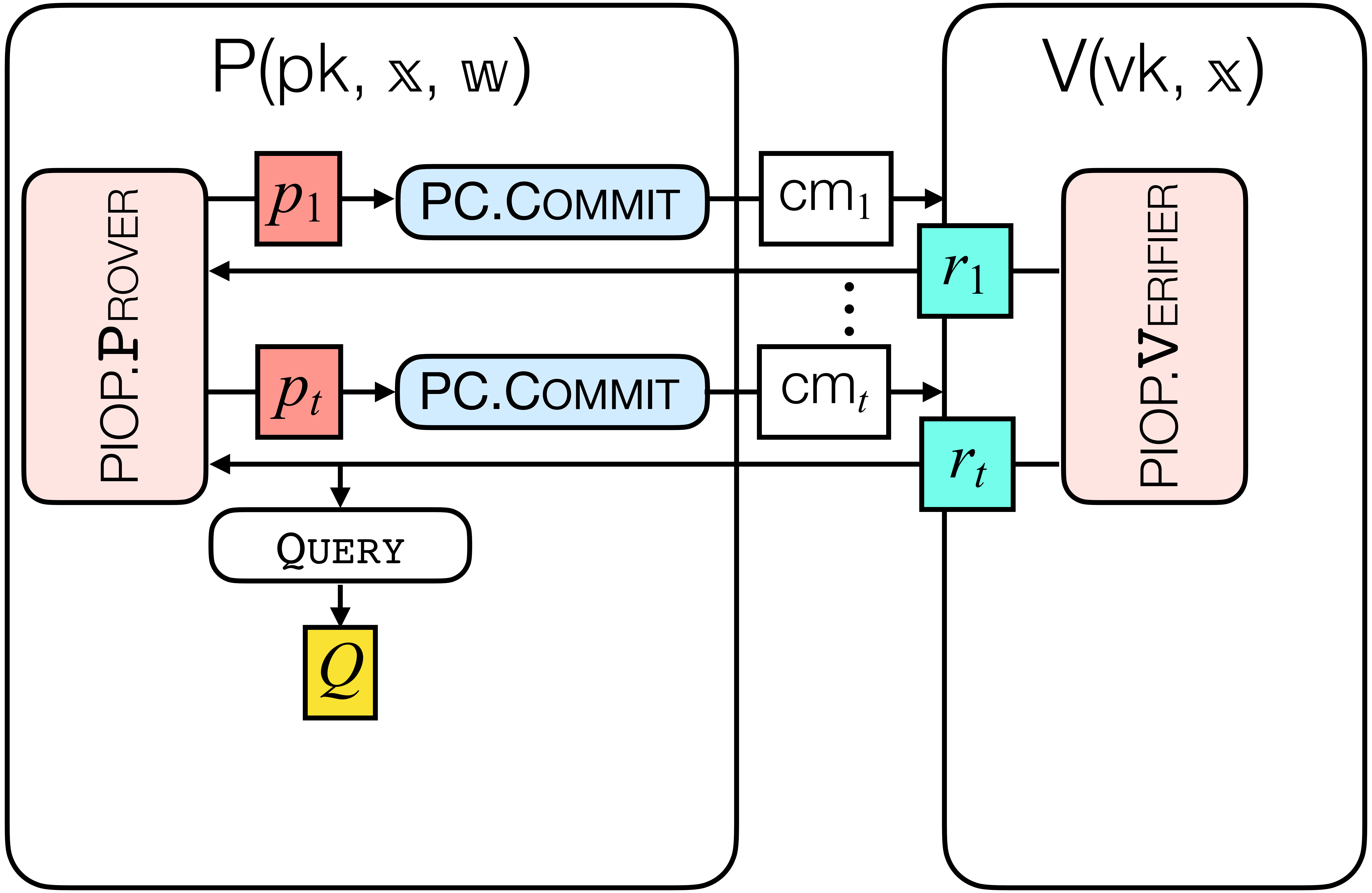




PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]



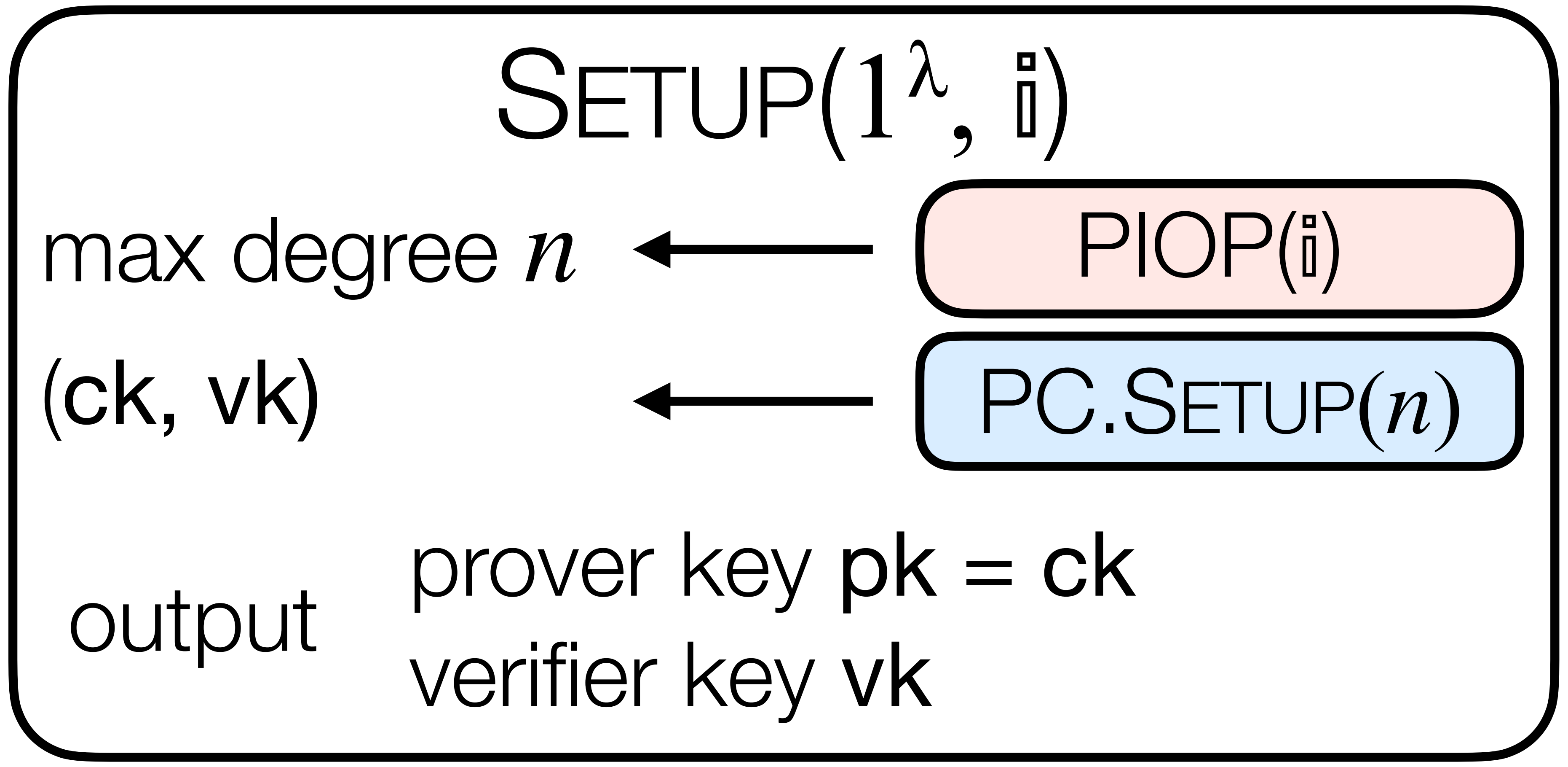


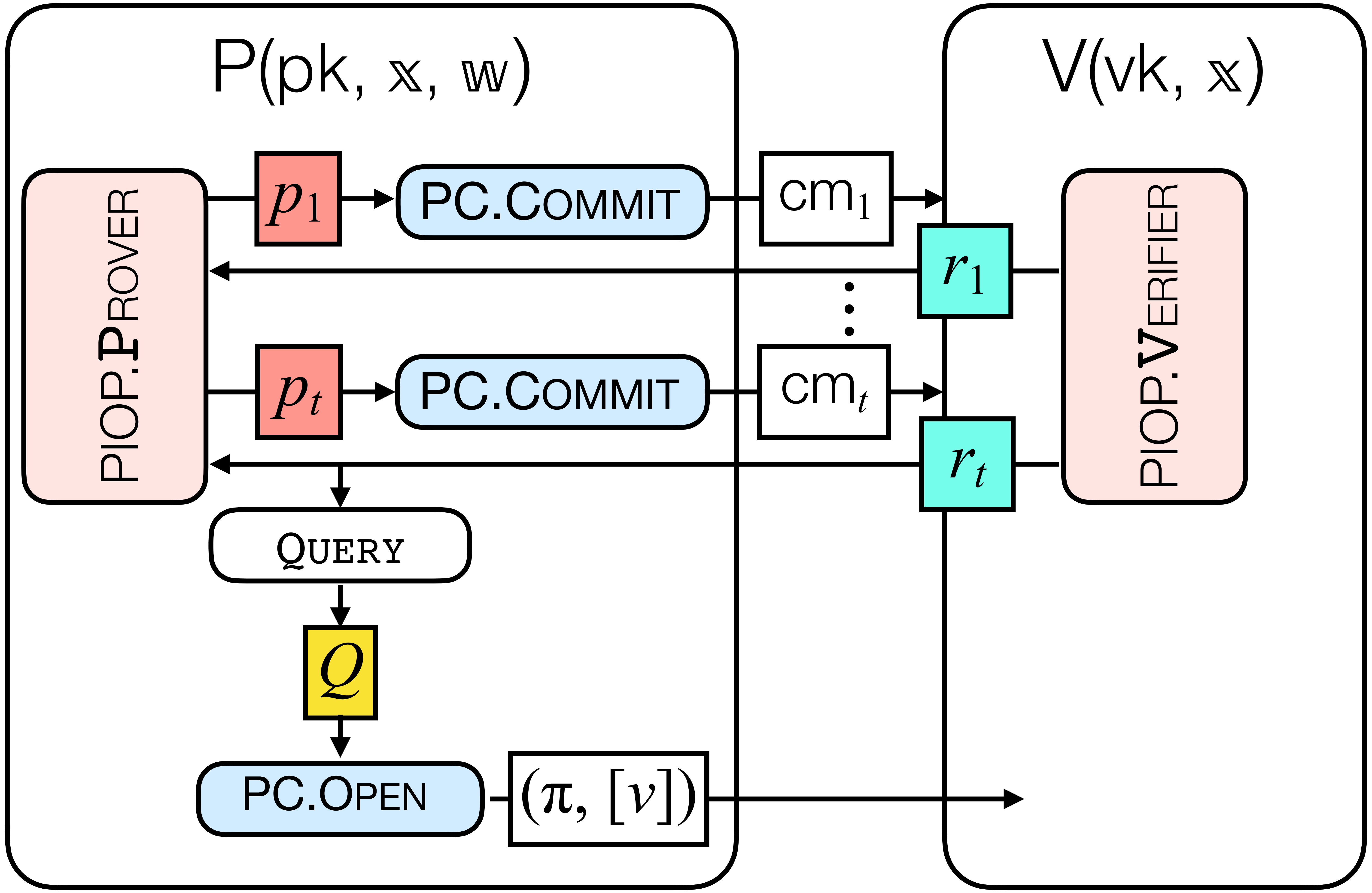
PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]

PIOPs + PC Schemes → SNARK

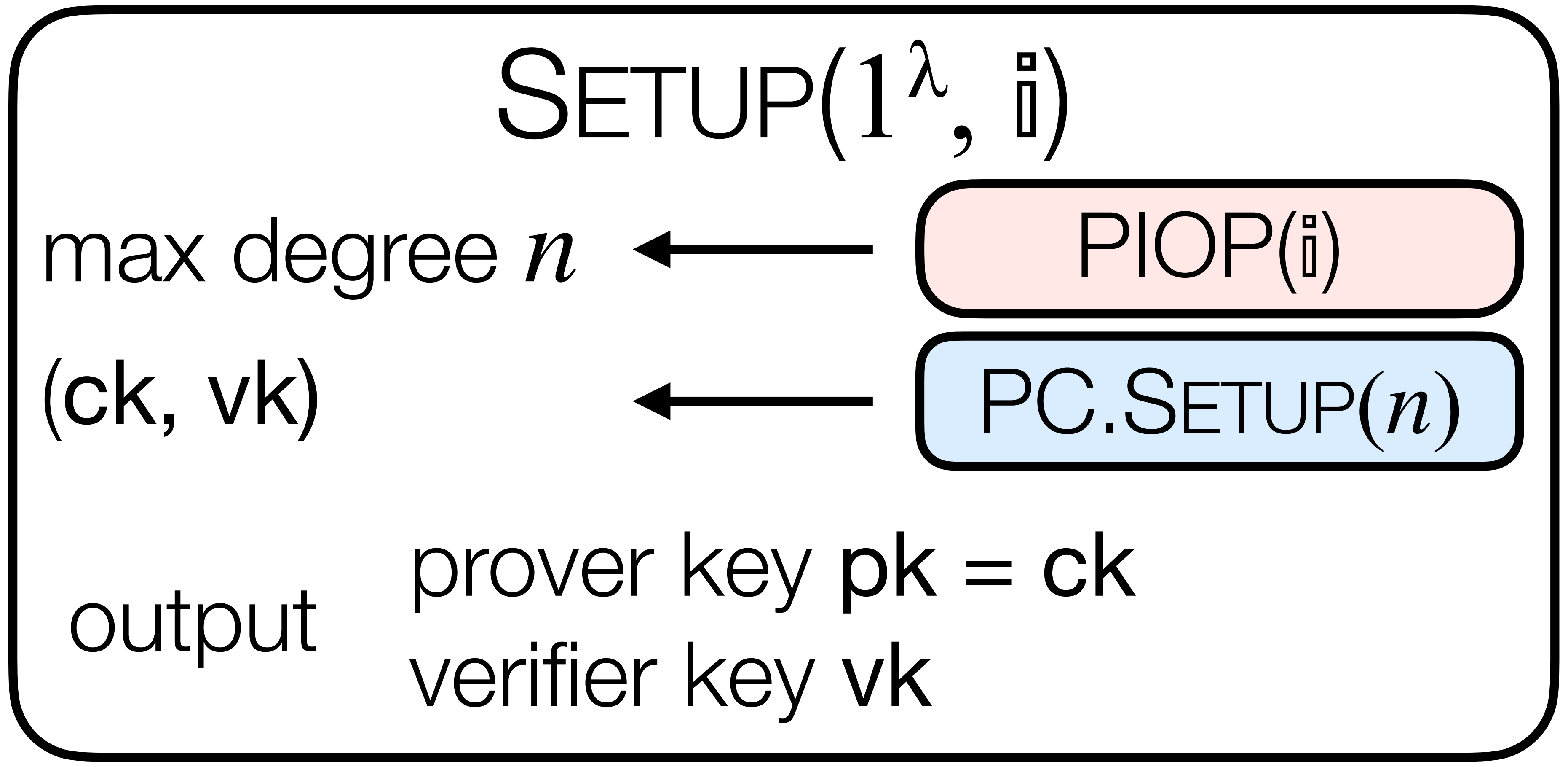
[CHMMVW20, BFS20]

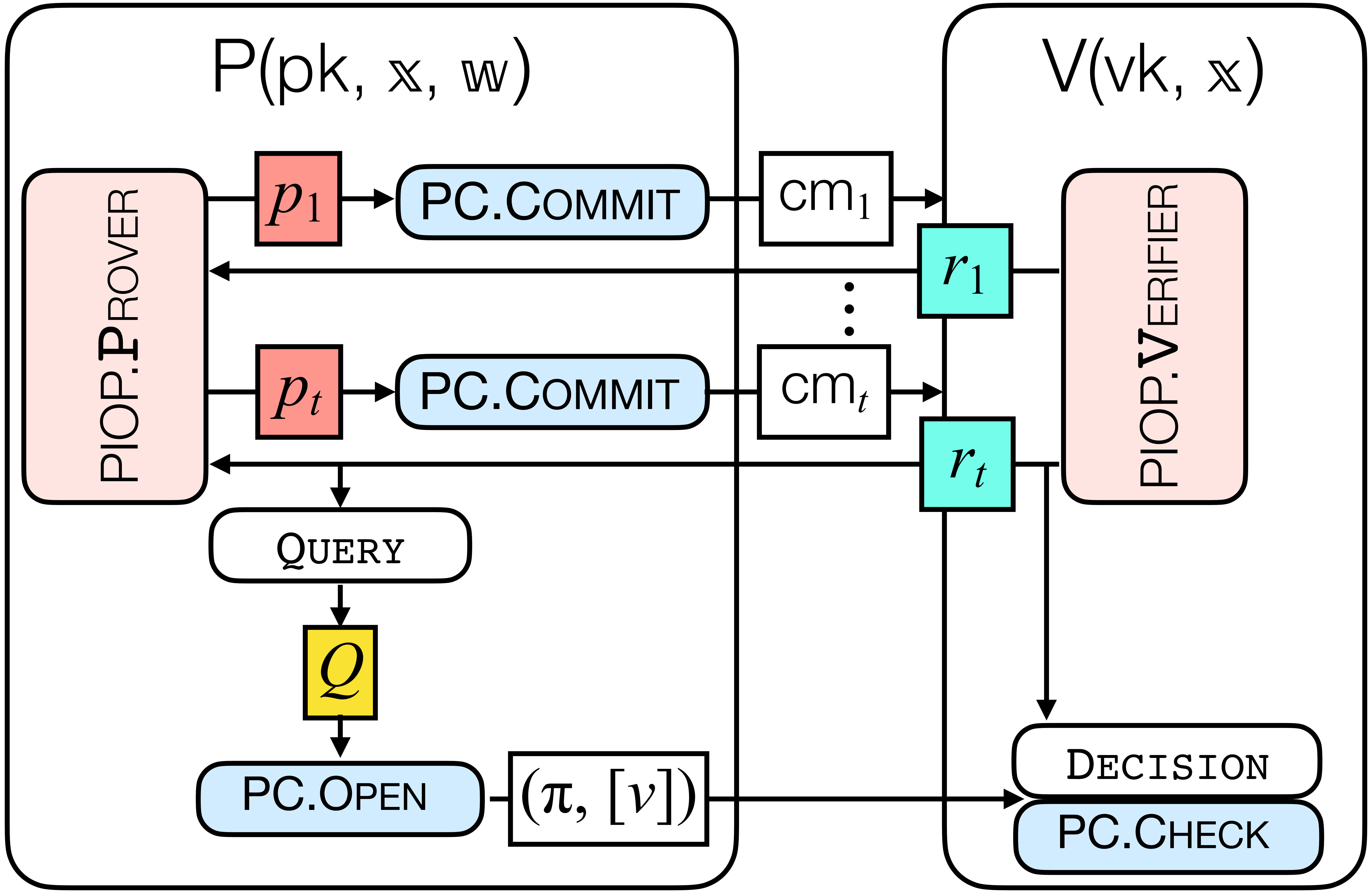




PIOPs + PC Schemes → SNARK

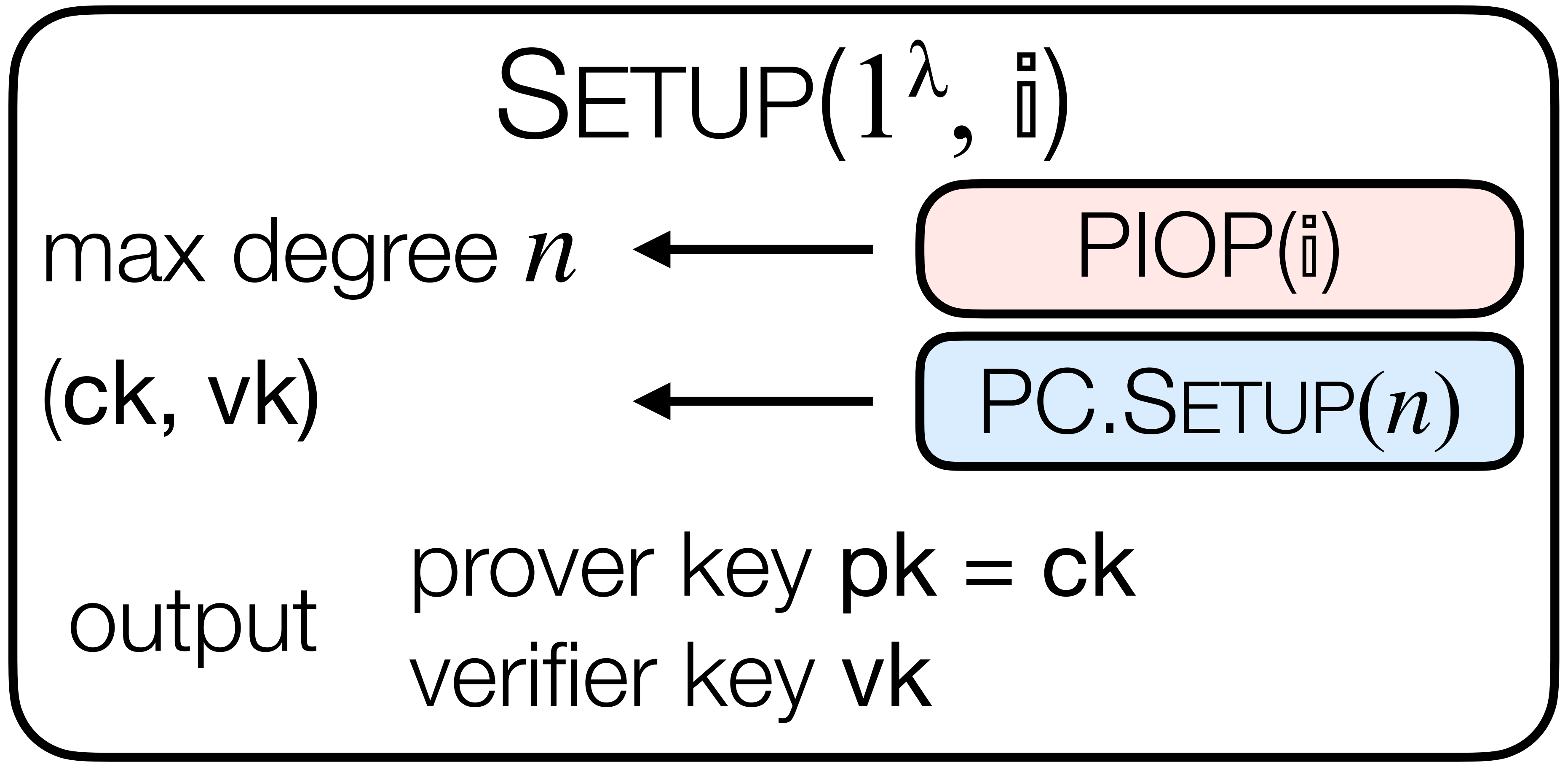
[CHMMVW20, BFS20]

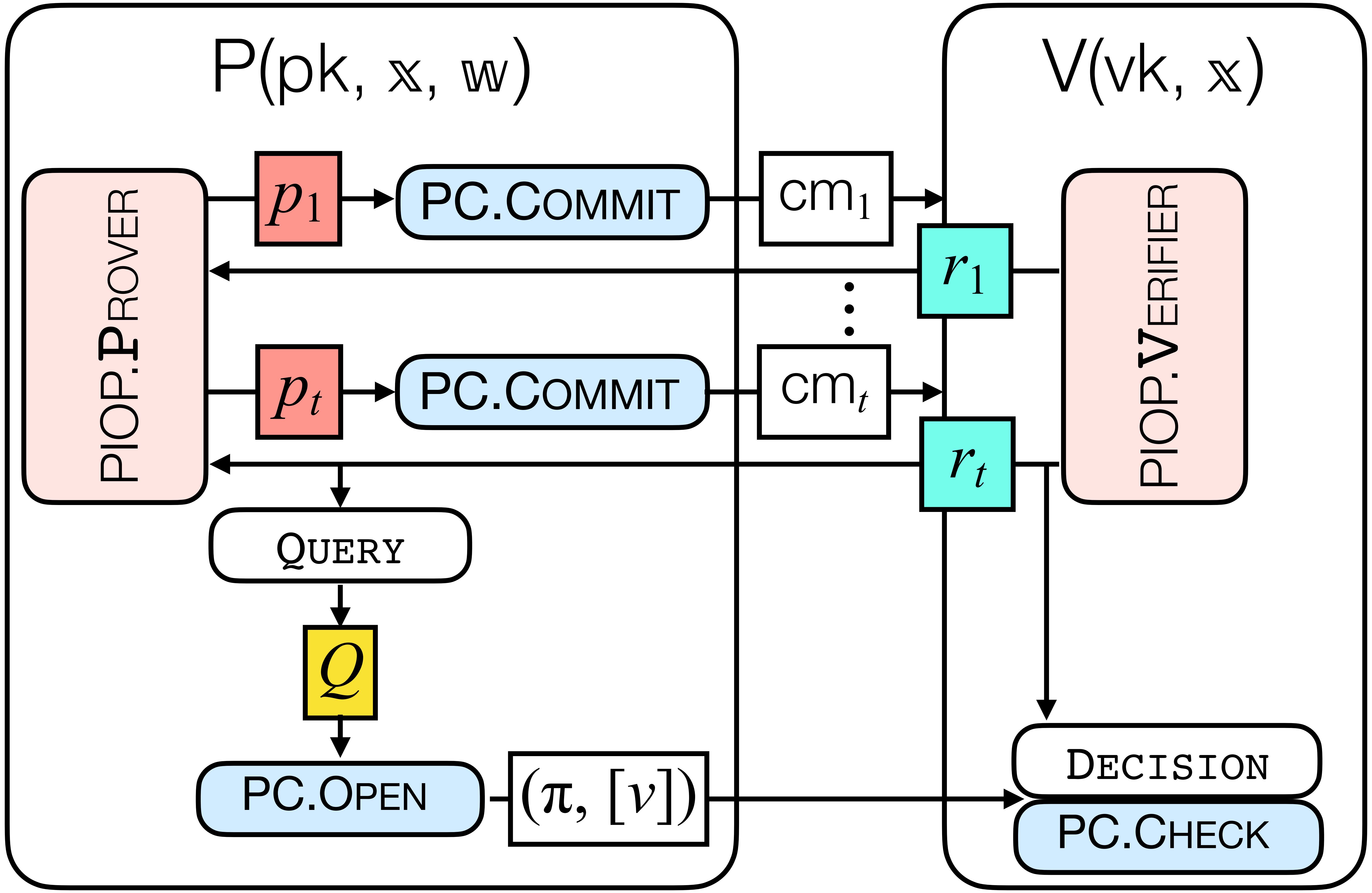




PIOPs + PC Schemes → SNARK

[CHMMVW20, BFS20]





+ Fiat—Shamir to get non-interactivity

Prior PIOP-based SNARKs for R1CS

[CHMMVW20, S21]

Prior PIOP-based SNARKs for R1CS

[CHMMVW20, S21]

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Prior PIOP-based SNARKs for R1CS

[CHMMVW20, S21]

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Linear checks:

$z_A = Az, z_B = Bz, z_C = Cz$

Prior PIOP-based SNARKs for R1CS

[CHMMVW20, S21]

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Linear checks:

$z_A = Az, z_B = Bz, z_C = Cz$

Nonlinear "row" checks:

$z_A \circ z_B = z_C$

Prior PIOP-based SNARKs for R1CS

[CHMMVW20, S21]

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Linear checks:

$z_A = Az, z_B = Bz, z_C = Cz$

Nonlinear "row" checks:

$z_A \circ z_B = z_C$

Prior PIOP-based SNARKs for R1CS

[CHMMVW20, S21]

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Linear checks:

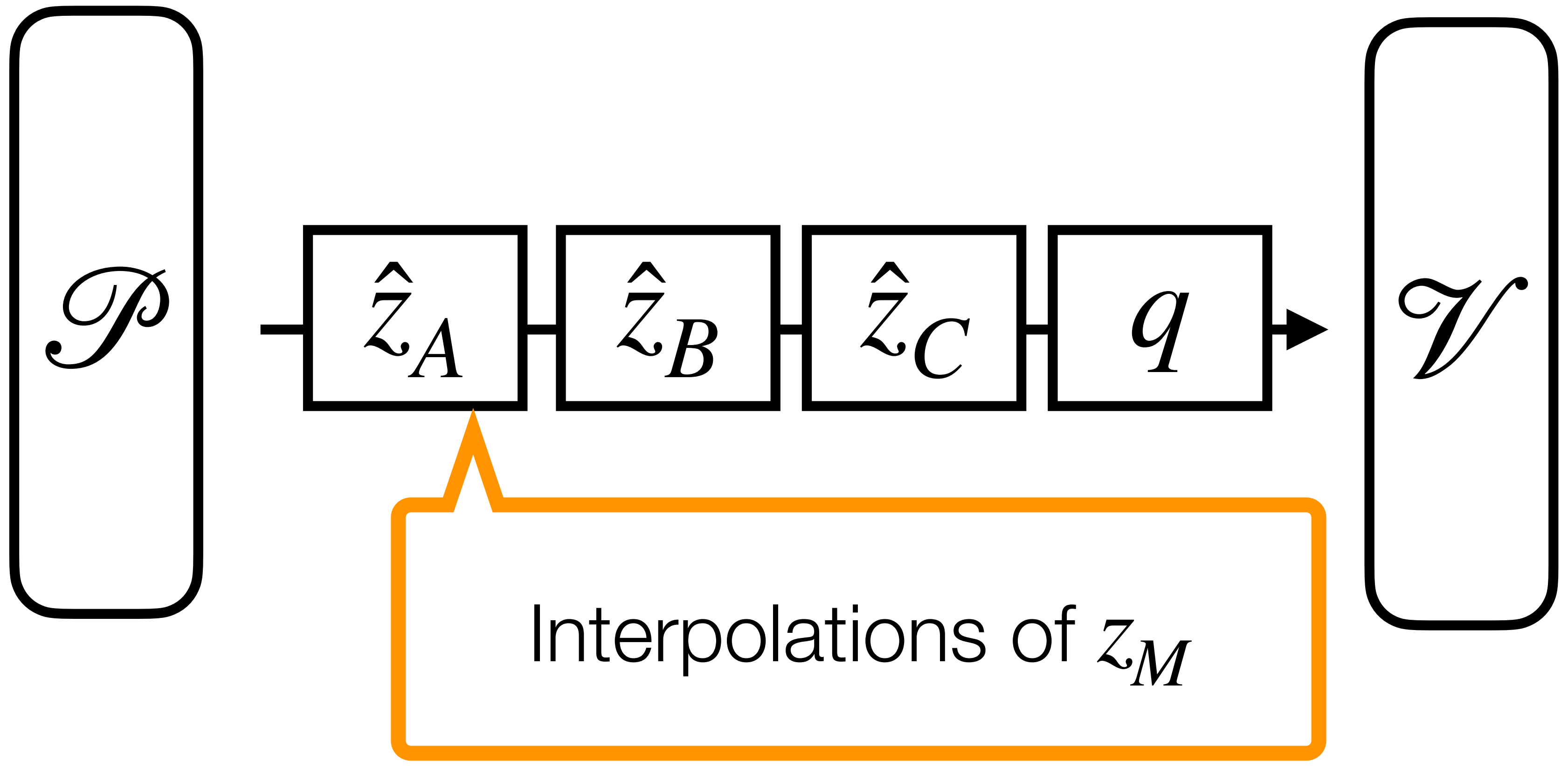
$$z_A = Az,\ z_B = Bz,\ z_C = Cz$$

Nonlinear "row" checks:

$$z_A \circ z_B = z_C$$

Rowcheck subPIOP

Usually quite cheap!



Prior PIOP-based SNARKs for R1CS

[CHMMVW20, S21]

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

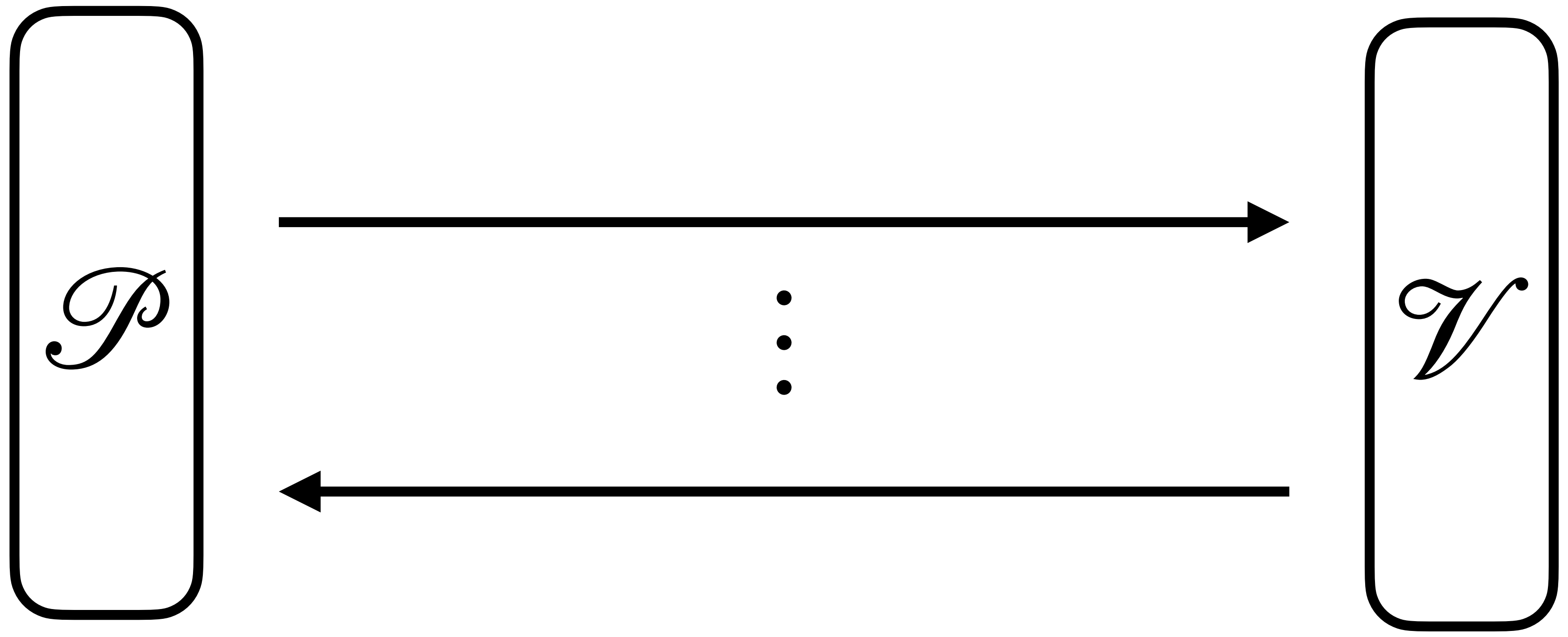
$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Linear checks:

$$z_A = Az,\ z_B = Bz,\ z_C = Cz$$

Lincheck subPIOP

Usually most expensive part!

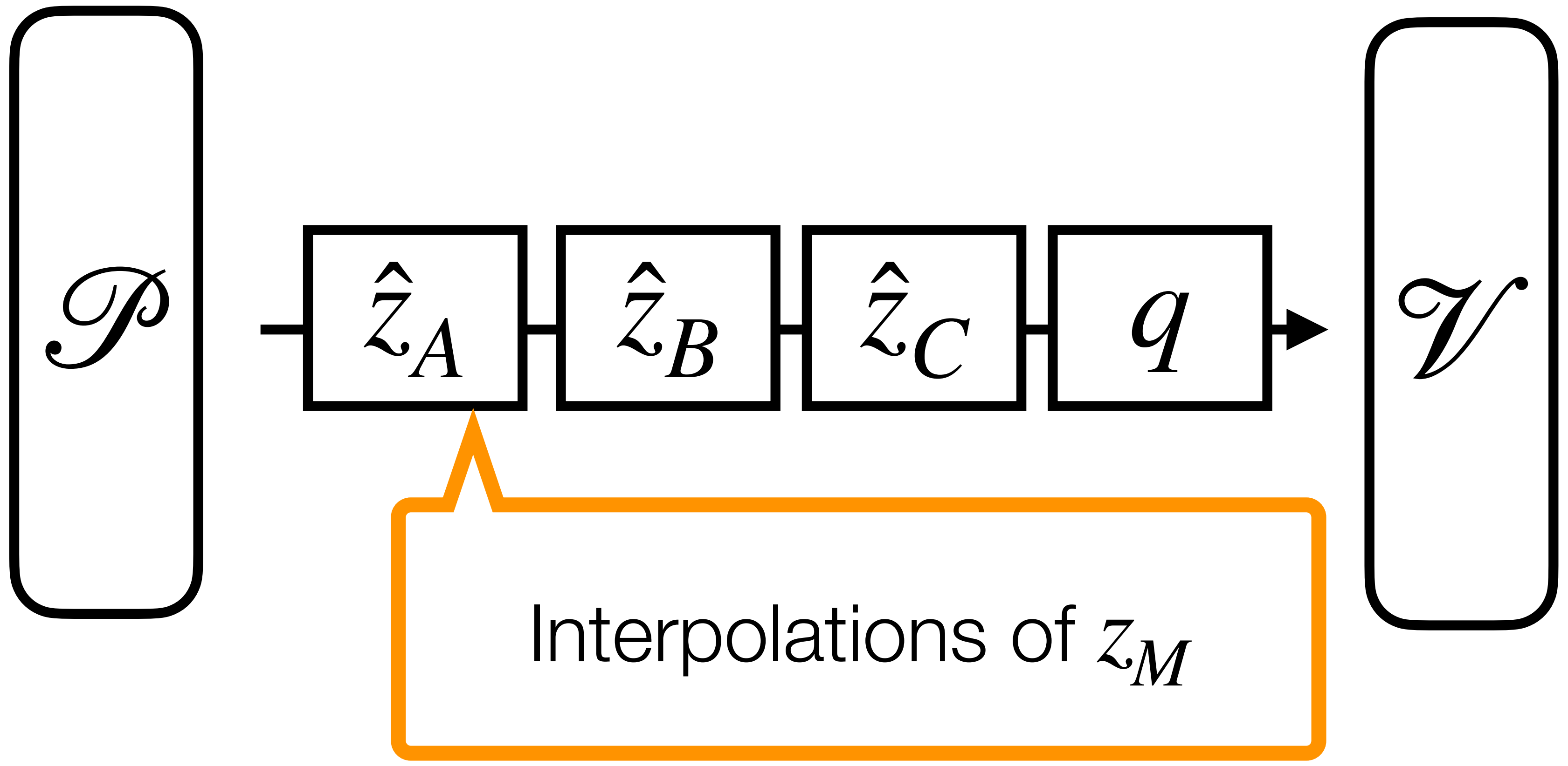


Nonlinear "row" checks:

$$z_A \circ z_B = z_C$$

Rowcheck subPIOP

Usually quite cheap!



Prior PIOP-based SNARKs for R1CS

[CHMMVW20, S21]

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Linear checks:

$$z_A = Az, z_B = Bz, z_C = Cz$$

Lincheck subPIOP

Usually most expensive part!

Requires numerous commitments, openings, and evaluation proofs

In contrast, circuit-specific SNARKs like Groth16 require no extra group elements

Nonlinear "row" checks:

$$z_A \circ z_B = z_C$$

Rowcheck subPIOP

Usually quite cheap!

Compiles to only 4 group elements!

A New Lincheck

Linchecks via coefficient-equality

Linchecks via coefficient-equality

Step 1 : Express Lincheck as a linear combination of matrix column vectors.

Linchecks via coefficient-equality

Step 1 : Express Lincheck as a linear combination of matrix column vectors.

$$A \quad \times \quad z \;=\; z_A$$

Linchecks via coefficient-equality

Step 1 : Express Lincheck as a linear combination of matrix column vectors.

$$A \quad \times \quad z = z_A$$

$\Longleftrightarrow$

Linchecks via coefficient-equality

Step 1 : Express Lincheck as a linear combination of matrix column vectors.

$$A \quad \times \quad z \quad = \quad z_A$$

$$\Longleftrightarrow \quad \begin{bmatrix} \uparrow & \uparrow & \uparrow & & \uparrow \\ a_1 & a_2 & a_3 & \cdots & a_n \\ \downarrow & \downarrow & \downarrow & & \downarrow \end{bmatrix} \times \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} \\ z_A \\ \\ \end{bmatrix}$$

Linchecks via coefficient-equality

Step 1 : Express Lincheck as a linear combination of matrix column vectors.

$$A \times z = z_A$$

$$\Longleftrightarrow \quad \begin{bmatrix} \updownarrow & \updownarrow & \updownarrow & & \updownarrow \\ a_1 & a_2 & a_3 & \cdots & a_n \\ \updownarrow & \updownarrow & \updownarrow & & \updownarrow \end{bmatrix} \times \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} z_A \end{bmatrix}$$

$$\Longleftrightarrow$$

Linchecks via coefficient-equality

Step 1 : Express Lincheck as a linear combination of matrix column vectors.

$$A \times z = z_A$$

$$\iff \begin{bmatrix} \updownarrow & \updownarrow & \updownarrow & & \updownarrow \\ a_1 & a_2 & a_3 & \cdots & a_n \\ \updownarrow & \updownarrow & \updownarrow & & \updownarrow \end{bmatrix} \times \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} z_A \end{bmatrix}$$

$$\iff z_1 \cdot \begin{bmatrix} \updownarrow \\ a_1 \\ \updownarrow \end{bmatrix} + z_2 \cdot \begin{bmatrix} \updownarrow \\ a_2 \\ \updownarrow \end{bmatrix} + \cdots + z_n \cdot \begin{bmatrix} \updownarrow \\ a_n \\ \updownarrow \end{bmatrix} = \begin{bmatrix} z_A \end{bmatrix}$$

Linchecks via coefficient-equality

Linchecks via coefficient-equality

Step 2 : Interpolate the column vectors using Lagrange interpolation.

Linchecks via coefficient-equality

Step 2 : Interpolate the column vectors using Lagrange interpolation.

$$z_1 \cdot \begin{bmatrix} \uparrow \\ a_1 \\ \downarrow \end{bmatrix} + z_2 \cdot \begin{bmatrix} \uparrow \\ a_2 \\ \downarrow \end{bmatrix} + \quad \cdots \quad + z_n \cdot \begin{bmatrix} \uparrow \\ a_n \\ \downarrow \end{bmatrix} = \begin{bmatrix} z_A \end{bmatrix}$$

Linchecks via coefficient-equality

Step 2 : Interpolate the column vectors using Lagrange interpolation.

$$z_1 \cdot \begin{bmatrix} a_1 \end{bmatrix} + z_2 \cdot \begin{bmatrix} a_2 \end{bmatrix} + \quad \dots \quad + z_n \cdot \begin{bmatrix} a_n \end{bmatrix} = \begin{bmatrix} z_A \end{bmatrix}$$

Interpolate a_i-s over $\{1, \dots, n\}$

Linchecks via coefficient-equality

Step 2 : Interpolate the column vectors using Lagrange interpolation.

$$z_1 \cdot \begin{bmatrix} a_1 \end{bmatrix} + z_2 \cdot \begin{bmatrix} a_2 \end{bmatrix} + \quad \dots \quad + z_n \cdot \begin{bmatrix} a_n \end{bmatrix} = \begin{bmatrix} z_A \end{bmatrix}$$

Interpolate a_i-s over $\{1,\ldots,n\}$

Interpolate z_A over $\{1,\ldots,n\}$

Linchecks via coefficient-equality

Step 2 : Interpolate the column vectors using Lagrange interpolation.

$$z_1 \cdot \begin{bmatrix} a_1 \end{bmatrix} + z_2 \cdot \begin{bmatrix} a_2 \end{bmatrix} + \quad \dots \quad + z_n \cdot \begin{bmatrix} a_n \end{bmatrix} = \begin{bmatrix} z_A \end{bmatrix}$$

Interpolate a_i-s over $\{1, \dots, n\}$

Interpolate z_A over $\{1, \dots, n\}$

$$z_1 \cdot \hat{a}_1(X) + z_2 \cdot \hat{a}_2(X) + \quad \dots \quad + z_n \cdot \hat{a}_n(X) = \hat{z}_A(X)$$

Linchecks via coefficient-equality

Step 2 : Interpolate the column vectors using Lagrange interpolation.

$$z_1 \cdot \begin{bmatrix} a_1 \end{bmatrix} + z_2 \cdot \begin{bmatrix} a_2 \end{bmatrix} + \quad \dots \quad + z_n \cdot \begin{bmatrix} a_n \end{bmatrix} = \begin{bmatrix} z_A \end{bmatrix}$$

Interpolate a_i-s over $\{1,\dots,n\}$

Interpolate z_A over $\{1,\dots,n\}$

$$z_1 \cdot \hat{a}_1(X) + z_2 \cdot \hat{a}_2(X) + \quad \dots \quad + z_n \cdot \hat{a}_n(X) = \hat{z}_A(X)$$

Now we can express the Lincheck in the language of polynomials:

$$\forall i \in \{1,\dots,n\}, \qquad \hat{z}_A(i) = \sum_j \hat{a}_j(i) \cdot z[j]$$

Linchecks via coefficient-equality

Linchecks via coefficient-equality

Step 3 : Now that Lincheck is written in the language of polynomials, we can argue that:

Linchecks via coefficient-equality

Step 3 : Now that Lincheck is written in the language of polynomials, we can argue that:

Lincheck

Prover knows $z \in \mathbb{F}^n$ such that

$$z_A = Az$$

$$z_B = Bz$$

$$z_C = Cz$$

Linchecks via coefficient-equality

Step 3 : Now that Lincheck is written in the language of polynomials, we can argue that:

Lincheck

Prover knows $z \in \mathbb{F}^n$ such that

$$z_A = Az$$

$$z_B = Bz$$

$$z_C = Cz$$

$\Longleftrightarrow$

Linchecks via coefficient-equality

Step 3 : Now that Lincheck is written in the language of polynomials, we can argue that:

Lincheck

Prover knows $z \in \mathbb{F}^n$ such that

$$z_A = Az$$

$$z_B = Bz$$

$$z_C = Cz$$

$$\Longleftrightarrow$$

Coefficient-equality constraint

Prover knows $z \in \mathbb{F}^n$ such that

Linchecks via coefficient-equality

Step 3 : Now that Lincheck is written in the language of polynomials, we can argue that:

Lincheck

Prover knows $z \in \mathbb{F}^n$ such that

$$z_A = Az$$

$$z_B = Bz$$

$$z_C = Cz$$

$$\Longleftrightarrow$$

Coefficient-equality constraint

Prover knows $z \in \mathbb{F}^n$ such that

$$\hat{z}_A(X) = z_1 \cdot \hat{a}_1(X) + \cdots + z_n \cdot \hat{a}_n(X)$$

$$\hat{z}_B(X) = z_1 \cdot \hat{b}_1(X) + \cdots + z_n \cdot \hat{b}_n(X)$$

$$\hat{z}_C(X) = z_1 \cdot \hat{c}_1(X) + \cdots + z_n \cdot \hat{c}_n(X)$$

Linchecks via coefficient-equality

Step 3 : Now that Lincheck is written in the language of polynomials, we can argue that:

Lincheck

Prover knows $z \in \mathbb{F}^n$ such that

$$z_A = Az$$

$$z_B = Bz$$

$$z_C = Cz$$

$$\Longleftrightarrow$$

Coefficient-equality constraint

Prover knows $z \in \mathbb{F}^n$ such that

$$\hat{z}_A(X) = z_1 \cdot \hat{a}_1(X) + \cdots + z_n \cdot \hat{a}_n(X)$$

$$\hat{z}_B(X) = z_1 \cdot \hat{b}_1(X) + \cdots + z_n \cdot \hat{b}_n(X)$$

$$\hat{z}_C(X) = z_1 \cdot \hat{c}_1(X) + \cdots + z_n \cdot \hat{c}_n(X)$$

Linchecks via coefficient-equality

Step 3 : Now that Lincheck is written in the language of polynomials, we can argue that:

Lincheck

Prover knows $z \in \mathbb{F}^n$ such that

$$z_A = Az$$

$$z_B = Bz$$

$$z_C = Cz$$

$\Longleftrightarrow$

Coefficient-equality constraint

Prover knows $z \in \mathbb{F}^n$ such that

$$\hat{z}_A(X) = z_1 \cdot \hat{a}_1(X) + \cdots + z_n \cdot \hat{a}_n(X)$$

$$\hat{z}_B(X) = z_1 \cdot \hat{b}_1(X) + \cdots + z_n \cdot \hat{b}_n(X)$$

$$\hat{z}_C(X) = z_1 \cdot \hat{c}_1(X) + \cdots + z_n \cdot \hat{c}_n(X)$$

Same coefficients in all!

New Approach for Lincheck

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Linear checks:

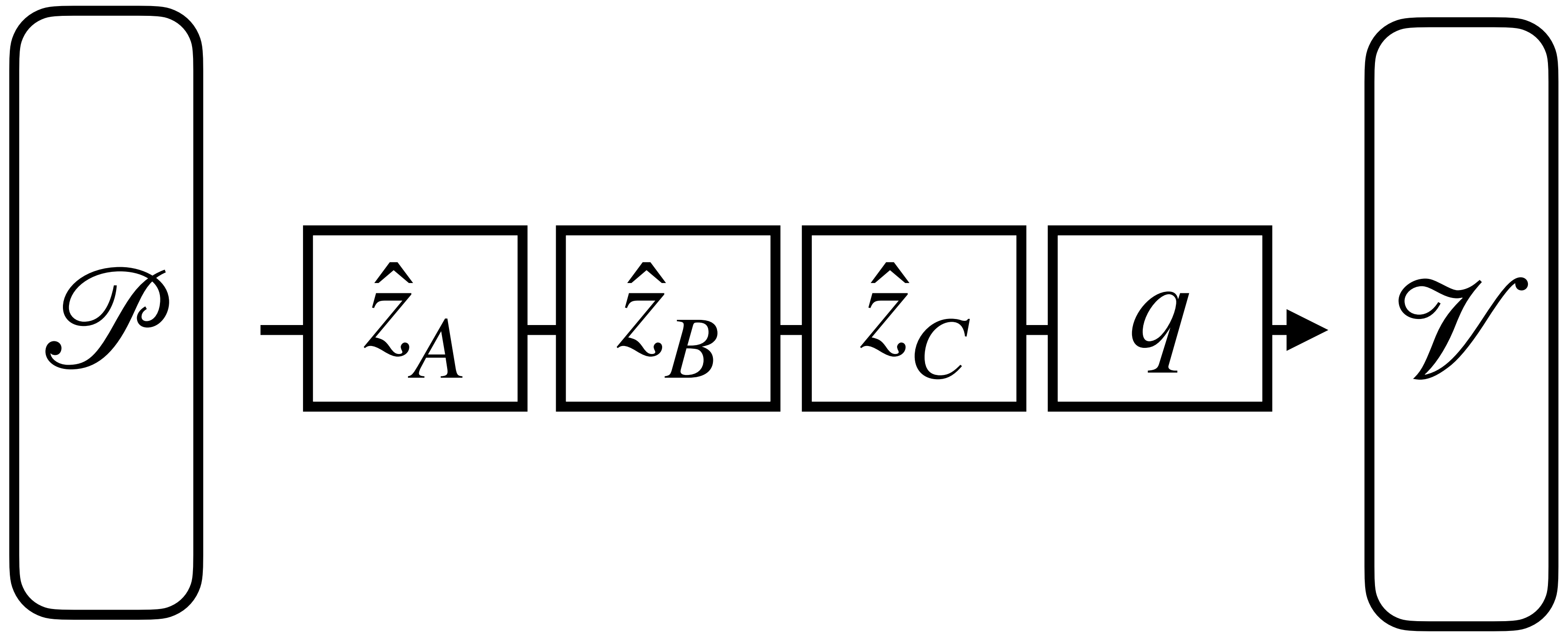
$z_A = Az, z_B = Bz, z_C = Cz$

Nonlinear "row" checks:

$z_A \circ z_B = z_C$

Rowcheck subPIOP

Usually quite cheap!



New Approach for Lincheck

R1CS consists of triples $((A, B, C), x, w)$ such that the following holds for $z = (x, w)$:

$Az \circ Bz = Cz$, or equivalently the following checks are satisfied:

Linear checks:

$z_A = Az,\, z_B = Bz,\, z_C = Cz$

Lincheck via coefficient-equality

How to enforce?

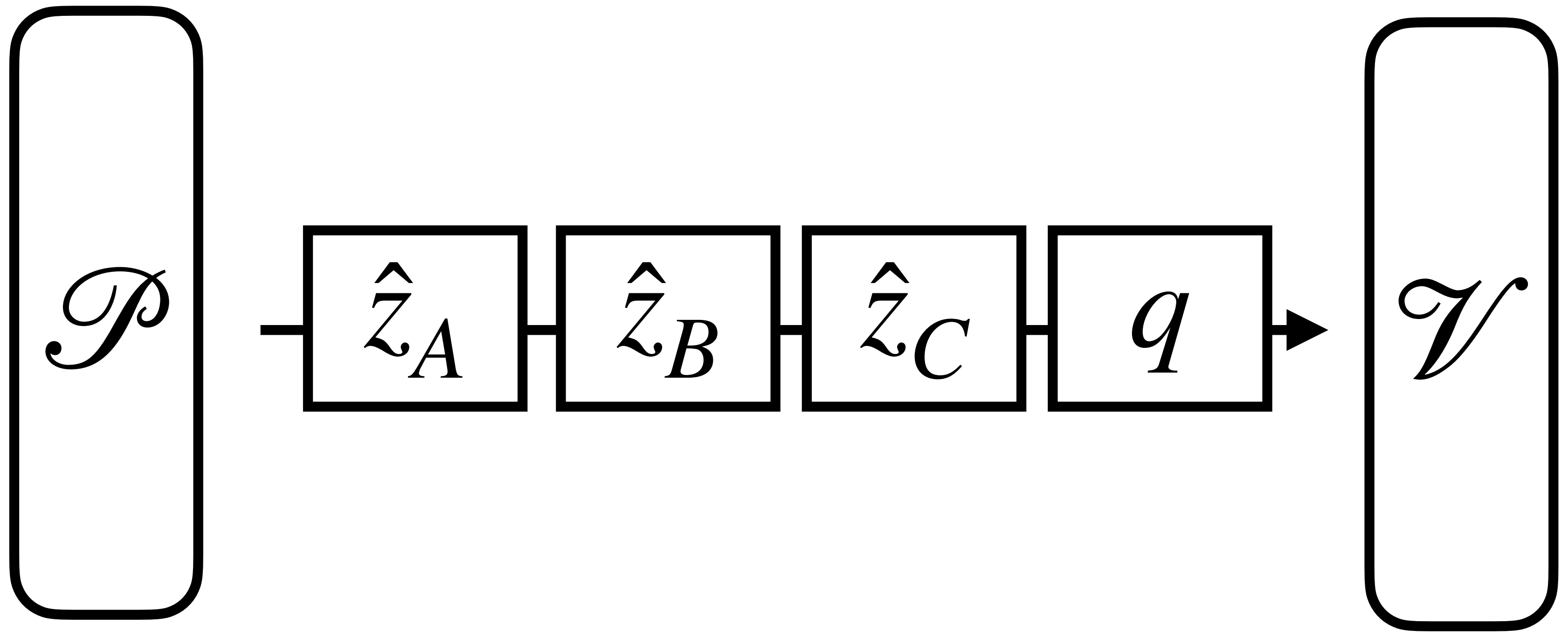
Equifficient Polynomial Commitment Schemes!

Nonlinear "row" checks:

$z_A \circ z_B = z_C$

Rowcheck subPIOP

Usually quite cheap!



New Tool: EPC schemes

Equifficient constraints

A coefficient-equality or "equifficient" constraint is a set of bases

$$E := \{\mathscr{A}, \mathscr{B}, \mathscr{C}\}$$

where $\mathscr{A} = \{a_1, \ldots, a_n\}$, $\mathscr{B} = \{b_1, \ldots, b_n\}$, $\mathscr{C} = \{c_1, \ldots, c_n\}$

Equifficient constraints

A coefficient-equality or "equifficient" constraint is a set of bases

$$E := \{\mathscr{A}, \mathscr{B}, \mathscr{C}\}$$

where $\mathscr{A} = \{a_1, \ldots, a_n\}$, $\mathscr{B} = \{b_1, \ldots, b_n\}$, $\mathscr{C} = \{c_1, \ldots, c_n\}$

Polynomials $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ are said to satisfy E if they have equal coefficient vectors under bases $\mathscr{A}, \mathscr{B}, \mathscr{C}$ respectively, i.e.:

$$\hat{z}_A = z_1 \cdot \hat{a}_1 + \cdots + z_n \cdot \hat{a}_n$$
$$\hat{z}_B = z_1 \cdot \hat{b}_1 + \cdots + z_n \cdot \hat{b}_n$$
$$\hat{z}_C = z_1 \cdot \hat{c}_1 + \cdots + z_n \cdot \hat{c}_n$$

Equifficient constraints

A coefficient-equality or "equifficient" constraint is a set of bases

$$E := \{\mathscr{A}, \mathscr{B}, \mathscr{C}\}$$

where $\mathscr{A} = \{a_1, \ldots, a_n\}$, $\mathscr{B} = \{b_1, \ldots, b_n\}$, $\mathscr{C} = \{c_1, \ldots, c_n\}$

Polynomials $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ are said to satisfy E if they have equal coefficient vectors under bases $\mathscr{A}, \mathscr{B}, \mathscr{C}$ respectively, i.e.:

$$\hat{z}_A = \boxed{z_1} \cdot \hat{a}_1 + \cdots + \boxed{z_n} \cdot \hat{a}_n$$

$$\hat{z}_B = \boxed{z_1} \cdot \hat{b}_1 + \cdots + \boxed{z_n} \cdot \hat{b}_n$$

$$\hat{z}_C = \boxed{z_1} \cdot \hat{c}_1 + \cdots + \boxed{z_n} \cdot \hat{c}_n$$

Equifficient PC (EPC) schemes

Equifficient PC (EPC) schemes

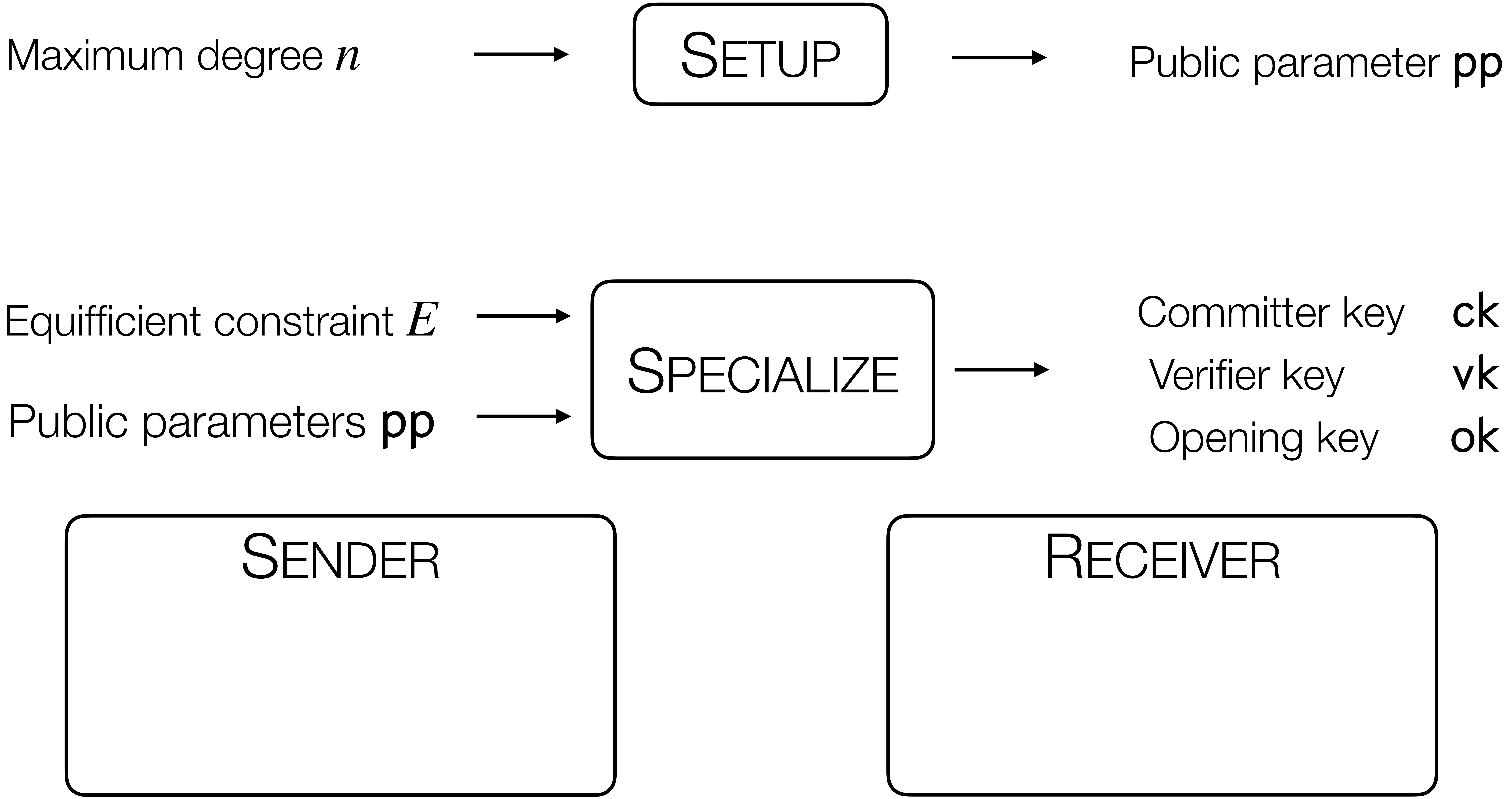
Maximum degree n ⟶ SETUP ⟶ Public parameter **pp**

Equifficient PC (EPC) schemes

Maximum degree n → SETUP → Public parameter pp

Equifficient constraint E
Public parameters pp
→ SPECIALIZE →
Committer key ck
Verifier key vk
Opening key ok

Equifficient PC (EPC) schemes



Equifficient PC (EPC) schemes

Maximum degree n → SETUP → Public parameter pp

Equifficient constraint E → SPECIALIZE → Committer key ck

Public parameters pp → SPECIALIZE → Verifier key vk

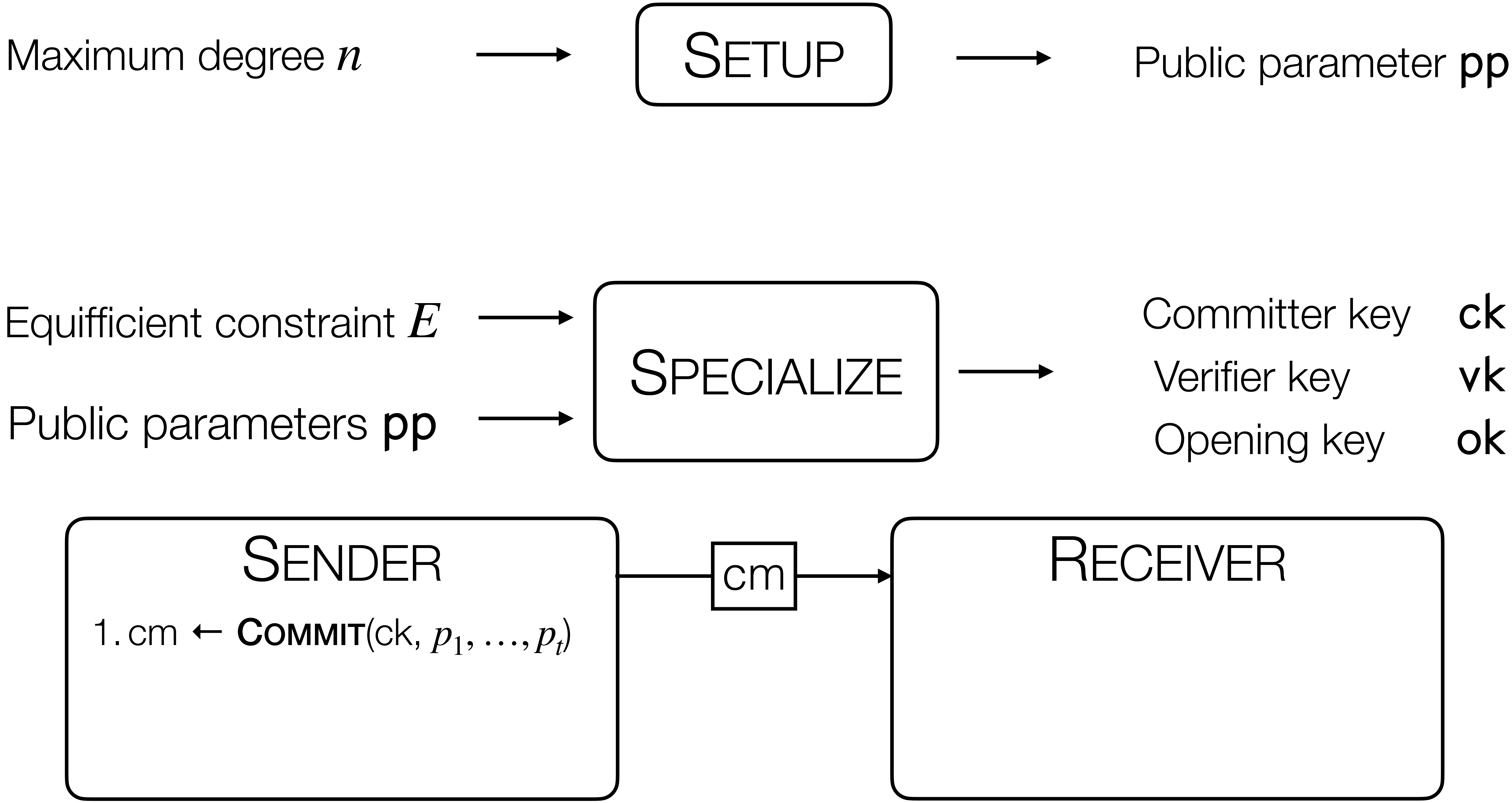
Opening key ok

SENDER

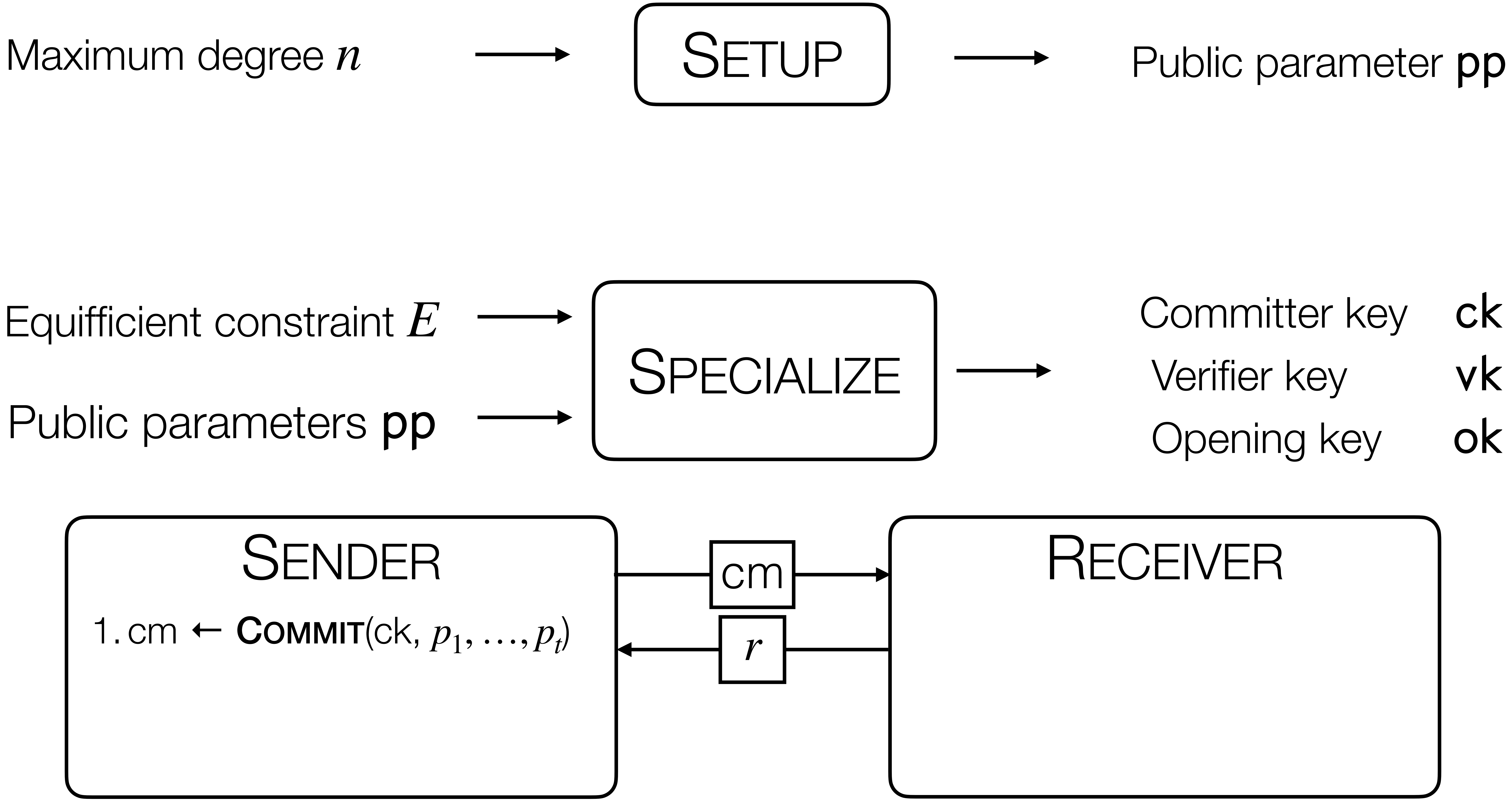
1. cm ← COMMIT(ck, $p_1, \ldots, p_t$)

RECEIVER

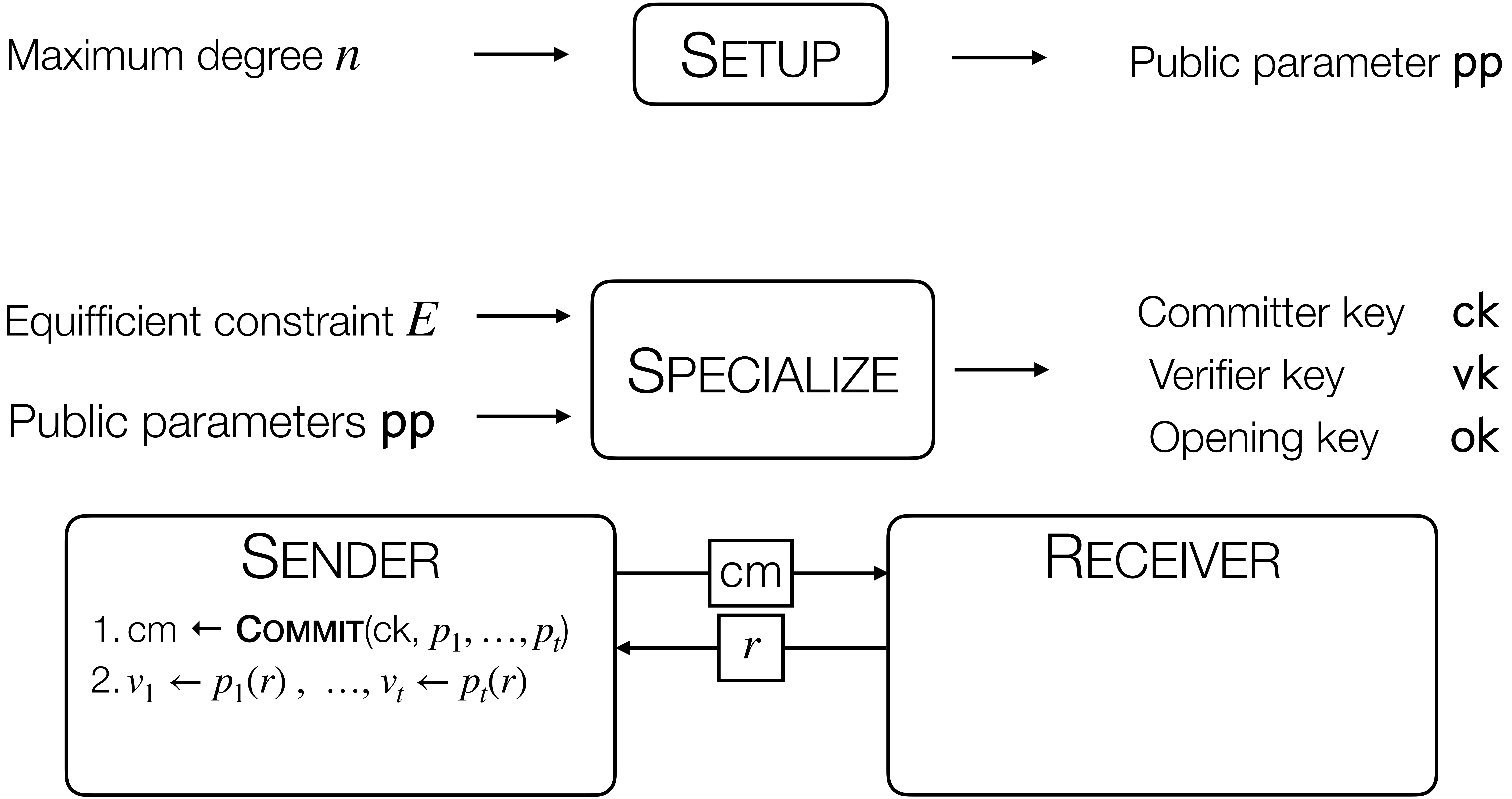
Equifficient PC (EPC) schemes



Equifficient PC (EPC) schemes



Equifficient PC (EPC) schemes



Equifficient PC (EPC) schemes

Maximum degree n → **SETUP** → Public parameter pp

Equifficient constraint E, Public parameters pp → **SPECIALIZE** → Committer key ck, Verifier key vk, Opening key ok

SENDER

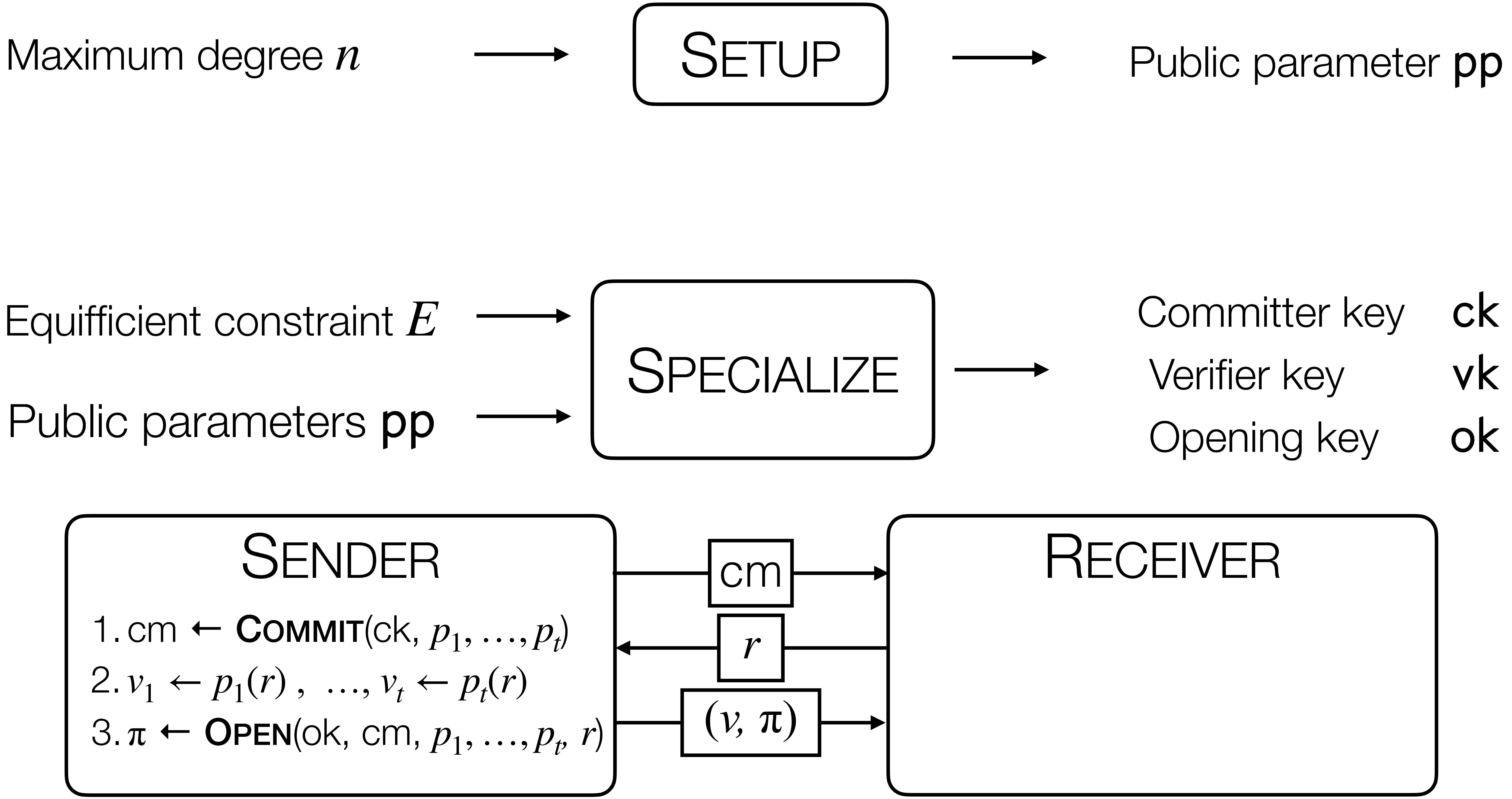
1. cm ← COMMIT(ck, $p_1, \ldots, p_t$)
2. $v_1 \leftarrow p_1(r)$, $\ldots$, $v_t \leftarrow p_t(r)$
3. π ← OPEN(ok, cm, $p_1, \ldots, p_t$, r)

SENDER → cm → RECEIVER

RECEIVER → r → SENDER

RECEIVER

Equifficient PC (EPC) schemes



Equifficient PC (EPC) schemes

Maximum degree n → **SETUP** → Public parameter pp

Equifficient constraint E → **SPECIALIZE**
Public parameters pp → **SPECIALIZE**

SPECIALIZE →
- Committer key ck
- Verifier key vk
- Opening key ok

SENDER

1. cm ← **COMMIT**(ck, $p_1, \ldots, p_t$)
2. $v_1 \leftarrow p_1(r)$, $\ldots$, $v_t \leftarrow p_t(r)$
3. π ← **OPEN**(ok, cm, $p_1, \ldots, p_t, r$)

SENDER → cm → RECEIVER

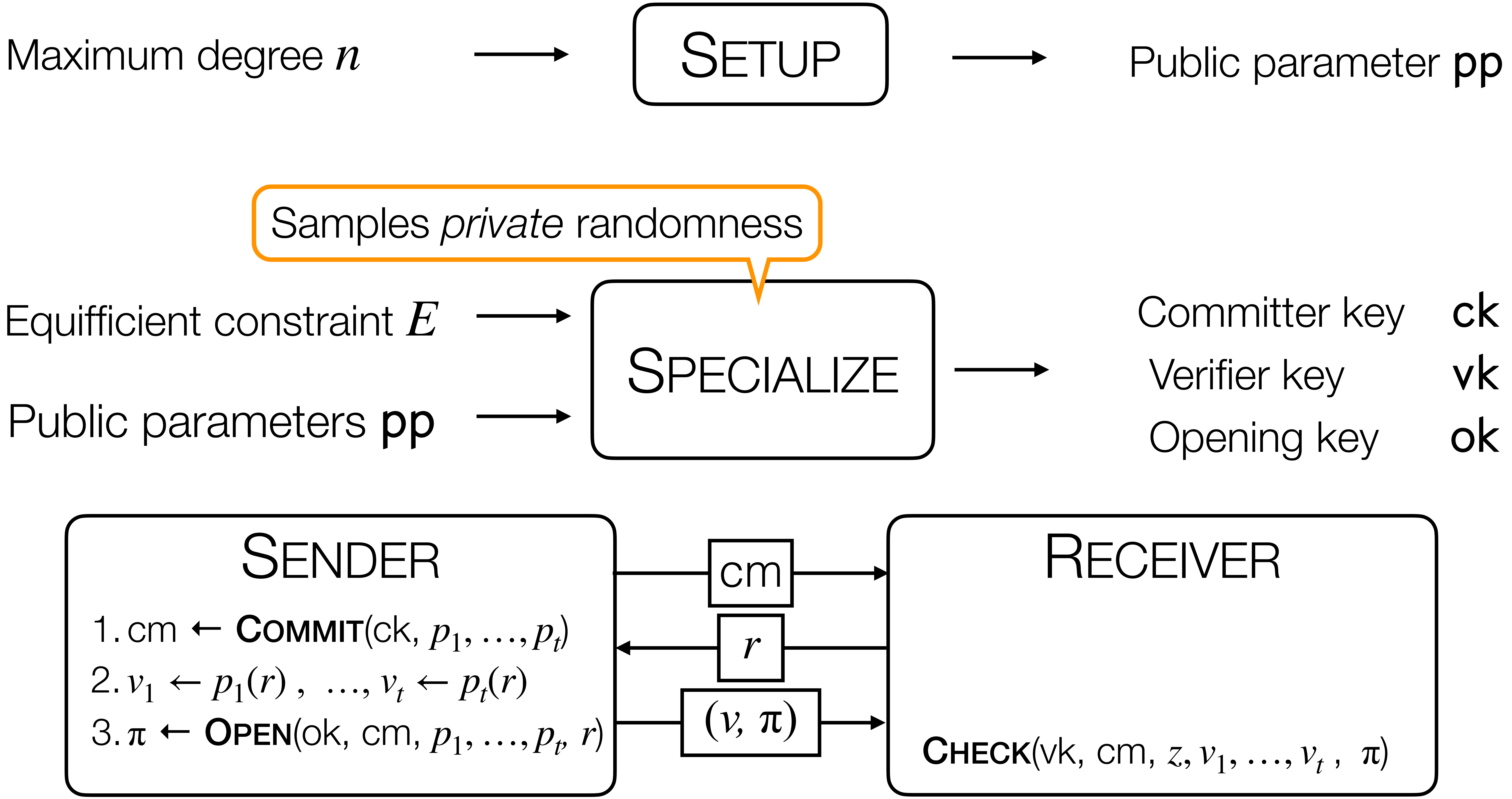
RECEIVER → r → SENDER

SENDER → (v, π) → RECEIVER

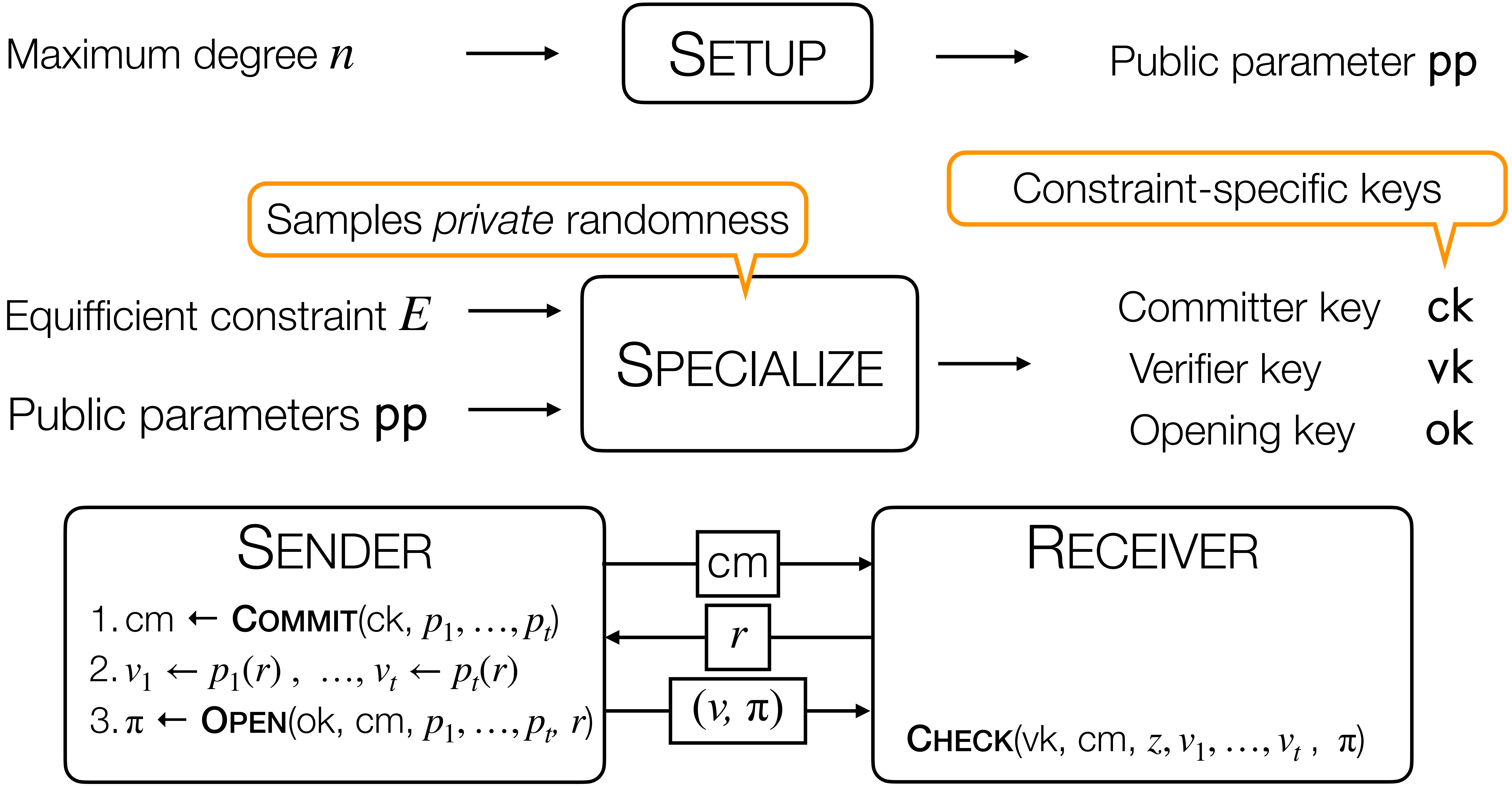
RECEIVER

CHECK(vk, cm, $z, v_1, \ldots, v_t$, π)

Equifficient PC (EPC) schemes



Equifficient PC (EPC) schemes



Properties of EPC schemes

Properties of EPC schemes

Completeness:

If the committed polynomials

- satisfy the evaluation claims ($p_1(z) = v_1, \ldots, p_n(z) = v_n$), and
- satisfy the equifficient constraints,

then the receiver accepts the evaluation proof

Properties of EPC schemes

Completeness:

If the committed polynomials

- satisfy the evaluation claims ($p_1(z) = v_1, \ldots, p_n(z) = v_n$), and
- satisfy the equifficient constraints,

then the receiver accepts the evaluation proof

Extractability

Properties of EPC schemes

Completeness:

If the committed polynomials

- satisfy the evaluation claims ($p_1(z) = v_1, \ldots, p_n(z) = v_n$), and
- satisfy the equifficient constraints,

then the receiver accepts the evaluation proof

Extractability

If adversary outputs a commitment & proof that convinces the receiver, then it must know $p_1, \ldots, p_n$ such that the following holds:

- **PC Extractability:** $p_1(z) = v_1, \ldots, p_n(z) = v_n$
- **Equifficient constraint satisfaction:** $p_1, \ldots, p_n$ are equifficient wrt E

KZG-based EPC Construction

KZG-based EPC Construction

Step 1: Using regular KZG, commit to the polynomials $\hat{z}_A$, $\hat{z}_B$, and $\hat{z}_C$

KZG-based EPC Construction

Step 1: Using regular KZG, commit to the polynomials $\hat{z}_A$, $\hat{z}_B$, and $\hat{z}_C$

$$\mathsf{ck} = (1 \cdot G,\ \tau \cdot G,\ \tau^2 \cdot G,\ \ldots,\ \tau^{n-1} \cdot G)$$

KZG-based EPC Construction

Step 1: Using regular KZG, commit to the polynomials $\hat{z}_A, \hat{z}_B$, and $\hat{z}_C$

$$\mathsf{ck} = (1 \cdot G,\ \tau \cdot G,\ \tau^2 \cdot G, \ldots,\ \tau^{n-1} \cdot G)$$

$$\mathsf{KZG.Commit}(\mathsf{ck}, \hat{z}_A) \to c_A$$
$$\mathsf{KZG.Commit}(\mathsf{ck}, \hat{z}_B) \to c_B$$
$$\mathsf{KZG.Commit}(\mathsf{ck}, \hat{z}_C) \to c_C$$

where $c_M := \sum_i z_M[i] \cdot \tau^i \cdot G$ for $M \in \{A, B, C\}$

KZG-based EPC Construction

KZG-based EPC Construction

Step 2: Enforce the coefficient-equality constraint.

KZG-based EPC Construction

Step 2: Enforce the coefficient-equality constraint.

To do this, first we construct committer keys that encode each basis…

$$E := \{\mathscr{A}, \mathscr{B}, \mathscr{C}\}$$

KZG-based EPC Construction

Step 2: Enforce the coefficient-equality constraint.

To do this, first we construct committer keys that encode each basis…

$$E := \{\mathscr{A}, \mathscr{B}, \mathscr{C}\}$$

$$\mathscr{A} = \{\hat{a}_1, \ldots, \hat{a}_n\} \quad \Longleftrightarrow \quad \mathsf{ck}_A = \left[\hat{a}_1(\tau)G,\ \ \hat{a}_2(\tau)G,\ \ \hat{a}_3(\tau)G,\ \ \ \ldots,\ \ \hat{a}_n(\tau)G\right]$$

$$\mathscr{B} = \{\hat{b}_1, \ldots, \hat{b}_n\} \quad \Longleftrightarrow \quad \mathsf{ck}_B = \left[\hat{b}_1(\tau)G,\ \ \hat{b}_2(\tau)G,\ \ \hat{b}_3(\tau)G,\ \ \ \ldots,\ \ \hat{b}_n(\tau)G\right]$$

$$\mathscr{C} = \{\hat{c}_1, \ldots, \hat{c}_n\} \quad \Longleftrightarrow \quad \mathsf{ck}_C = \left[\hat{c}_1(\tau)G,\ \ \hat{c}_2(\tau)G,\ \ \hat{c}_3(\tau)G,\ \ \ \ldots,\ \ \hat{c}_n(\tau)G\right]$$

KZG-based EPC Construction

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \begin{matrix} \alpha \cdot \mathsf{ck}_A + \\ \beta \cdot \mathsf{ck}_B + \\ \gamma \cdot \mathsf{ck}_C \end{matrix} = \begin{matrix} \alpha \cdot [\, \hat{a}_1(\tau)\,G, & \hat{a}_2(\tau)\,G, & \ldots & \hat{a}_n(\tau)\,G \,] + \\ \beta \cdot [\, \hat{b}_1(\tau)\,G, & \hat{b}_2(\tau)\,G, & \ldots & \hat{b}_n(\tau)\,G \,] + \\ \gamma \cdot [\, \hat{c}_1(\tau)\,G, & \hat{c}_2(\tau)\,G, & \ldots & \hat{c}_n(\tau)\,G \,] \end{matrix}$$

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \begin{matrix} \alpha \cdot \mathsf{ck}_A + \\ \beta \cdot \mathsf{ck}_B + \\ \gamma \cdot \mathsf{ck}_C \end{matrix} = \begin{matrix} \alpha \cdot \left[\, \hat{a}_1(\tau)\, G, \quad \hat{a}_2(\tau)\, G, \ \ldots \quad \hat{a}_n(\tau)\, G \,\right] + \\ \beta \cdot \left[\, \hat{b}_1(\tau)\, G, \quad \hat{b}_2(\tau)\, G, \ \ldots \quad \hat{b}_n(\tau)\, G \,\right] + \\ \gamma \cdot \left[\, \hat{c}_1(\tau)\, G, \quad \hat{c}_2(\tau)\, G, \ \ldots \quad \hat{c}_n(\tau)\, G \,\right] \end{matrix}$$

Now, commit to z, which recall is the equal-coefficient representation of $\hat{z}_A$, $\hat{z}_B$, and $\hat{z}_C$

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \begin{matrix} \alpha \cdot \mathsf{ck}_A + \\ \beta \cdot \mathsf{ck}_B + \\ \gamma \cdot \mathsf{ck}_C \end{matrix} = \begin{matrix} \alpha \cdot [\, \hat{a}_1(\tau)\, G, & \hat{a}_2(\tau)\, G, & \ldots & \hat{a}_n(\tau)\, G \,] + \\ \beta \cdot [\, \hat{b}_1(\tau)\, G, & \hat{b}_2(\tau)\, G, & \ldots & \hat{b}_n(\tau)\, G \,] + \\ \gamma \cdot [\, \hat{c}_1(\tau)\, G, & \hat{c}_2(\tau)\, G, & \ldots & \hat{c}_n(\tau)\, G \,] \end{matrix}$$

Now, commit to z, which recall is the equal-coefficient representation of $\hat{z}_A$, $\hat{z}_B$, and $\hat{z}_C$

$$c^* = \langle z, \mathsf{ck}^* \rangle = \begin{matrix} \alpha \cdot (\, z_1 \cdot \hat{a}_1(\tau) + z_2 \cdot \hat{a}_2(\tau) + \ldots + z_n \cdot \hat{a}_n(\tau)) \cdot G + \\ \beta \cdot (\, z_1 \cdot \hat{b}_1(\tau) + z_2 \cdot \hat{b}_2(\tau) + \ldots + z_n \cdot \hat{b}_n(\tau)) \cdot G + \\ \gamma \cdot (\, z_1 \cdot \hat{c}_1(\tau) + z_2 \cdot \hat{c}_2(\tau) + \ldots + z_n \cdot \hat{c}_n(\tau)) \cdot G \end{matrix}$$

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \begin{matrix} \alpha \cdot \mathsf{ck}_A + \\ \beta \cdot \mathsf{ck}_B + \\ \gamma \cdot \mathsf{ck}_C \end{matrix} = \begin{matrix} \alpha \cdot [\, \hat{a}_1(\tau)\, G, & \hat{a}_2(\tau)\, G, & \dots & \hat{a}_n(\tau)\, G \,] + \\ \beta \cdot [\, \hat{b}_1(\tau)\, G, & \hat{b}_2(\tau)\, G, & \dots & \hat{b}_n(\tau)\, G \,] + \\ \gamma \cdot [\, \hat{c}_1(\tau)\, G, & \hat{c}_2(\tau)\, G, & \dots & \hat{c}_n(\tau)\, G \,] \end{matrix}$$

Now, commit to z, which recall is the equal-coefficient representation of $\hat{z}_A$, $\hat{z}_B$, and $\hat{z}_C$

$$c^* = \langle z, \mathsf{ck}^* \rangle = \begin{matrix} \alpha \cdot (z_1 \cdot \hat{a}_1(\tau) + z_2 \cdot \hat{a}_2(\tau) + \dots + z_n \cdot \hat{a}_n(\tau)) \cdot G + \\ \beta \cdot (z_1 \cdot \hat{b}_1(\tau) + z_2 \cdot \hat{b}_2(\tau) + \dots + z_n \cdot \hat{b}_n(\tau)) \cdot G + \\ \gamma \cdot (z_1 \cdot \hat{c}_1(\tau) + z_2 \cdot \hat{c}_2(\tau) + \dots + z_n \cdot \hat{c}_n(\tau)) \cdot G \end{matrix}$$

KZG-based EPC Construction

KZG-based EPC Construction

$$\text{Commit}(\text{ck}, (z_A(X), z_B(X), z_C(X))) \to c_A, c_B, c_C , c^*$$

$$c_A = z_A(\tau) \cdot G, \quad c_B = z_B(\tau) \cdot G , c_C = z_C(\tau) \cdot G$$

Consistency Commitment

$$c^* = (\alpha \cdot \hat{z}_A(\tau) + \beta \cdot \hat{z}_B(\tau) + \gamma \cdot \hat{z}_C(\tau)) \cdot G$$

KZG-based EPC Construction

Equifficient

$$\text{Commit}(\text{ck}, (z_A(X), z_B(X), z_C(X))) \to c_A, c_B, c_C\ , c^*$$

$$c_A = z_A(\tau) \cdot G, \quad c_B = z_B(\tau) \cdot G\ , c_C = z_C(\tau) \cdot G$$

Consistency Commitment

$$c^* = (\ \alpha\ \cdot \hat{z}_A(\tau) + \beta\ \cdot \hat{z}_B(\tau) + \gamma \cdot \hat{z}_C(\tau)\ ) \cdot G$$

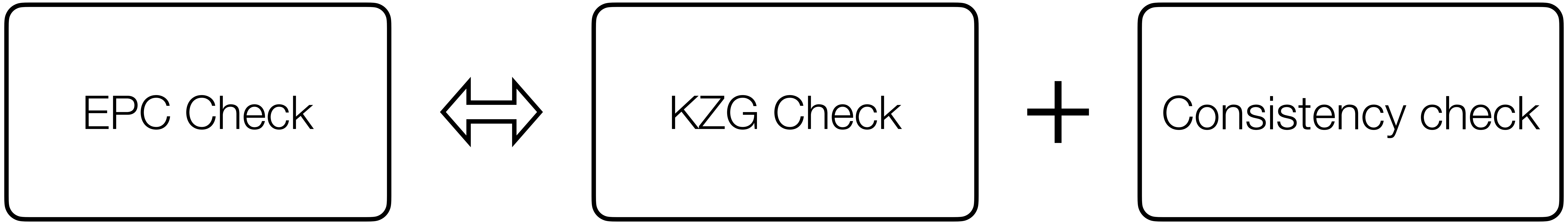
KZG-based EPC Construction

KZG-based EPC Construction

Step 4: Now, in the EPC check, do regular KZG verifications for each of c_A, c_B and c_C plus a consistency check using our new commitment $c*$

KZG-based EPC Construction

Step 4: Now, in the EPC check, do regular KZG verifications for each of c_A, c_B and c_C plus a consistency check using our new commitment c^*



KZG-based EPC Construction

Step 4: Now, in the EPC check, do regular KZG verifications for each of c_A, c_B and c_C plus a consistency check using our new commitment c^*

EPC Check $\Longleftrightarrow$ KZG Check $+$ Consistency check

KZG Check

Pass/fail ← **KZG.CHECK**(vk, c_A, v_A, π_A)

Pass/fail ← **KZG.CHECK**(vk, c_B, v_B, π_B)

Pass/fail ← **KZG.CHECK**(vk, c_C, v_C, π_C)

KZG-based EPC Construction

Step 4: Now, in the EPC check, do regular KZG verifications for each of c_A, c_B and c_C plus a consistency check using our new commitment $c*$

EPC Check $\Leftrightarrow$ KZG Check + Consistency check

KZG Check

Pass/fail ← **KZG.CHECK**(vk, c_A, v_A, π_A)

Pass/fail ← **KZG.CHECK**(vk, c_B, v_B, π_B)

Pass/fail ← **KZG.CHECK**(vk, c_C, v_C, π_C)

KZG-based EPC Construction

Step 4: Now, in the EPC check, do regular KZG verifications for each of c_A, c_B and c_C plus a consistency check using our new commitment $c*$

EPC Check $\Leftrightarrow$ KZG Check + Consistency check

KZG Check

Pass/fail ← **KZG.CHECK**(vk, c_A, v_A, π_A)

Pass/fail ← **KZG.CHECK**(vk, c_B, v_B, π_B)

Pass/fail ← **KZG.CHECK**(vk, c_C, v_C, π_C)

Consistency check

$$e(c^*, H) \stackrel{?}{=} \prod_{M \in \{A,B,C\}} e(c_M, \mathsf{vk}_M)$$

KZG-based EPC Construction

Step 4: Now, in the EPC check, do regular KZG verifications for each of c_A, c_B and c_C plus a consistency check using our new commitment c^*

EPC Check $\Leftrightarrow$ KZG Check $+$ Consistency check

KZG Check

Pass/fail $\leftarrow$ **KZG.CHECK**(vk, c_A, v_A, π_A)

Pass/fail $\leftarrow$ **KZG.CHECK**(vk, c_B, v_B, π_B)

Pass/fail $\leftarrow$ **KZG.CHECK**(vk, c_C, v_C, π_C)

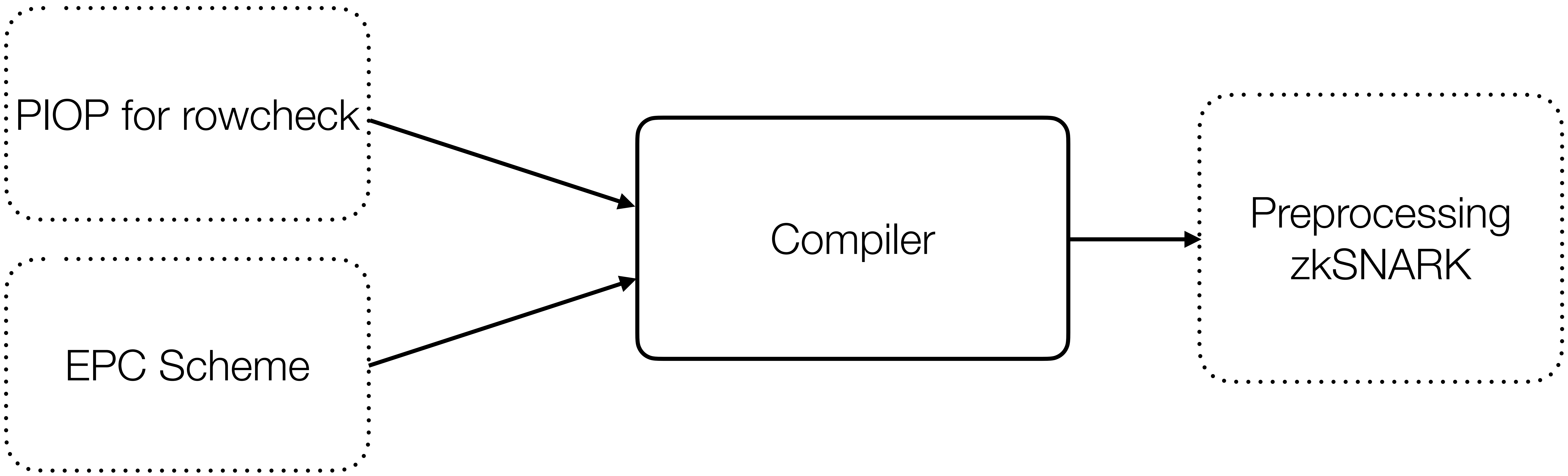
These proofs are computed by KZG.Open, which we omit for simplicity!

Consistency check

$$e(c^*, H) \stackrel{?}{=} \prod_{M \in \{A,B,C\}} e(c_M, \mathsf{vk}_M)$$

Our SNARK Construction

R1CS SNARKs from PIOPs + EPC Schemes



PIOPs + EPC Schemes → SNARK

SETUP($1^\lambda, A, B, C$)

n ⟵ PIOP(A, B, C)

pp ⟵ EPC.SETUP(n)

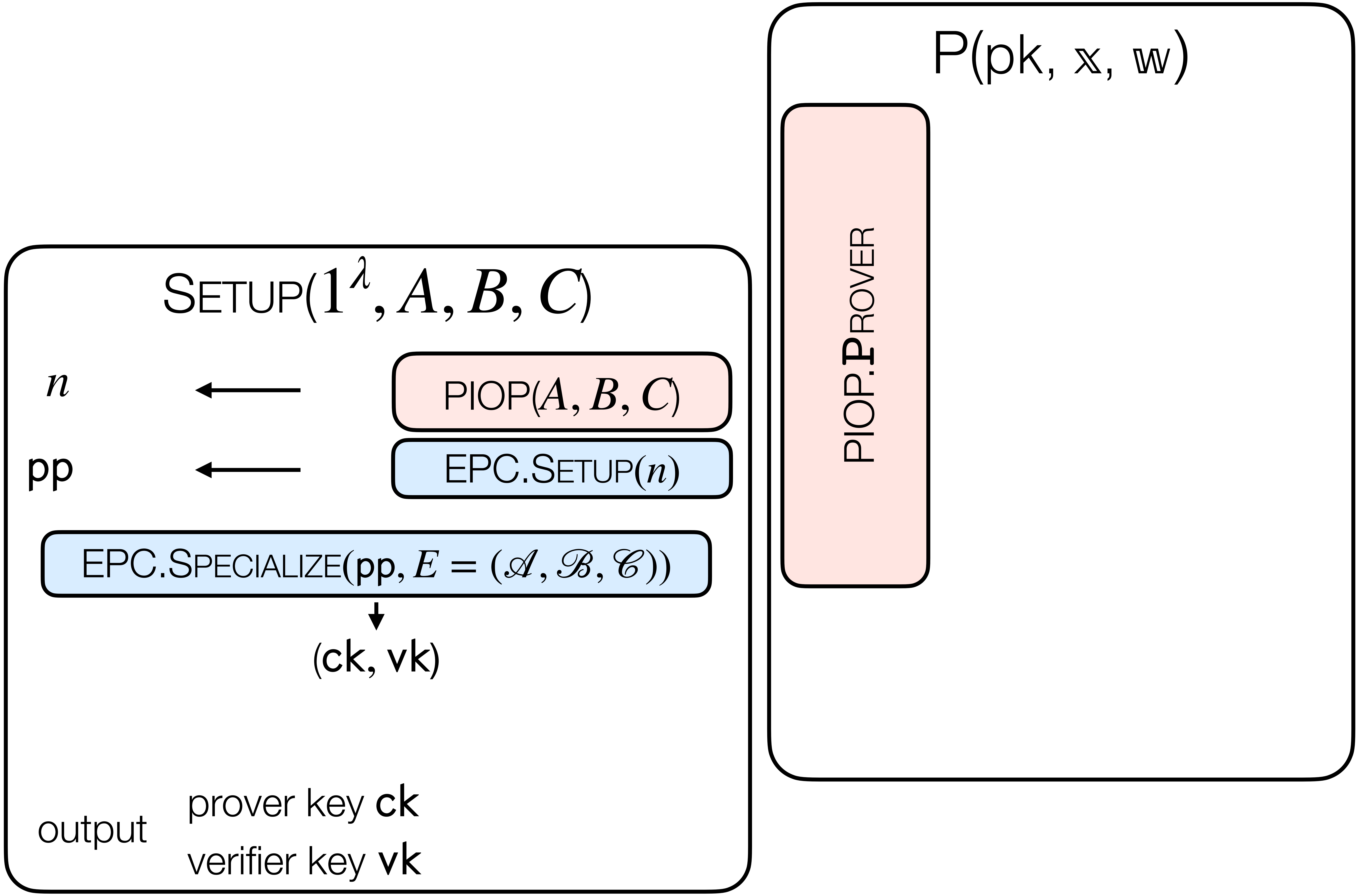
EPC.SPECIALIZE(pp, $E = (\mathscr{A}, \mathscr{B}, \mathscr{C})$)

↓

(ck, vk)

output
prover key ck
verifier key vk

PIOPs + EPC Schemes → SNARK



PIOPs + EPC Schemes → SNARK

PIOPs + EPC Schemes → SNARK

PIOPs + EPC Schemes → SNARK

PIOPs + EPC Schemes → SNARK

PIOPs + EPC Schemes → SNARK

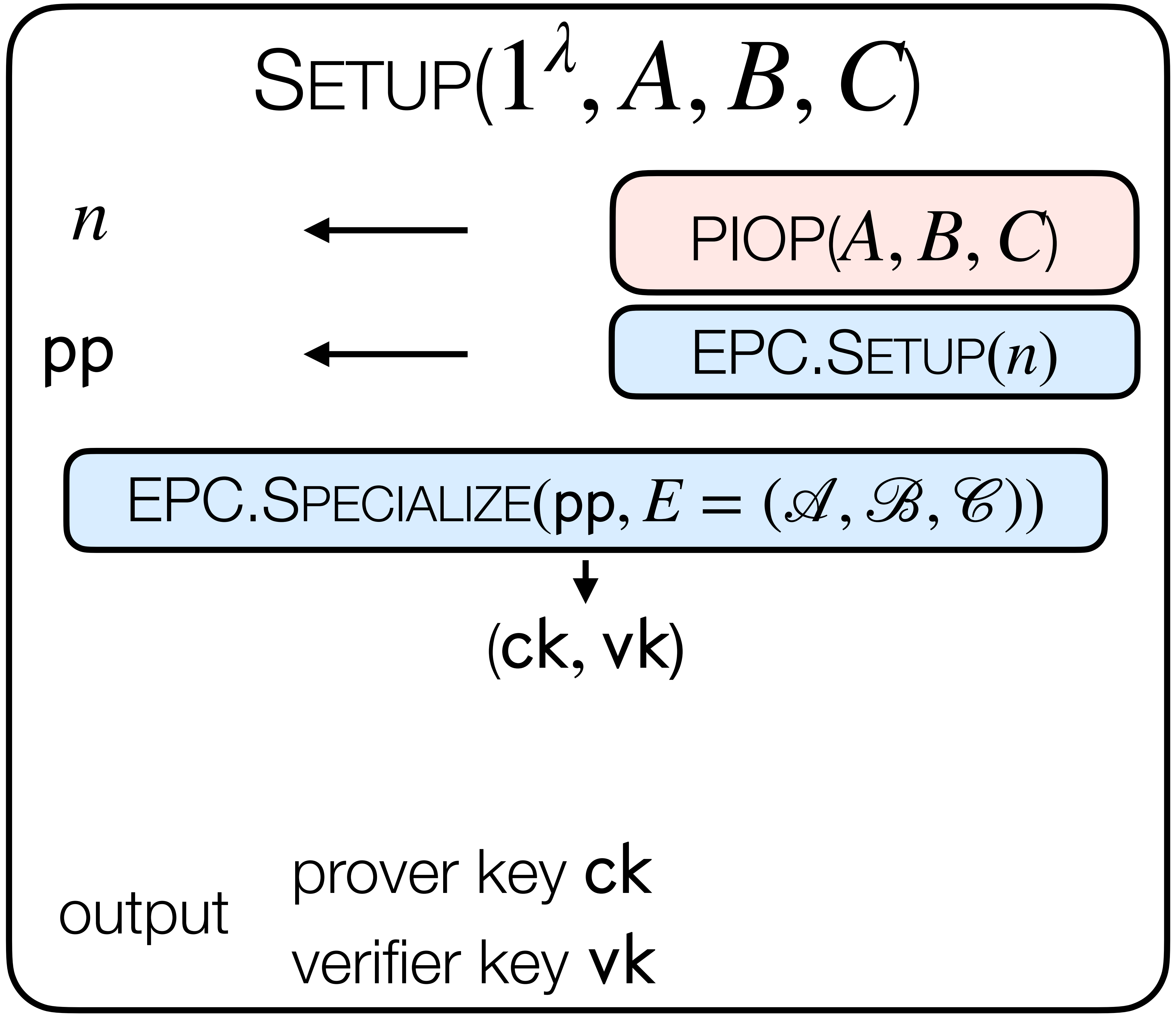
PIOPs + EPC Schemes → SNARK

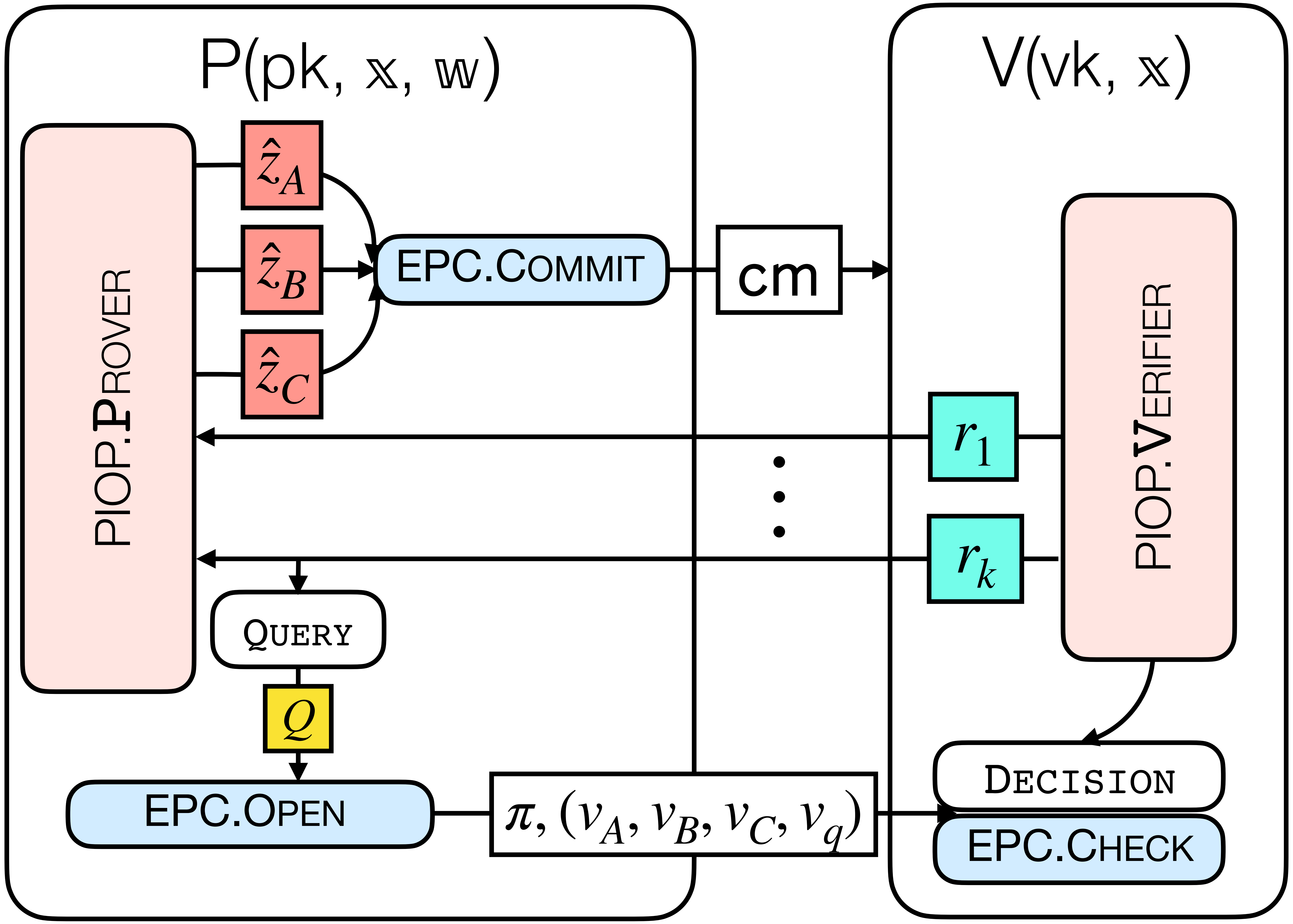
PIOPs + EPC Schemes → SNARK

PIOPs + EPC Schemes → SNARK

PIOPs + EPC Schemes → SNARK

PIOPs + EPC Schemes → SNARK





+ Fiat—Shamir to get non-interactivity

Properties of our construction

Properties of our construction

Completeness: follows from completeness of PIOP + EPC

Properties of our construction

Completeness: follows from completeness of PIOP + EPC

Proof of knowledge:
Whenever Arg.V accepts but R1CS is not satisfied, then we can construct an adversary that either breaks PIOP soundness or EPC extractability.

Additionally, we show that if
PIOP has round-by-round soundness → ARG has state-restoration PoK [BCS16]
Enables safe application of Fiat—Shamir transform in ROM!

Properties of our construction

Completeness: follows from completeness of PIOP + EPC

Proof of knowledge:
Whenever Arg.V accepts but R1CS is not satisfied, then we can construct an adversary that either breaks PIOP soundness or EPC extractability.

Additionally, we show that if
PIOP has round-by-round soundness → ARG has state-restoration PoK [BCS16]
Enables safe application of Fiat—Shamir transform in ROM!

Efficiency:

Properties of our construction

Completeness: follows from completeness of PIOP + EPC

Proof of knowledge:
Whenever Arg.V accepts but R1CS is not satisfied, then we can construct an adversary that either breaks PIOP soundness or EPC extractability.

Additionally, we show that if
PIOP has round-by-round soundness → ARG has state-restoration PoK [BCS16]
Enables safe application of Fiat—Shamir transform in ROM!

Efficiency:

- Proof size: # commitments + # evals + evaluation proof

Properties of our construction

Completeness: follows from completeness of PIOP + EPC

Proof of knowledge:
Whenever Arg.V accepts but R1CS is not satisfied, then we can construct an adversary that either breaks PIOP soundness or EPC extractability.

Additionally, we show that if
PIOP has round-by-round soundness → ARG has state-restoration PoK [BCS16]
Enables safe application of Fiat—Shamir transform in ROM!

Efficiency:

- Proof size: # commitments + # evals + evaluation proof
- Prover time: time for PIOP prover + time to EPC.Commit and EPC.Open

Properties of our construction

Completeness: follows from completeness of PIOP + EPC

Proof of knowledge:
Whenever Arg.V accepts but R1CS is not satisfied, then we can construct an adversary that either breaks PIOP soundness or EPC extractability.

Additionally, we show that if
PIOP has round-by-round soundness → ARG has state-restoration PoK [BCS16]
Enables safe application of Fiat—Shamir transform in ROM!

Efficiency:

- Proof size: # commitments + # evals + evaluation proof
- Prover time: time for PIOP prover + time to EPC.Commit and EPC.Open
- Verifier time: time for PIOP verifier + time for EPC.Check

Properties of our construction

Completeness: follows from completeness of PIOP + EPC

Proof of knowledge:
Whenever Arg.V accepts but R1CS is not satisfied, then we can construct an adversary that either breaks PIOP soundness or EPC extractability.

Additionally, we show that if
PIOP has round-by-round soundness → ARG has state-restoration PoK [BCS16]
Enables safe application of Fiat—Shamir transform in ROM!

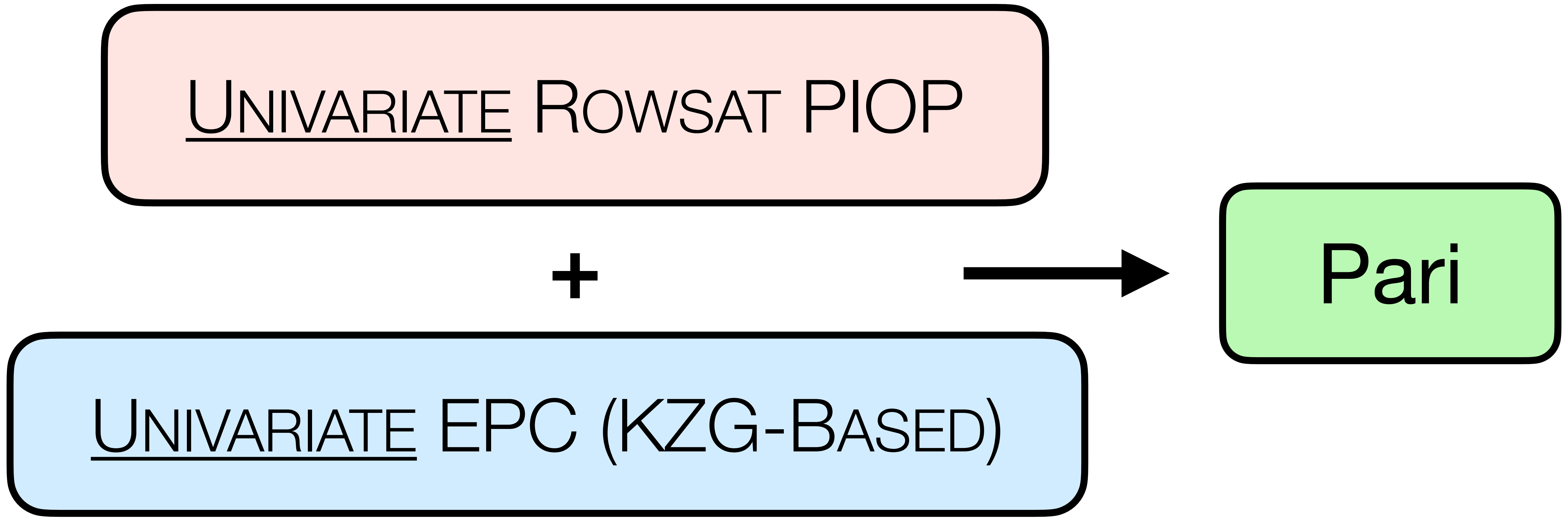
Efficiency:

- Proof size: # commitments + # evals + evaluation proof
- Prover time: time for PIOP prover + time to EPC.Commit and EPC.Open
- Verifier time: time for PIOP verifier + time for EPC.Check

Note: Our construction does not achieve Zero-knowledge; we leave this to future work

Instantiations: Garuda and Pari

Instantiations: Garuda and Pari



- SNARK for Square R1CS [GM17]
- Quasi-Linear Prover
- Verification needs 3 pairings
- Proof size 2 field + 2 group elements

Instantiations: Garuda and Pari

UNIVARIATE ROWSAT PIOP + UNIVARIATE EPC (KZG-BASED) → Pari

- SNARK for Square R1CS [GM17]
- Quasi-Linear Prover
- Verification needs 3 pairings
- Proof size 2 field + 2 group elements

MULTIVARIATE ROWSAT PIOP + MULTIVARIATE EPC (PST-BASED) → Garuda

- Generalize R1CS for custom gates
- *Linear-Time* Prover
- Logarithmic verifier and proof size
- Free addition gates

Instantiations: Garuda and Pari

Univariate RowSat PIOP
+
Univariate EPC (KZG-based)
→ Pari

- SNARK for Square R1CS [GM17]
- Quasi-Linear Prover
- Verification needs 3 pairings
- Proof size 2 field + 2 group elements

Multivariate RowSat PIOP
+
Multivariate EPC (PST-based)
→ Garuda

- Generalize R1CS for custom gates
- *Linear-Time* Prover
- Logarithmic verifier and proof size
- Free addition gates

Both require circuit-specific trusted setup =(

Implementation and Evaluation

Implementation in arkworks

GR1CS programming infrastructure, backward-compatible with R1CS

arkworks-rs/snark

Interfaces for Relations and SNARKs for these relations

Rust ☆ 844 234

arkworks-rs/r1cs-std

R1CS constraints for bits, fields, and elliptic curves

Rust ☆ 157 79

Garuda Implementation + Pari Implementation

Automatic Solidity Smart contract generator for Pari

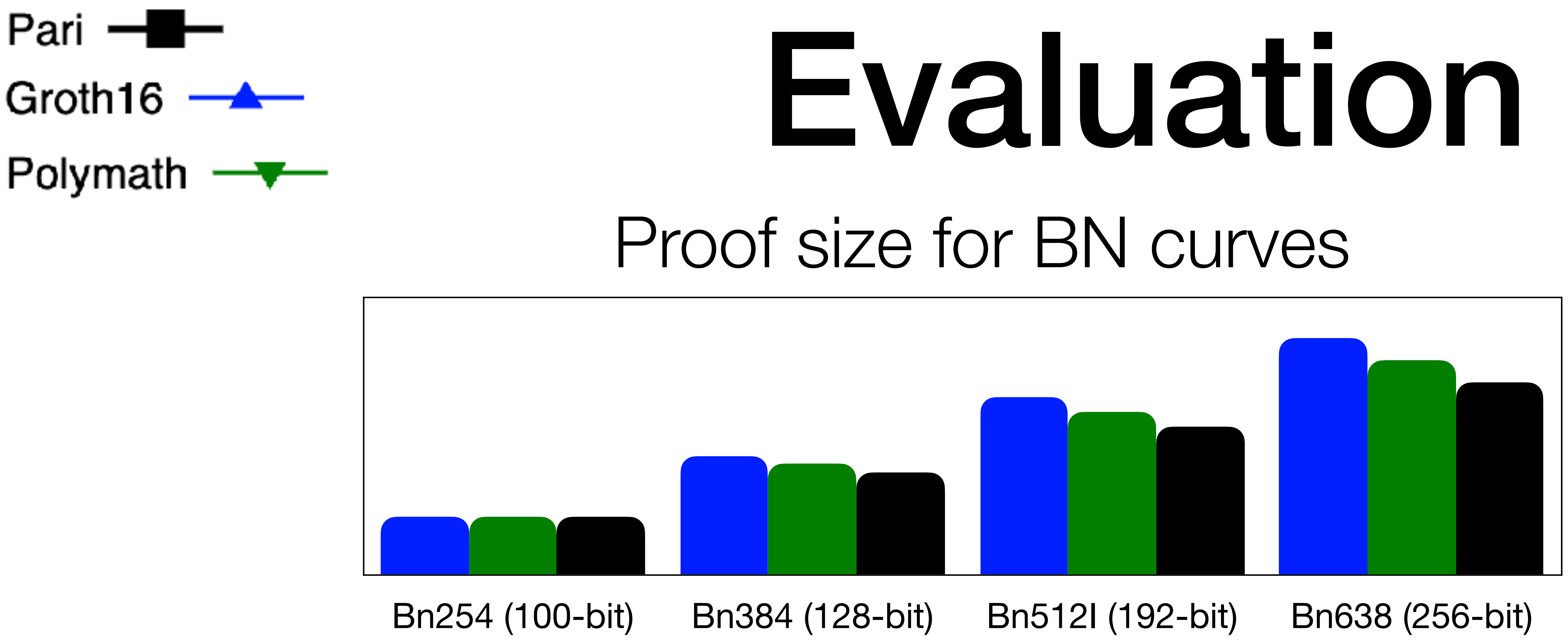
garuda-pari

A Rust implementation of the Garuda SNARK

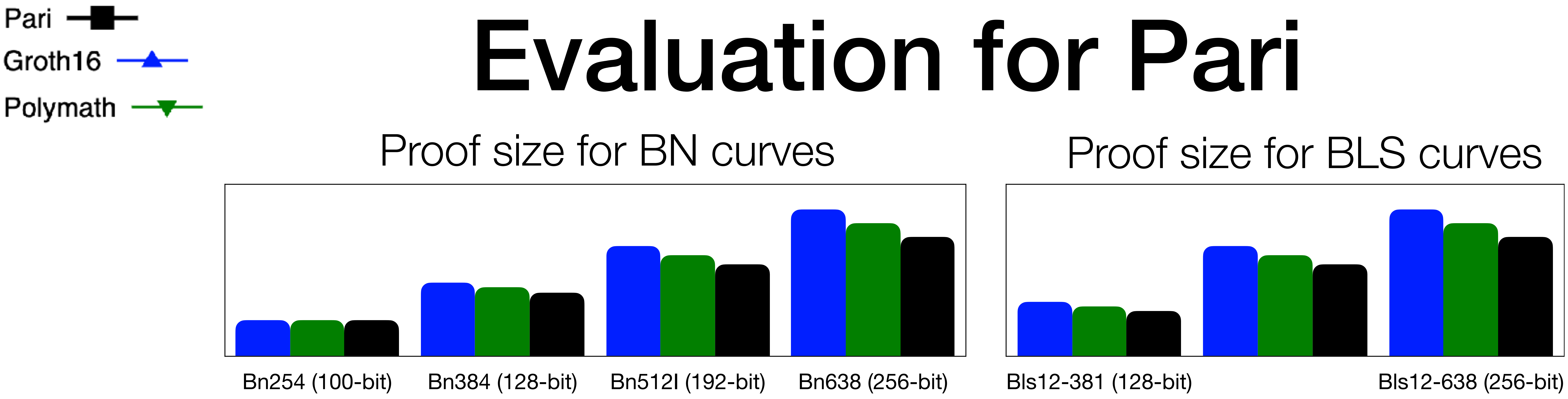
Rust

Evaluation for Pari

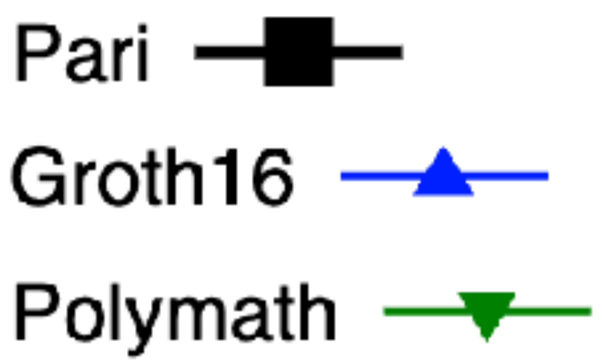
Evaluation for Pari



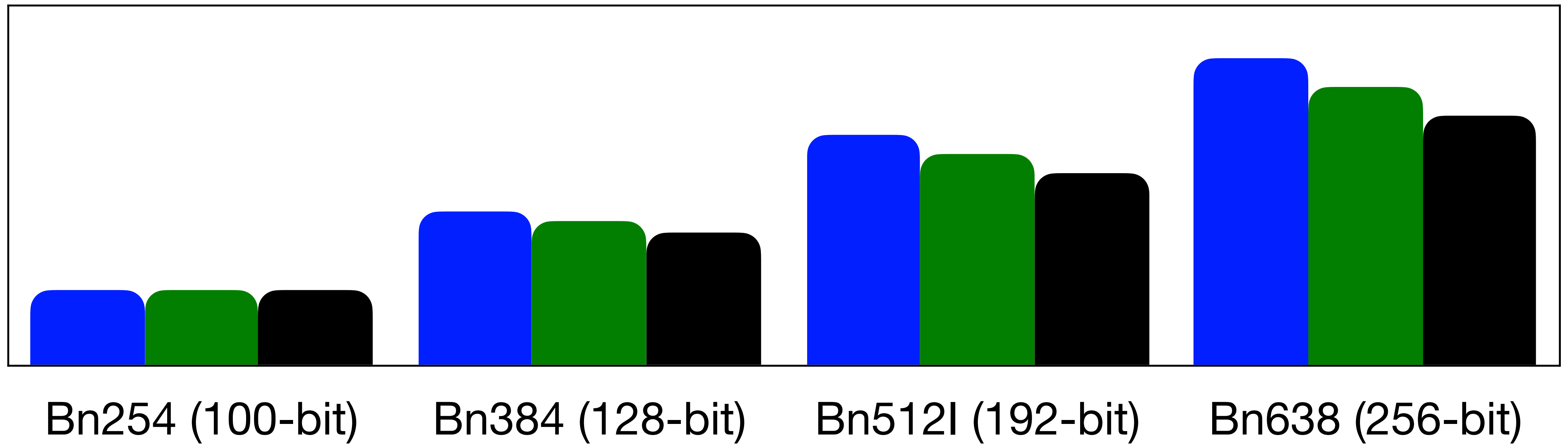
Evaluation for Pari



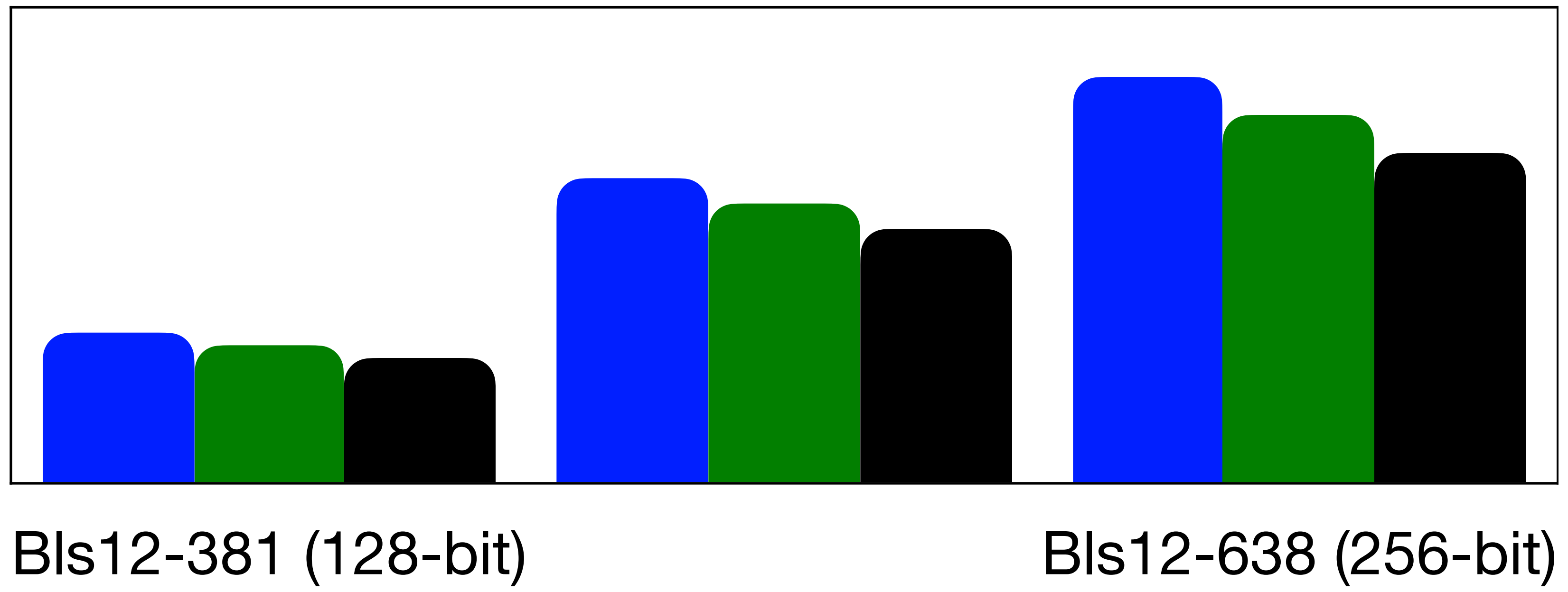
Evaluation for Pari



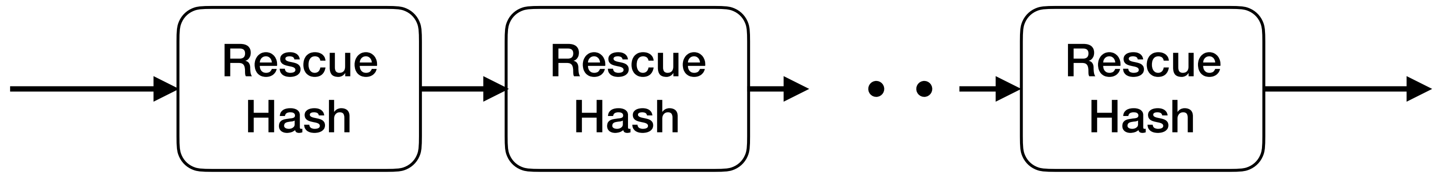
Proof size for BN curves

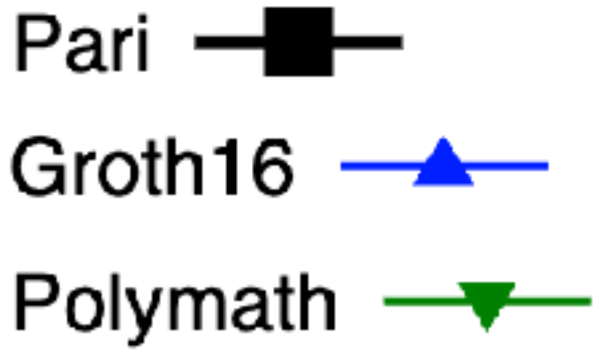


Proof size for BLS curves



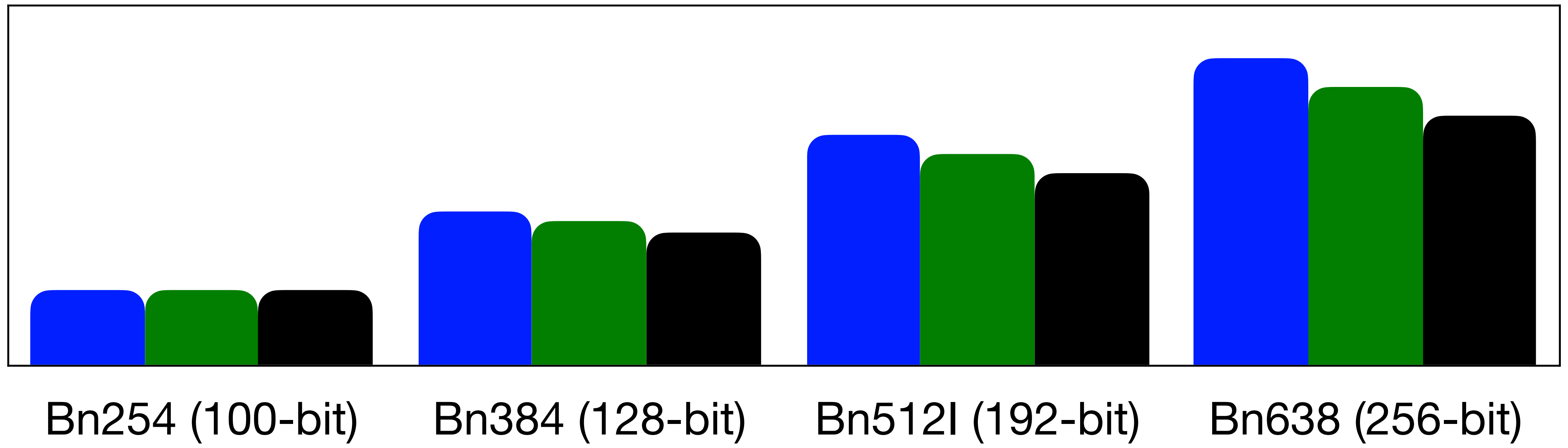
Benchmark results for a Hash-Chain circuit



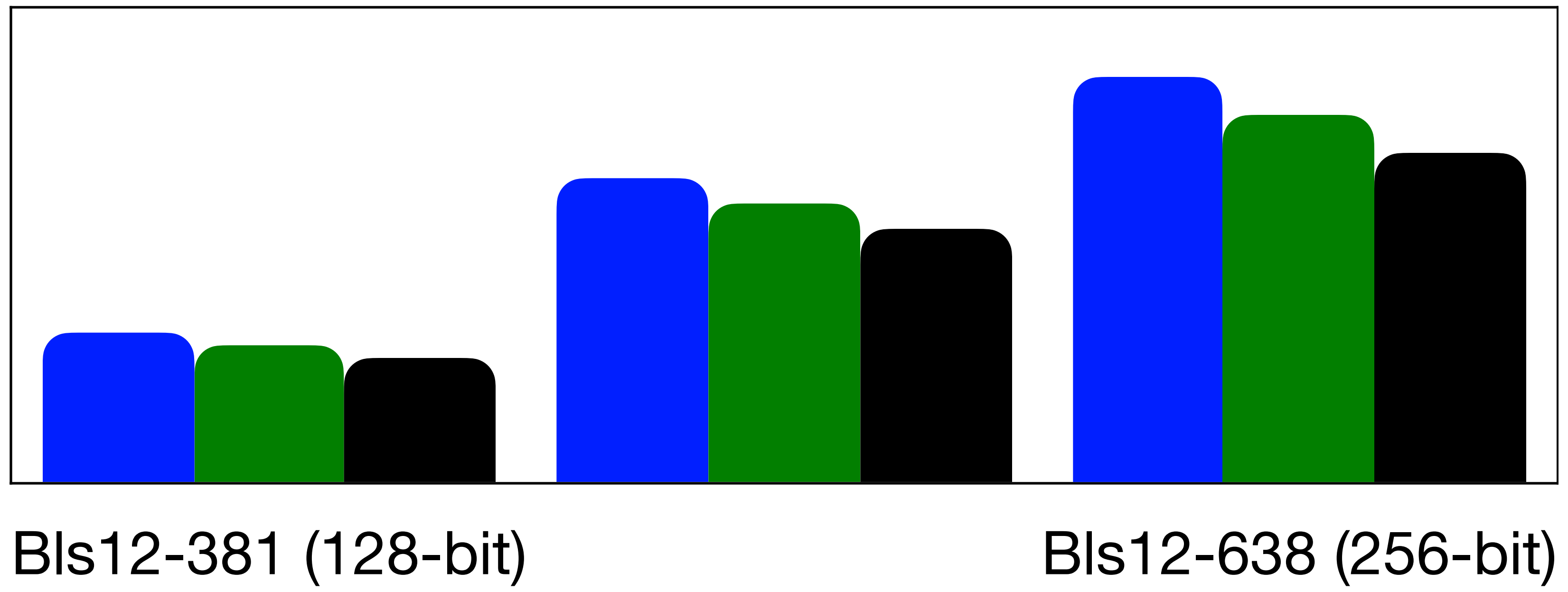


Evaluation for Pari

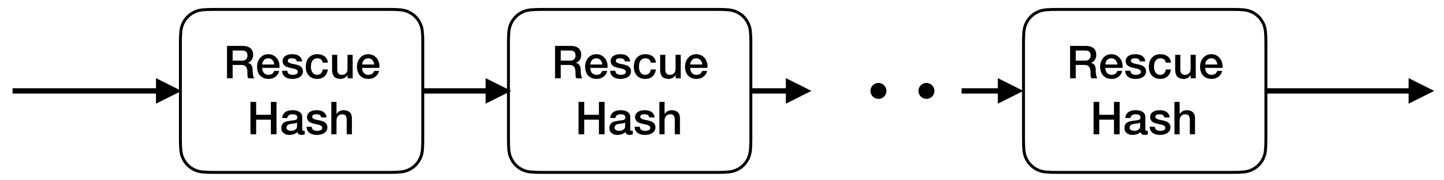
Proof size for BN curves

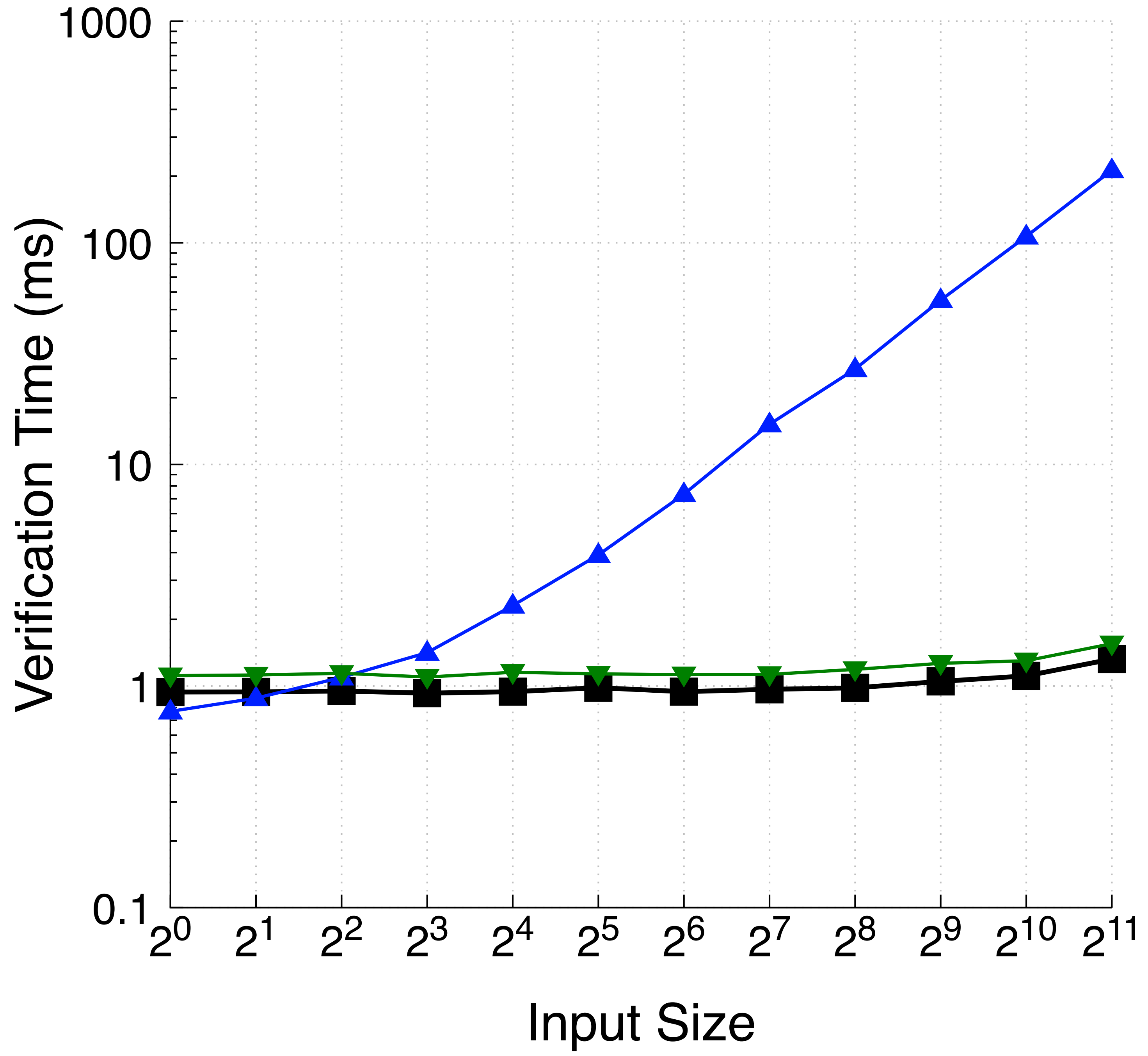


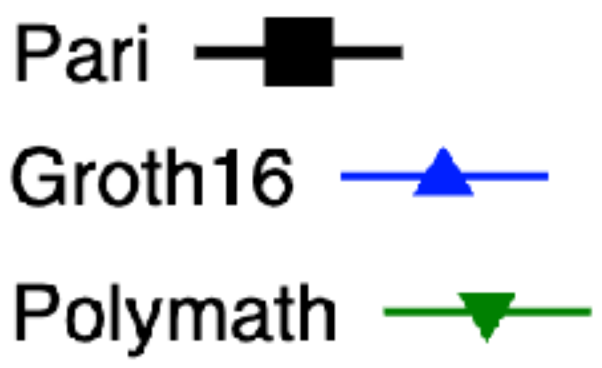
Proof size for BLS curves



Benchmark results for a Hash-Chain circuit

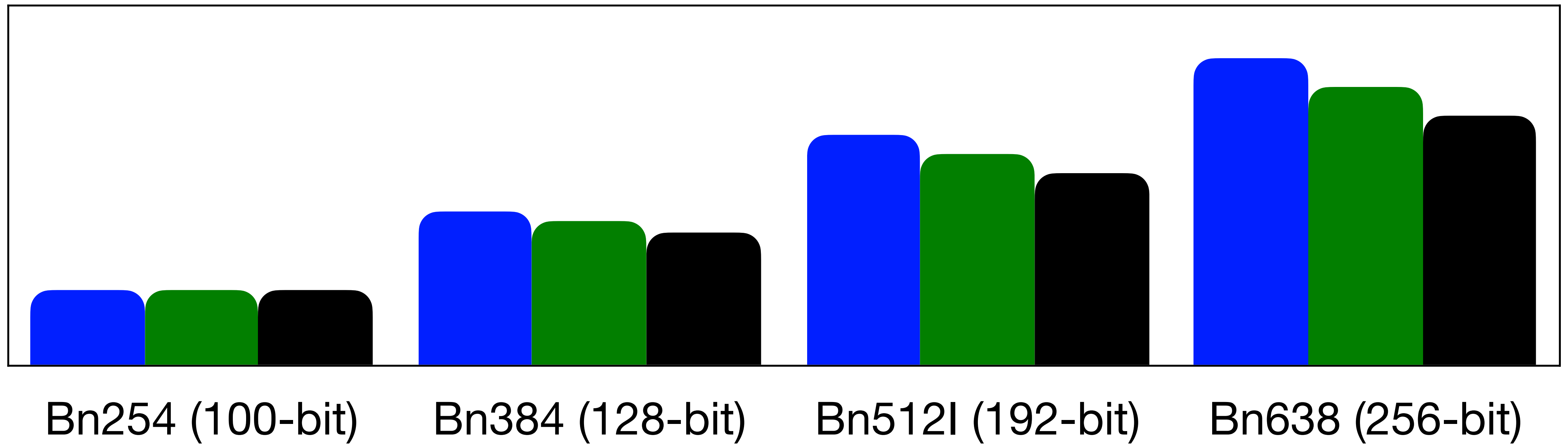




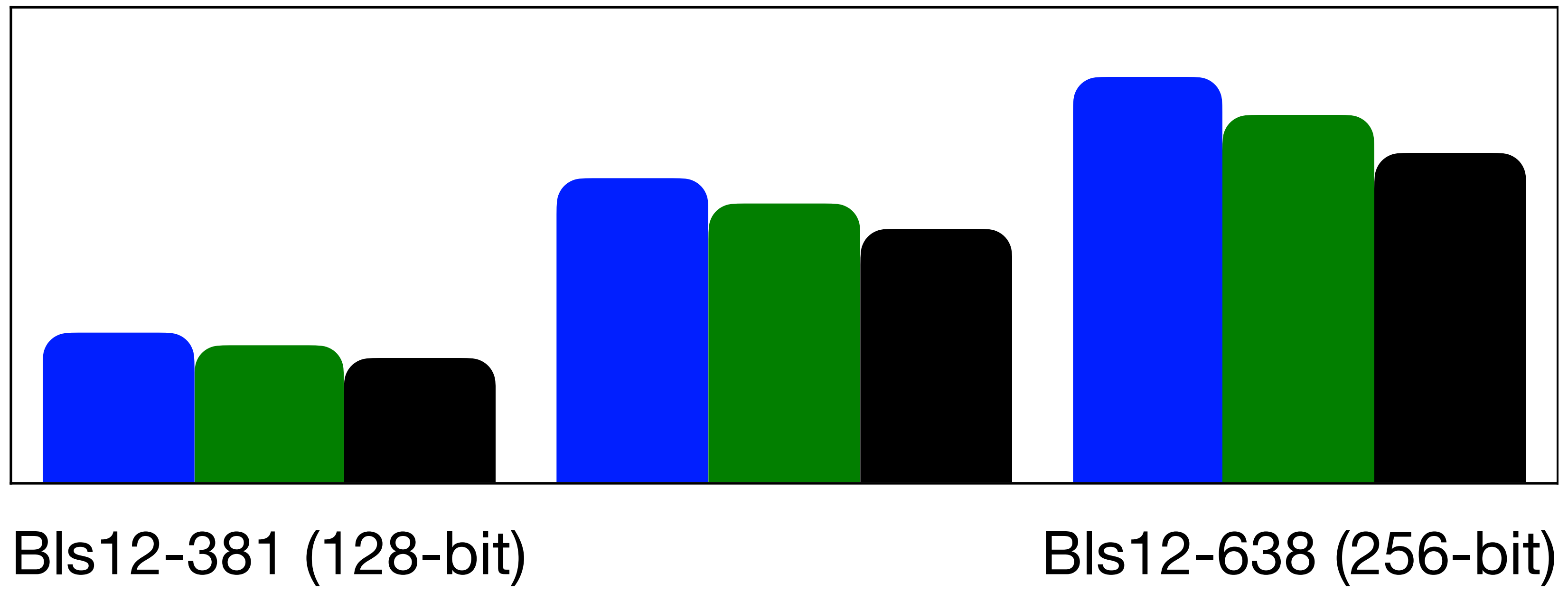


Evaluation for Pari

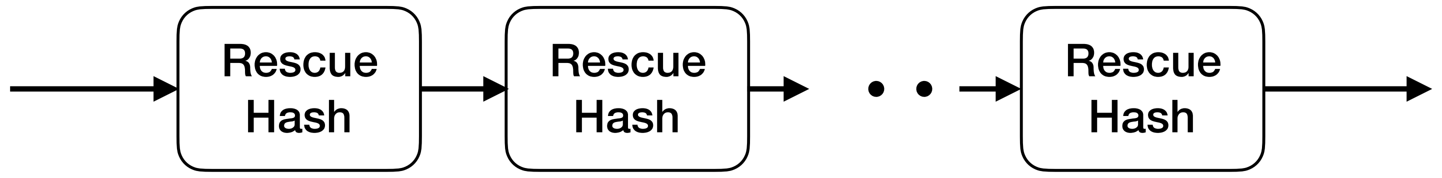
Proof size for BN curves

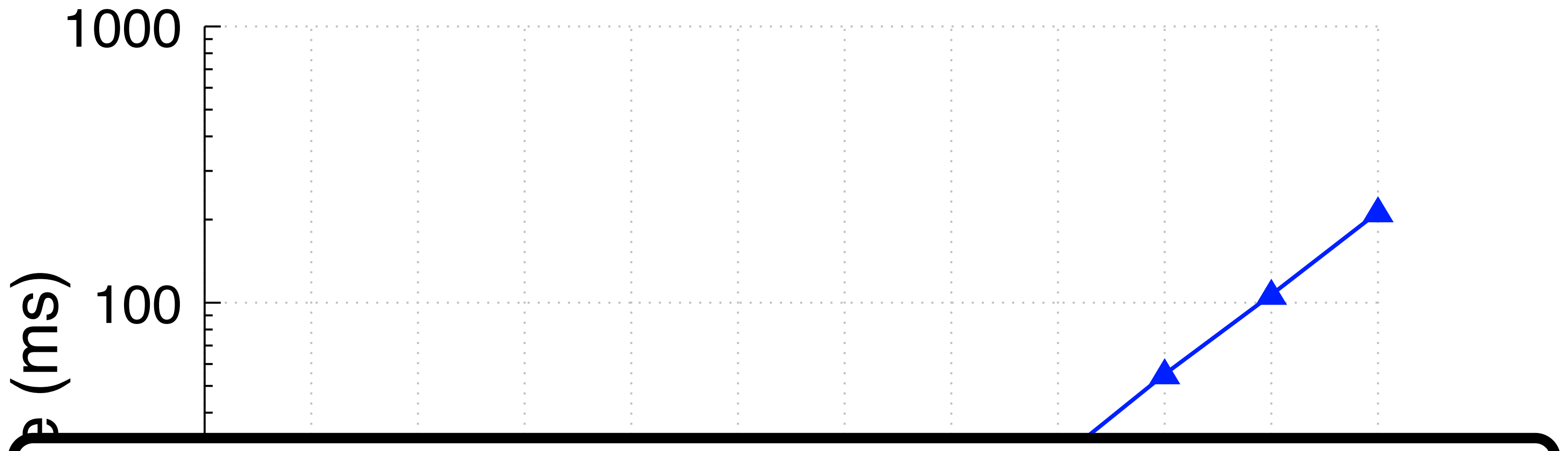


Proof size for BLS curves



Benchmark results for a Hash-Chain circuit

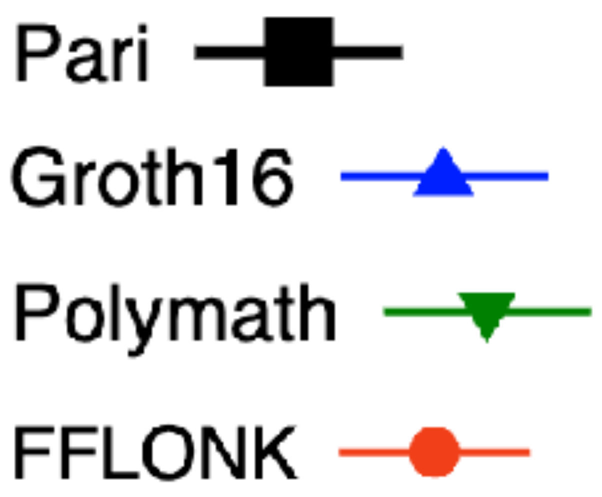




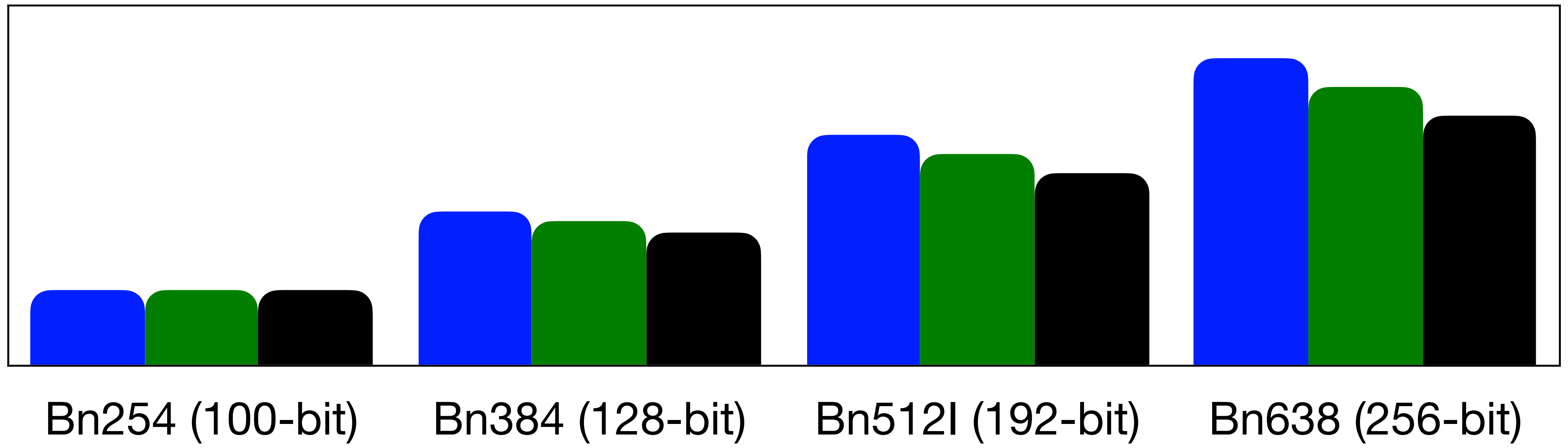
Comparison with Groth16:
No verifier MSM
0.2 ms worse for #io=1

Comparison with Polymath:
~ 15% faster verifier

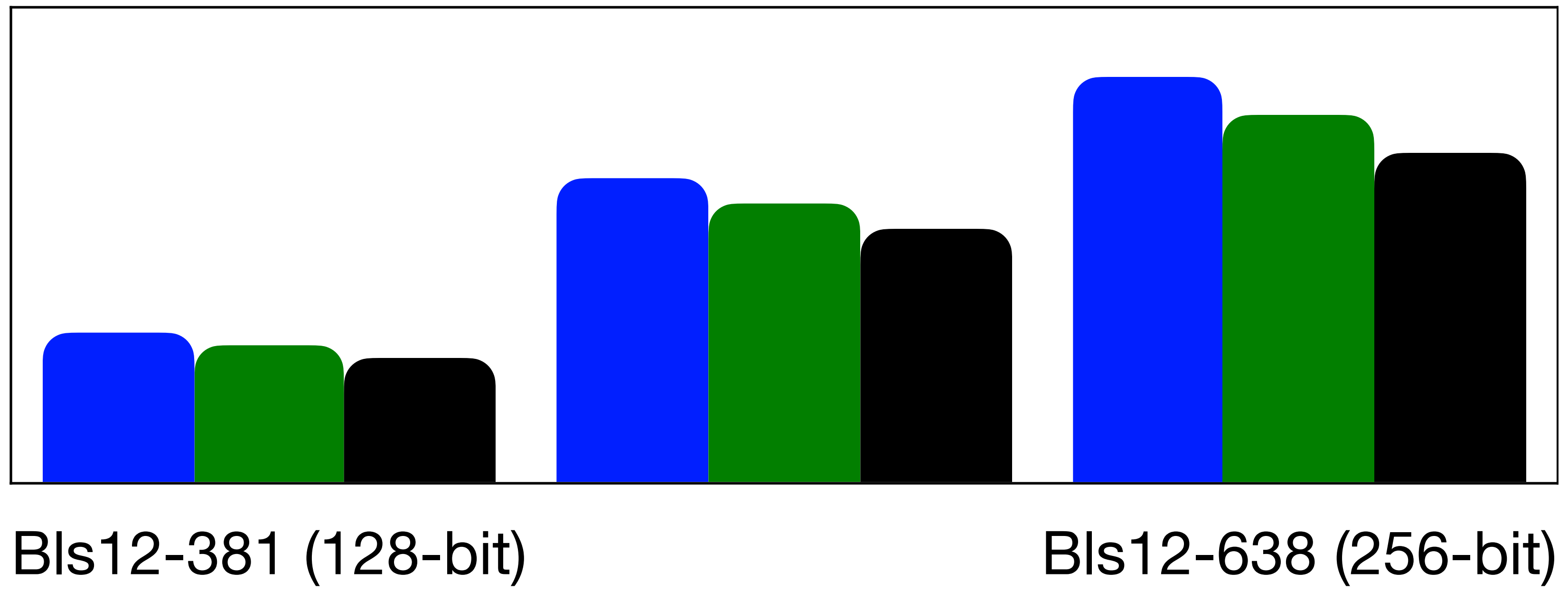
Evaluation for Pari



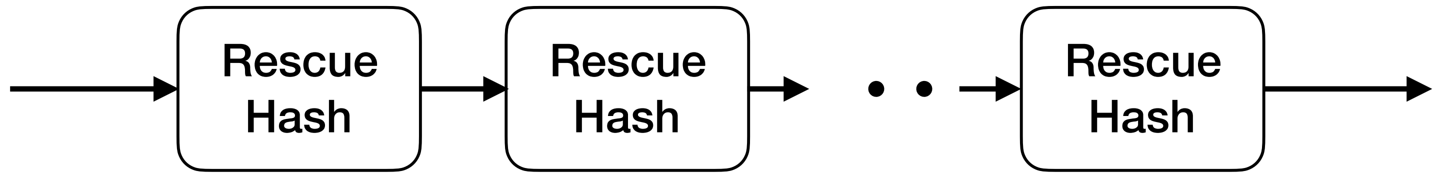
Proof size for BN curves



Proof size for BLS curves

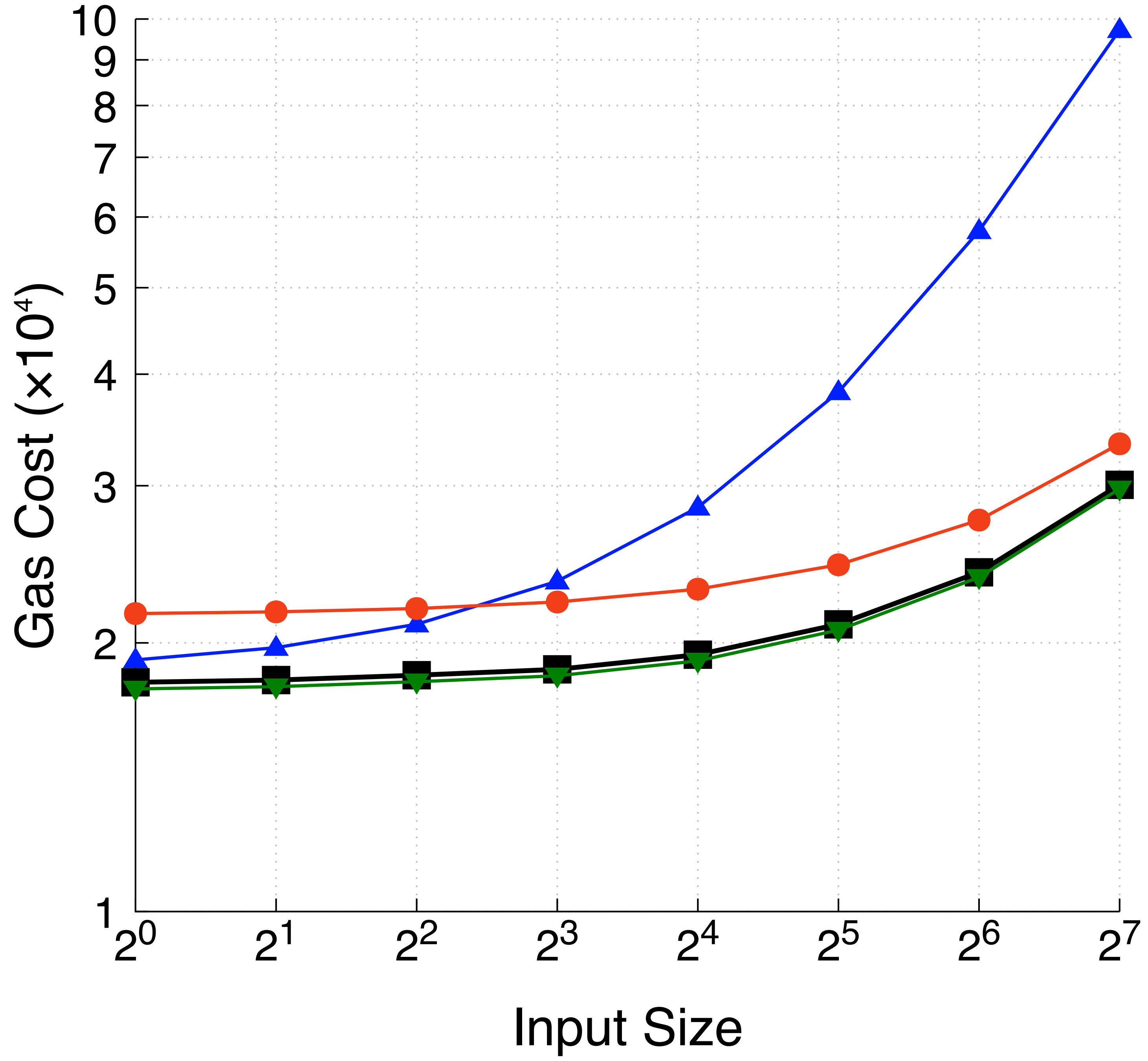


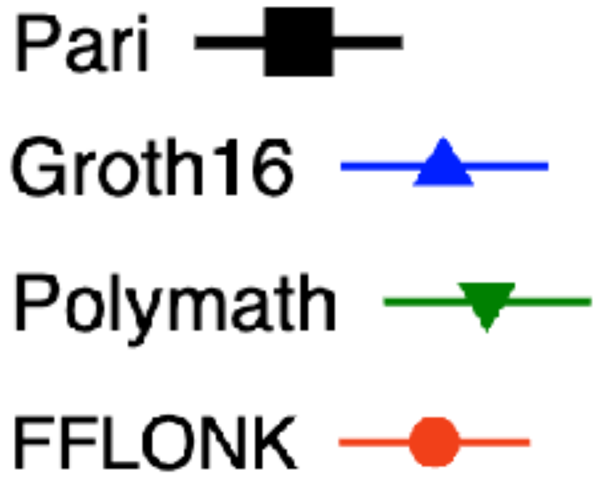
Benchmark results for a Hash-Chain circuit



Comparison with Groth16:
No verifier MSM
0.2 ms worse for #io=1

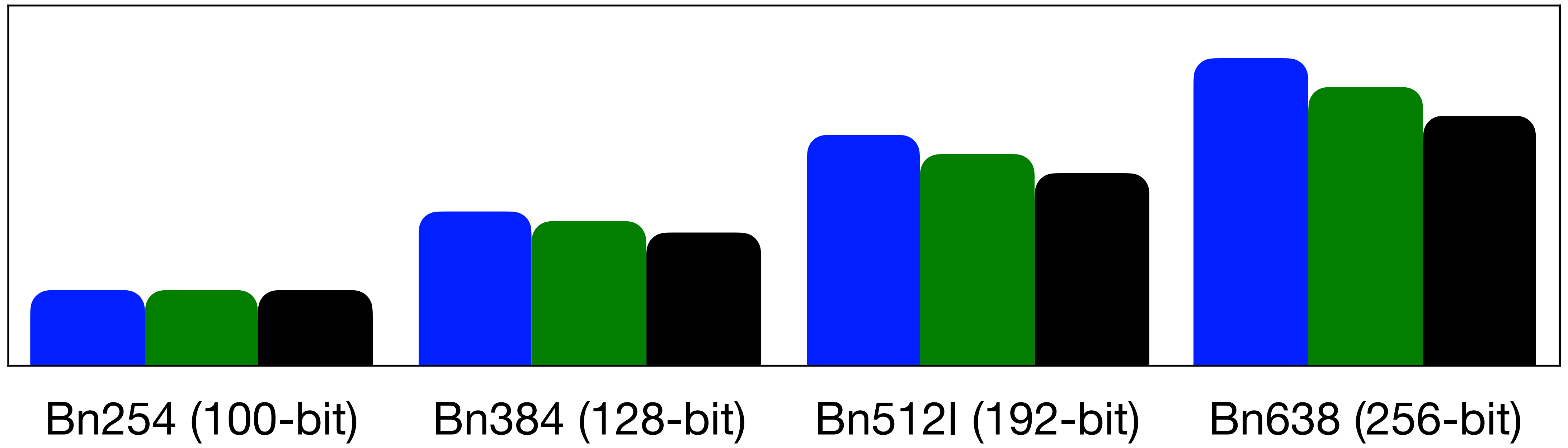
Comparison with Polymath:
~ 15% faster verifier



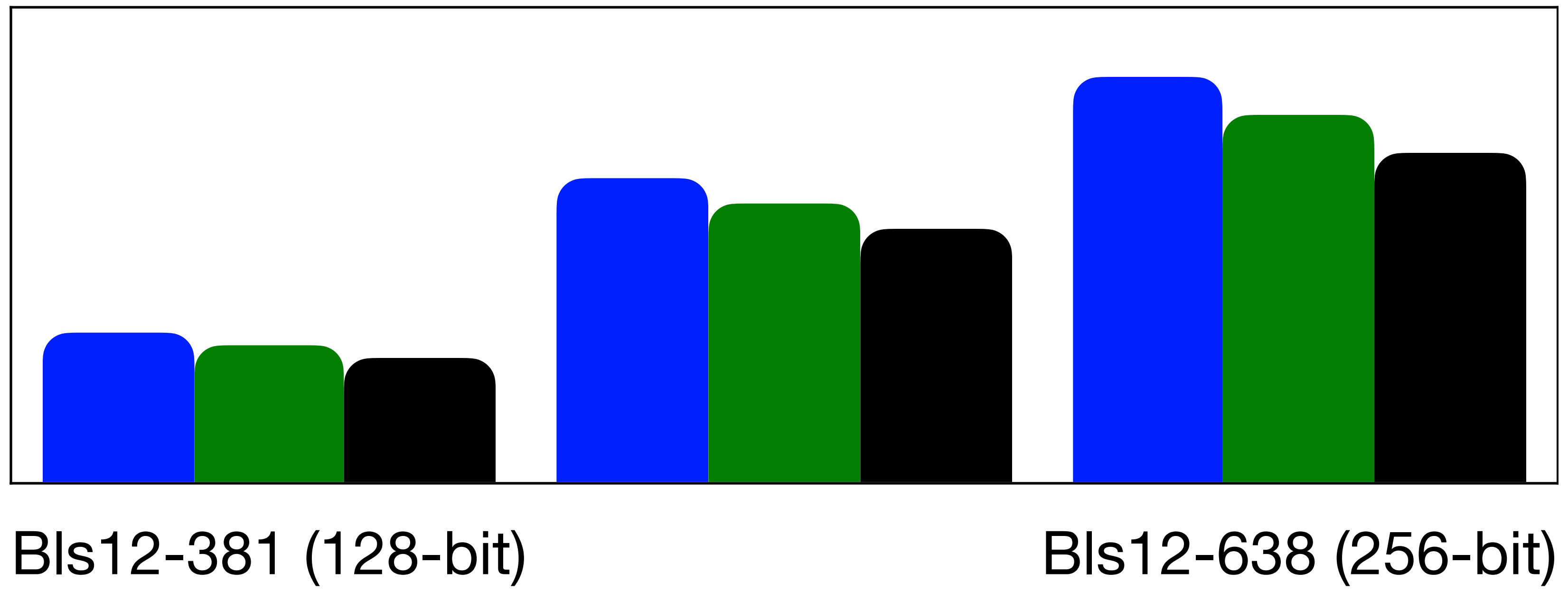


Evaluation for Pari

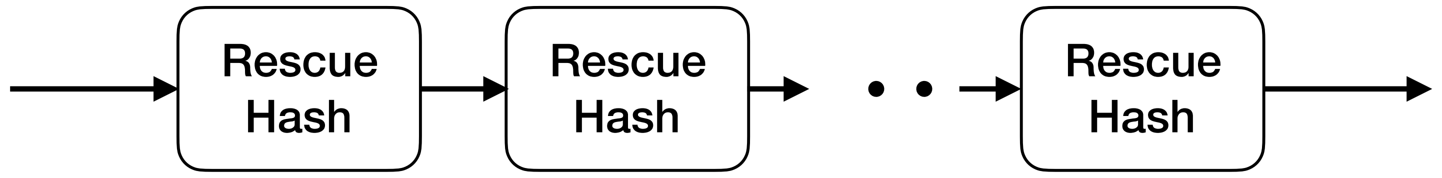
Proof size for BN curves



Proof size for BLS curves

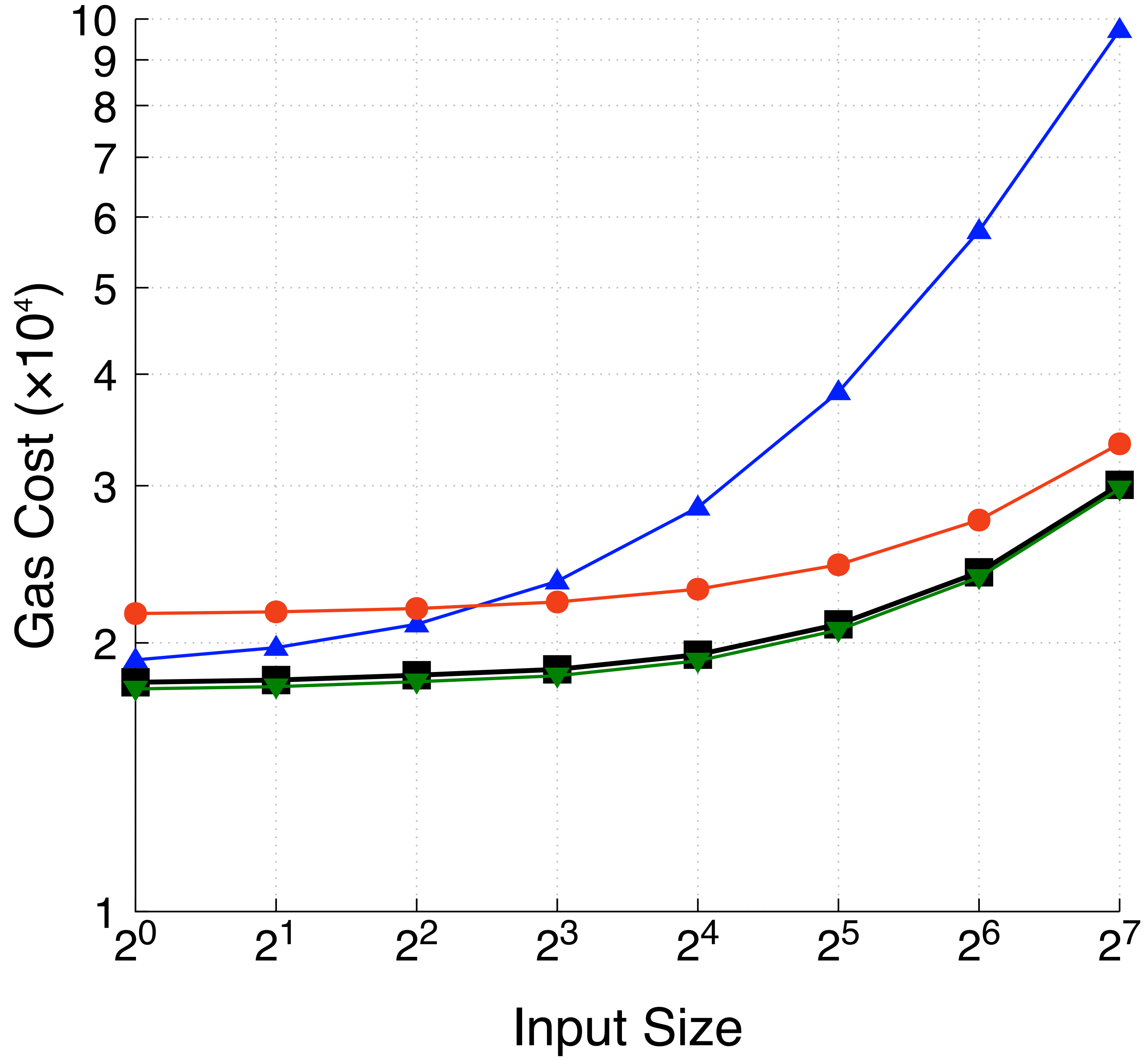


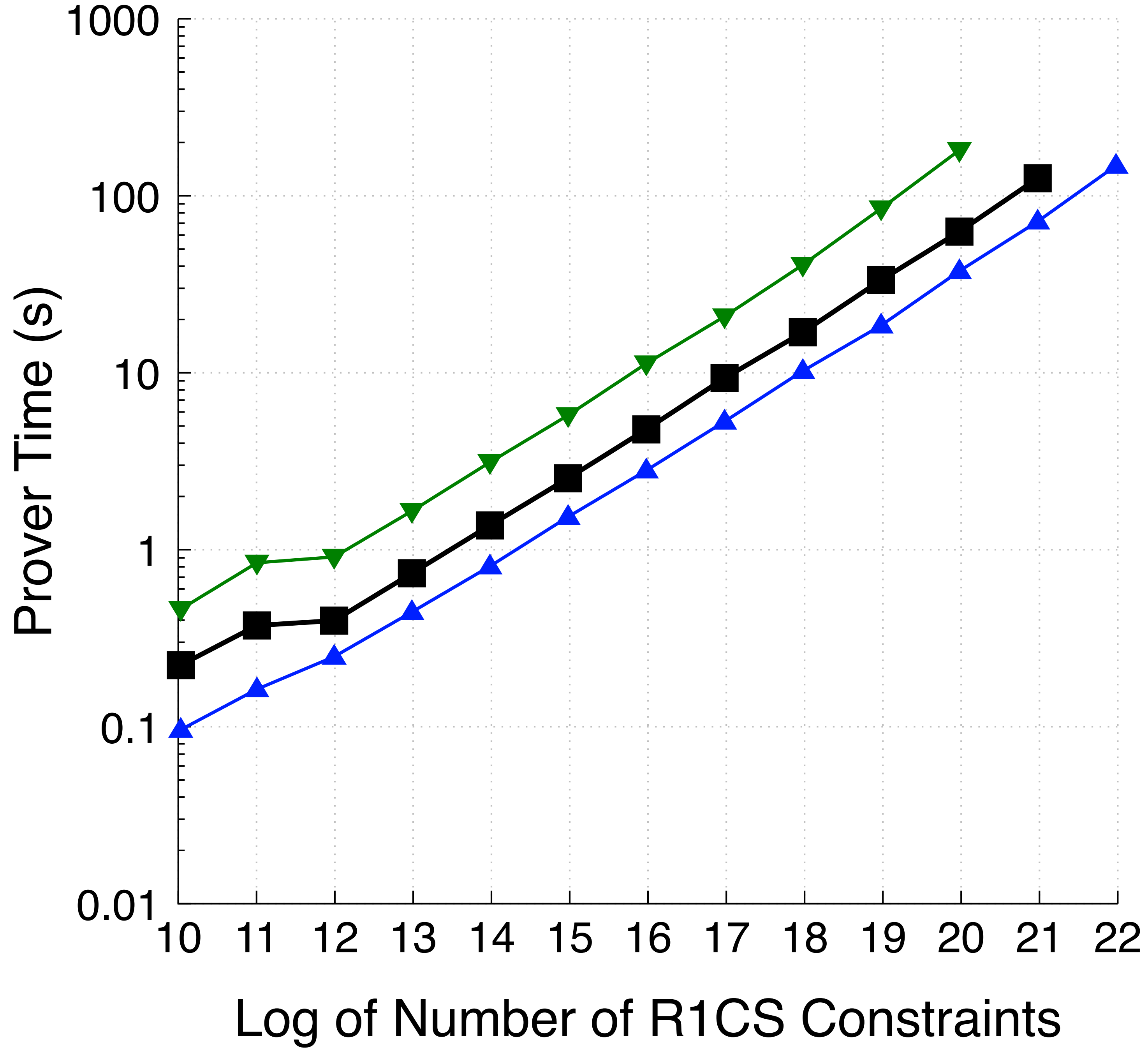
Benchmark results for a Hash-Chain circuit

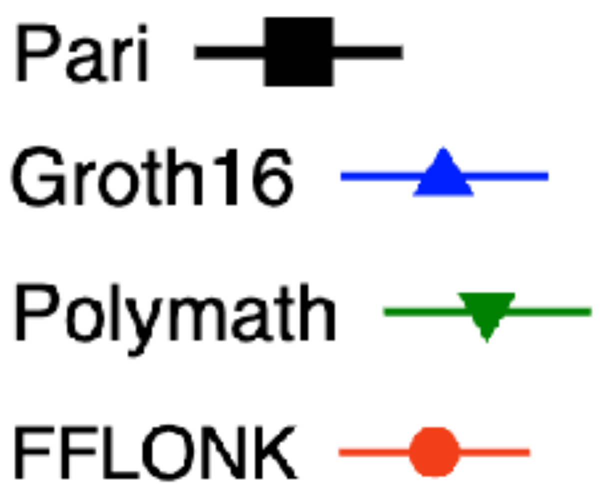


Comparison with Groth16:
No verifier MSM
0.2 ms worse for #io=1

Comparison with Polymath:
~ 15% faster verifier

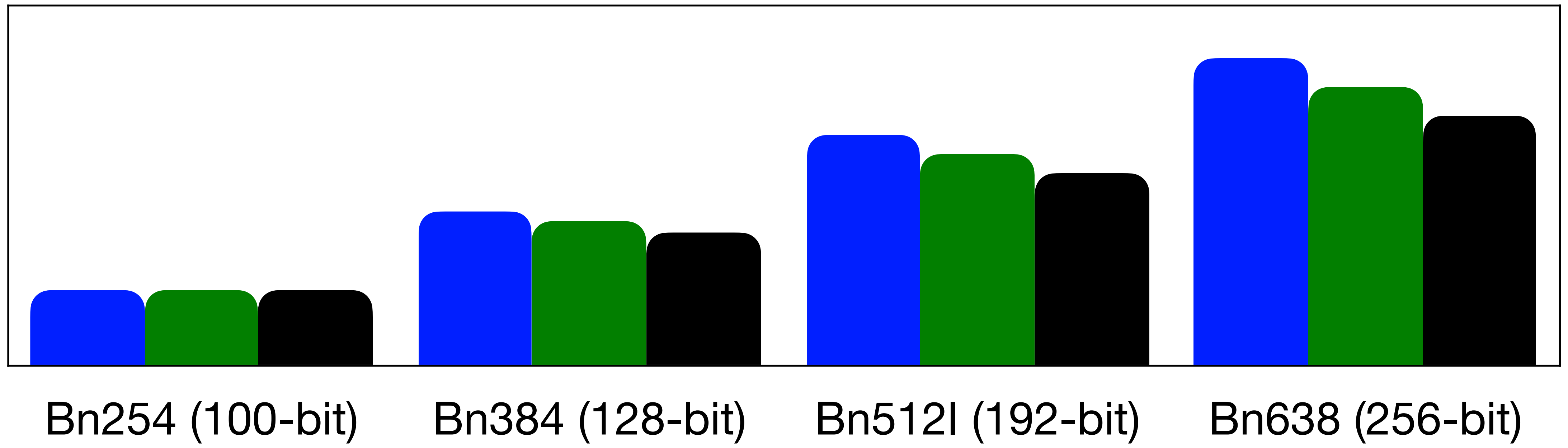




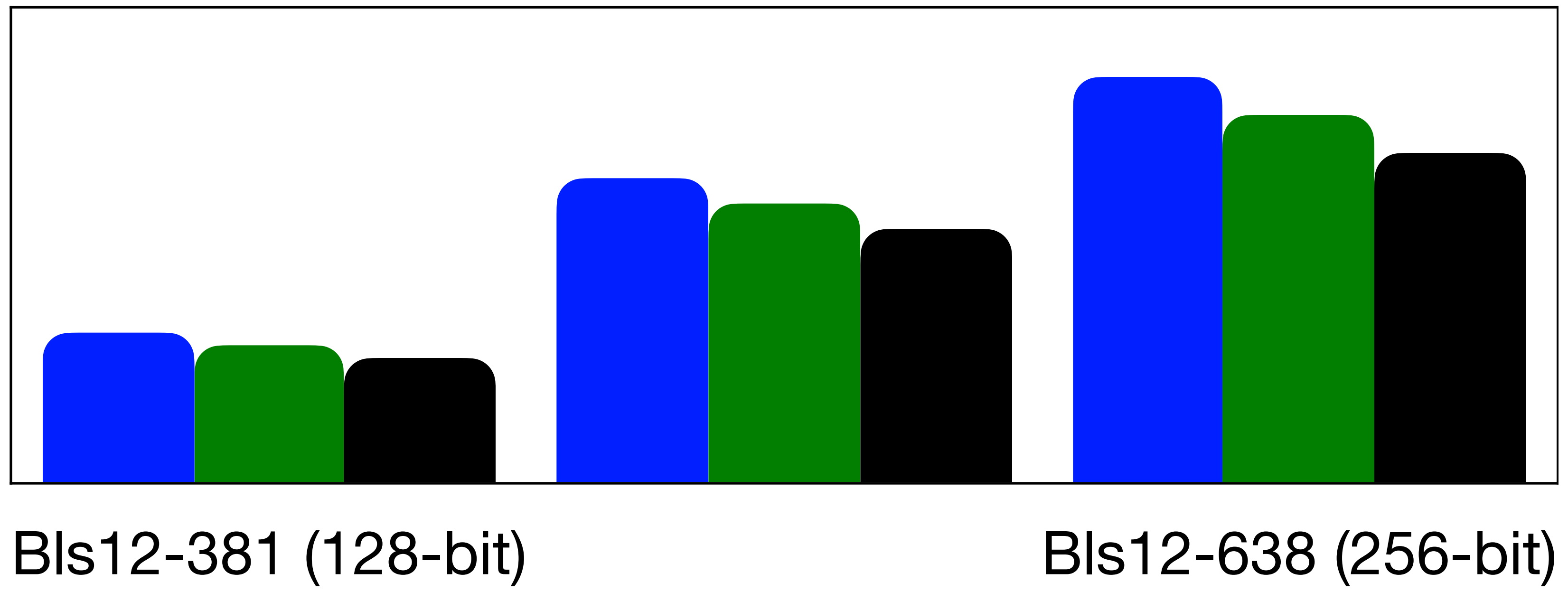


Evaluation for Pari

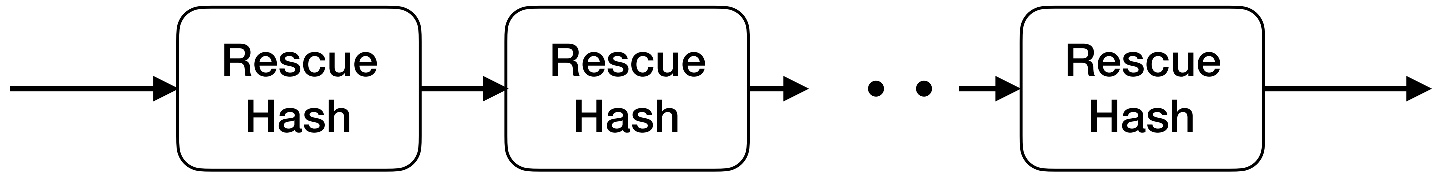
Proof size for BN curves



Proof size for BLS curves

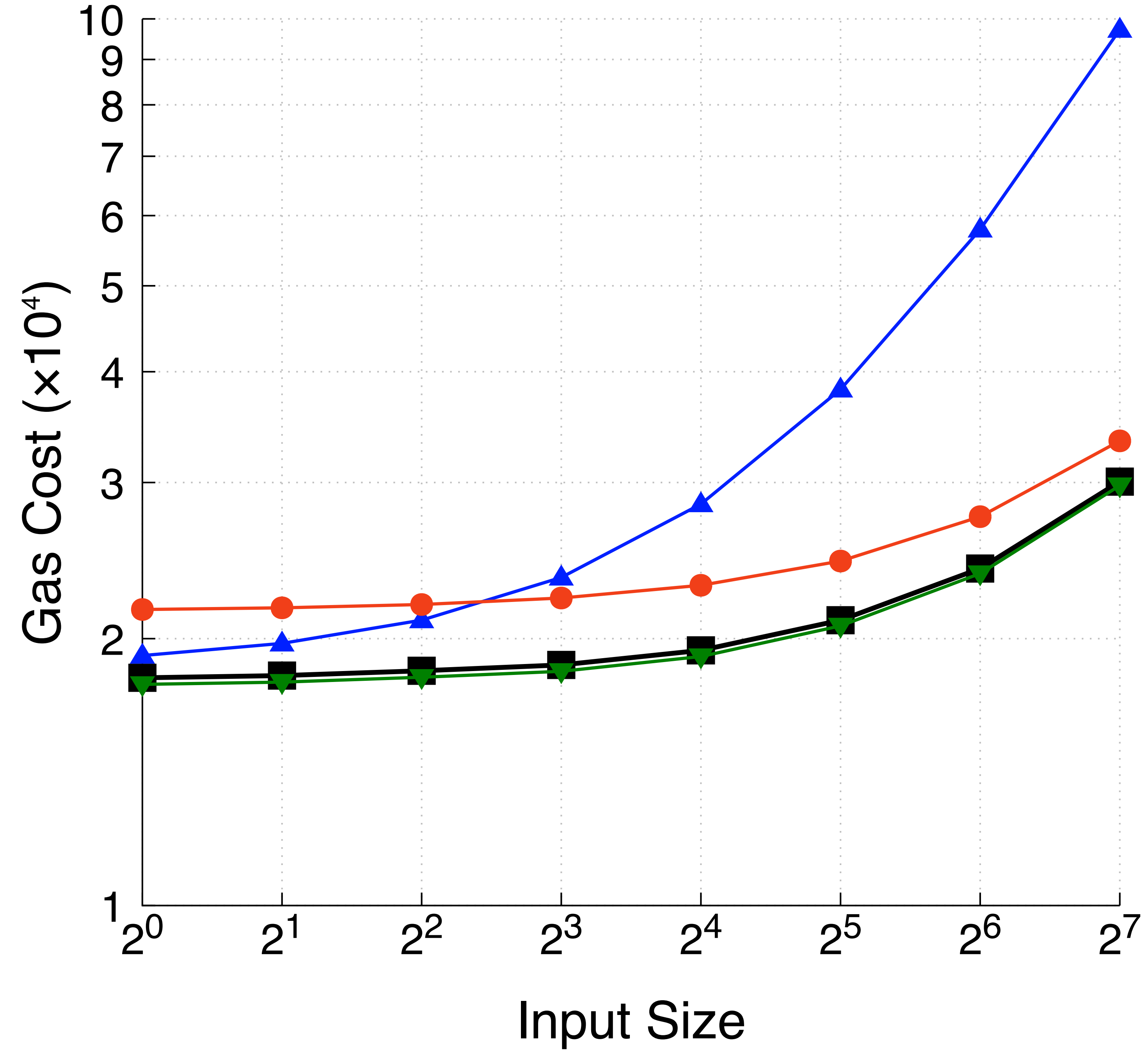


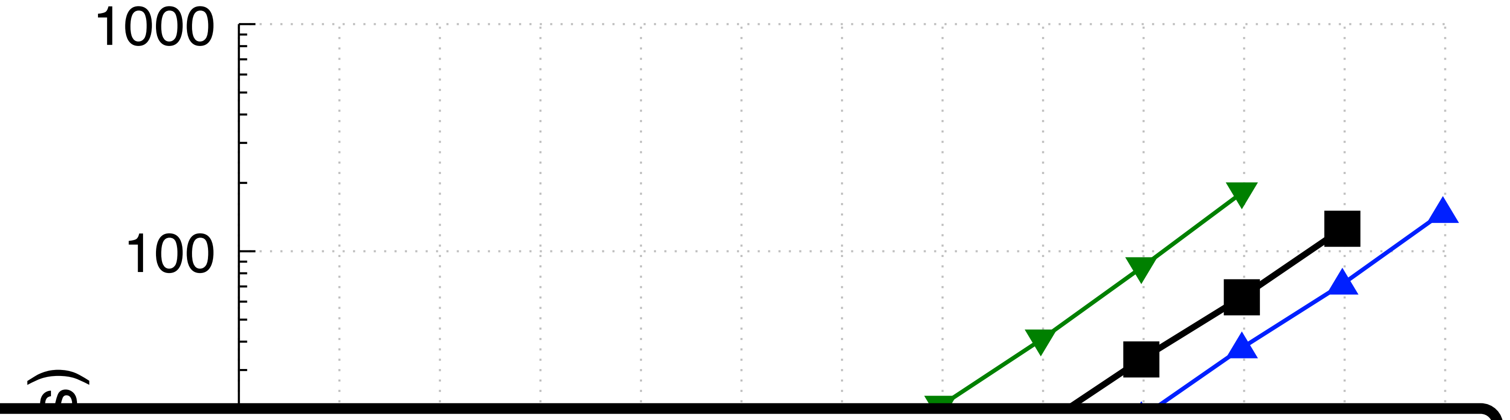
Benchmark results for a Hash-Chain circuit



Comparison with Groth16:
No verifier MSM
0.2 ms worse for #io=1

Comparison with Polymath:
~ 15% faster verifier





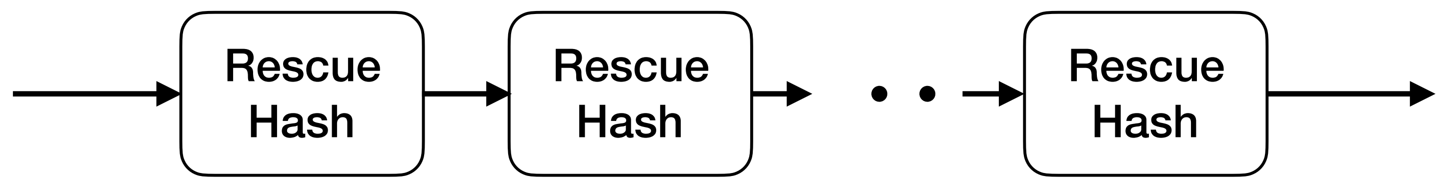
Comparison with Groth16:
< 2x slower prover

Comparison with Polymath:
~ 30% faster prover

Evaluation for Garuda

Evaluation for Garuda

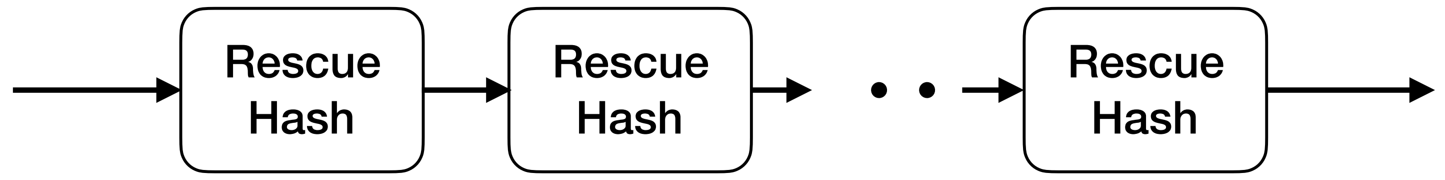
Same Hash-Chain circuit

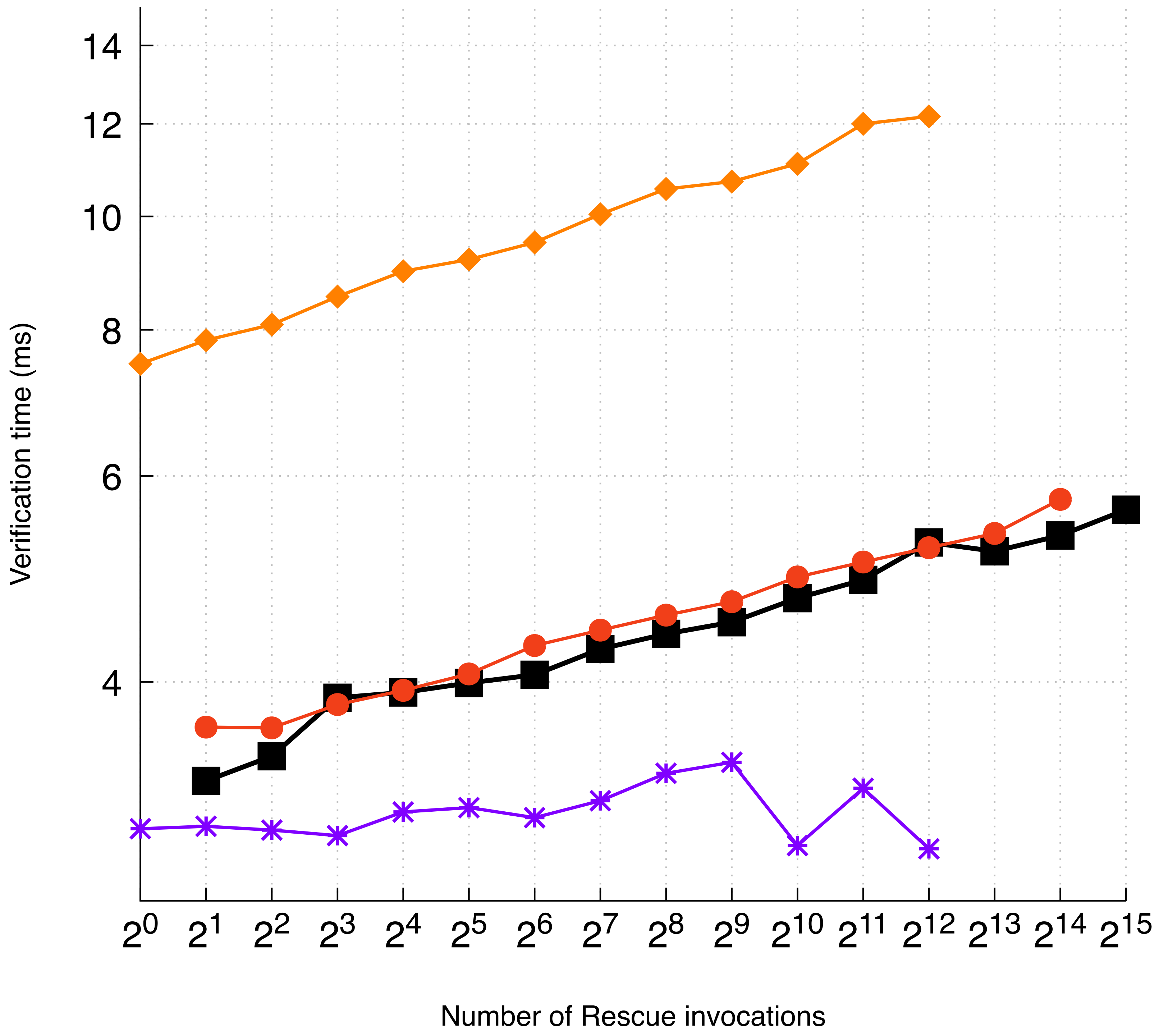


Evaluation for Garuda

Garuda (R1CS)
Garuda (GR1CS)
Groth16 (R1CS)
Hyperplonk (Plonkish)

Same Hash-Chain circuit



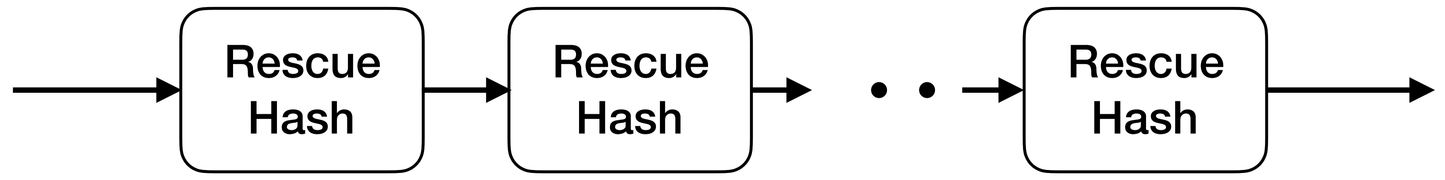


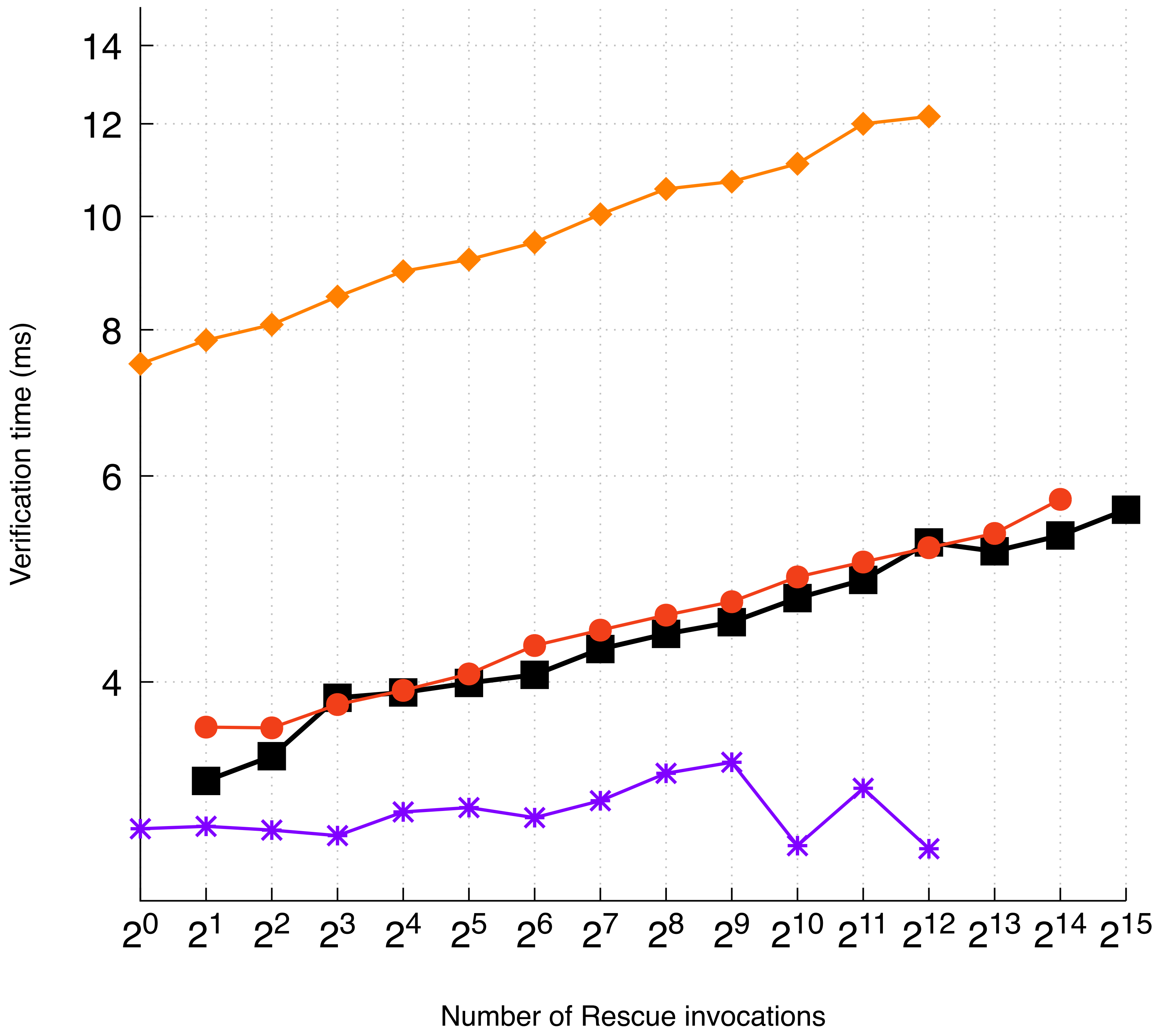
Garuda (R1CS)
Garuda (GR1CS)

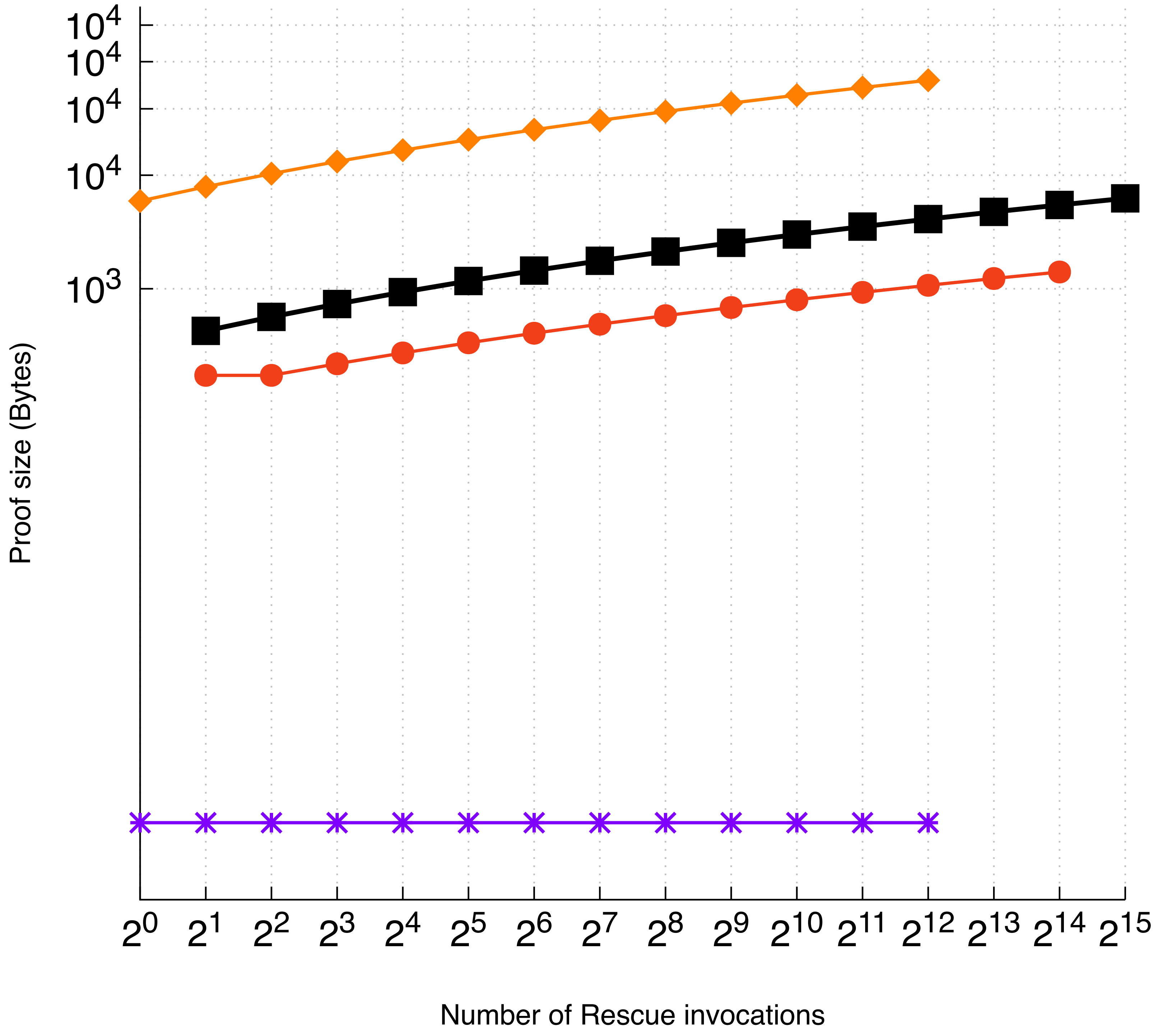
Evaluation for Garuda

Groth16 (R1CS)
Hyperplonk (Plonkish)

Same Hash-Chain circuit

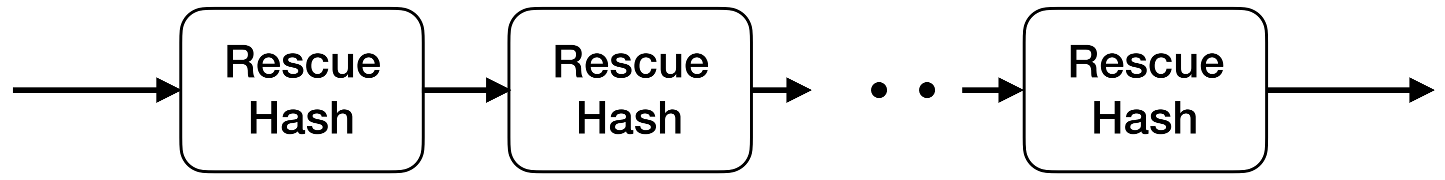






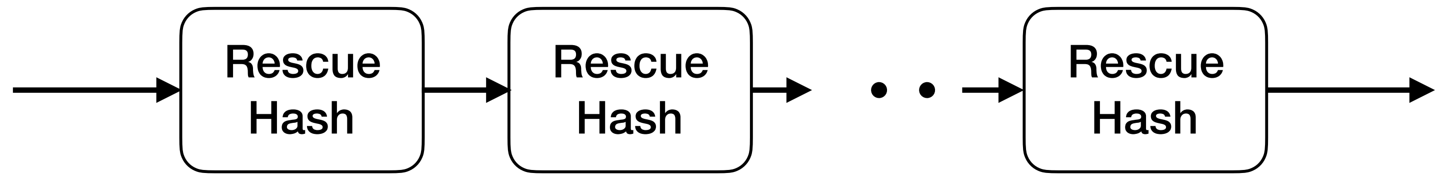
Evaluation for Garuda

Same Hash-Chain circuit



Evaluation for Garuda

Same Hash-Chain circuit



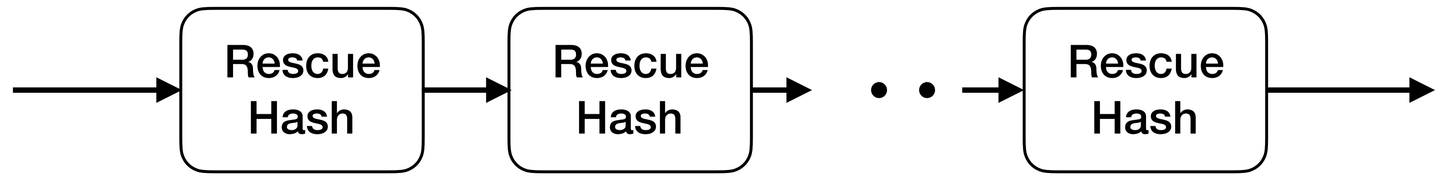
Comparison with Hyperplonk:
~ 2x faster verifier

Comparison with Groth16:
~ 2x slower verifier

Evaluation for Garuda

Garuda (R1CS)
Garuda (GR1CS)
Spartan (R1CS)
Groth16 (R1CS)
Hyperplonk (Plonkish)
SuperSpartan (CCS)

Same Hash-Chain circuit

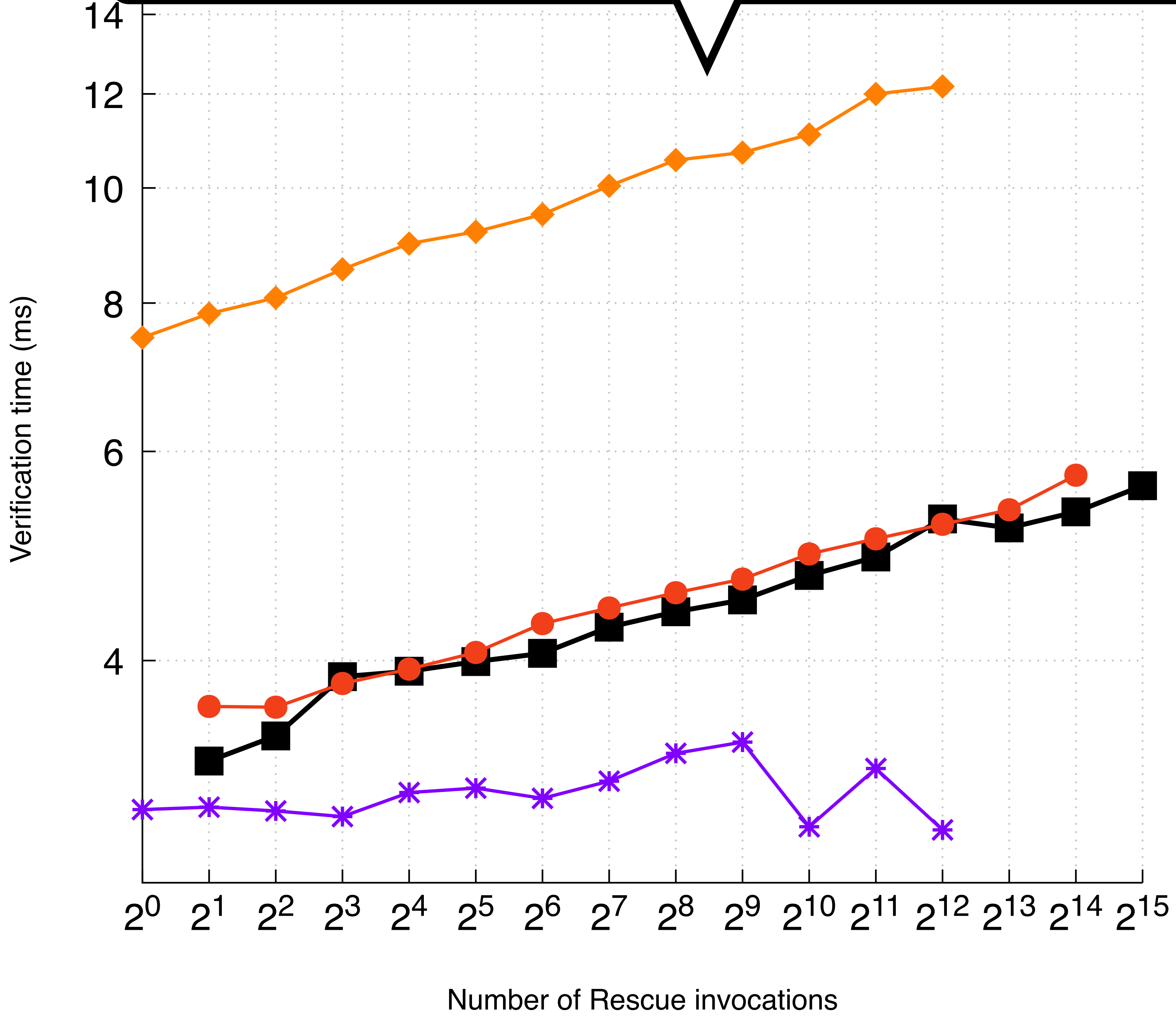


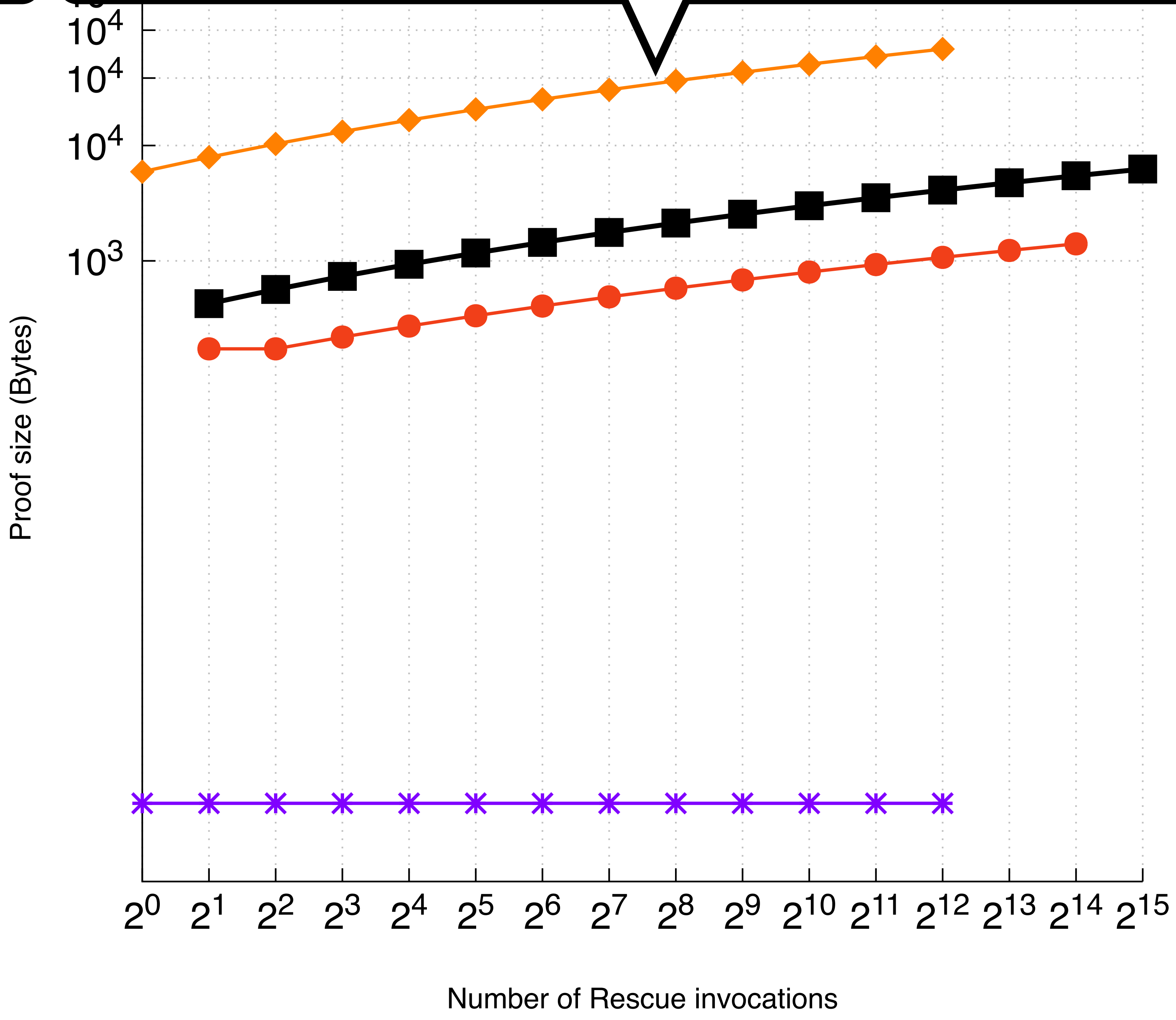
Comparison with Hyperplonk:
~ 2x faster verifier

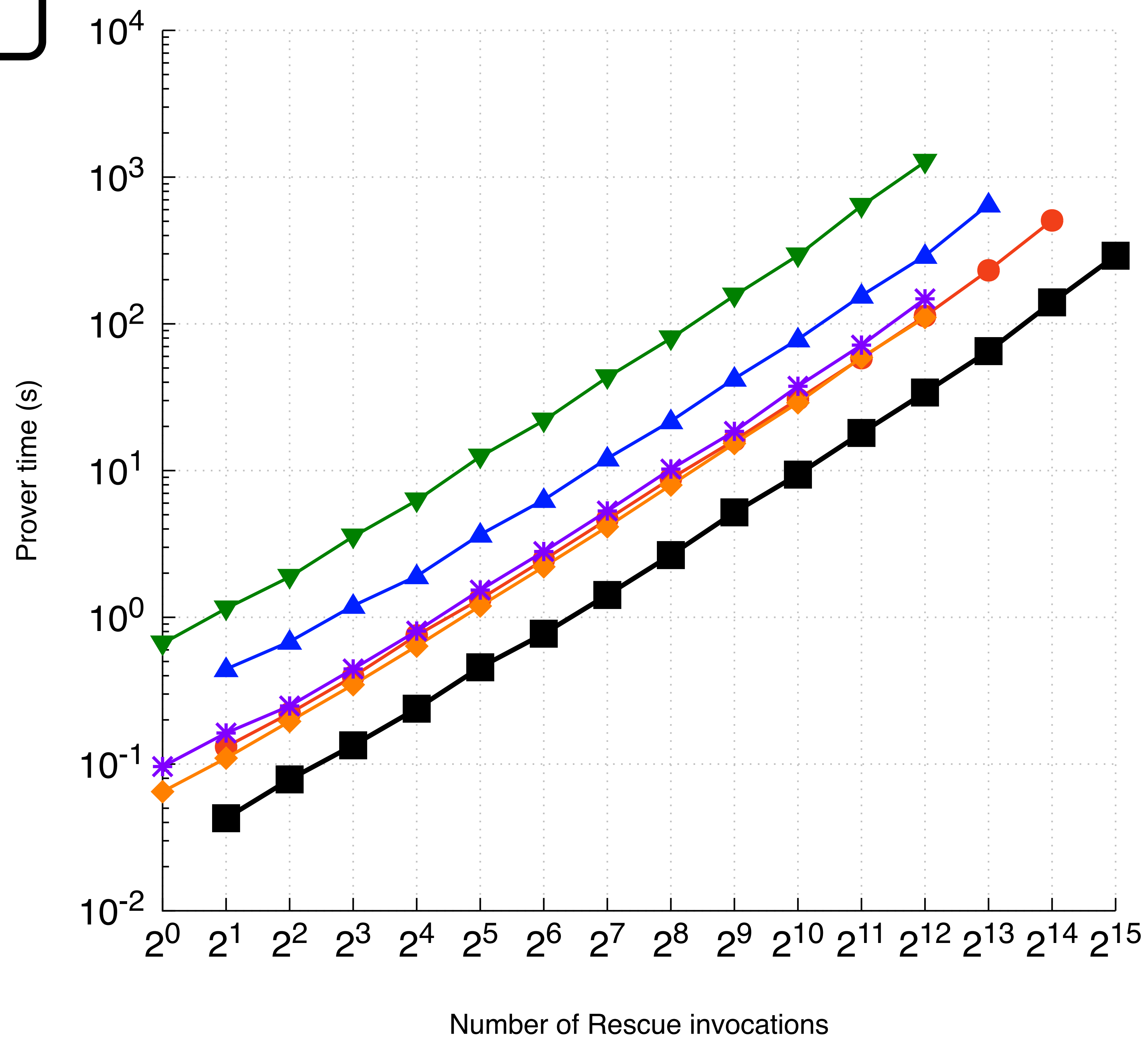
Comparison with Groth16:
~ 2x slower verifier

Comparison with Groth16:
~ 30x bigger

Comparison with Polymath:
~ 2x smaller



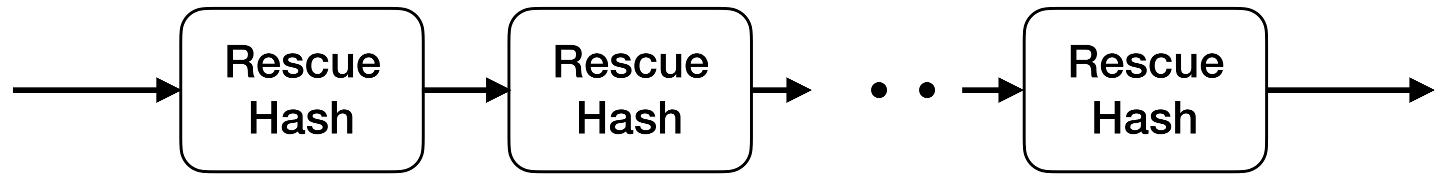




Evaluation for Garuda

Garuda (R1CS)
Garuda (GR1CS)
Spartan (R1CS)
Groth16 (R1CS)
Hyperplonk (Plonkish)
SuperSpartan (CCS)

Same Hash-Chain circuit



Comparison with Hyperplonk:
~ 2x faster verifier

Comparison with Groth16:
~ 2x slower verifier

Comparison with Groth16:
~ 30x bigger

Comparison with Polymath:
~ 2x smaller

Comparison with Groth16:
3x faster

Comparison with Polymath:
2x faster

Thanks!

github: *github.com/alireza-shirzad/garuda-pari*

Open questions

- Our EPC constructions imply circuit-specific setup
  Q: can we construct EPC schemes that achieve universal setup?
- What other applications of EPC schemes can we find?
  Ideas: Verifiable Secret Sharing, Accumulators, etc?
- Our SNARKs don't achieve ZK.
  Q: How can we demonstrate ZK without increasing the proof size?

Thanks!

ePrint: *https://eprint.iacr.org/2024/1245*

github: *github.com/alireza-shirzad/garuda-pari*

Open questions

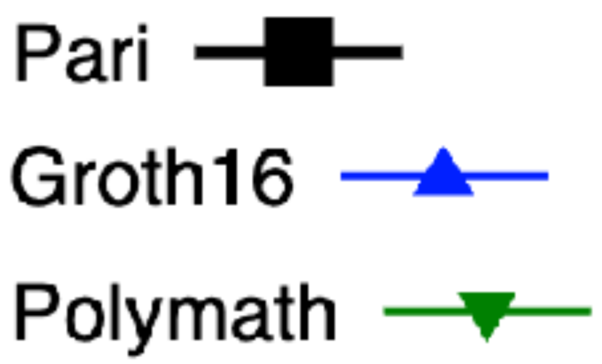
- Our EPC constructions imply circuit-specific setup

  Q: can we construct EPC schemes that achieve universal setup?
- What other applications of EPC schemes can we find?

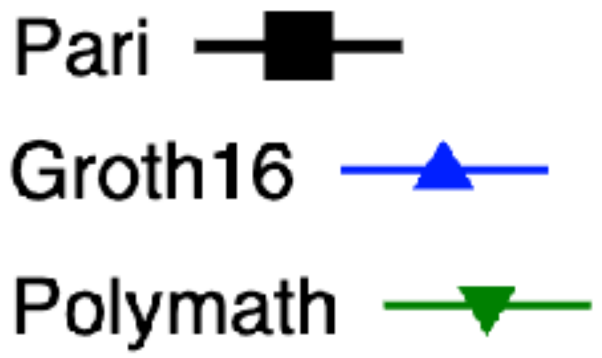
  Ideas: Verifiable Secret Sharing, Accumulators, etc?
- Our SNARKs don't achieve ZK.

  Q: How can we demonstrate ZK without increasing the proof size?

Evaluation for Pari

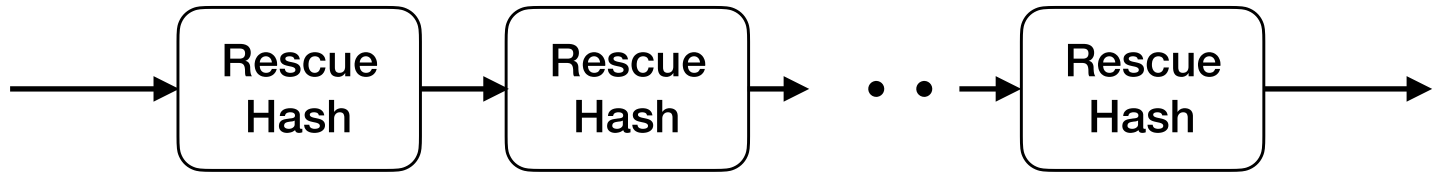


Evaluation for Pari



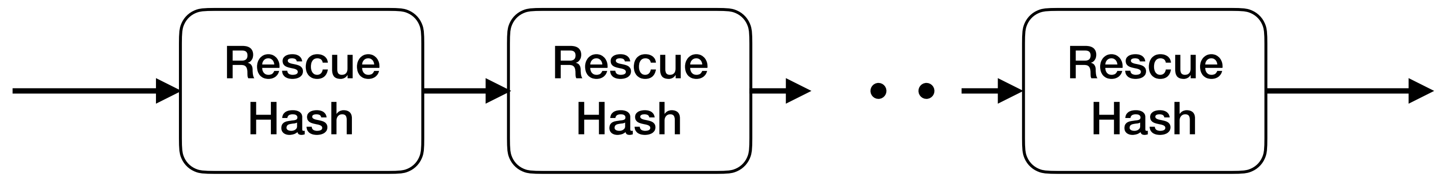
Evaluation for Pari

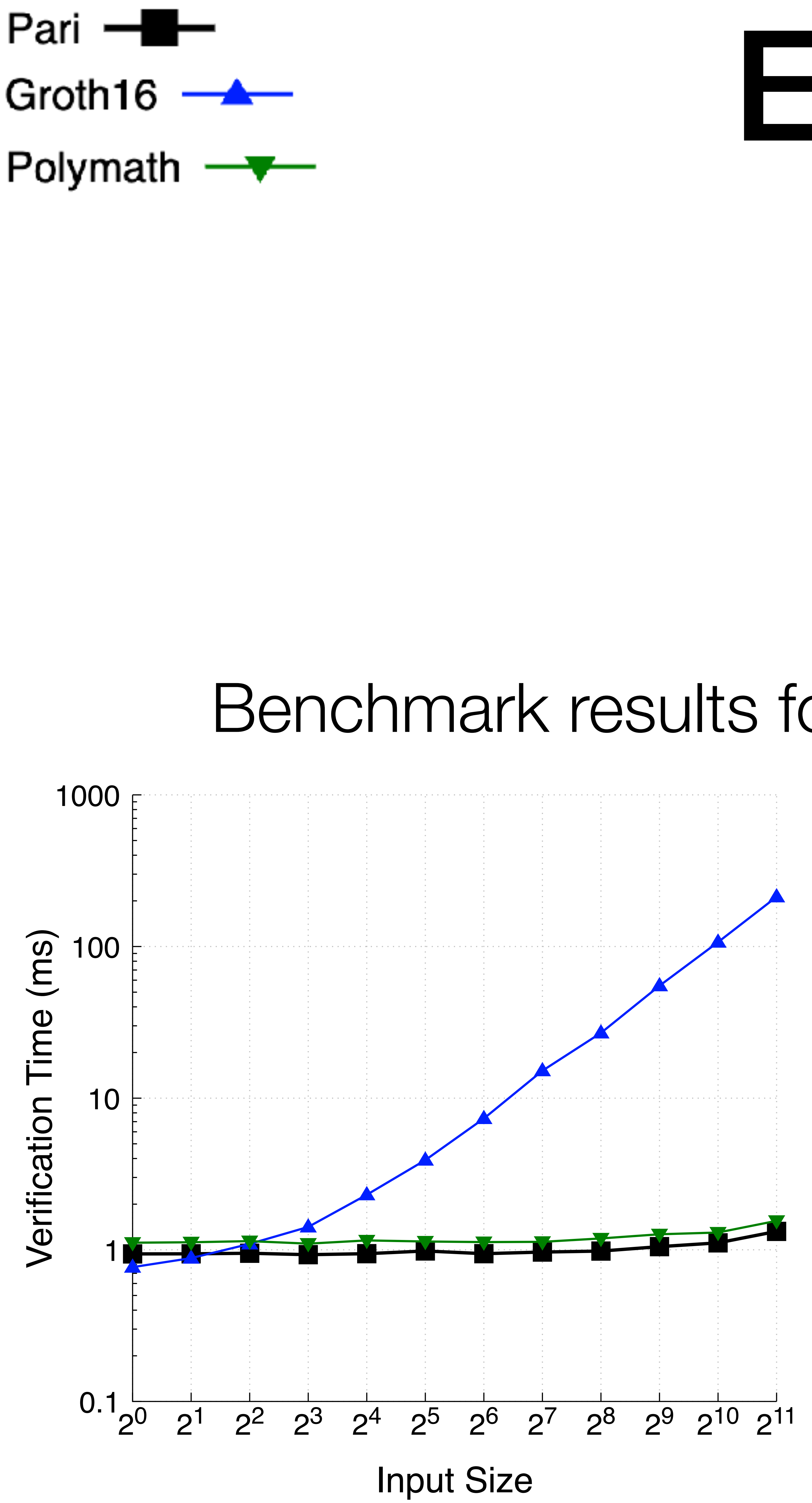
Benchmark results for a Hash-Chain circuit



Evaluation for Pari

Benchmark results for a Hash-Chain circuit



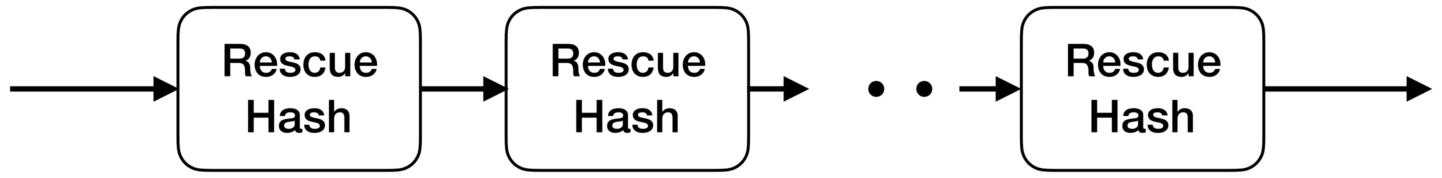


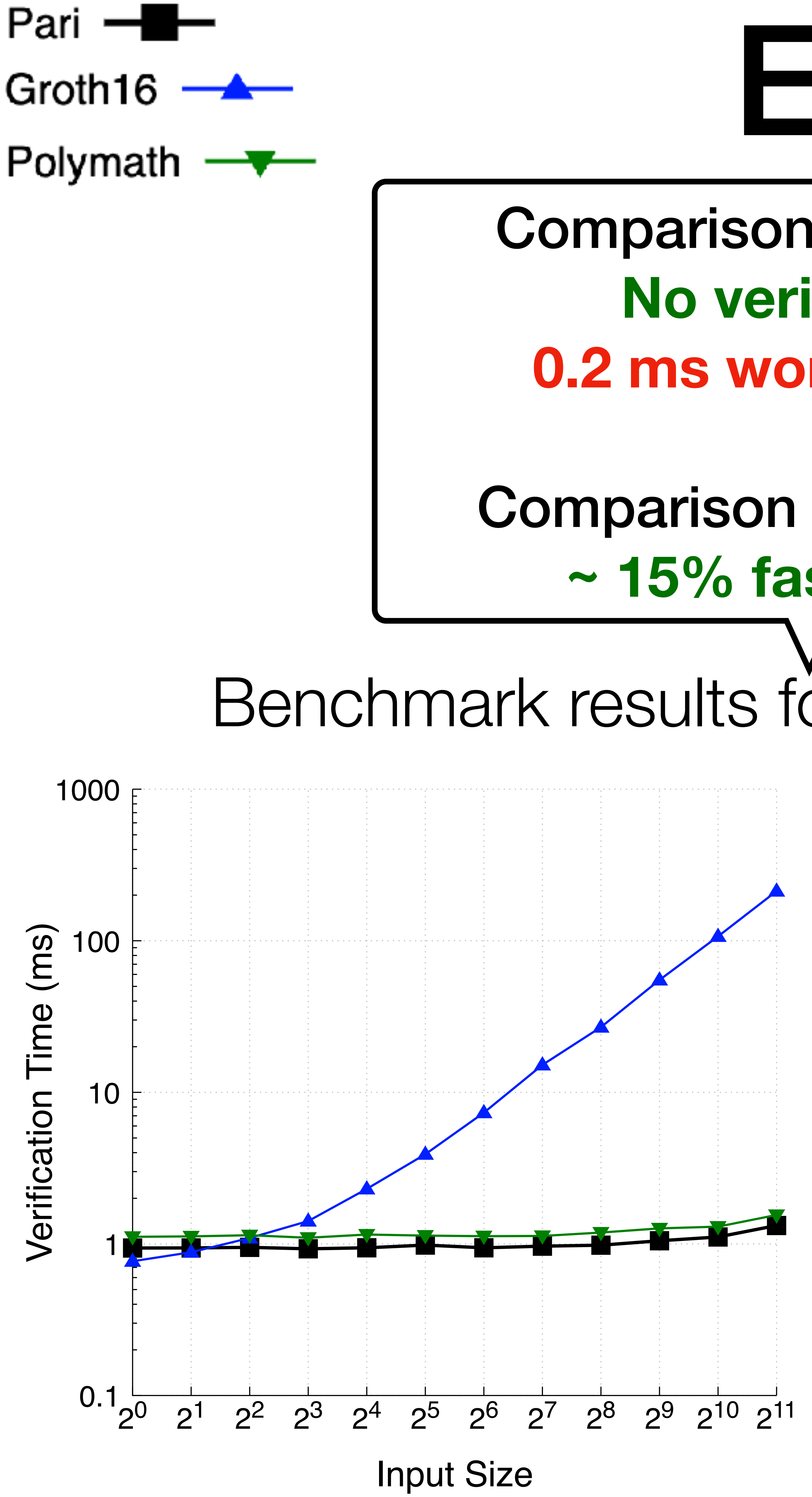
Evaluation for Pari

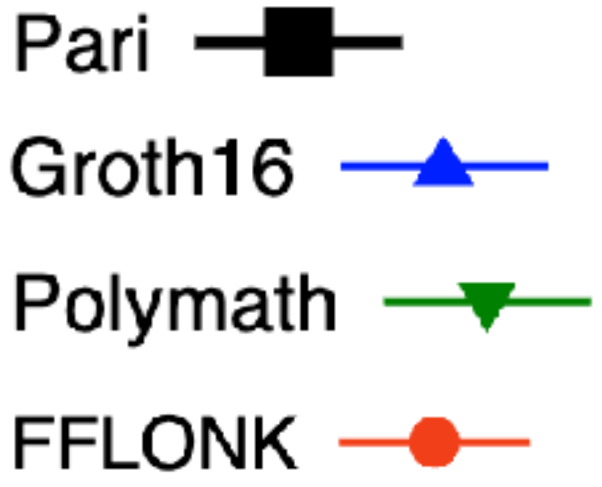
Comparison with Groth16:
No verifier MSM
0.2 ms worse for #io=1

Comparison with Polymath:
~ 15% faster verifier

Benchmark results for a Hash-Chain circuit





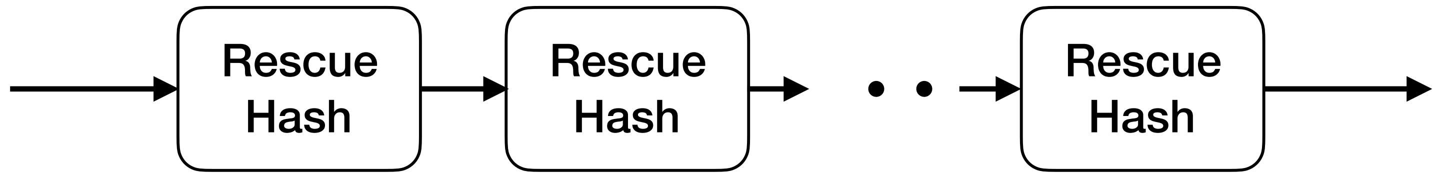


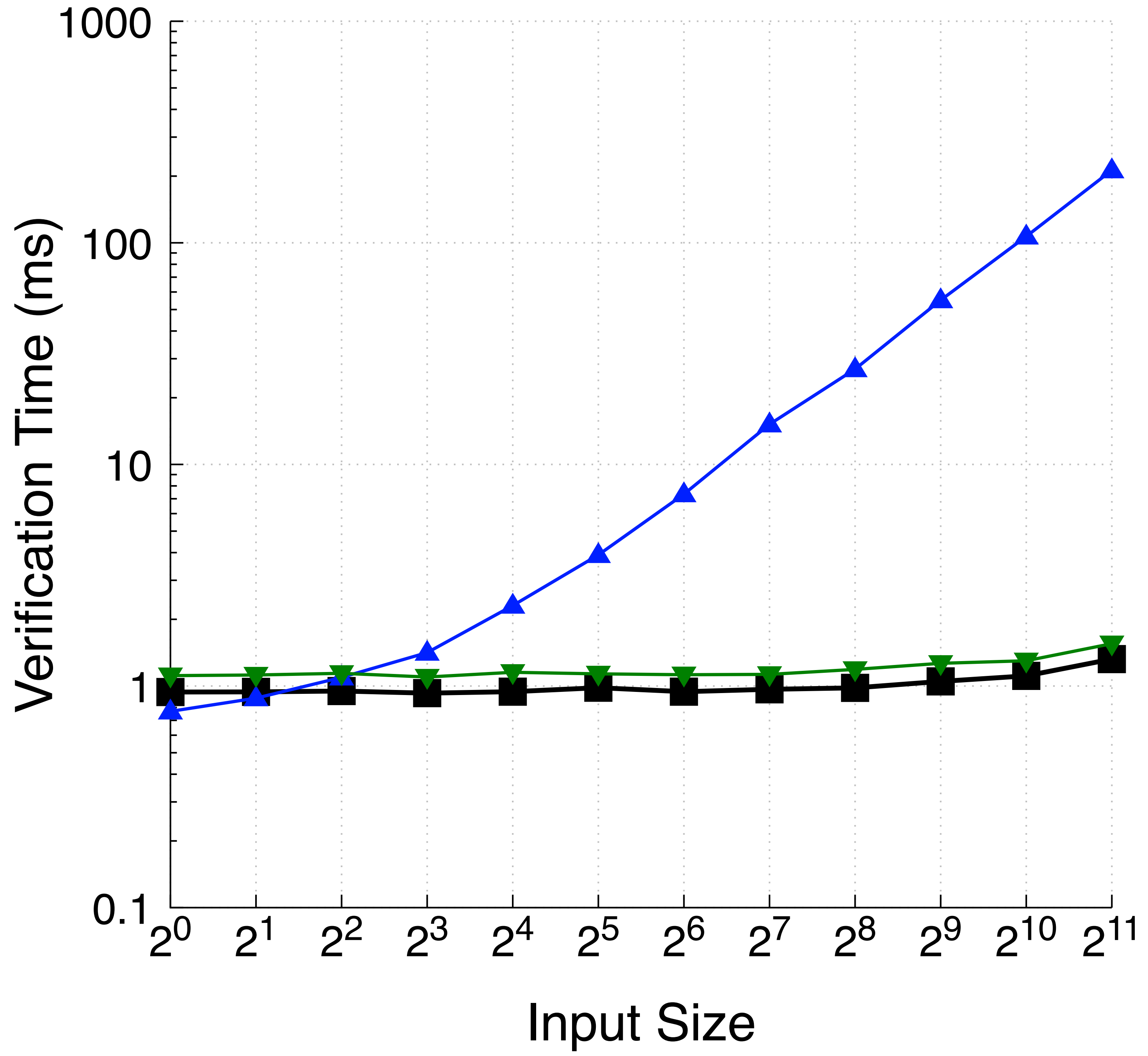
Evaluation for Pari

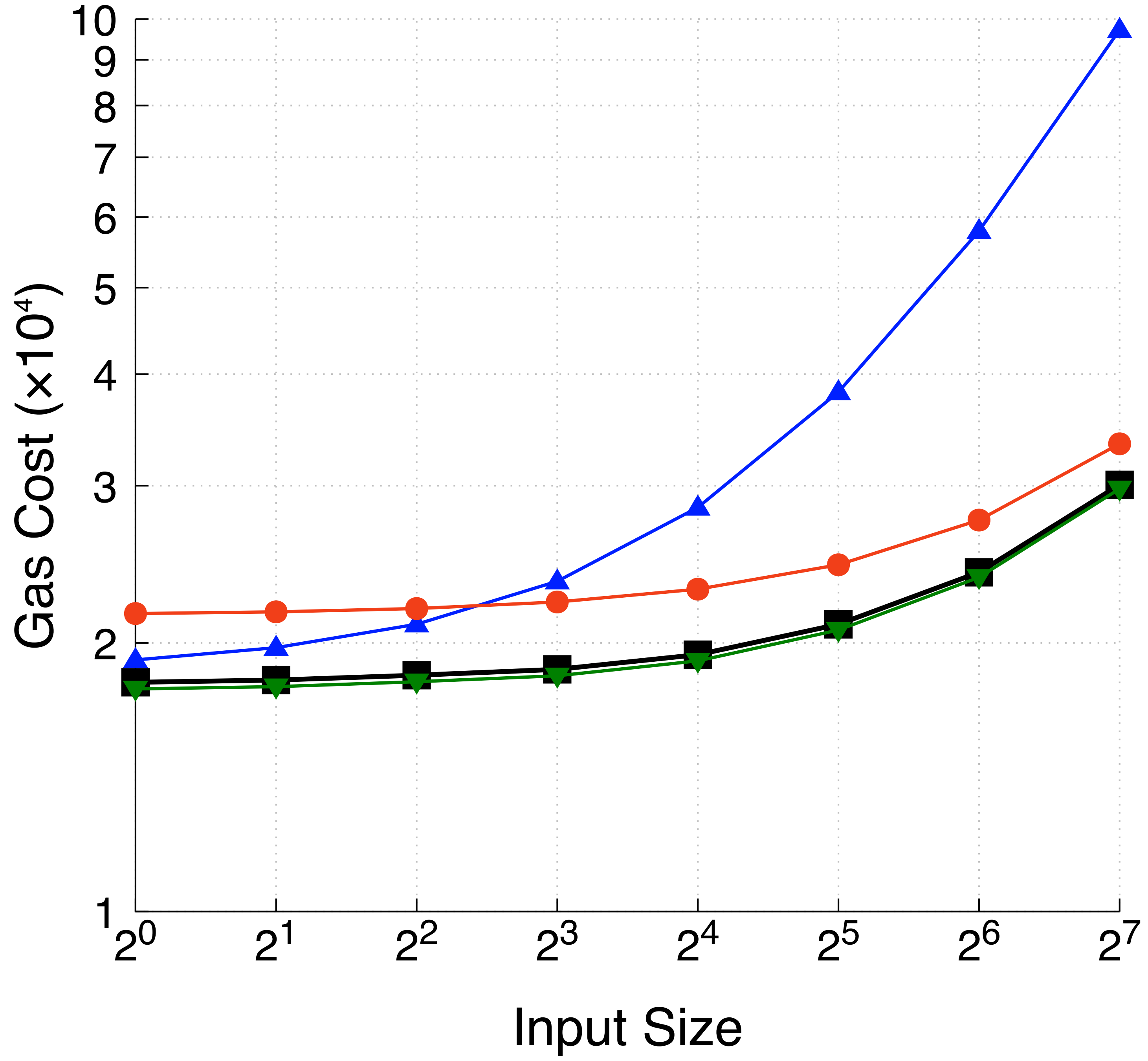
Comparison with Groth16:
No verifier MSM
0.2 ms worse for #io=1

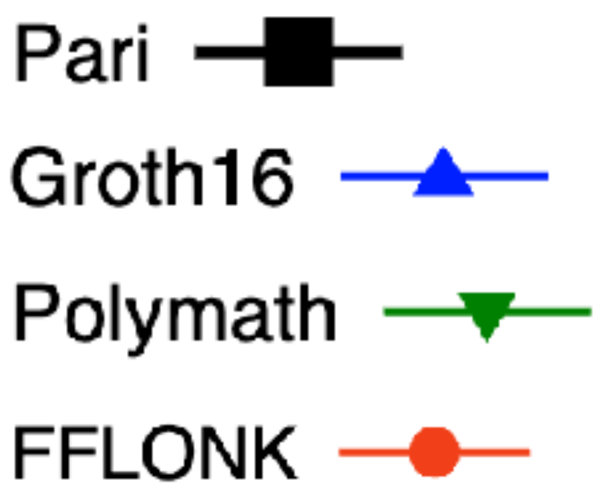
Comparison with Polymath:
~ 15% faster verifier

Benchmark results for a Hash-Chain circuit







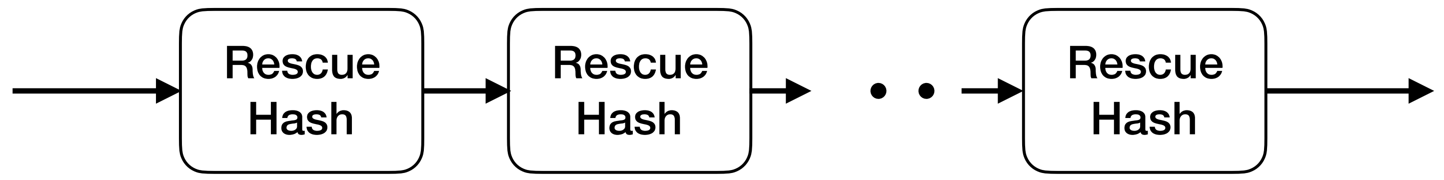


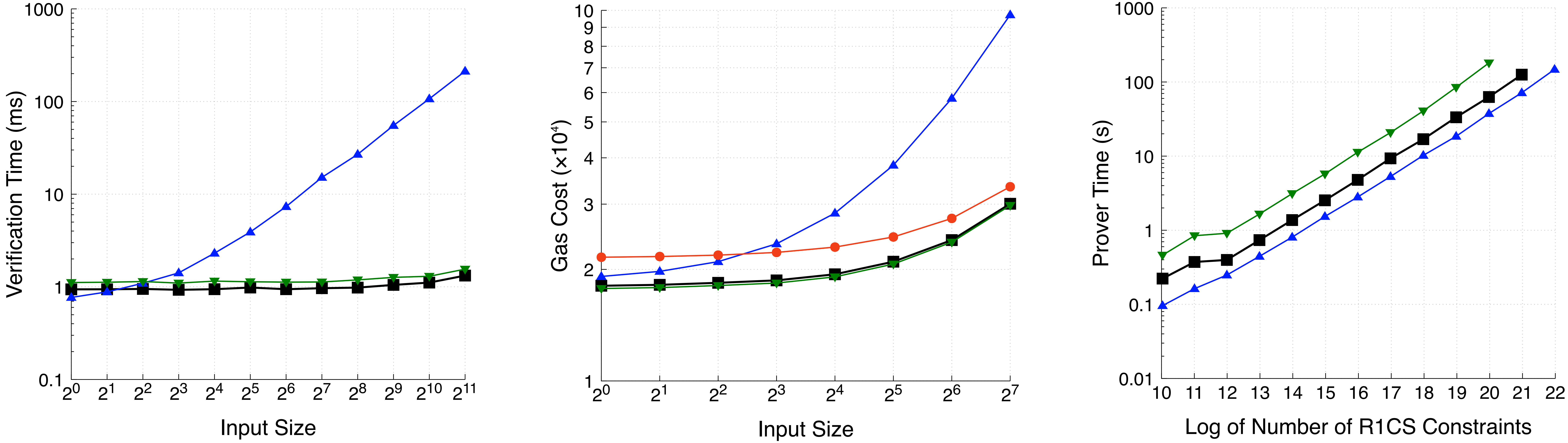
Evaluation for Pari

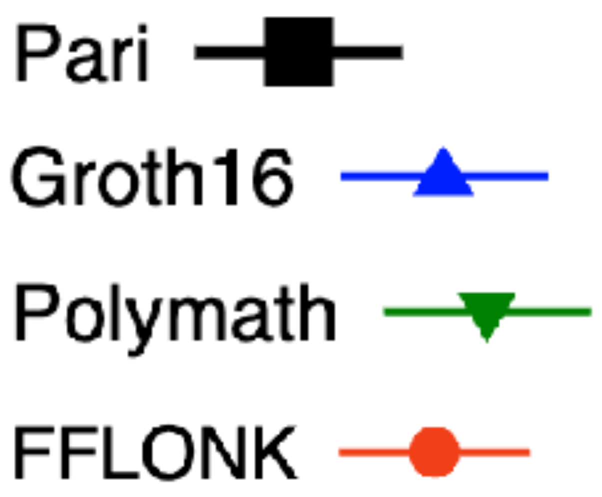
Comparison with Groth16:
No verifier MSM
0.2 ms worse for #io=1

Comparison with Polymath:
~ 15% faster verifier

Benchmark results for a Hash-Chain circuit







Evaluation for Pari

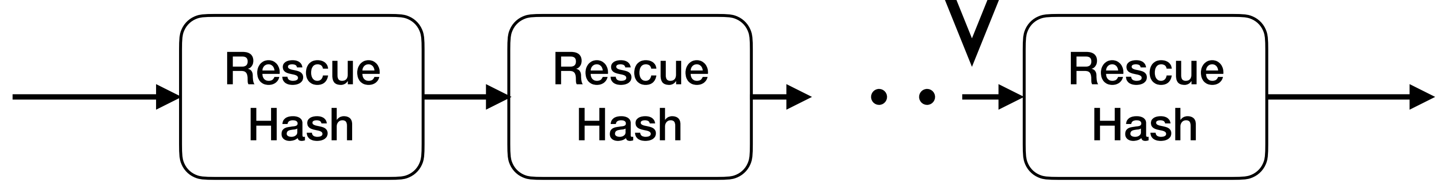
Comparison with Groth16:
No verifier MSM
0.2 ms worse for #io=1

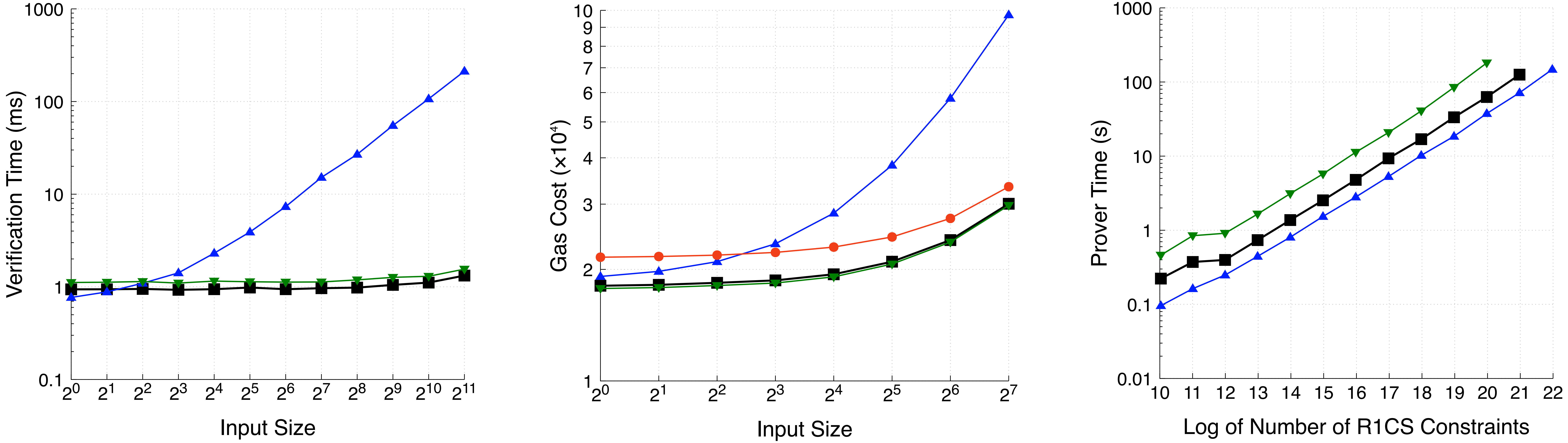
Comparison with Polymath:
~ 15% faster verifier

Comparison with Groth16:
< 2x slower prover

Comparison with Polymath:
~ 30% faster prover

Benchmark results for a Hash-Chain circuit





KZG-based EPC Construction

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \alpha \cdot \mathsf{ck}_A + \beta \cdot \mathsf{ck}_B + \gamma \cdot \mathsf{ck}_C \quad =$$

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \alpha \cdot \mathsf{ck}_A + \beta \cdot \mathsf{ck}_B + \gamma \cdot \mathsf{ck}_C \quad = \quad \begin{array}{l} \alpha \cdot \left[\, \hat{a}_1(\tau)\, G, \quad \hat{a}_2(\tau)\, G, \; \ldots \quad \hat{a}_n(\tau)\, G \,\right] + \\ \beta \cdot \left[\, \hat{b}_1(\tau)\, G, \quad \hat{b}_2(\tau)\, G, \; \ldots \quad \hat{b}_n(\tau)\, G \,\right] + \\ \gamma \cdot \left[\, \hat{c}_1(\tau)\, G, \quad \hat{c}_2(\tau)\, G, \; \ldots \quad \hat{c}_n(\tau)\, G \,\right] \end{array}$$

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \alpha \cdot \mathsf{ck}_A + \beta \cdot \mathsf{ck}_B + \gamma \cdot \mathsf{ck}_C = \begin{array}{l} \alpha \cdot [\ \hat{a}_1(\tau)\,G, \quad \hat{a}_2(\tau)\,G, \ \ldots \quad \hat{a}_n(\tau)\,G\ ] \ + \\ \beta \cdot [\ \hat{b}_1(\tau)\,G, \quad \hat{b}_2(\tau)\,G, \ \ldots \quad \hat{b}_n(\tau)\,G\ ] \ + \\ \gamma \cdot [\ \hat{c}_1(\tau)\,G, \quad \hat{c}_2(\tau)\,G, \ \ldots \quad \hat{c}_n(\tau)\,G\ ] \end{array}$$

These are random numbers in {1,…, n}

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \alpha \cdot \mathsf{ck}_A + \beta \cdot \mathsf{ck}_B + \gamma \cdot \mathsf{ck}_C \quad = \quad \begin{aligned} &\alpha \cdot \big[\, \hat{a}_1(\tau)\, G, \quad \hat{a}_2(\tau)\, G, \;\ldots\quad \hat{a}_n(\tau)\, G \,\big] + \\ &\beta \cdot \big[\, \hat{b}_1(\tau)\, G, \quad \hat{b}_2(\tau)\, G, \;\ldots\quad \hat{b}_n(\tau)\, G \,\big] + \\ &\gamma \cdot \big[\, \hat{c}_1(\tau)\, G, \quad \hat{c}_2(\tau)\, G, \;\ldots\quad \hat{c}_n(\tau)\, G \,\big] \end{aligned}$$

These are random numbers in {1,…, n}

Now, commit to z, which recall is the equal-coefficient representation of $\hat{z}_A$, $\hat{z}_B$, and $\hat{z}_C$

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \alpha \cdot \mathsf{ck}_A + \beta \cdot \mathsf{ck}_B + \gamma \cdot \mathsf{ck}_C = \begin{aligned} &\alpha \cdot [\ \hat{a}_1(\tau)\,G, \quad \hat{a}_2(\tau)\,G, \ \ldots \quad \hat{a}_n(\tau)\,G\ ] \ + \\ &\beta \cdot [\ \hat{b}_1(\tau)\,G, \quad \hat{b}_2(\tau)\,G, \ \ldots \quad \hat{b}_n(\tau)\,G\ ] \ + \\ &\gamma \cdot [\ \hat{c}_1(\tau)\,G, \quad \hat{c}_2(\tau)\,G, \ \ldots \quad \hat{c}_n(\tau)\,G\ ] \end{aligned}$$

These are random numbers in {1,…, n}

Now, commit to z, which recall is the equal-coefficient representation of $\hat{z}_A, \hat{z}_B$, and $\hat{z}_C$

$$\begin{aligned} c^* = \langle z, \mathsf{ck}^* \rangle = \ &\alpha \cdot \ (\,z_1 \cdot \hat{a}_1(\tau) + z_2 \cdot \hat{a}_2(\tau) + \ldots + z_n \cdot \hat{a}_n(\tau)) \cdot G \ + \\ &\beta \cdot \ (\,z_1 \cdot \hat{b}_1(\tau) + z_2 \cdot \hat{b}_2(\tau) + \ldots + z_n \cdot \hat{b}_n(\tau)) \cdot G \ + \\ &\gamma \cdot \ (\,z_1 \cdot \hat{c}_1(\tau) + z_2 \cdot \hat{c}_2(\tau) + \ldots + z_n \cdot \hat{c}_n(\tau)) \cdot G \end{aligned}$$

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \alpha \cdot \mathsf{ck}_A + \beta \cdot \mathsf{ck}_B + \gamma \cdot \mathsf{ck}_C \quad = \quad \begin{matrix} \alpha \cdot [\, \hat{a}_1(\tau)\, G, & \hat{a}_2(\tau)\, G, & \ldots & \hat{a}_n(\tau)\, G \,] + \\ \beta \cdot [\, \hat{b}_1(\tau)\, G, & \hat{b}_2(\tau)\, G, & \ldots & \hat{b}_n(\tau)\, G \,] + \\ \gamma \cdot [\, \hat{c}_1(\tau)\, G, & \hat{c}_2(\tau)\, G, & \ldots & \hat{c}_n(\tau)\, G \,] \end{matrix}$$

These are random numbers in {1,…, n}

Now, commit to z, which recall is the equal-coefficient representation of $\hat{z}_A$, $\hat{z}_B$, and $\hat{z}_C$

$$\begin{aligned} c^* = \langle z, \mathsf{ck}^* \rangle = \alpha \cdot & \; (z_1 \cdot \hat{a}_1(\tau) + z_2 \cdot \hat{a}_2(\tau) + \ldots + z_n \cdot \hat{a}_n(\tau)) \cdot G \; + \\ \beta \cdot & \; (z_1 \cdot \hat{b}_1(\tau) + z_2 \cdot \hat{b}_2(\tau) + \ldots + z_n \cdot \hat{b}_n(\tau)) \cdot G \; + \\ \gamma \cdot & \; (z_1 \cdot \hat{c}_1(\tau) + z_2 \cdot \hat{c}_2(\tau) + \ldots + z_n \cdot \hat{c}_n(\tau)) \cdot G \end{aligned}$$

KZG-based EPC Construction

⋮

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$\vdots$

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}^* = \alpha \cdot \mathsf{ck}_A + \beta \cdot \mathsf{ck}_B + \gamma \cdot \mathsf{ck}_C = \begin{bmatrix} \left(\alpha \cdot a_1(\tau) + \beta \cdot b_1(\tau) + \gamma \cdot c_1(\tau)\right) \cdot G \\ \vdots \\ \left(\alpha \cdot a_n(\tau) + \beta \cdot b_n(\tau) + \gamma \cdot c_n(\tau)\right) \cdot G \end{bmatrix}$$

These are random numbers in {1,…, n}

KZG-based EPC Construction

Step 3: Next, we take a random linear combination of these committer keys to get the "consistency" committer key!

$$\mathsf{ck}* = \alpha \cdot \mathsf{ck}_A + \beta \cdot \mathsf{ck}_B + \gamma \cdot \mathsf{ck}_C = \begin{bmatrix} (\alpha \cdot a_1(\tau) + \beta \cdot b_1(\tau) + \gamma \cdot c_1(\tau)) \cdot G \\ \vdots \\ (\alpha \cdot a_n(\tau) + \beta \cdot b_n(\tau) + \gamma \cdot c_n(\tau)) \cdot G \end{bmatrix}$$

These are random numbers in {1,…, n}

Now, commit to z, which recall is the equal-coefficient representation of $\hat{z}_A$, $\hat{z}_B$, and $\hat{z}_C$

$$c* = \langle z, \mathsf{ck}^* \rangle = \left(\alpha \cdot \hat{z}_A(\tau) + \beta \cdot \hat{z}_B(\tau) + \gamma \cdot \hat{z}_C(\tau)\right) \cdot G$$

This is the consistency commitment!

Pari

Roadmap for Pari

1. Design a univariate EPC scheme
2. Design a univariate PIOP for R1CS rowcheck
3. Compile the PIOP with EPC
4. Optimize to reduce proof size

Roadmap for Pari

1. Design a univariate EPC scheme
2. Design a univariate PIOP for R1CS rowcheck
3. Compile the PIOP with EPC
4. Optimize to reduce proof size

Roadmap for Pari

1. Design a univariate EPC scheme
2. Design a univariate PIOP for R1CS rowcheck
3. Compile the PIOP with EPC
4. Optimize to reduce proof size

Note: For simplicity, we assume that public input length is 0.

KZG Polynomial Commitment Scheme for p_1 and p_2

KZG Polynomial Commitment Scheme for p_1 and p_2

- Setup(D) $\rightarrow$ pp $= (G, \tau G, \tau^2 G, \ldots, \tau^n G)$

KZG Polynomial Commitment Scheme for p_1 and p_2

- Setup$(D) \rightarrow$ pp $= (G, \tau G, \tau^2 G, \ldots, \tau^n G)$
- Commit(ck, $\mathbf{p} = (p_1(X), p_2(X))) \rightarrow$ cm $:= (c_1 = p_1(\tau)G, c_2 = p_2(\tau)G)$

KZG Polynomial Commitment Scheme for p_1 and p_2

- Setup(D) → pp $= (G, \tau G, \tau^2 G, \ldots, \tau^n G)$

- Commit(ck, $\mathbf{p} = (p_1(X), p_2(X))$) → cm $:= (c_1 = p_1(\tau)G, c_2 = p_2(\tau)G)$

- Open(ck, $\mathbf{p}, z$) → $\pi = (\ \pi_1 = w_1(\tau)G\ ,\ \pi_2 = w_2(\tau)G\ )$

$$w_1(X) = \frac{p_1(X) - p_1(z)}{X - z} \qquad W_2(X) = \frac{p_2(X) - p_2(z)}{X - z}$$

KZG Polynomial Commitment Scheme for p_1 and p_2

- Setup(D) → pp $= (G, \tau G, \tau^2 G, \ldots, \tau^n G)$

- Commit(ck, $\mathbf{p} = (p_1(X), p_2(X))$) → cm := $(c_1 = p_1(\tau)G, c_2 = p_2(\tau)G)$

- Open(ck, $\mathbf{p}, z$) → $\pi = (\ \pi_1 = w_1(\tau)G\ ,\ \pi_2 = w_2(\tau)G\ )$

$$w_1(X) = \frac{p_1(X) - p_1(z)}{X - z} \qquad W_2(X) = \frac{p_2(X) - p_2(z)}{X - z}$$

- Verify(vk, cm, $z, \mathbf{v} = (v_1, v_2)$) → $\{0,1\}$

$$e(c_1, H) \stackrel{?}{=} e(\pi_1, \tau H - zH) \cdot e(p_1(z)G, H)$$
$$e(c_2, H) \stackrel{?}{=} e(\pi_2, \tau H - zH) \cdot e(p_2(z)G, H)$$

KZG-based EPC (Setup and Specialize)

KZG-based EPC (Setup and Specialize)

$$\mathsf{Setup}(n) \rightarrow \mathsf{pp}$$

$$\mathsf{pp} = (G, \tau G, \tau^2 G, \ldots, \tau^n G)$$

KZG-based EPC (Setup and Specialize)

$\mathsf{Setup}(n) \rightarrow \mathsf{pp}$

$\mathsf{pp} = (G, \tau G, \tau^2 G, \ldots, \tau^n G)$

$\mathscr{A} = (a_i(x))_{i=1}^n$

$\mathscr{B} = (b_i(x))_{i=1}^n$

$\mathsf{Specialize}(\mathsf{pp}, E = (\mathscr{A}, \mathscr{B})) \rightarrow (\mathsf{ck}, \mathsf{vk})$

Sample $\alpha, \beta \in \mathbb{F}$

$\mathsf{ck} = (G, \tau G, \tau^2 G, \ldots, \tau^n G) \cup \left((\alpha a_i(\tau) + \beta\, b_i(\tau))G\right)_{1=1}^n$

$\mathsf{vk} := \tau H, \alpha H, \beta H$

KZG-based EPC (Commit)

KZG-based EPC (Commit)

Equifficient

$\text{Commit}(\text{ck}, \mathbf{p} = (p_1(X), p_2(X))) \to c_1, c_2, c^*$

$$c_1 = p_1(\tau)G, \quad c_2 = p_2(\tau)G$$

KZG-based EPC (Commit)

Equifficient

$$\text{Commit}(\text{ck}, \mathbf{p} = (p_1(X), p_2(X))) \to c_1, c_2, c^*$$

$$c_1 = p_1(\tau)G, \quad c_2 = p_2(\tau)G$$

Consistency Commitment

$$c^* = (\alpha\, p_1(\tau) + \beta\, p_2(\tau))G$$

$$= \langle p_1, \left((\alpha\, a_i(\tau) + \beta\, b_i(\tau))G\right)_{1=1}^{n} \rangle$$

$$= \langle p_2, \left((\alpha\, a_i(\tau) + \beta\, b_i(\tau))G\right)_{1=1}^{n} \rangle$$

KZG-based EPC (Open & Verify)

KZG-based EPC (Open & Verify)

$$\text{Open}(\text{ck}, \mathbf{p}, z) \rightarrow \pi = (\pi_1 = w_1(\tau)G \ , \ \pi_2 = w_2(\tau)G)$$

$$w_1(x) = \frac{p_1(X) - p_1(z)}{X - z} \qquad w_2(X) = \frac{p_2(X) - p_2(z)}{X - z}$$

KZG-based EPC (Open & Verify)

$\text{Open}(\text{ck}, \mathbf{p}, z) \rightarrow \pi = (\pi_1 = w_1(\tau)G\ ,\ \pi_2 = w_2(\tau)G)$

$$w_1(x) = \frac{p_1(X) - p_1(z)}{X - z} \qquad w_2(X) = \frac{p_2(X) - p_2(z)}{X - z}$$

$\text{Verify}(\text{vk}, \text{cm}, z, \mathbf{v} = \{v_1, v_2\}) \rightarrow \{0,1\}$

$$e(c_1, H) \stackrel{?}{=} e(\pi_1, \tau H - zH) \cdot e(p_1(z)G, H)$$
$$e(c_2, H) \stackrel{?}{=} e(\pi_2, \tau H - zH) \cdot e(p_2(z)G, H)$$
$$e(c^*G, H) \stackrel{?}{=} e(c_1, \alpha H) + e(c_2, \beta H)$$

KZG-based EPC for p_1 and p_2

- Setup$(D) \rightarrow$ pp $= (G, \tau G, \tau^2 G, \ldots, \tau^n G)$
- Specialize(pp, $E = (\mathscr{A}, \mathscr{B})) \rightarrow ck, vk$
- Commit(ck, $\mathbf{p} = (p_1(X), p_2(X))) \rightarrow$ cm $= (c_1, c_2, c^*)$
- Open(ck, $\mathbf{p}, z) \rightarrow \pi = (\pi_1, \pi_2)$
- Verify(vk, cm, $z, \mathbf{v} = (v_1, v_2)) \rightarrow \{0,1\}$

$$e(c_1, H) \stackrel{?}{=} e(\pi_1, \tau H - zH) \cdot e(p(z)G, H)$$

$$e(c_2, H) \stackrel{?}{=} e(\pi_2, \tau H - zH) \cdot e(p(z)G, H)$$

$$e(c^*, H) \stackrel{?}{=} e(c_1, \alpha H) + e(c_2, \beta H)$$

KZG-based EPC for p_1 and p_2

- Setup(D) → pp $= (G, \tau G, \tau^2 G, \ldots, \tau^n G)$
- Specialize(pp, $E = (\mathscr{A}, \mathscr{B})$) → ck, vk
- Commit(ck, $\mathbf{p} = (p_1(X), p_2(X))$) → cm $= (c_1, c_2, c^*)$
- Open(ck, $\mathbf{p}$, z) → $\pi = (\pi_1, \pi_2)$
- Verify(vk, cm, z, $\mathbf{v} = (v_1, v_2)$) → $\{0,1\}$

$$e(c_1, H) \stackrel{?}{=} e(\pi_1, \tau H - zH) \cdot e(p(z)G, H)$$

$$e(c_2, H) \stackrel{?}{=} e(\pi_2, \tau H - zH) \cdot e(p(z)G, H)$$

$$e(c^*, H) \stackrel{?}{=} e(c_1, \alpha H) + e(c_2, \beta H)$$

$3\mathbb{G}$ elements for commitment

KZG-based EPC for p_1 and p_2

- Setup$(D) \rightarrow$ pp $= (G, \tau G, \tau^2 G, \ldots, \tau^n G)$
- Specialize(pp, $E = (\mathscr{A}, \mathscr{B})) \rightarrow ck, vk$
- Commit(ck, $\mathbf{p} = (p_1(X), p_2(X))) \rightarrow$ cm $= (c_1, c_2, c^*)$
- Open(ck, $\mathbf{p}, z) \rightarrow \pi = (\pi_1, \pi_2)$
- Verify(vk, cm, $z, \mathbf{v} = (v_1, v_2)) \rightarrow \{0,1\}$

3$\mathbb{G}$ elements for commitment

2$\mathbb{G}$ elements for opening

$$e(c_1, H) \stackrel{?}{=} e(\pi_1, \tau H - zH) \cdot e(p(z)G, H)$$

$$e(c_2, H) \stackrel{?}{=} e(\pi_2, \tau H - zH) \cdot e(p(z)G, H)$$

$$e(c^*, H) \stackrel{?}{=} e(c_1, \alpha H) + e(c_2, \beta H)$$

Roadmap for Pari

1. Design a univariate EPC scheme
2. Design a univariate PIOP for R1CS rowcheck
3. Compile the PIOP with EPC
4. Optimize to reduce proof size

Roadmap for Pari

1. Design a univariate EPC scheme
2. Design a univariate PIOP for R1CS rowcheck
3. Compile the PIOP with EPC
4. Optimize to reduce proof size

RowCheck PIOP

RowCheck PIOP

PIOP to check that for three polynomials $\hat{z}_a(X), \hat{z}_b(X), \hat{z}_c(X)$

it holds that for each $i \in \{1,...,m\}$: $\hat{z}_a(i) * \hat{z}_b(i) = \hat{z}_c(i)$

RowCheck PIOP

PIOP to check that for three polynomials $\hat{z}_a(X), \hat{z}_b(X), \hat{z}_c(X)$

it holds that for each $i \in \{1,...,m\}$: $\hat{z}_a(i) * \hat{z}_b(i) = \hat{z}_c(i)$

Let $t(X) = (X-1)(X-2)\ldots(X-m)$ be the vanishing polynomial

RowCheck PIOP

PIOP to check that for three polynomials $\hat{z}_a(X), \hat{z}_b(X), \hat{z}_c(X)$

it holds that for each $i \in \{1,...,m\}$: $\hat{z}_a(i) * \hat{z}_b(i) = \hat{z}_c(i)$

Let $t(X) = (X-1)(X-2)\ldots(X-m)$ be the vanishing polynomial

$$t(X) \mid \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) \;\leftrightarrow\; \exists q(X)\text{:}\;\; \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = t(X)q(X)$$

RowCheck PIOP

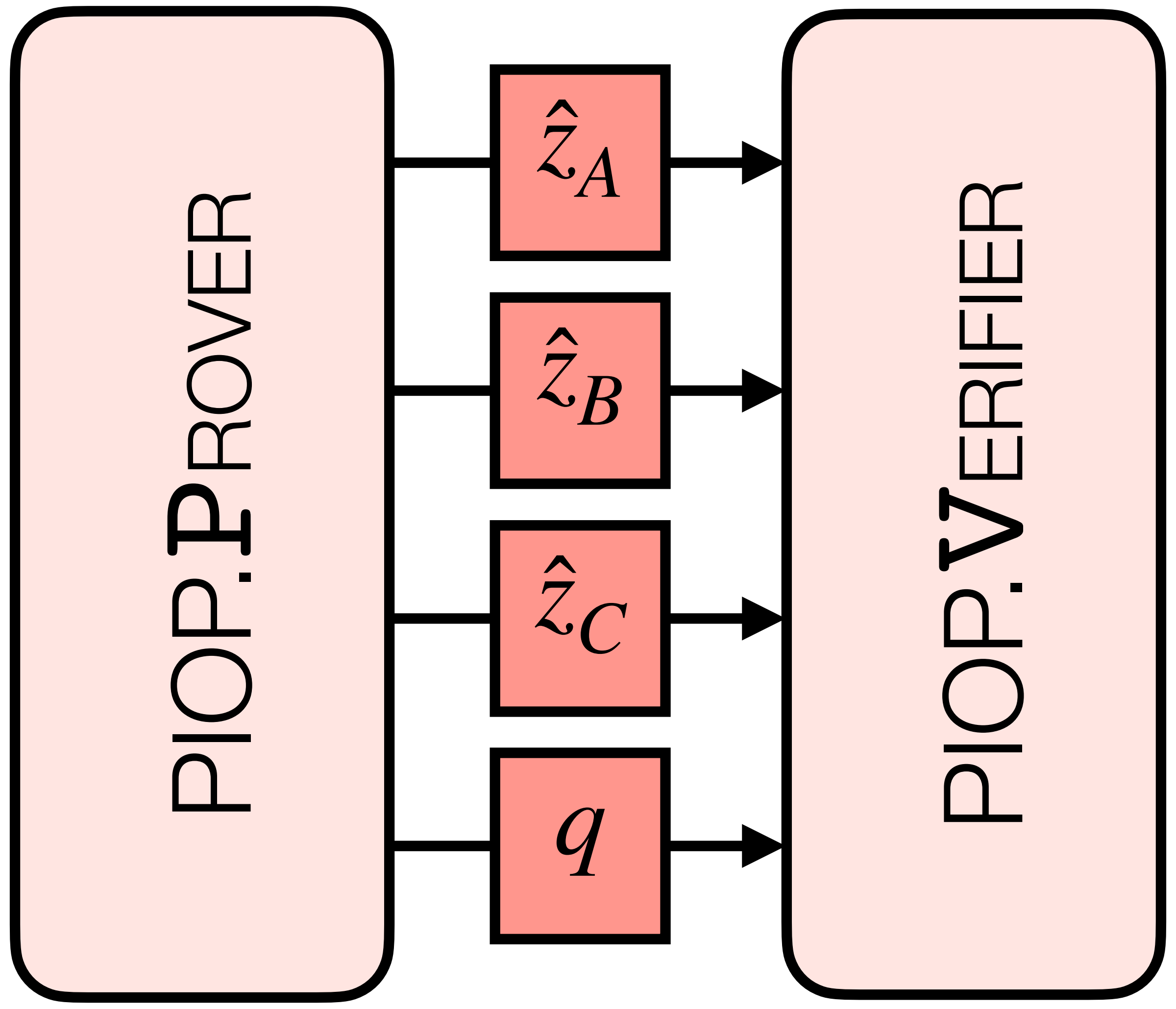
PIOP to check that for three polynomials $\hat{z}_a(X), \hat{z}_b(X), \hat{z}_c(X)$

it holds that for each $i \in \{1,...,m\}$: $\hat{z}_a(i) * \hat{z}_b(i) = \hat{z}_c(i)$

Let $t(X) = (X-1)(X-2)\ldots(X-m)$ be the vanishing polynomial

$$t(X) \mid \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) \leftrightarrow \exists q(X): \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = t(X)q(X)$$

$$q(X) = \frac{\hat{z}_a(X) \cdot \hat{z}_b(X)}{\hat{z}_c(X)}$$



RowCheck PIOP

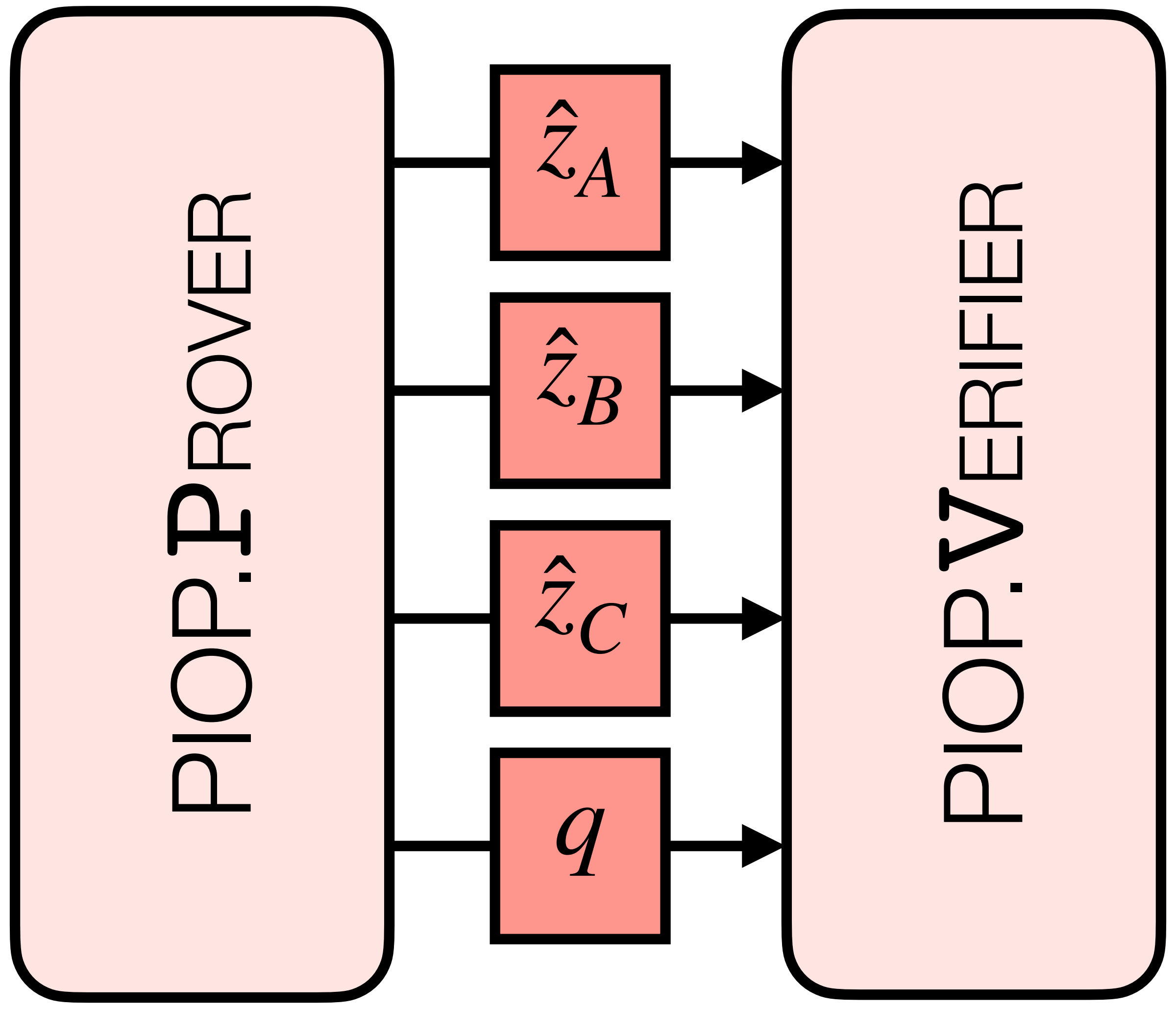
PIOP to check that for three polynomials $\hat{z}_a(X), \hat{z}_b(X), \hat{z}_c(X)$

it holds that for each $i \in \{1,\ldots,m\}$: $\hat{z}_a(i) * \hat{z}_b(i) = \hat{z}_c(i)$

Let $t(X) = (X-1)(X-2)\ldots(X-m)$ be the vanishing polynomial

$$t(X) \mid \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) \leftrightarrow \exists q(X): \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = t(X)q(X)$$

$$q(X) = \frac{\hat{z}_a(X) \cdot \hat{z}_b(X)}{\hat{z}_c(X)}$$



$$\hat{z}_A(r) \cdot \hat{z}_B(r) - \hat{z}_C(r) \stackrel{?}{=} t(r) \cdot q(r)$$

RowCheck PIOP

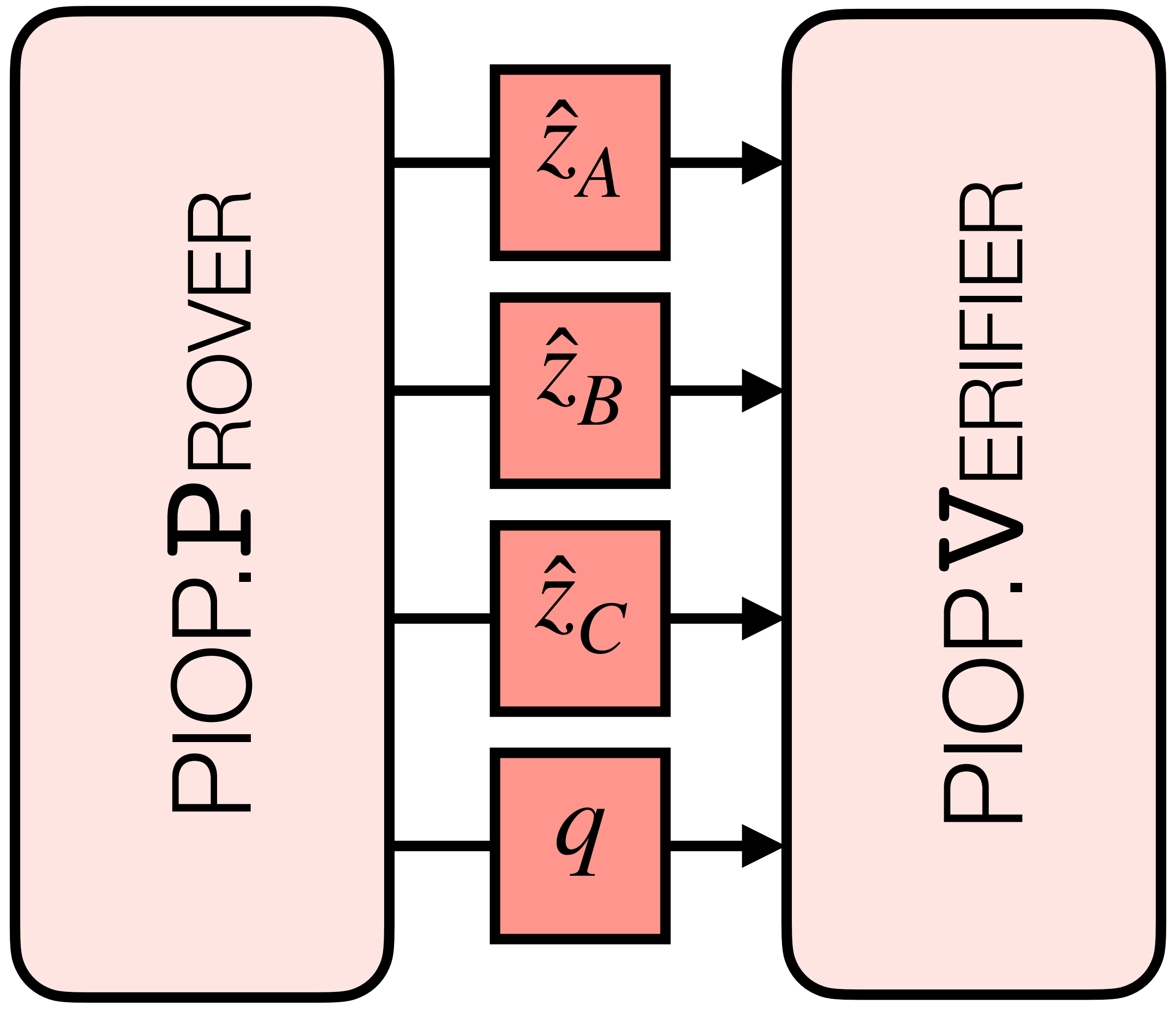
PIOP to check that for three polynomials $\hat{z}_a(X), \hat{z}_b(X), \hat{z}_c(X)$

it holds that for each $i \in \{1,...,m\}$: $\hat{z}_a(i) * \hat{z}_b(i) = \hat{z}_c(i)$

Let $t(X) = (X-1)(X-2)\ldots(X-m)$ be the vanishing polynomial

$$t(X) \mid \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) \leftrightarrow \exists q(X): \; \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = t(X)q(X)$$

$$q(X) = \frac{\hat{z}_a(X) \cdot \hat{z}_b(X)}{\hat{z}_c(X)}$$



$$\hat{z}_A(r) \cdot \hat{z}_B(r) - \hat{z}_C(r) \stackrel{?}{=} t(r) \cdot q(r)$$

Note: In practice, We replace $\{1,\ldots,m\}$ with a smooth multiplicative subgroup

Roadmap for Pari

1. Design a univariate EPC scheme

2. Design a univariate PIOP for R1CS rowcheck

3. Compile the PIOP with EPC

4. Optimize to reduce proof size

Roadmap for Pari

1. Design a univariate EPC scheme
2. Design a univariate PIOP for R1CS rowcheck
3. Compile the PIOP with EPC
4. Optimize to reduce proof size

Result of Compilation

Result of Compilation

After compiling the Rowcheck with univariate EPC We achieve a SNARK with the proof size $|\pi| =$

Result of Compilation

After compiling the Rowcheck with univariate EPC We achieve a SNARK with the proof size $|\pi| =$

$$4\mathbb{G}$$

(opening proofs for polynomials $\hat{z}_A, \hat{z}_B, \hat{z}_C, q$)

Result of Compilation

After compiling the Rowcheck with univariate EPC We achieve a SNARK with the proof size $|\pi| =$

$$4\mathbb{G}$$

(opening proofs for polynomials $\hat{z}_A, \hat{z}_B, \hat{z}_C, q$)

+

$$4\mathbb{F}$$

for the evaluations: v_A, v_B, v_C, v_q

Result of Compilation

After compiling the Rowcheck with univariate EPC We achieve a SNARK with the proof size $|\pi| =$

$$5\mathbb{G}$$

(commitment to polynomials $\hat{z}_A, \hat{z}_B, \hat{z}_C, q$ + consistency commitment)

$$4\mathbb{G}$$

(opening proofs for polynomials $\hat{z}_A, \hat{z}_B, \hat{z}_C, q$)

+

$$4\mathbb{F}$$

for the evaluations: v_A, v_B, v_C, v_q

Result of Compilation

After compiling the Rowcheck with univariate EPC We achieve a SNARK with the proof size $|\pi| =$

$$5\mathbb{G}$$

(commitment to polynomials $\hat{z}_A, \hat{z}_B, \hat{z}_C, q$ + consistency commitment)

+

$$4\mathbb{G}$$

(opening proofs for polynomials $\hat{z}_A, \hat{z}_B, \hat{z}_C, q$)

+

$$4\mathbb{F}$$

for the evaluations: v_A, v_B, v_C, v_q

Roadmap for Pari

1. Design a univariate EPC scheme
2. Design a univariate PIOP for R1CS rowcheck
3. Compile the PIOP with EPC
4. Optimize to reduce proof size

Roadmap for Pari

1. Design a univariate EPC scheme
2. Design a univariate PIOP for R1CS rowcheck
3. Compile the PIOP with EPC
4. Optimize to reduce proof size

Optimizations to reduce proof size

Optimizations to reduce proof size

1. We use batch commitment and batch opening for EPC which reduces the number of group elements to $2\mathbb{G}_1$

Optimizations to reduce proof size

1. We use batch commitment and batch opening for EPC which reduces the number of group elements to $2\mathbb{G}_1$
2. We use Square R1CS (SR1CS) [GM17] as the NP-Complete language, which checks

$$(Az)^2 - Bz = 0$$

to only send v_A, v_B, v_q , which reduces the number of field elements to $3\mathbb{F}$

Optimizations to reduce proof size

1. We use batch commitment and batch opening for EPC which reduces the number of group elements to $2\mathbb{G}_1$
2. We use Square R1CS (SR1CS) [GM17] as the NP-Complete language, which checks

$$(Az)^2 - Bz = 0$$

   to only send v_A, v_B, v_q , which reduces the number of field elements to $3\mathbb{F}$
3. We can also avoid sending v_q because $v_A^2 - v_B = v_q v_t$ if and only if $v_q = (v_A^2 - v_B)/v_t$ and so the verifier can compute it from v_a, v_b

Optimizations to reduce proof size

1. We use batch commitment and batch opening for EPC which reduces the number of group elements to $2\mathbb{G}_1$
2. We use Square R1CS (SR1CS) [GM17] as the NP-Complete language, which checks

$$(Az)^2 - Bz = 0$$

   to only send v_A, v_B, v_q , which reduces the number of field elements to $3\mathbb{F}$

3. We can also avoid sending v_q because $v_A^2 - v_B = v_q v_t$ if and only if $v_q = (v_A^2 - v_B)/v_t$ and so the verifier can compute it from v_a, v_b

Hence: $|\pi| = 2\mathbb{G} + 2\mathbb{F}$

Garuda

Diff of Pari and Garuda

Pari	Garuda
Square R1CS (SR1CS)	Generalized R1CS (GR1CS)
Univariate EPC (Batched)	Multivariate EPC (Non-Batched)
Univariate Rowcheck PIOP	Multivariate Rowcheck PIOP Using sumcheck protocol

Diff of Pari and Garuda

Pari	Garuda
Square R1CS (SR1CS)	Generalized R1CS (GR1CS)
Univariate EPC (Batched)	Multivariate EPC (Non-Batched)
Univariate Rowcheck PIOP	Multivariate Rowcheck PIOP Using sumcheck protocol

Support for any custom gates, e.g. Lookups

Diff of Pari and Garuda

Pari	Garuda
Square R1CS (SR1CS)	Generalized R1CS (GR1CS)
Univariate EPC (Batched)	Multivariate EPC (Non-Batched)
Univariate Rowcheck PIOP	Multivariate Rowcheck PIOP Using sumcheck protocol

Support for any custom gates, e.g. Lookups

Free addition gates

Diff of Pari and Garuda

Pari	Garuda
Square R1CS (SR1CS)	Generalized R1CS (GR1CS)
Univariate EPC (Batched)	Multivariate EPC (Non-Batched)
Univariate Rowcheck PIOP	Multivariate Rowcheck PIOP Using sumcheck protocol

Support for any custom gates, e.g. Lookups

Free addition gates

Linear-time prover

Generalized R1CS (GR1CS)

Generalized R1CS (GR1CS)

R1CS: $z = (x, w)$ should satisfy $Az \circ Bz = Cz$

Sr1CS: $z = (x, w)$ should satisfy $(Az)^2 = Cz$

Generalized R1CS (GR1CS)

R1CS: $z = (x, w)$ should satisfy $Az \circ Bz = Cz$

Sr1CS: $z = (x, w)$ should satisfy $(Az)^2 = Cz$

We can extend this to an arbitrary expression of the form like:

- High degree gate: $Az \circ Bz \circ (Cz)^4 - (Dz)^3 \circ (Ez)^4 \circ Fz + 7 = 0$
- Lookup table: $\mathcal{T}(Az, Bz, Cz, Dz, ..) = 0$

Generalized R1CS (GR1CS)

R1CS: $z = (x, w)$ should satisfy $Az \circ Bz = Cz$

Sr1CS: $z = (x, w)$ should satisfy $(Az)^2 = Cz$

We can extend this to an arbitrary expression of the form like:

- High degree gate: $Az \circ Bz \circ (Cz)^4 - (Dz)^3 \circ (Ez)^4 \circ Fz + 7 = 0$
- Lookup table: $\mathcal{T}(Az, Bz, Cz, Dz, ..) = 0$

In general, a constraint system is satisfied if $z = (x, w)$ satisfies:

$$\mathcal{L}(M_1 z, M_2 z, \ldots, M_t z) = 0$$

Generalized R1CS (GR1CS)

R1CS: $z = (x, w)$ should satisfy $Az \circ Bz = Cz$

Sr1CS: $z = (x, w)$ should satisfy $(Az)^2 = Cz$

We can extend this to an arbitrary expression of the form like:

- High degree gate: $Az \circ Bz \circ (Cz)^4 - (Dz)^3 \circ (Ez)^4 \circ Fz + 7 = 0$
- Lookup table: $\mathcal{T}(Az, Bz, Cz, Dz, ..) = 0$

In general, a constraint system is satisfied if $z = (x, w)$ satisfies:

$$\mathscr{L}(M_1 z, M_2 z, \ldots, M_t z) = 0$$

A GR1CS instance is composed of local predicates

$$\mathscr{C} = \left(\mathscr{L}_i : \mathbb{F}^{t_i} \to \{0,1\}, (M_{i,1}, \ldots, M_{i,t_i}) \right)_{i \in [c]}$$

We say $\mathscr{C}$ is satisfied iff for all $i \in [c]$: $\mathscr{L}_i(M_{i,1} z, \ldots, M_{i,t_i}) = 0$

Grand multivariate zerocheck

Grand multivariate zerocheck

In GR1CS, we have c different local predicates:

Grand multivariate zerocheck

In GR1CS, we have c different local predicates:

$$\mathcal{L}_1 \to \quad M_{1,1} \quad , \ldots, \quad M_{1,t_1}$$

$$\vdots$$

$$\mathcal{L}_c \to \quad M_{c,1} \quad , \ldots, \quad M_{c,t_c}$$

Grand multivariate zerocheck

In GR1CS, we have c different local predicates:

$$\mathscr{L}_1 \to \quad M_{1,1} \quad , \ldots, \quad M_{1,t_1}$$

$$\vdots$$

$$\mathscr{L}_c \to \quad M_{c,1} \quad , \ldots, \quad M_{c,t_c}$$

We stack these matrices on top of each other

$$\mathscr{L}^* \to \quad M^*_{1,1} \quad , \ldots, \quad M^*_{1,t}$$

Grand multivariate zerocheck

In GR1CS, we have c different local predicates:

$$\mathscr{L}_1 \rightarrow \boxed{M_{1,1}}, \ldots, \boxed{M_{1,t_1}}$$

$$\vdots$$

$$\mathscr{L}_c \rightarrow \boxed{M_{c,1}}, \ldots, \boxed{M_{c,t_c}}$$

We stack these matrices on top of each other

$$\mathscr{L}^* \rightarrow M^*_{1,1}, \ldots, M^*_{1,t}$$

The Rowcheck PIOP checks the following grand multivariate zerocheck:

$$\mathscr{L}^*(M^*_{1,1}z, \ldots, M^*_{1,t}z)$$

$$= S_1 \cdot \mathscr{L}_1(M^*_{1,1}z, \ldots, M^*_{1,t}z) + S_c \cdot \mathscr{L}_c(M^*_{1,1}z, \ldots, M^*_{1,t}z) = 0$$

Grand multivariate zerocheck

In GR1CS, we have c different local predicates:

$$\begin{array}{lccc} \mathscr{L}_1 \rightarrow & M_{1,1} & ,\ldots, & M_{1,t_1} \\ & \vdots & & \\ \mathscr{L}_c \rightarrow & M_{c,1} & ,\ldots, & M_{c,t_c} \end{array}$$

We stack these matrices on top of each other

$$\mathscr{L}^* \rightarrow \quad M^*_{1,1} \quad ,\ldots, \quad M^*_{1,t}$$

The Rowcheck PIOP checks the following grand multivariate zerocheck:

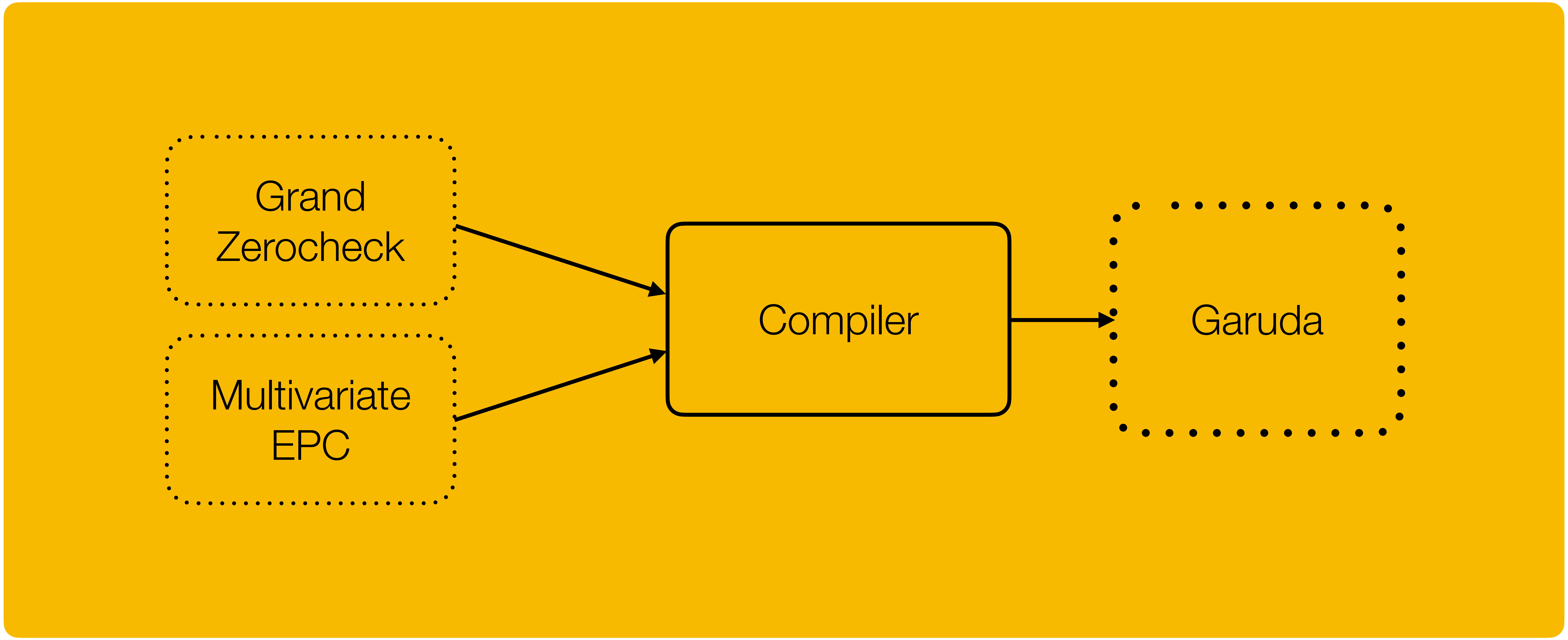
$$\mathscr{L}^*(M^*_{1,1}z, \ldots, M^*_{1,t}z)$$

$$= S_1 \cdot \mathscr{L}_1(M^*_{1,1}z, \ldots, M^*_{1,t}z) + S_c \cdot \mathscr{L}_c(M^*_{1,1}z, \ldots, M^*_{1,t}z) = 0$$

Selector for the 1st predicate

Selector for the c-th predicate

Garuda



$$= S_1 \cdot \mathscr{L}_1(M^*_{1,1}z, \ldots, M^*_{1,t}z) + S_c \cdot \mathscr{L}_c(M^*_{1,1}z, \ldots, M^*_{1,t}z) = 0$$

Selector for the 1st predicate

Selector for the c-th predicate

Thanks!

Open questions

- Our EPC constructions imply circuit-specific setup

  Q: can we construct EPC schemes that achieve universal setup?
- What other applications of EPC schemes can we find?

  Ideas: Verifiable Secret Sharing, Accumulators, etc?
- Our SNARKs don't achieve ZK.

  Q: How can we demonstrate ZK without increasing the proof size?

Thanks!

ePrint: *https://eprint.iacr.org/2024/1245*

Open questions

- Our EPC constructions imply circuit-specific setup

  Q: can we construct EPC schemes that achieve universal setup?
- What other applications of EPC schemes can we find?

  Ideas: Verifiable Secret Sharing, Accumulators, etc?
- Our SNARKs don't achieve ZK.

  Q: How can we demonstrate ZK without increasing the proof size?