History of Succinct Arguments

Nicholas Ward

zkSNARKs

zero **k**nowledge

succinct

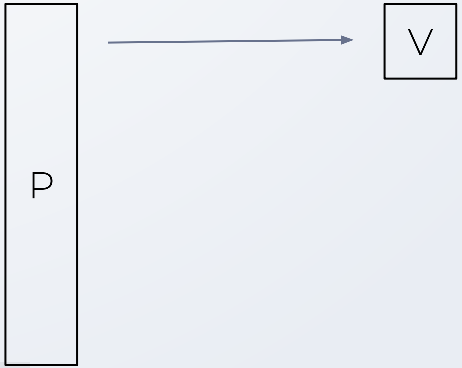
non-interactive

argument

of **k**nowledge

Why?

Delegation of Computation



History of Zero Knowledge

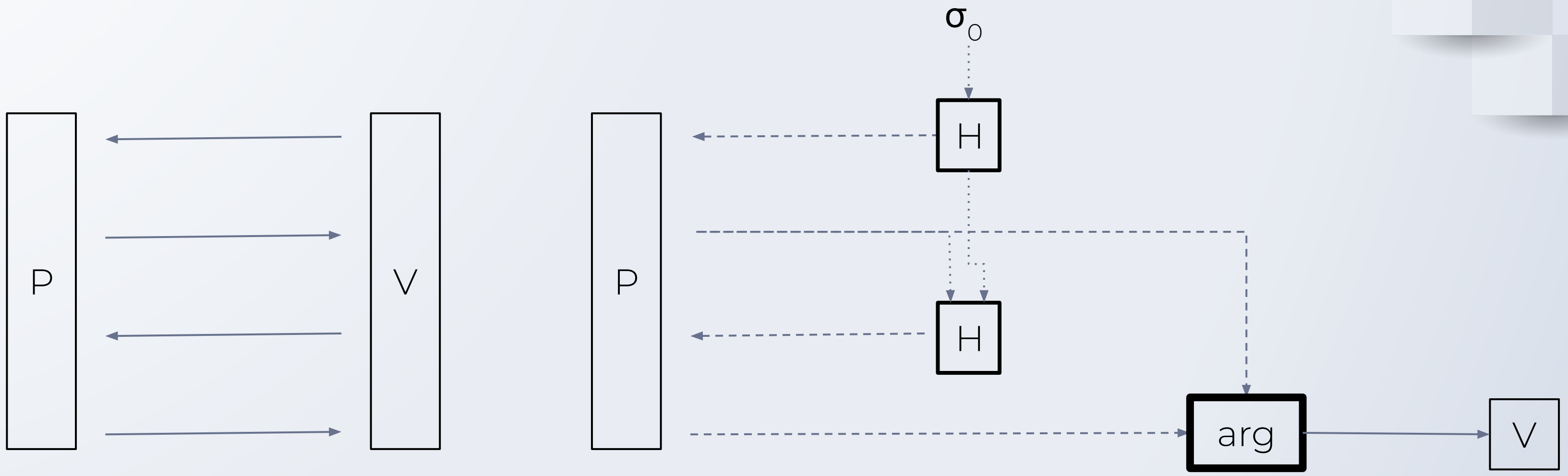
[GMR85]: introduced ZKP, with its simulator-based definition, and gave an example

[GKR89]: gave ZKP for an NP-complete problem (3-colorability), and thus for all of NP

History of Non-Interactive Arguments

[FS87]: generically turn public-coin IP into non-interactive proof by generating verifier's next queries from input and conversation so far

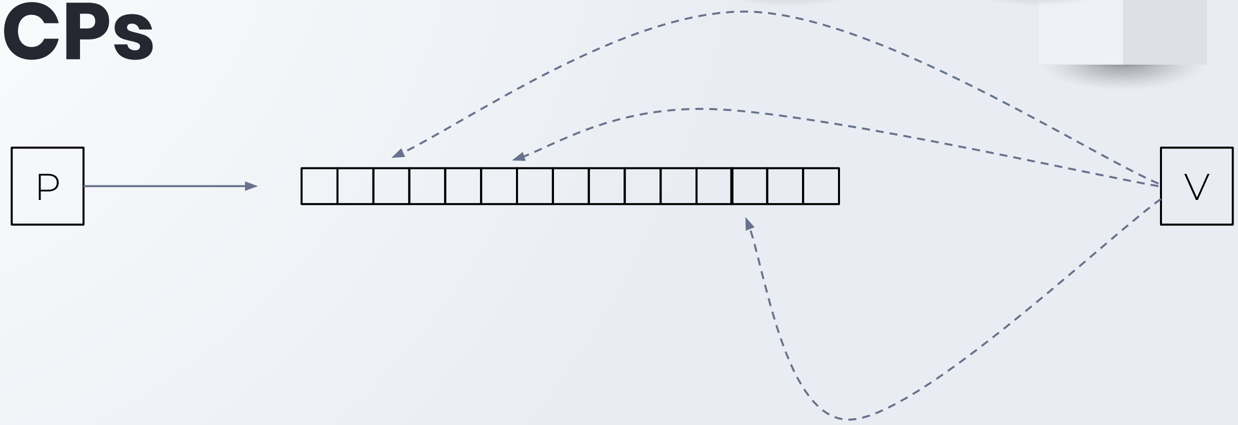
History of Non-Interactive Arguments



(secure assuming *random oracle*, but heuristically valid)

History of Succinct Arguments

PCPs



The PCP Theorem

$NP \subseteq PCP[\underbrace{O(\log n)}_{\text{Randomnes}}, \underbrace{O(1)}_{\text{Queries}}]$

& easy to add ZK!

From PCP to Succinct Argument

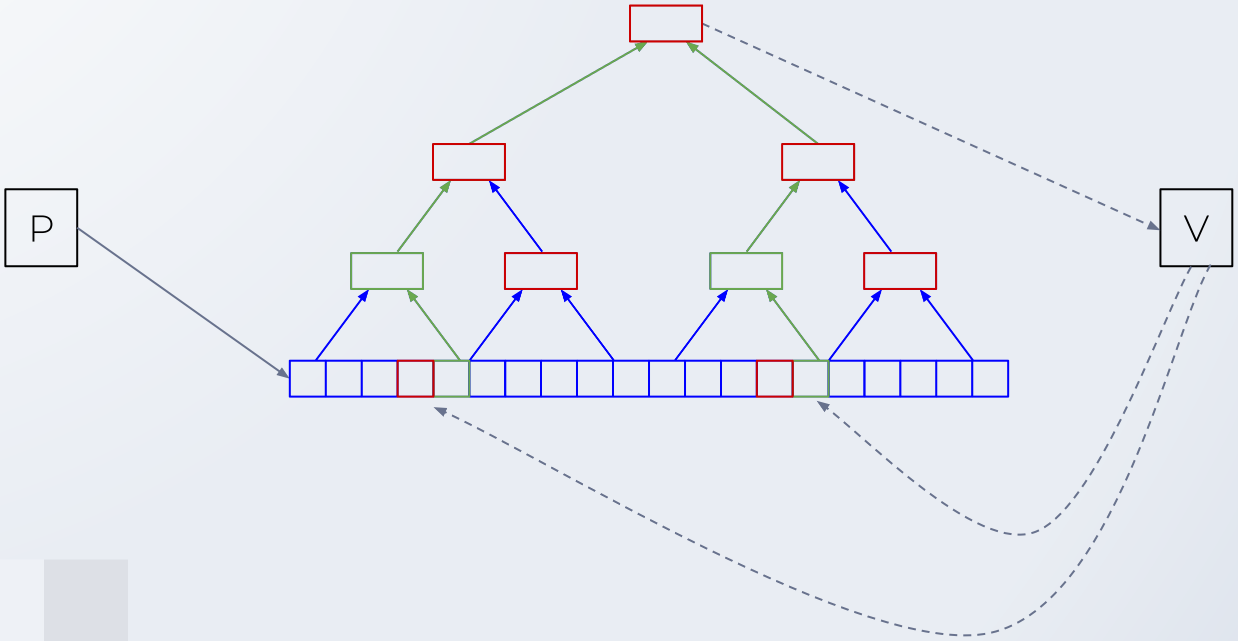
Send entire PCP? Not succinct!

Verifier sends query locations? Easy for prover to cheat!

From PCP to Succinct Argument

[Kil93, Mic94]: Encode PCP over Merkle tree

From PCP to Succinct Argument

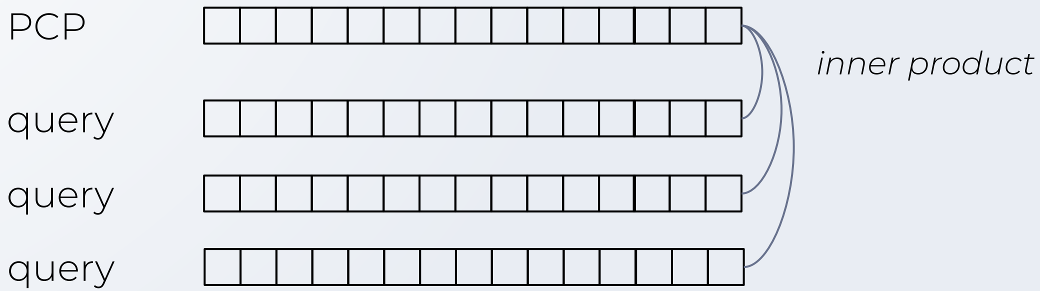


From PCP to Succinct Argument

[Kil93, Mic94]: Encode PCP over Merkle tree

Gives good asymptotics, but *bad* practical efficiency

Linear PCPs



Achieved by *moving to the exponent* of a group with hard discrete logs

Requires shared *structured reference string*, involving *trusted setup*

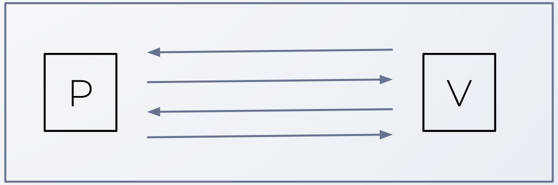
Introduced in [IKO07], made efficient in [GGPR13] using *pairings*

Linear PCPs

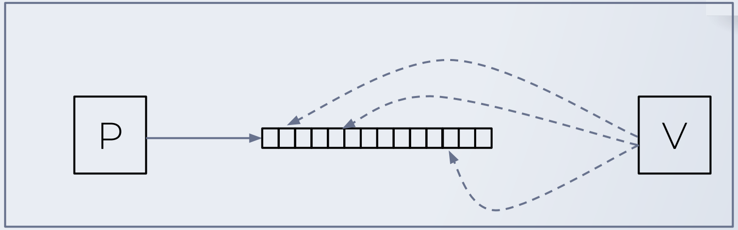
- Computation
- Algebraic Circuit
- R1CS
- QAP
- Linear PCP
- zkSNARK

IOPs

IP

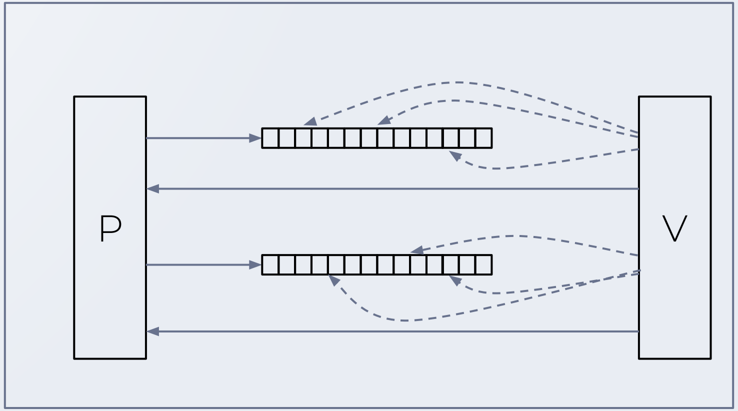


PCP



IOPs

IOP



Linear IOPs

IOP where each PCP is linear

[GKR08] & protocols based off it

Polynomial IOPs

Special case of linear IOP:

PCP is *coefficients*

Query is of the form

1	z	z^2	z^3	z^4	$\cdots$	z^n

STARK

DARK

PLONK

Marlin!

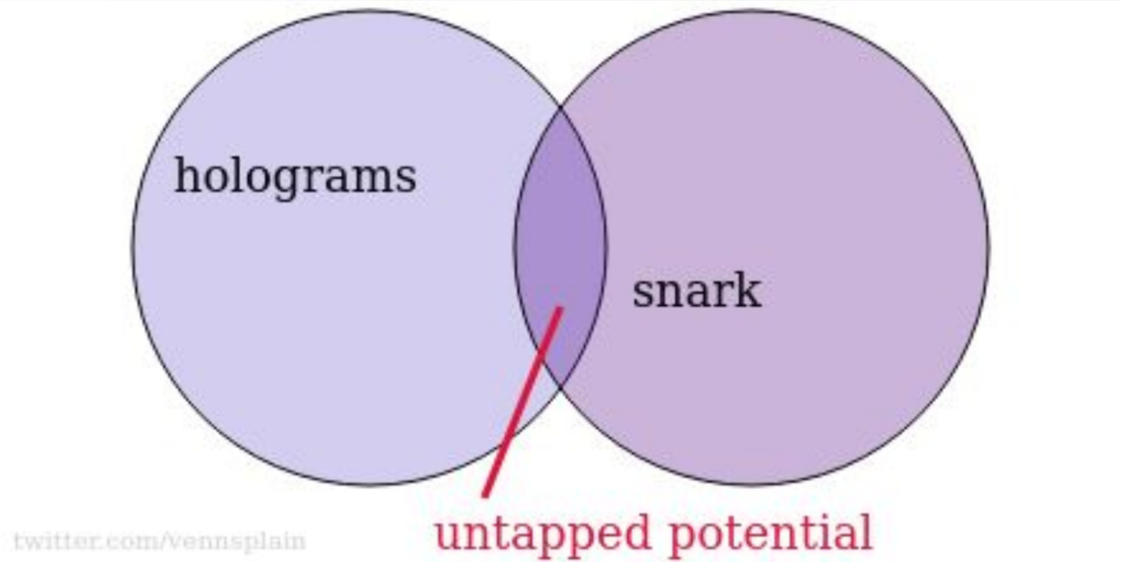
Polynomial IOPs

- Computation
- Algebraic Circuit
- R1CS
- Polynomial IOP
- *(using polynomial commitments)*
- zkSNARK

MARLIN: Preprocessing zkSNARKs with Universal and Updatable Setup

Alessandro Chiesa	UC Berkeley
Yuncong Hu	UC Berkeley
Mary Maller	University College London
Pratyush Mishra	UC Berkeley
Noah Vesely	University College London
Nicholas Ward	UC Berkeley

https://erint.iacr.org/2019/1047


hologram
snark
untapped potential
twitter.com/vennsplain

Preprocessing zkSNARKs with circuit-specific setup

$$C(x, w) = 1$$

Circuit

Public Input

Private Input

Argument system for CSAT

$\textsc{Setup}(1^\lambda, C) \to (\mathsf{pk}_C, \mathsf{vk}_C)$

$\textsc{Prove}(\mathsf{pk}_C, x, w) \to \pi$

$\textsc{Verify}(\mathsf{vk}_C, x, \pi) \to b$

Problem: new setup for *every* circuit; to be trustworthy, this requires a *global* MPC

Goal: universal setup

Universal Trusted Setup:
$\textsc{USetup}(1^\lambda, N) \rightarrow (\mathrm{upk}, \mathrm{uvk})$

Circuit-specific deterministic preprocessing:

$\textsc{CProcess}(\mathrm{upk}, C_1) \downarrow (\mathrm{cpk}_1, \mathrm{cvk}_1)$	$\textsc{CProcess}(\mathrm{upk}, C_2) \downarrow (\mathrm{cpk}_2, \mathrm{cvk}_2)$	$\textsc{CProcess}(\mathrm{upk}, C_3) \downarrow (\mathrm{cpk}_3, \mathrm{cvk}_3)$

$\textsc{Prove}(\mathrm{cpk}_2, x, w) \rightarrow \pi$

$\textsc{Verify}(\mathrm{cvk}_2, x, \pi) \rightarrow b$

Goal: *updatable* setup

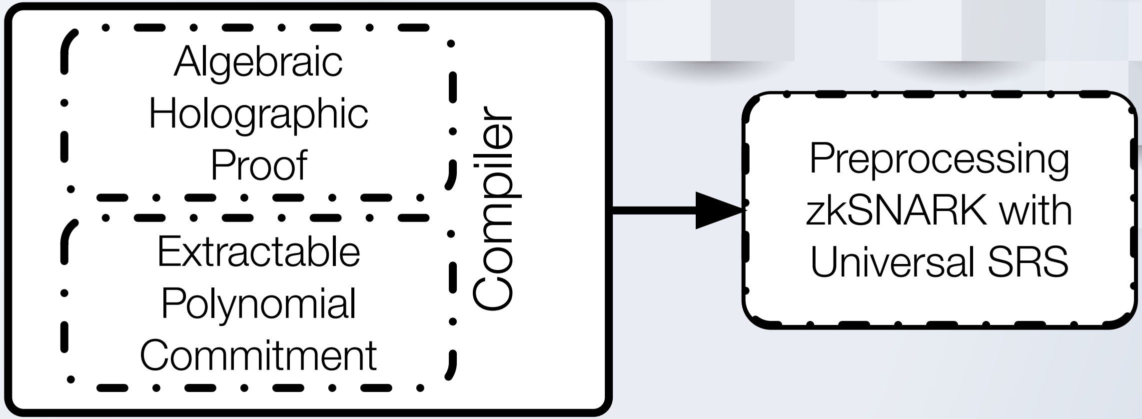
Initial Setup:
$\textsc{Setup}(1^\lambda) \rightarrow (\mathbf{srs}, \rho)$

Each update:
$\textsc{Update}(1^\lambda, \mathbf{srs}, (\rho_i)_{i=1,..,n}) \rightarrow (\mathbf{srs'}, \rho')$

Verification:
$\textsc{Verify}(1^\lambda, \mathbf{srs}, (\rho_i)_{i=1,..,n}) \rightarrow b$

Contributions

1. Methodology



2. Efficient ingredients for an efficient SNARK

AHP for R1CS

Extended KZG10 Scheme

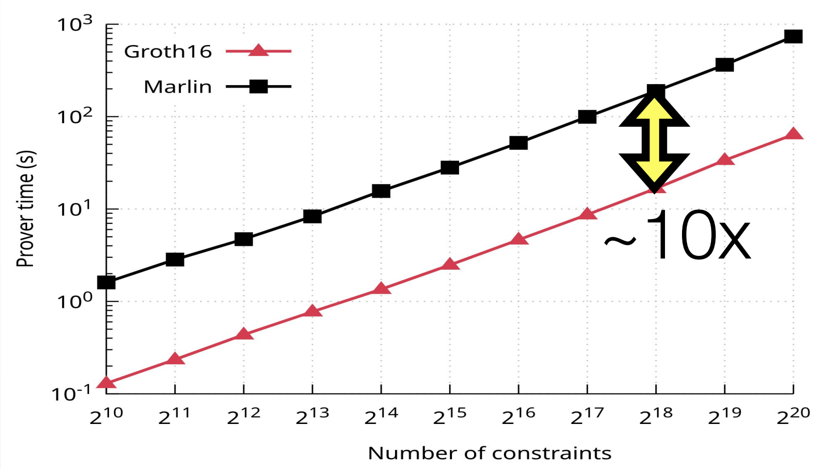
3. Rust implementation

https://github.com/scipr-lab/marlin

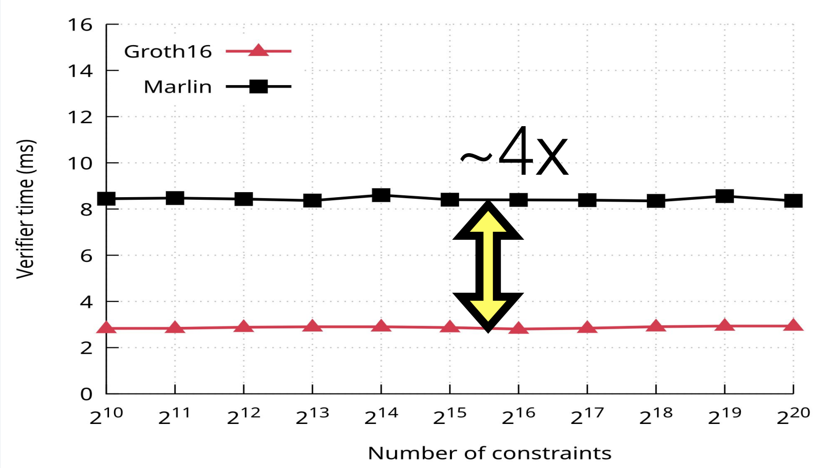
Evaluation over BLS12-381

Proof size: Marlin: 1296B
Groth16: 192B

Prover time



Verifier time



Concurrent Work:

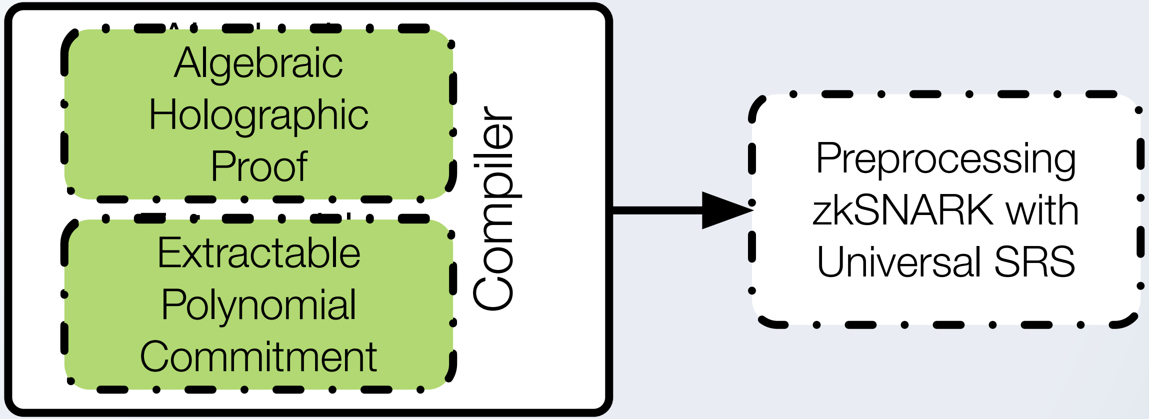
Marlin: good for R1CS

PLONK: good for CSAT

This Talk

1. Methodology

A. Provides a clean and straightforward way to construct preprocessing SNARKs
B. Shows that the key to achieving preprocessing is holography


Algebraic Holographic Proof
Extractable Polynomial Commitment
Compiler
Preprocessing zkSNARK with Universal SRS

A set of triples *(i, x, w)* satisfying a prescribed condition

Index | Public Input | Private Input

Example: Arithmetic Circuit Satisfaction (CSAT):

$$\left\{(C, x, w) \mid C(x, w) = 1\right\}$$

Index | Public Input | Private Input

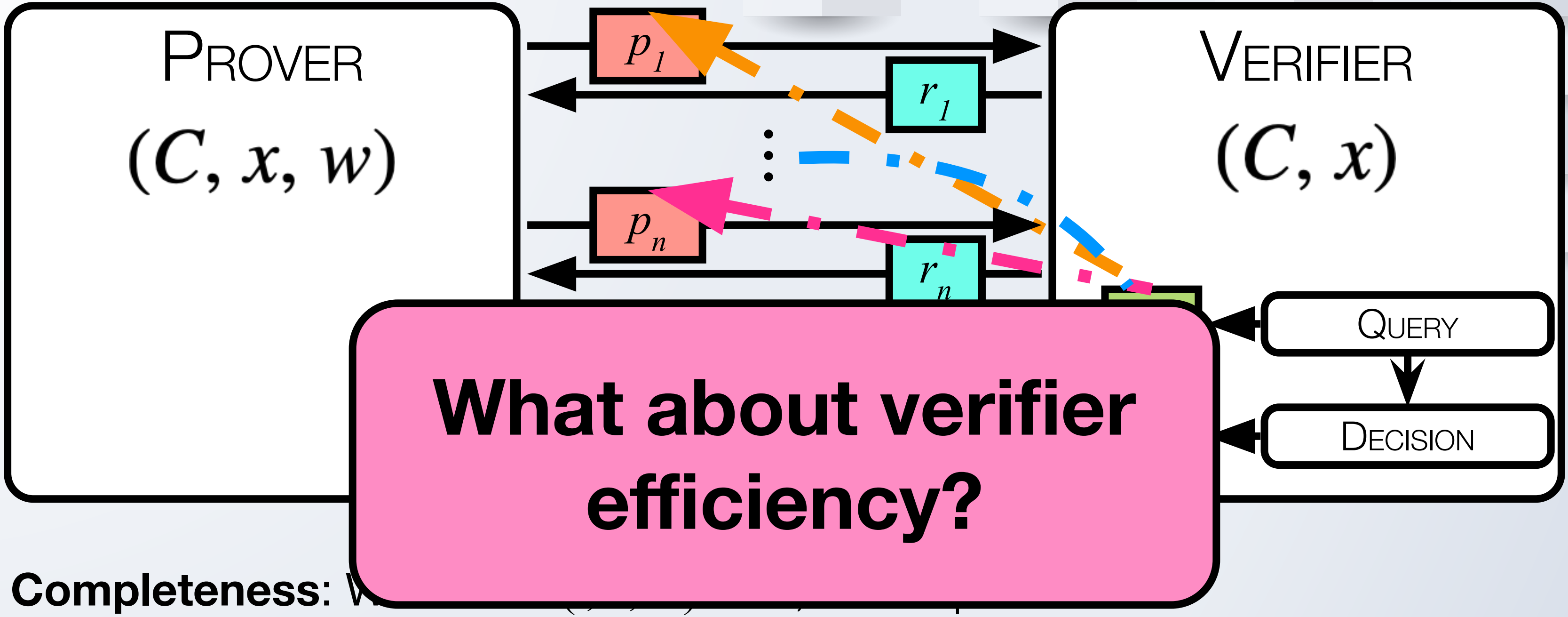
Example: Rank-1 Constraint System (R1CS):

$$\left\{((A, B, C), x, w) \mid Az \circ Bz = Cz\right\}$$

Index | Public Input | Private Input

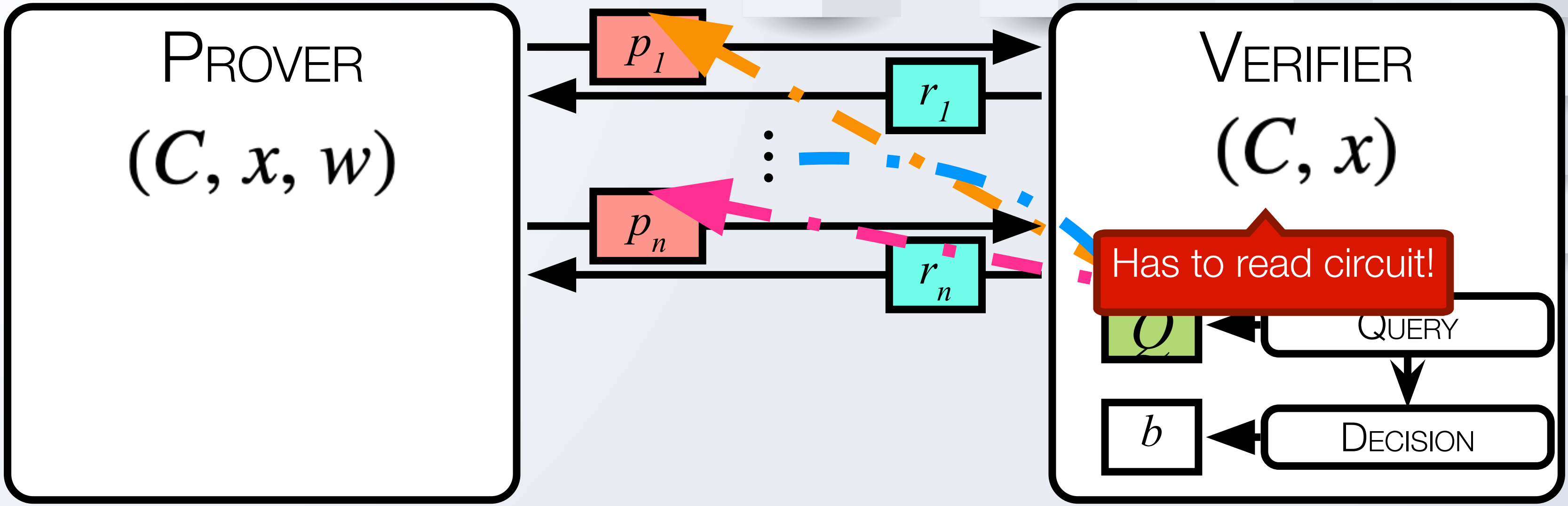
where $z = \begin{bmatrix} x \\ w \end{bmatrix}$

Algebraic ~~Holographic~~ Proofs



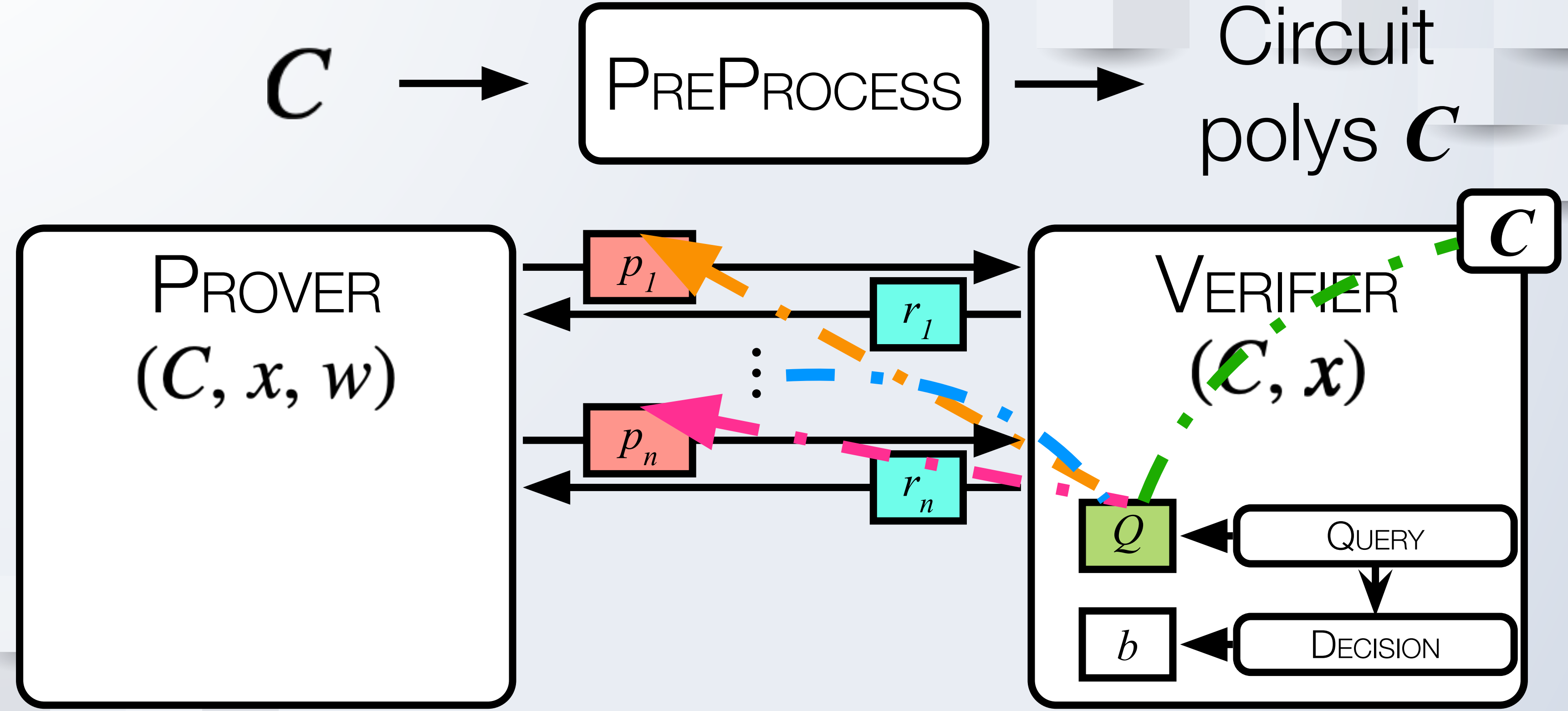
- **Completeness**: W
- **Proof of Knowledge**: Whenever **V** accepts, **P** "knows" w such that $(i, x, w) \in R$.
- **Bounded-query ZK**: Whenever $(i, x, w) \in R$, a verifier that makes up to b queries to polys learns nothing about w.

Problem: Verifier is linear in circuit!

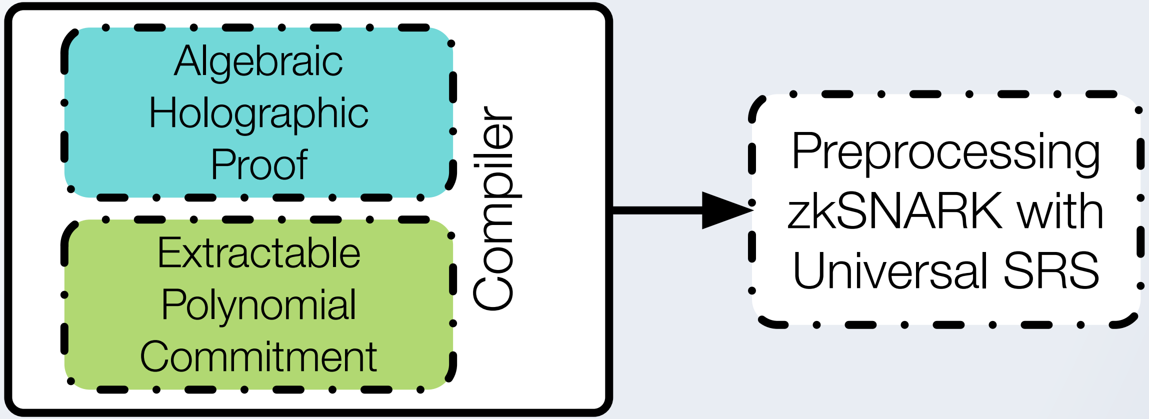


- When size of circuit << size of computation (like in machine computations), this is OK.
- When size of circuit = size of computation (like in CSAT/R1CS), this is bad!

Algebraic Holographic Proofs

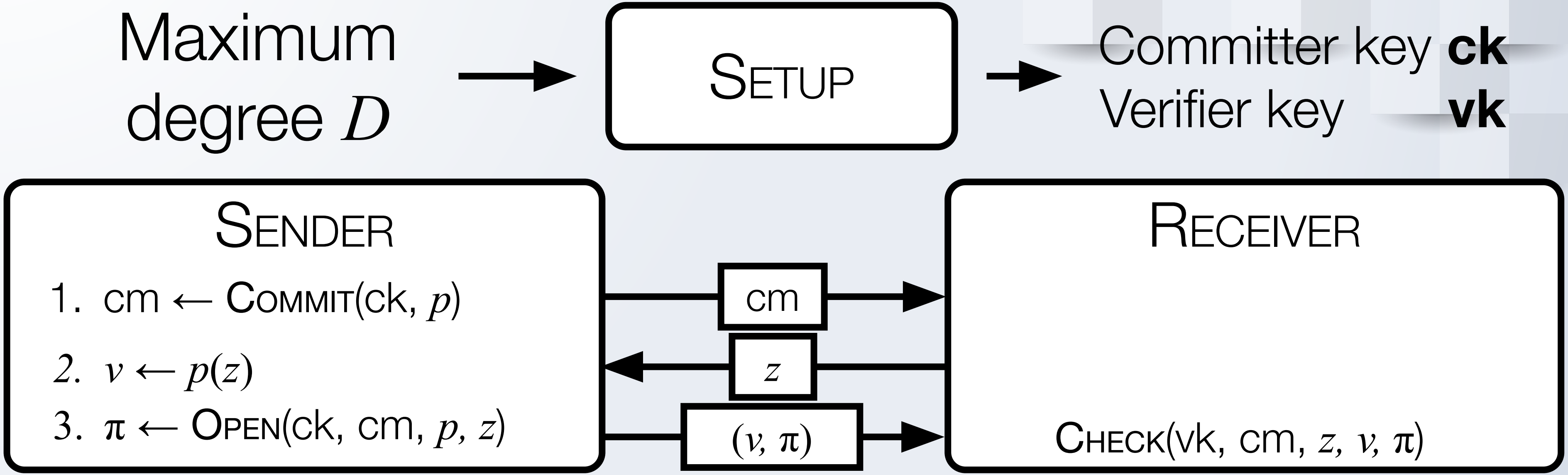


Verifier efficiency: $|x|$ +T(Interaction) + T(QUERY) + T(DECISION)



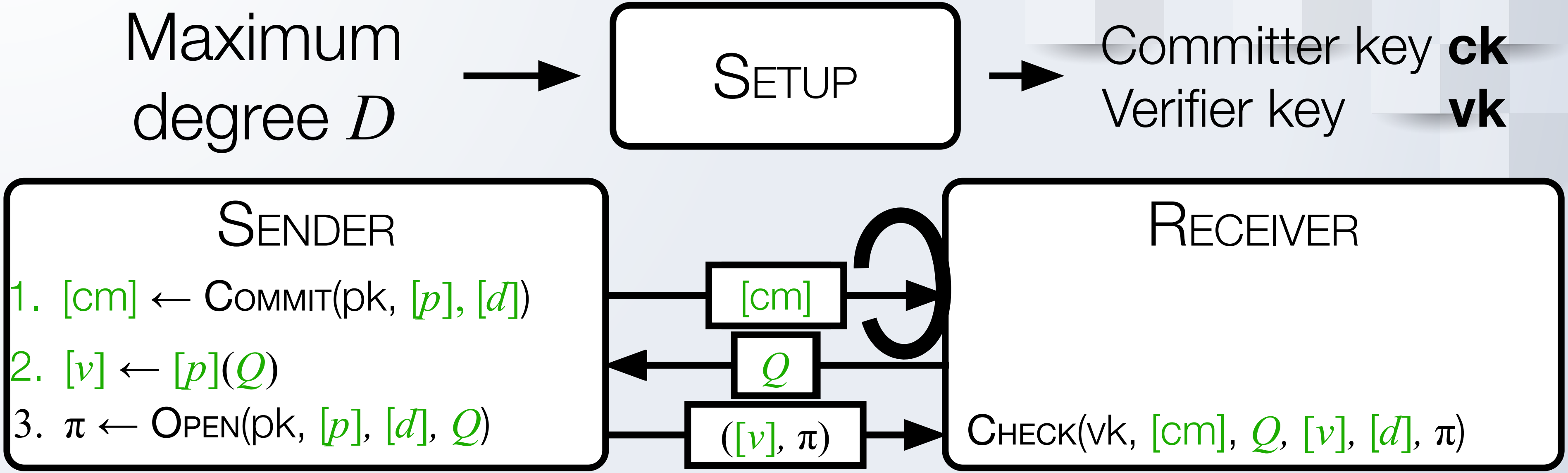
Algebraic Holographic Proof
Extractable Polynomial Commitment
Compiler
Preprocessing zkSNARK with Universal SRS

Polynomial Commitments



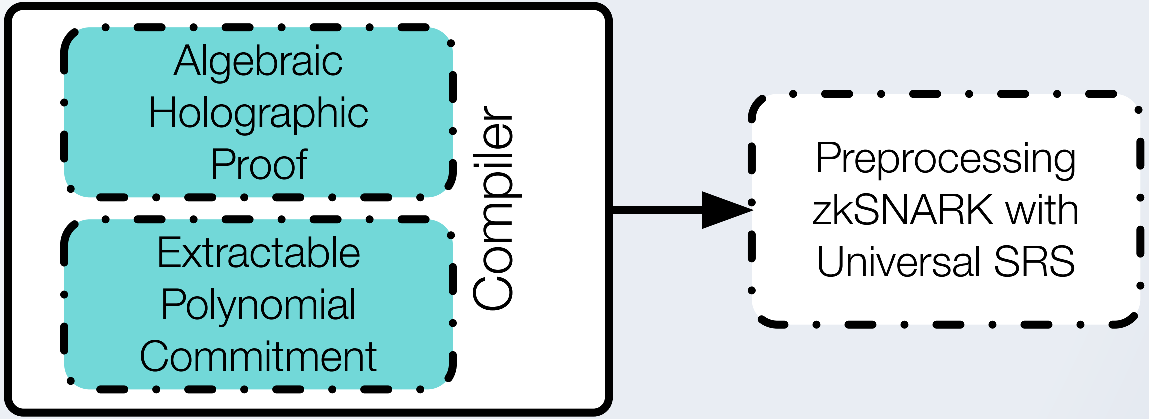
- **Completeness**: Whenever $p(z) = v$, **R** accepts.
- **Extractability**: Whenever **R** accepts, **S**’s commitment cm “contains” a polynomial p of degree at most D.
- **Hiding**: If **R** makes up to b queries, it learns nothing about p.

Polynomial Commitments



Our compiler needs more

- Batch commitment
- Multiple rounds
- Batch opening
- Per-poly degree bounds



Idea underlying compiler:

Holography ⇒ Preprocessing

Preprocessing zkSNARKs

$$\text{ARG.}\textsc{Setup}(1^\lambda, N) \to (\text{upk}, \text{uvk})$$

$$\text{ARG.}\textsc{Index}(\text{upk}, i) \to (\text{ipk}, \text{ivk})$$

$$\text{ARG.}\textsc{Prove}(\text{ipk}, x, w) \to \pi$$

$$\text{ARG.}\textsc{Verify}(\text{ivk}, x, \pi) \to b \in \{0, 1\}$$

- **Completeness**: Whenever $(i, x, w) \in R$, **V** accepts.
- **Proof of Knowledge**: Whenever **V** accepts, **P** “knows” w such that $(i, x, w) \in R$.
- **Zero Knowledge**: Whenever $(i, x, w) \in R$, **V** learns nothing about w.
- **Verifier efficiency**: $T(\mathbf{V}) = O(\log(|i|) + |x|)$

Universal Setup

$\text{ARG.S}\textsc{etup}(1^\lambda, N)$

1. Maximum degree D ← $\text{AHP}(N)$
2. Committer key **ck**, Verifier key **vk** ← $\text{PC.S}\textsc{etup}(D)$
3. Output
   - Universal prover key **upk = (ck, vk)**
   - Universal verifier key **uvk = vk**

Index-specific Setup

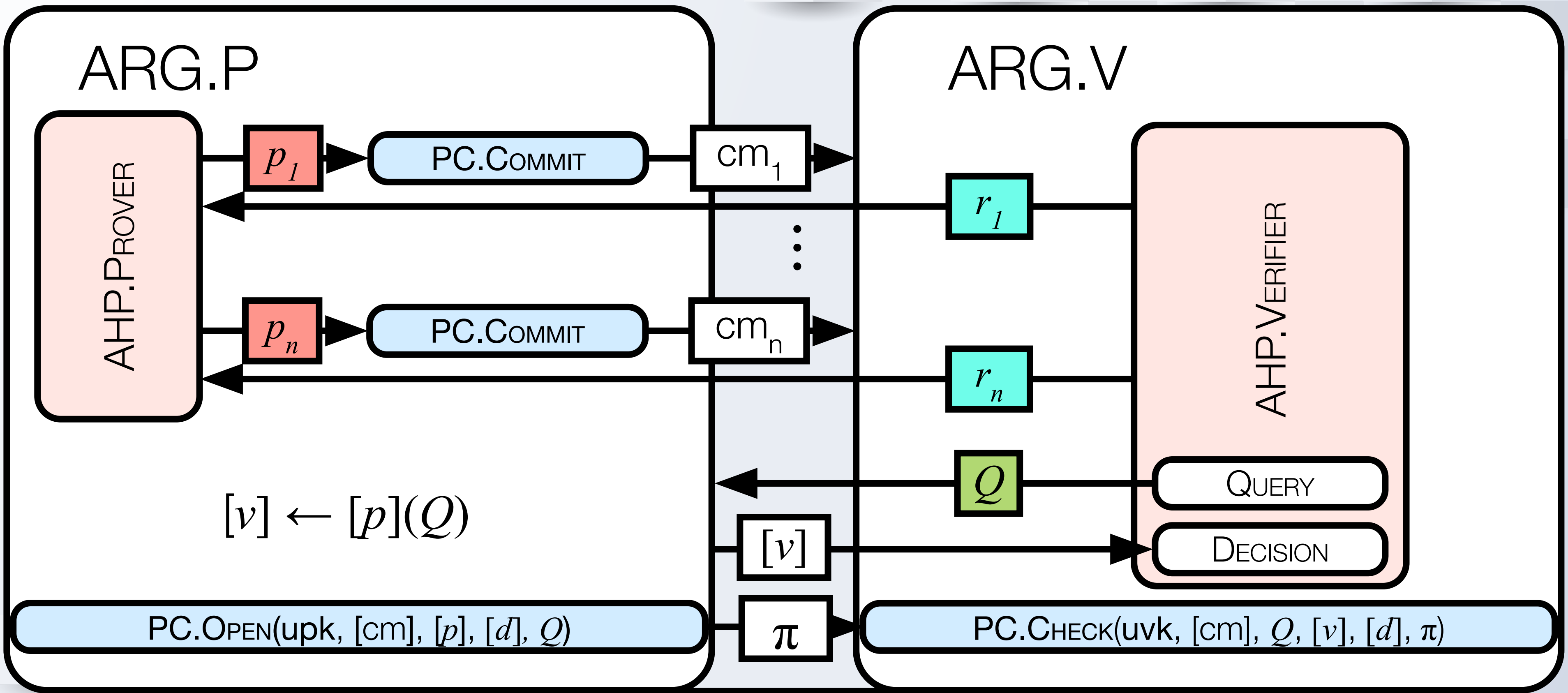
ARG.INDEXER(upk, i)

1. Index polys $\boldsymbol{I}$ ⟵ AHP.INDEXER(i)

2. Index comms. [cm] ⟵ PC.COMMIT(upk, I)

3. Output
   Index verifier key **ivk** = (**upk.uvk, [cm]**)
   Index prover key **ipk** = (**ivk, upk,** I)

Prove and Verify



+ Fiat-Shamir to get non-interactivity

Properties

- **Completeness**: Follows from completeness of **PC** and **AHP**.
- **Proof of Knowledge**: Whenever **ARG.Verify** accepts but $(i, x, w) \notin R$, we can construct either an adversarial prover against **AHP**, or an adversary that breaks extractability of **PC**.
- **Zero Knowledge**: Follows from hiding of **PC** and bounded-query ZK of **AHP**.
- **Verifier efficiency**: $T($**ARG.Verify**$) = T($**AHP.Verifier**$) + T($**PC.Check**$)$

Conclusion

In the talk:

algebraic holographic proof

+

extractable polynomial commitment scheme

into a

universal preprocessing zkSNARK

In the paper: **Efficient AHP for R1CS:**

- Protocol to evaluate low-degree extension for arbitrary R1CS matrices

Extending KZG10 to achieve:

- Extractability across multiple rounds
- Batch commitment and opening
- Individual degree bounds

Paper: https://eprint.iacr.org/2019/1047

Code: https://github.com/scipr-lab/marlin

[KZG10] Polynomial Commitments

Polynomial Commitments: Definition

PC.Setup(1^λ, degree bound **D**) → (committer key **ck**, receiver key **rk**)

PC.Commit(**ck**, polynomial **p**) → commitment **c**

PC.Open(**ck**, **p**, eval point **z**) → proof **π**

PC.Check(**rk**, **c**, **z**, claimed value **v**, **π**) → bit **b**

Polynomial Commitments: Security

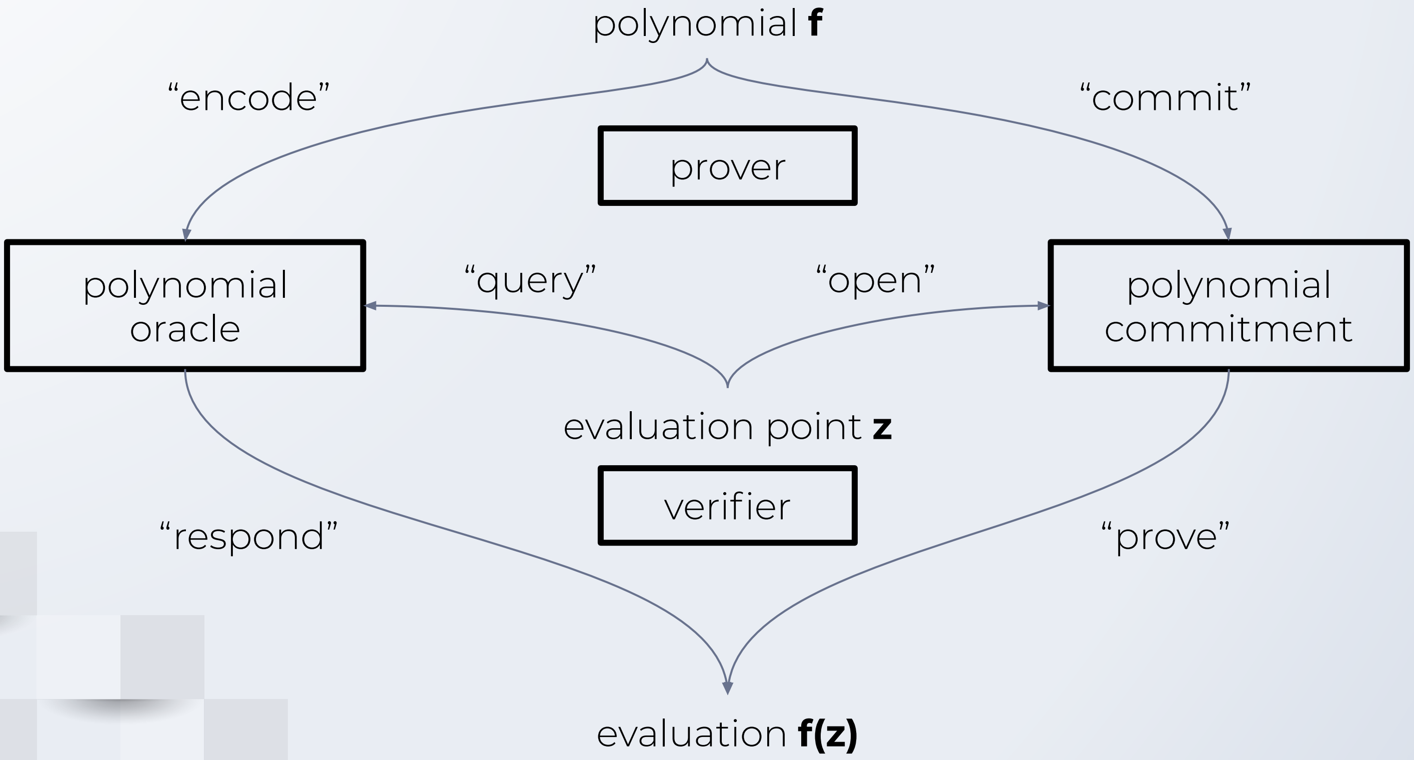
Completeness: if **v = p(z)**, then PC.Check outputs **1**

Extractability: anyone who produces a commitment **c** that cause PC.Check to accept “knows” a corresponding poly **p**

Succinctness: **c** and **π** sizes, PC.Check time independent of D

Hiding: commitment reveals no information about polynomial

What Are Polynomial Commitments?



A Wider View: Oracles & Primitives

oracle type	cryptographic primitive
set	accumulator
vector	vector commitment
low-degree	low-degree test
polynomial	polynomial commitment
inner product	inner product argument

more expressive

more expensive

Kate et al. Polynomial Commitments

Setup(1^λ, **D**):

- computes groups $\mathbf{G}$, $\mathbf{G_T}$ of prime order p with pairing $\mathbf{e}$
- chooses generator $\mathbf{g} \in \mathbf{G}$, random $\boldsymbol{\alpha}$ from $\{1, \ldots, p\text{-}1\}$
- outputs $\mathbf{pk} = \mathbf{rk} = (\mathbf{g}, \boldsymbol{\alpha}\mathbf{g}, \boldsymbol{\alpha}^2\mathbf{g}, \ldots, \boldsymbol{\alpha}^t\mathbf{g})$

Kate et al. Polynomial Commitments

Commit(**ck**, **p**):

- outputs **c** = **p**(**α**)**g**, pulling monomials from **ck**

Kate et al. Polynomial Commitments

Open(**ck**, **p**, **z**):

- computes witness poly $\boldsymbol{\phi}(\mathbf{x}) := (\mathbf{p}(\mathbf{x}) - \mathbf{p}(\mathbf{z}))/(\mathbf{x} - \mathbf{z})$
- outputs proof $\boldsymbol{\pi} = \boldsymbol{\phi}(\boldsymbol{\alpha})\mathbf{g}$

witness poly because it shows the value for $\mathbf{p}(\mathbf{z})$ is correct

Kate et al. Polynomial Commitments

Check($\mathbf{rk}$, $\mathbf{c}$, $\mathbf{z}$, $\mathbf{v}$, $\boldsymbol{\pi}$):

- checks whether

$$\mathbf{e}(\mathbf{c}, \mathbf{g}) = \mathbf{e}(\boldsymbol{\pi}, (\boldsymbol{\alpha} - \mathbf{z})\mathbf{g})\ \mathbf{e}(\mathbf{g}, \mathbf{g})^{\mathbf{v}}$$

$$\mathbf{e}(\mathbf{c}, \mathbf{g}) = \mathbf{e}(\mathbf{p}(\boldsymbol{\alpha})\mathbf{g}, \mathbf{g}) = \mathbf{e}(\mathbf{g}, \mathbf{g})^{\mathbf{p}(\boldsymbol{\alpha})} = \mathbf{e}(\mathbf{g}, \mathbf{g})^{\boldsymbol{\phi}(\boldsymbol{\alpha})(\boldsymbol{\alpha}-\mathbf{z})+\mathbf{p}(\mathbf{z})}$$

$$= \mathbf{e}(\boldsymbol{\phi}(\boldsymbol{\alpha})\mathbf{g}, (\boldsymbol{\alpha}-\mathbf{z})\mathbf{g})\ \mathbf{e}(\mathbf{g}, \mathbf{g})^{\mathbf{p}(\mathbf{z})}$$

$$= \mathbf{e}(\boldsymbol{\pi}, (\boldsymbol{\alpha}-\mathbf{z})\mathbf{g})\ \mathbf{e}(\mathbf{g}, \mathbf{g})^{\mathbf{v}} \quad \text{if } \mathbf{c} = \mathbf{p}(\boldsymbol{\alpha})\mathbf{g},\ \mathbf{v} = \mathbf{p}(\mathbf{z}),\ \boldsymbol{\pi} = \boldsymbol{\phi}(\boldsymbol{\alpha})\mathbf{g}$$