Beyond Ligero and Brakedown:

Building a Fast Prover Based on List-Polynomial Commitments

Azam Soleimanian Bogdan Ursu

About Us

Staff Cryptography Researcher

Consensys, for 3 years

PhD in Cryptography

Cryptography Researcher

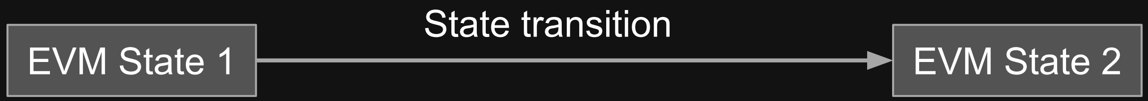
Linea, Prover Team for 1.5 years

PhD in Cryptography, ETH Zurich

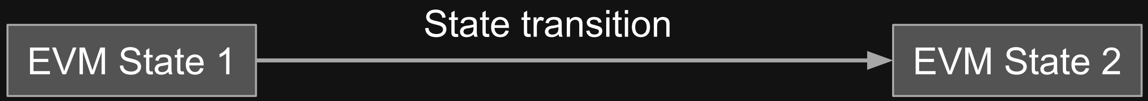
Outline

How to make proofs?

Outline



Outline



Layer 2: zero-knowledge rollups arithmetise the state transition, and compute a proof.

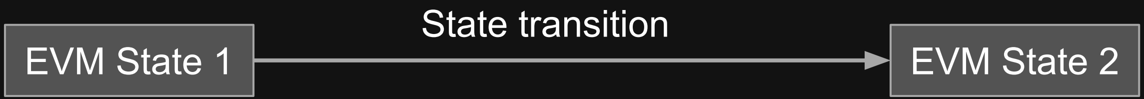
Outline

$$\text{EVM State 1} \xrightarrow{\text{State transition}} \text{EVM State 2}$$

Layer 2: zero-knowledge rollups arithmetise the state transition, and compute a proof.

Not all nodes need to perform the transitions themselves. Instead, they can just verify proofs (a much cheaper operation).

Outline



Layer 2: zero-knowledge rollups arithmetise the state transition, and compute a proof.

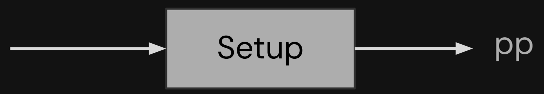
Not all nodes need to perform the transitions themselves. Instead, they can just verify proofs (a much cheaper operation).

How to construct such a proof system?

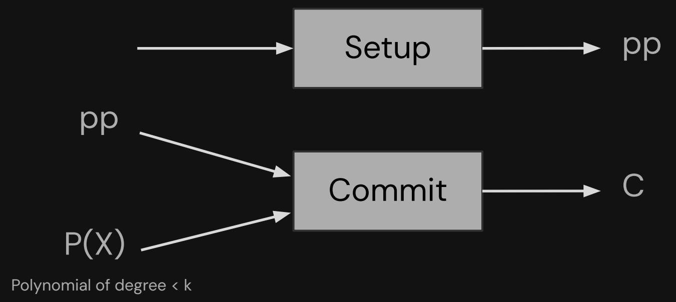
Outline

- Part 1: Our polynomial commitment scheme (Vortex)
- Part 2: From EVM execution to proof generation using Vortex and other components.

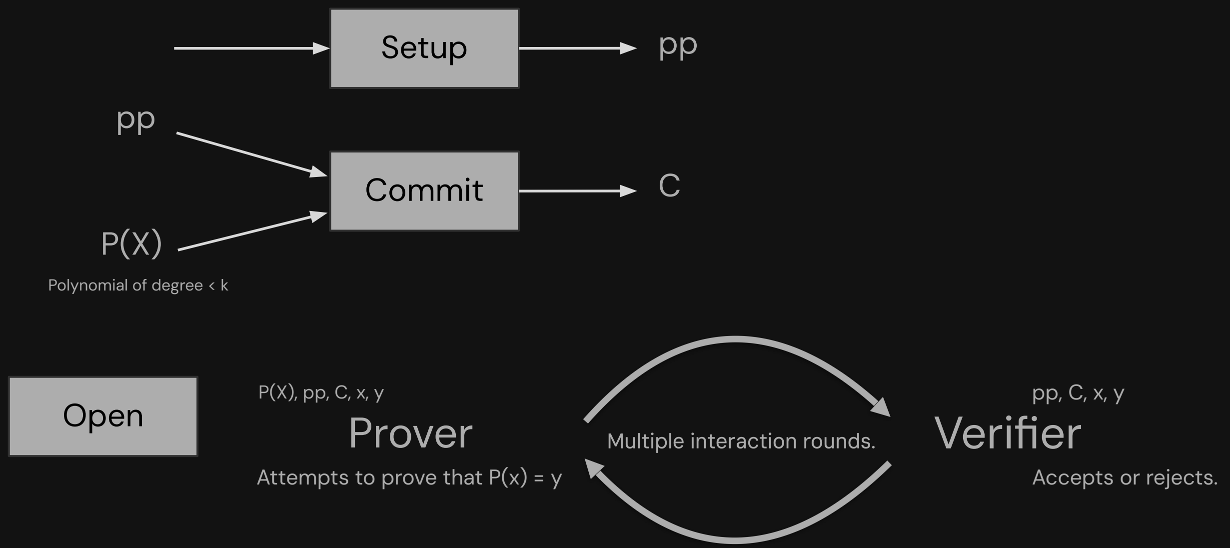
Polynomial Commitments



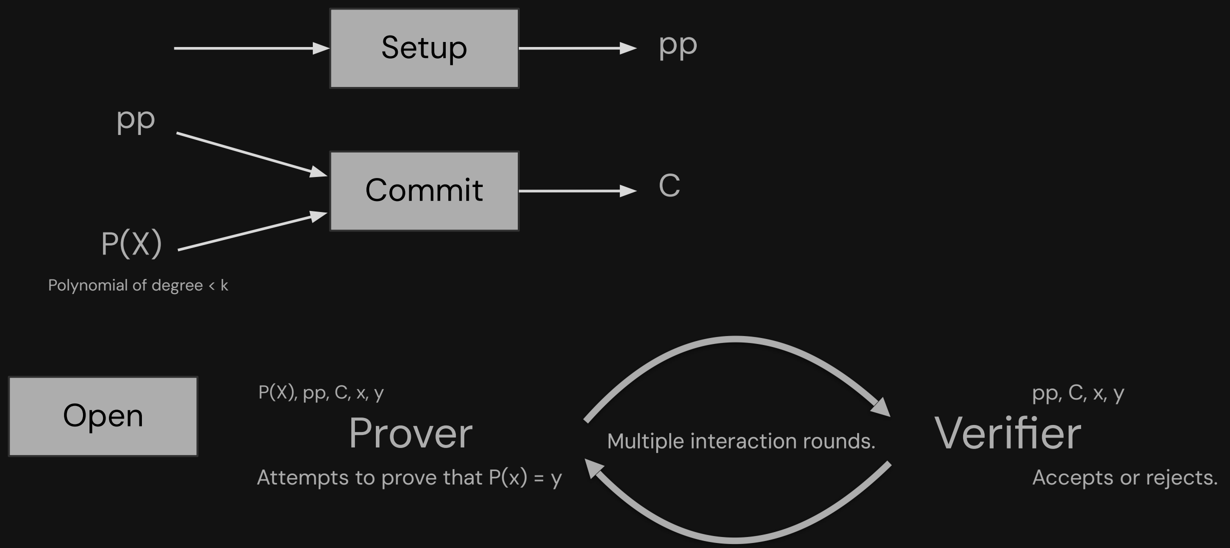
Polynomial Commitments



Polynomial Commitments

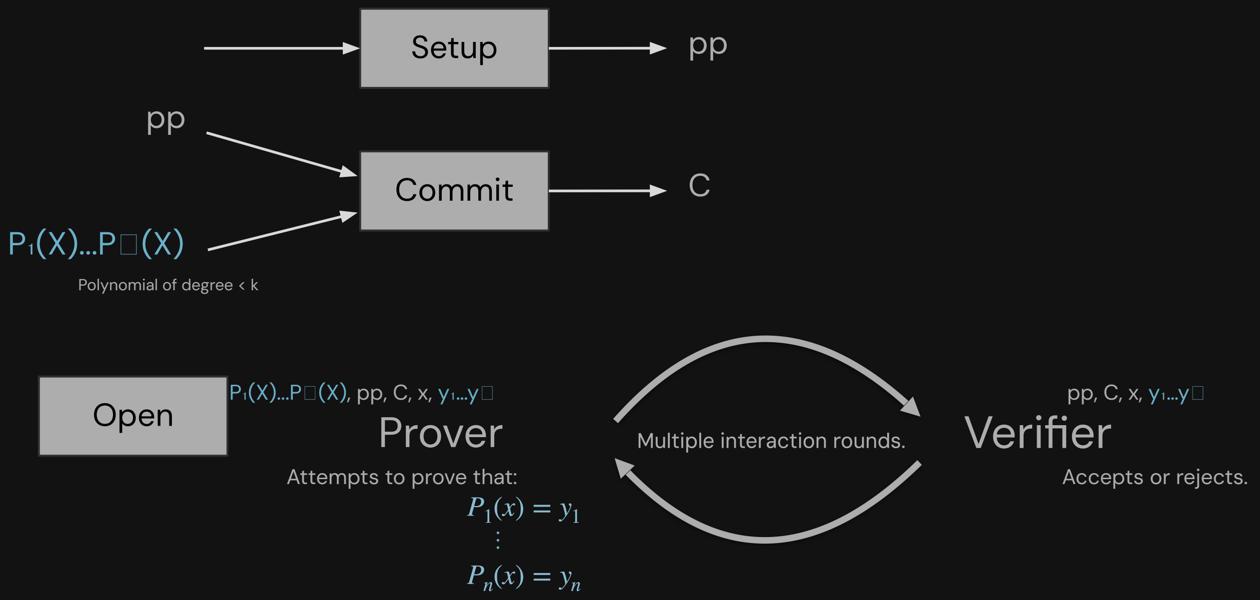


Security of Polynomial Commitments



The prover cannot convince the verifier in the case when $P(x) \neq y$.

Batched Polynomial Commitments



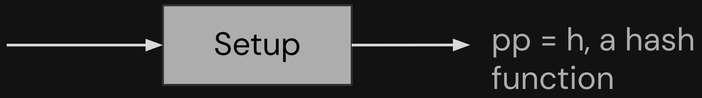
Reed-Solomon Codes

Finite field $\mathbb{F}$ and a subset of elements $a_1 \ldots a_n \in \mathbb{F}$.

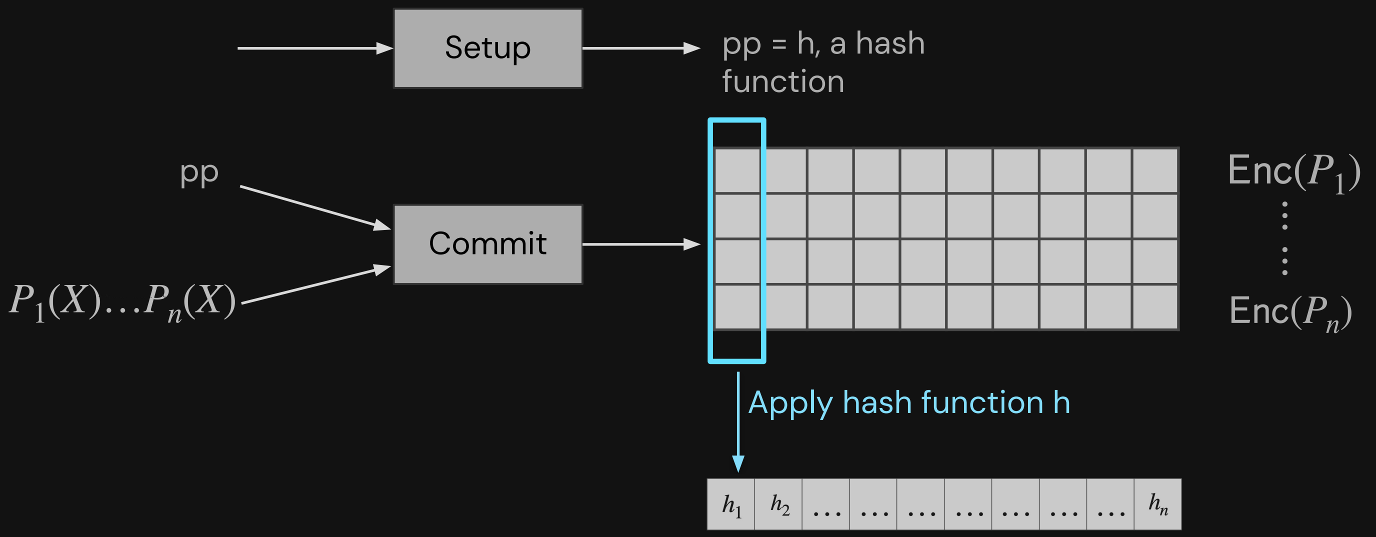
$$P(X) \longrightarrow \boxed{\text{Encode}} \longrightarrow P(a_1) \ldots P(a_n)$$

The degree of P must be ≤ n for the encoding to contain enough information about P.

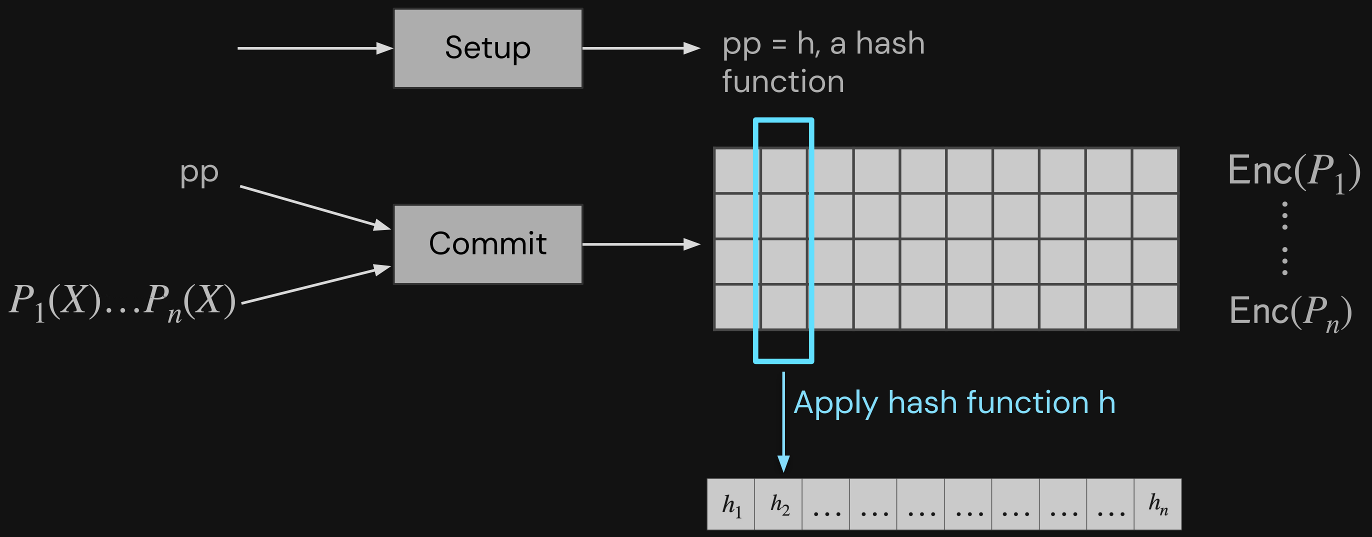
The Ligero/Breakdown Protocol



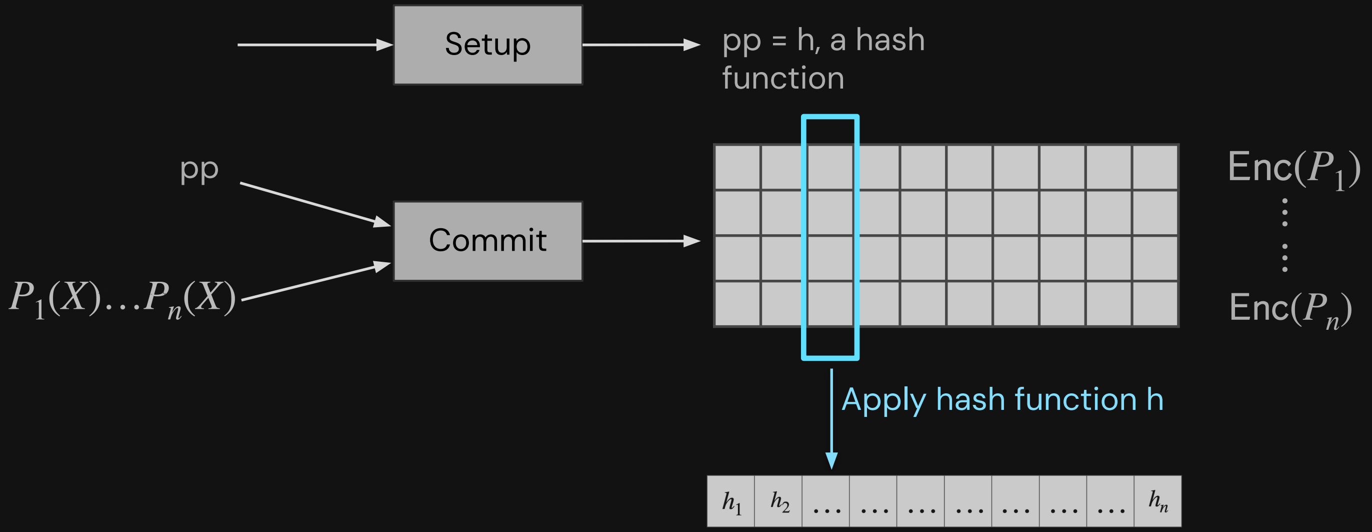
The Ligero/Breakdown Protocol



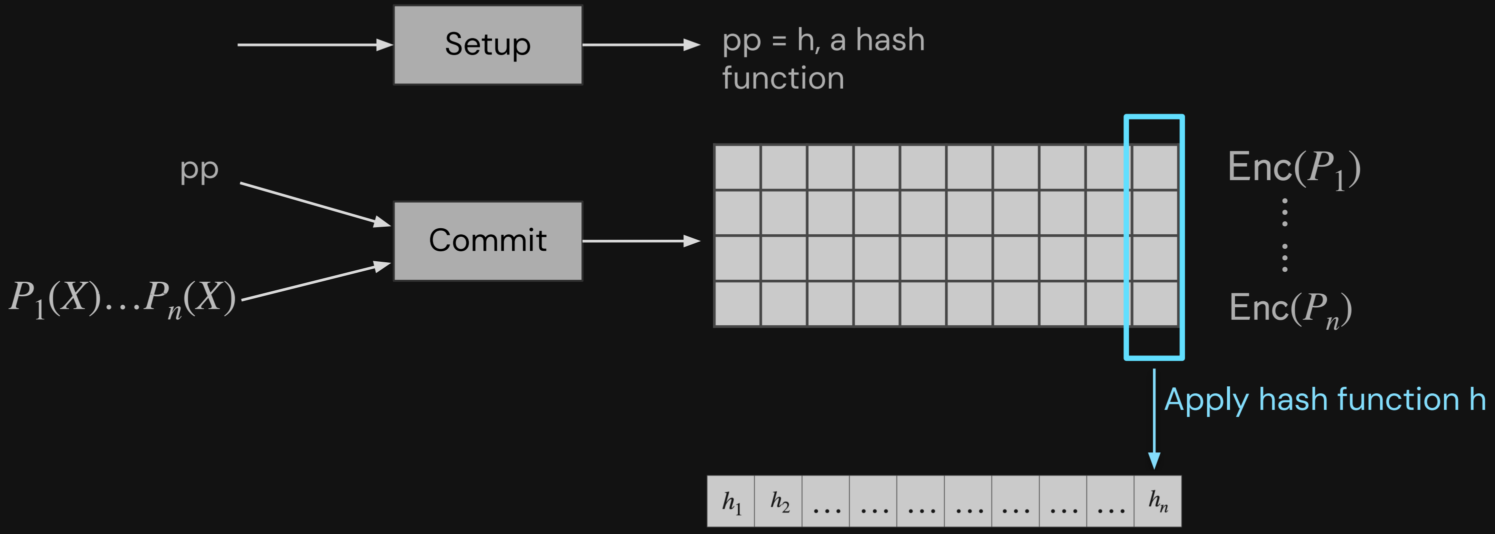
The Ligero/Breakdown Protocol



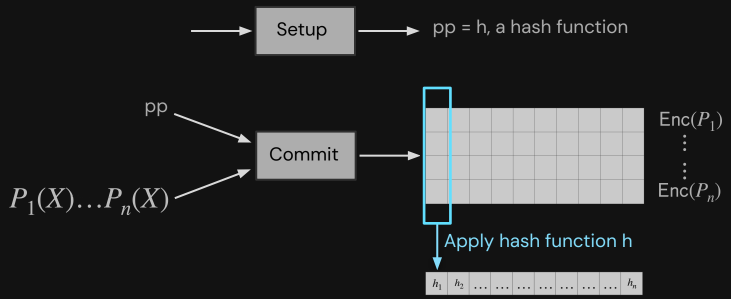
The Ligero/Breakdown Protocol



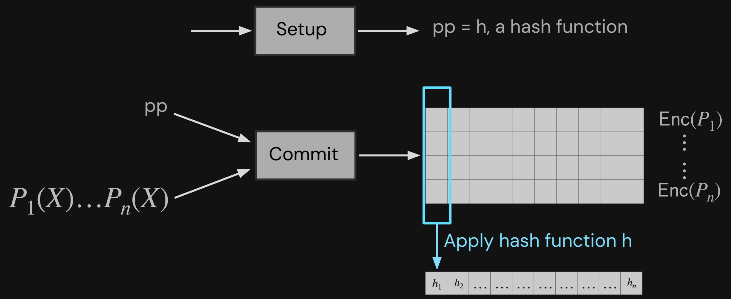
The Ligero/Breakdown Protocol



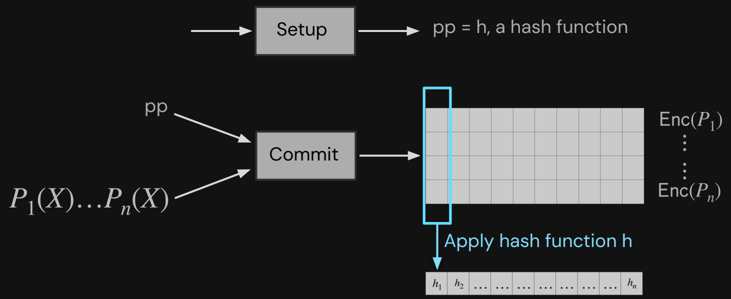
The Ligero/Breakdown Protocol



The Ligero/Breakdown Protocol



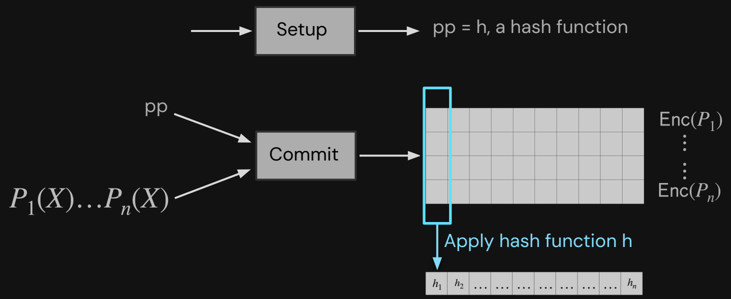
The Ligero/Breakdown Protocol



Open

Prover $\xleftarrow{\beta}$ Verifier

$$\mathbf{u} = \text{Enc}(P_0) + \beta \cdot \text{Enc}(P_1) + \ldots + \beta^{n-1} \cdot \text{Enc}(P_{n-1}) \xrightarrow{\quad \mathbf{u} \quad}$$



The Ligero/Breakdown Protocol

Open

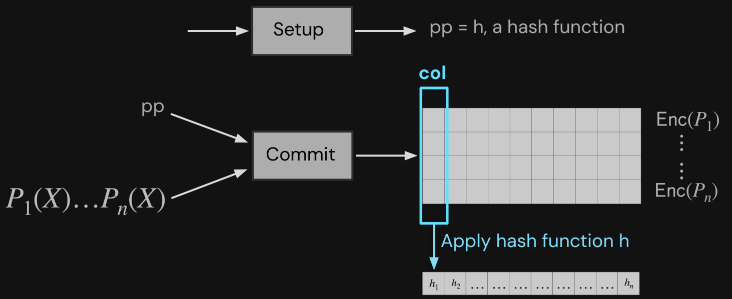
Prover ← β — Verifier

$$\mathbf{u} = \mathrm{Enc}(P_0) + \beta \cdot \mathrm{Enc}(P_1) + \dots + \beta^{n-1} \cdot \mathrm{Enc}(P_{n-1})$$

Prover — $\mathbf{u}$ → Verifier

Prover ← i — Verifier: Pick an index i ∈ {0…n}

The Ligero/Breakdown Protocol



Open

Prover — Verifier

Verifier → Prover: β

Prover: $\mathbf{u} = \text{Enc}(P_0) + \beta \cdot \text{Enc}(P_1) + \ldots + \beta^{n-1} \cdot \text{Enc}(P_{n-1})$

Prover → Verifier: **u**

Verifier: Pick an index i ∈ {0…n}

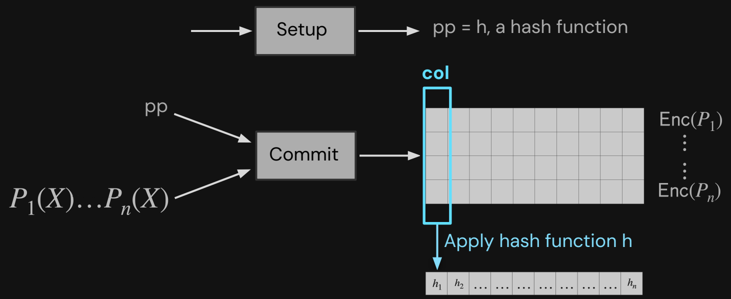
Verifier → Prover: i

Prover → Verifier: **col**

Verifier: Check $h(\mathbf{col}_i) \stackrel{?}{=} h_i$ $\quad \sum_i \beta^i \cdot \mathbf{col}_i \stackrel{?}{=} \mathbf{u}_i$

This only ensures that all the rows are codewords.

The Ligero/Breakdown Protocol



Open

Prover

Verifier

β (Verifier → Prover)

$$\mathbf{u} = \mathrm{Enc}(P_0) + \beta \cdot \mathrm{Enc}(P_1) + \ldots + \beta^{n-1} \cdot \mathrm{Enc}(P_{n-1})$$

$\mathbf{u}$ (Prover → Verifier)

Pick an index i ∈ {0…n}

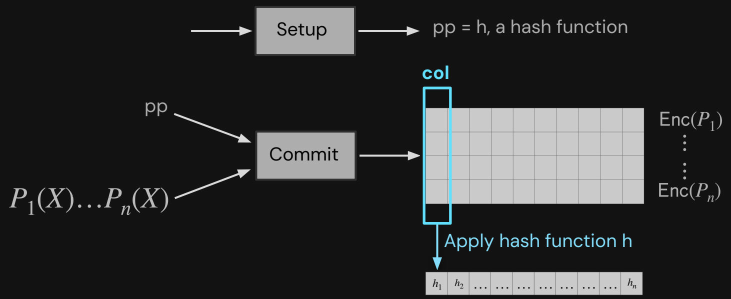
i (Verifier → Prover)

col (Prover → Verifier)

Check $h(\mathbf{col}_i) \stackrel{?}{=} h_i$ $\quad \sum_i \beta^i \cdot \mathbf{col}_i \stackrel{?}{=} \mathbf{u}_i$

Observation: interpolate $\mathbf{u}$ to obtain $P_\mathbf{u}$
check $P_\mathbf{u}(x) = \sum_i \beta^i \cdot y_i$

The Vortex Protocol



Open

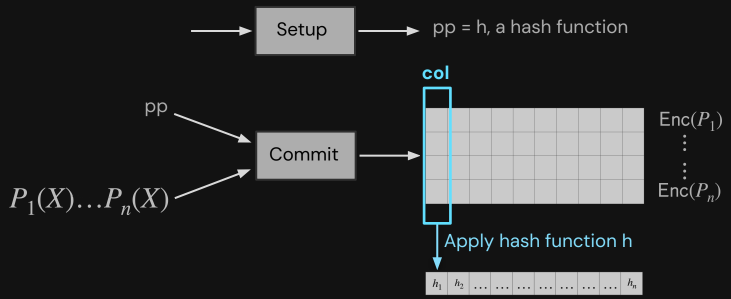
Prover		Verifier
$\xleftarrow{\beta}$
$\mathbf{u} = \text{Enc}(P_0) + \beta \cdot \text{Enc}(P_1) + \ldots + \beta^{n-1} \cdot \text{Enc}(P_{n-1})$	$\xrightarrow{\mathbf{u}}$
$\xleftarrow{i}$	Pick an index i ∈ {0…n}
$\xrightarrow{\textbf{col}}$	Check $h(\textbf{col}_i) \stackrel{?}{=} h_i$ $\quad \sum_i \beta^i \cdot \textbf{col}_i \stackrel{?}{=} \mathbf{u}_i$

Observation: interpolate $\mathbf{u}$ to obtain $P_{\mathbf{u}}$

check $P_{\mathbf{u}}(x) = \sum_i \beta^i \cdot y_i$

This additional check ensures that $P_1(x) = y_1 \ldots P_n(x) = y_n$.

The Vortex Protocol



Open

Prover — Verifier

Verifier → Prover: β

$$\mathbf{u} = \mathrm{Enc}(P_0) + \beta \cdot \mathrm{Enc}(P_1) + \ldots + \beta^{n-1} \cdot \mathrm{Enc}(P_{n-1})$$

Prover → Verifier: **u**

Verifier: Pick an index i ∈ {0…n}

Verifier → Prover: i

Prover → Verifier: **col**

Check $h(\mathbf{col}_i) \stackrel{?}{=} h_i$ $\quad \sum_i \beta^i \cdot \mathbf{col}_i \stackrel{?}{=} \mathbf{u}_i$

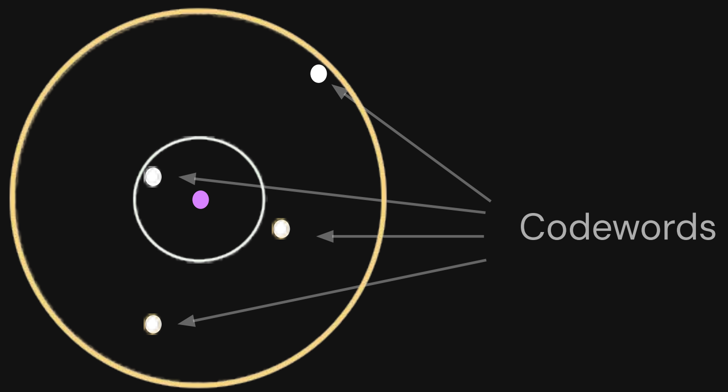
Observation: interpolate $\mathbf{u}$ to obtain $P_{\mathbf{u}}$

check $P_{\mathbf{u}}(x) = \sum_i \beta^i \cdot y_i$

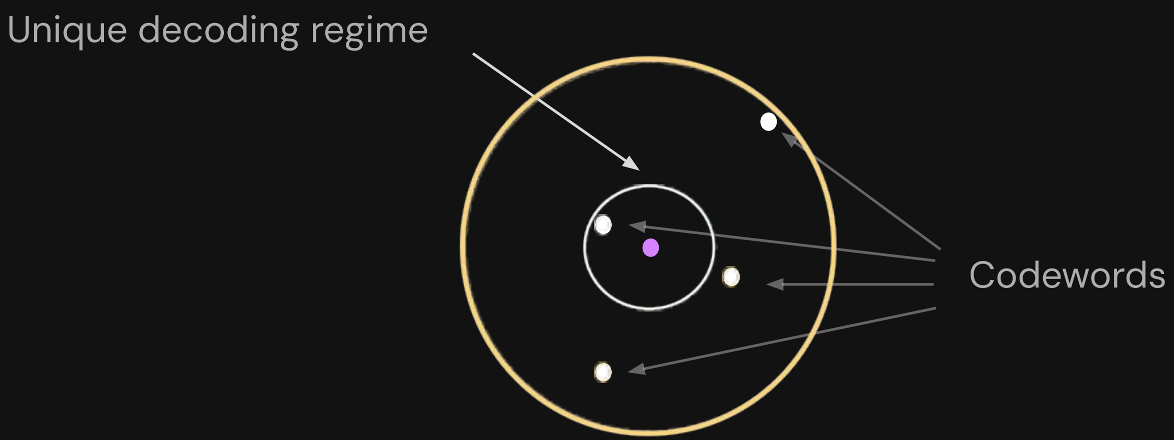
This additional check ensures that $P_1(x) = y_1 \ldots P_n(x) = y_n$.

However, the code parameters are not optimal.

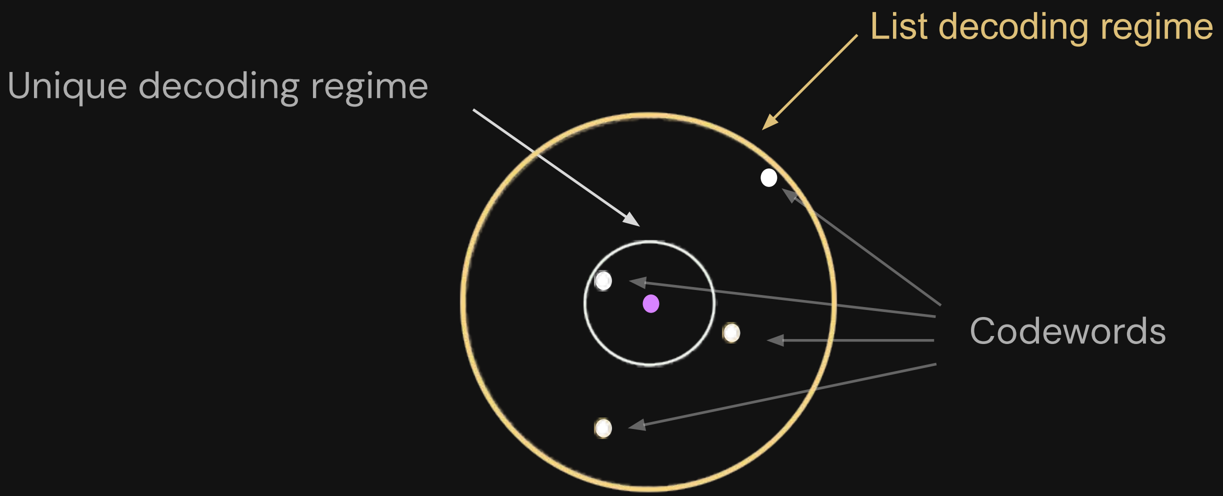
From Unique Decoding to List Decoding



From Unique Decoding to List Decoding



From Unique Decoding to List Decoding



From Unique Decoding to List Decoding

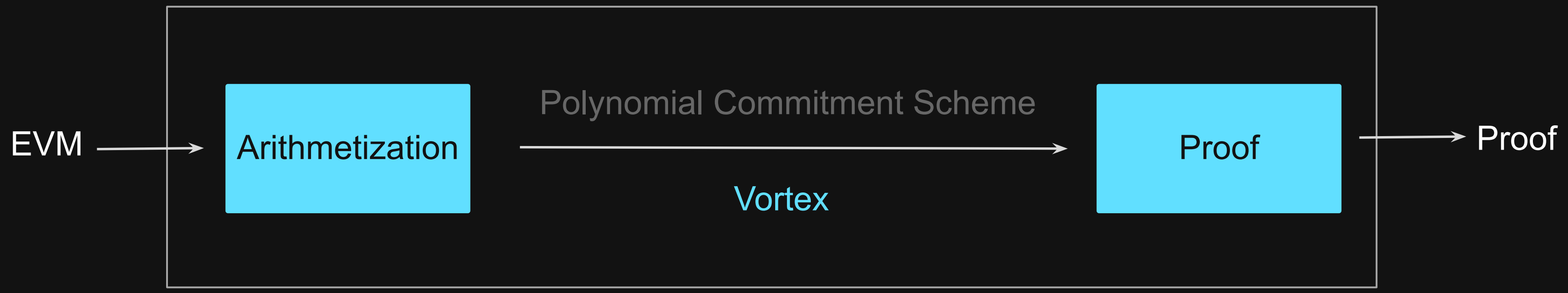
In the list decoding regime, the security guarantee is that there exist polynomials of small degree which evaluate correctly:

$$P_1(x) = y_1 \ldots P_n(x) = y_n.$$

And the commitments to the polynomials have only a small number of coordinates different from the target commitment: $h_1 \ldots h_n.$

Proof of security in eprint.iacr.org/2024/185

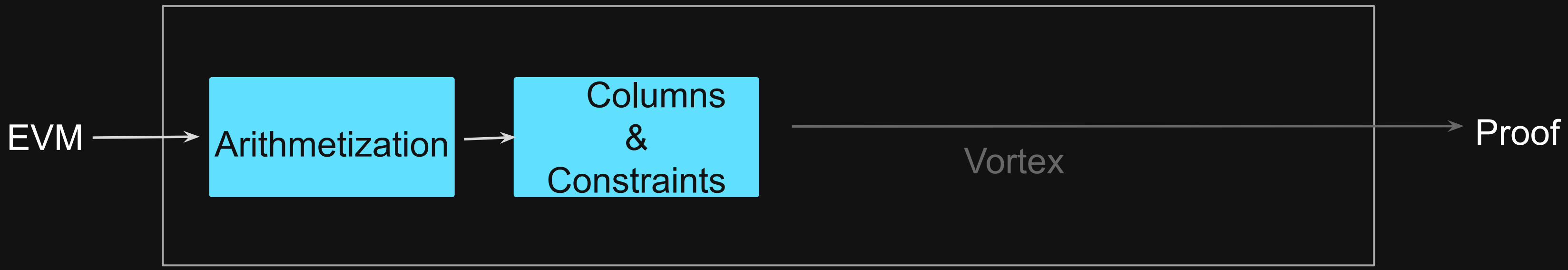
Part 2: from EVM Execution to Proof Generation



The journey of Linea

- Arithmetization: mathematical modeling of EVM via columns
- Constraints among columns
- H(x)=y

- ColumnX, ColumnX_1, ColumnX_2, $\ldots$, ColumnX_n, ColumnY



Type of constraints

- **LookUps:** Column A is Included in Column B
- **Local:** ColumnA$[0]$ $=$ ColumnB$[0]$
- **Global:** columnA$[i]$ $=$ columnA$[i-1]$ $+$ columnA$[i-2]$

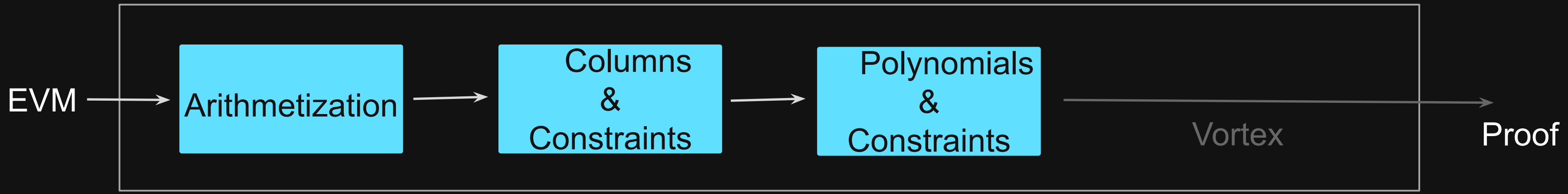
1
1
2
3
5
8
13
21

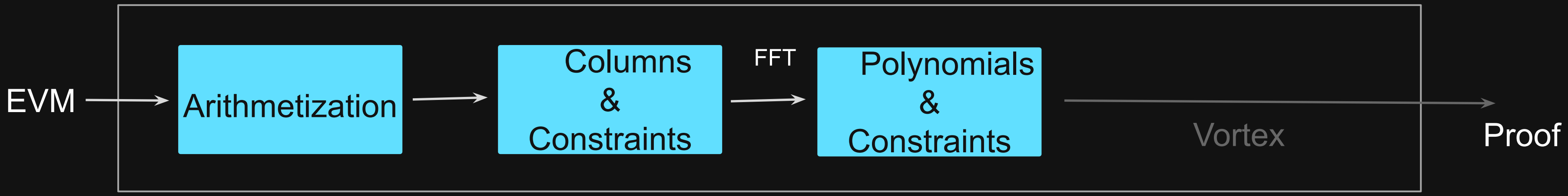
- columnX, columnX$_1$, … ,columnX$_1$, columnY

Many efforts!

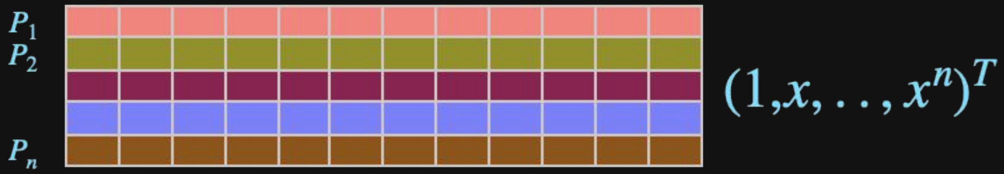
Polynomial Commitment

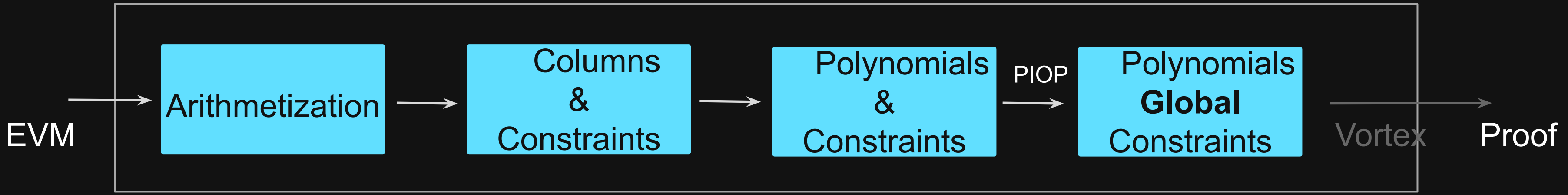
- Column = Polynomial
- Polynomial :
  - $F(P_1(X), \ldots, P_n(X)) \stackrel{?}{=} (X^n - 1) . Q(X)$ for every X
- Commitment: c_i instead of $P_i(X)$, can not be changed later.
- Schwartz–Zippel lemma: $F(P_1(\alpha), \ldots, P_n(\alpha)) \stackrel{?}{=} (\alpha^n - 1)Q(\alpha)$



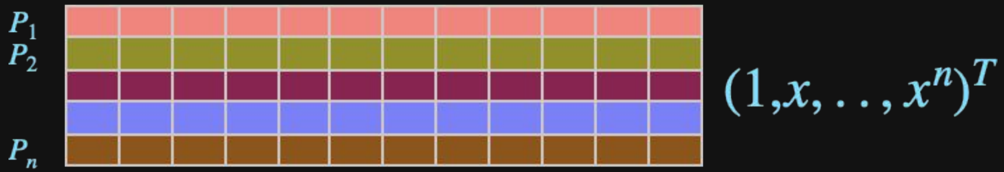

EVM
Arithmetization
Columns
&
Constraints
FFT
Polynomials
&
Constraints
Vortex
Proof

Vortex

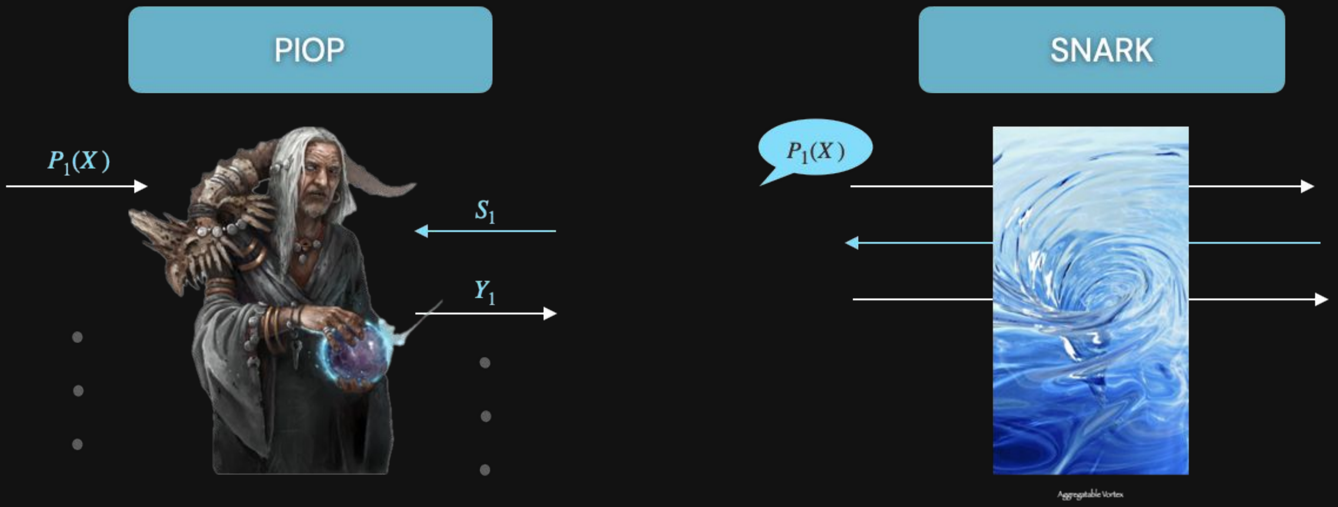

P_1
P_2
P_n
$(1, x, \ldots, x^n)^T$


EVM
Arithmetization
Columns & Constraints
Polynomials & Constraints
PIOP
Polynomials **Global** Constraints
Vortex
Proof

Vortex


P_1
P_2
P_n
$(1, x, \ldots, x^n)^T$

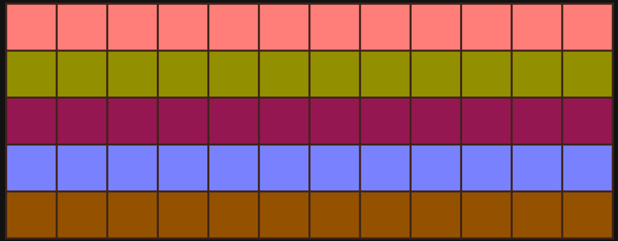
SNARK from PIOP

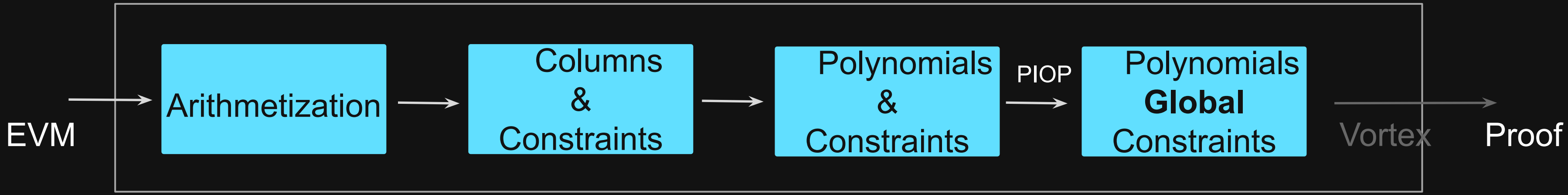


Theorem: Polynomial Commitment resembles to Oracle, thus it is a good replacement.

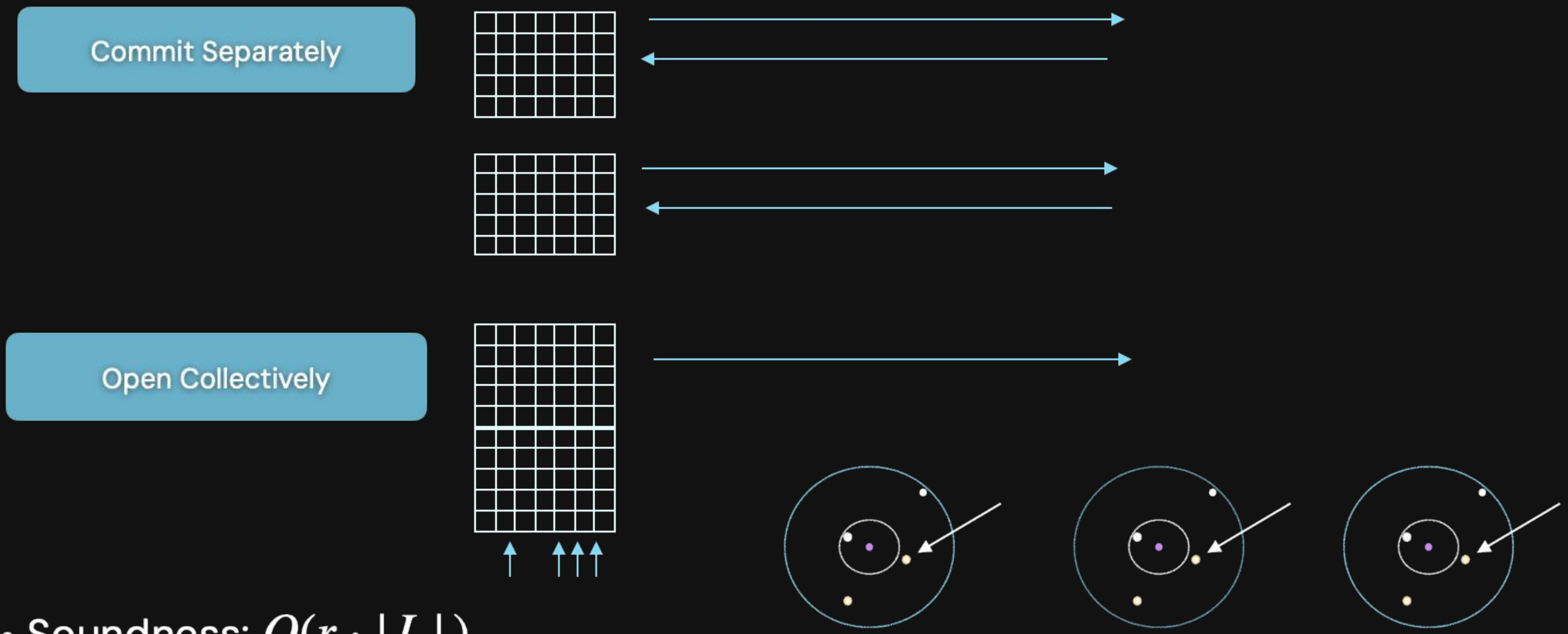
List Property of Vortex

Vortex





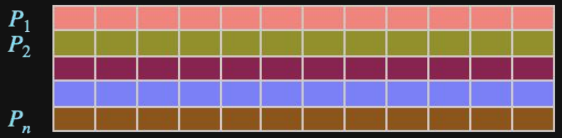
Aggregatable LPC



- Soundness: $O(r \cdot |L|)$

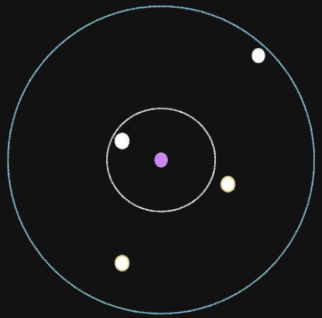
Batching at the Same-Point

Vortex



$(1, x, \ldots, x^n)^T$

Evaluation over the same point



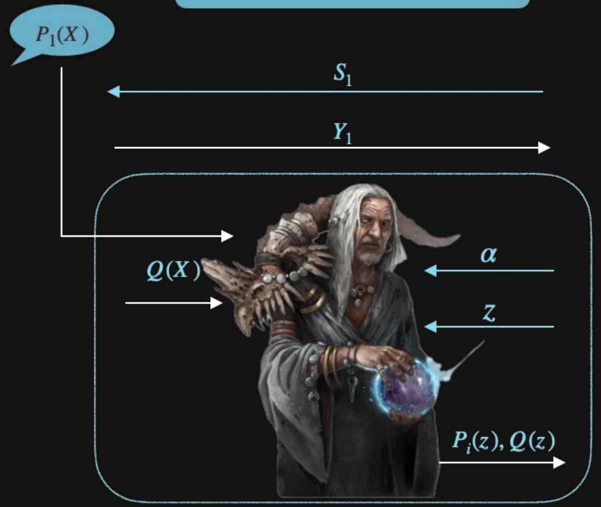
Vortex → $P_1(x), P_2(x), \ldots, P_n(x)$

Applications → $P_1(x_1), P_2(x_2), \ldots, P_n(x_n)$

UniEval Compiler

PIOP

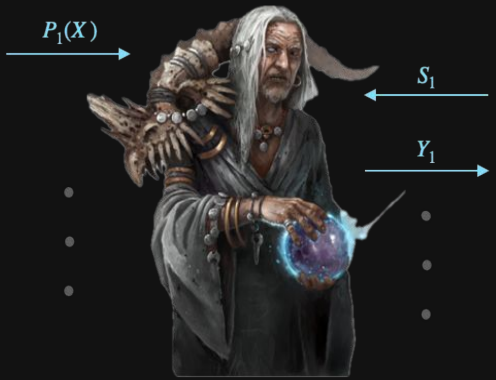
UniEval-PIOP



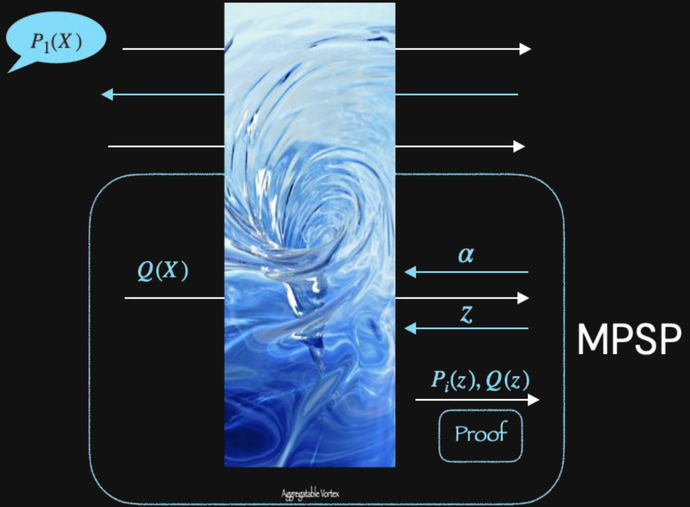
Multi-Point to Single-Point reduction

Putting All Together

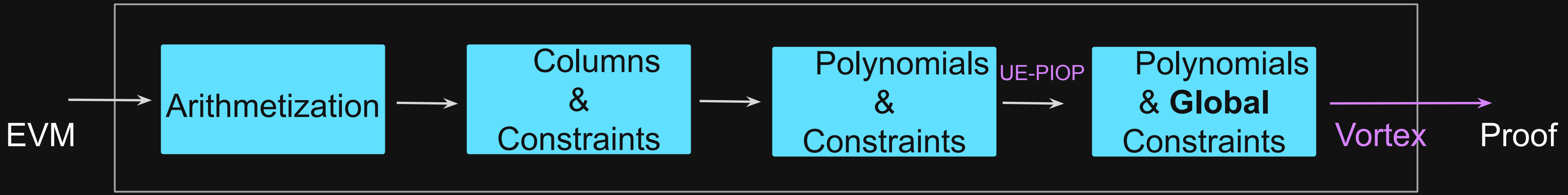
PIOP



LPC-SNARK



Finally, Here is the Proof



Example of Parameters

$$E_{\text{soundness}} \le E_{\text{collision}} + (1-\theta)^t + E$$

$$\theta = 1 - \sqrt{\rho} - \frac{\sqrt{\rho}}{2m'}$$

$$E \le (m-1)\frac{(m'+1/2)^7}{3\rho^{3/2}}\frac{|D|^2}{|\mathbb{F}|}$$

- Security bits = 128 bits
- Field Size =256 bits
- Codeword size = 2^20
- Blow-up factor of the code = 2^8
- Number of Polynomials =2^20
- Degree of polynomials = 2^ 12
- Number of chosen columns = 44 (versus 190 for unique decoding)
- List size = 151
- Soundness loss in PIOP = 8 bits of security? Maybe No!
- 15M constraints for the verifier of Vortex in Plonk with 100 bits of security.

$$\Pr(E^{\text{AoK}}) \le |L| \cdot \underbrace{\Pr(E^{\text{PIOP}})}_{k \cdot d / \mathbb{F}} + \Pr(E^{\text{aLPC}}) + \Pr(z \in D)$$

Further Reading

For more details,. see eprint.iacr.org/2024/185

Thank You!