

Zinc

Succinct Arguments with Small Arithmetization
Overheads from IOPs of Proximity to the Integers

Albert Garreta, Hendrik Waldner, Katerina Hristova, Luca Dall'ava

Arguments of knowledge

Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

x is a public **instance**, w is a **witness**

Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

x is a public **instance**, w is a **witness**

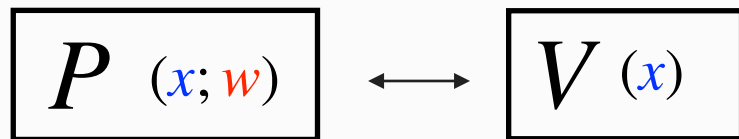
In a **SNARK for R** , given x , P convinces V that they know w such that $(x; w) \in R$.

Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

x is a public **instance**, w is a **witness**

In a **SNARK for R** , given x , P convinces V that they know w such that $(x; w) \in R$.

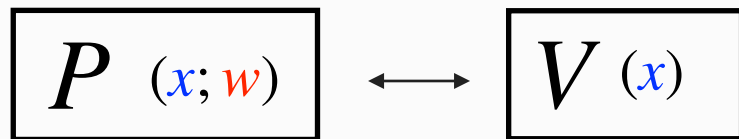


Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

x is a public **instance**, w is a **witness**

In a **SNARK for R** , given x , P convinces V that they know w such that $(x; w) \in R$.



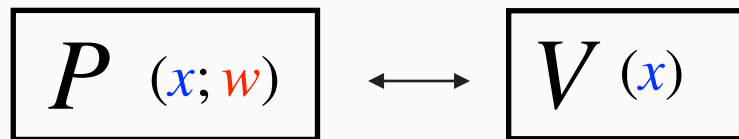
Often, R consists of algebraic equations over a finite field \mathbb{F} .

Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

x is a public **instance**, w is a **witness**

In a **SNARK for R** , given x , P convinces V that they know w such that $(x; w) \in R$.



Often, R consists of algebraic equations over a finite field \mathbb{F} .

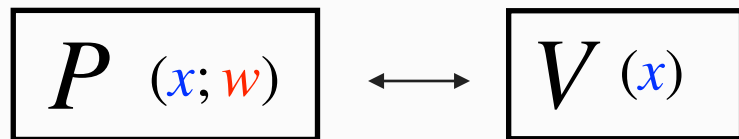
Example: the R1CS relation over \mathbb{F} :

Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

x is a public **instance**, w is a **witness**

In a **SNARK for R** , given x , P convinces V that they know w such that $(x; w) \in R$.



Often, R consists of algebraic equations over a finite field \mathbb{F} .

Example: the R1CS relation over \mathbb{F} :

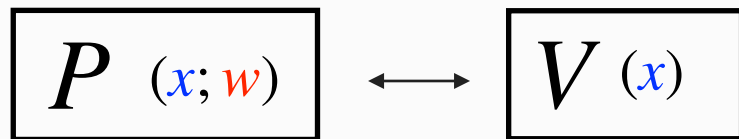
$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_1(x, w) = \dots = q_m(x, w) = 0\}$$

Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

x is a public **instance**, w is a **witness**

In a **SNARK for R** , given x , P convinces V that they know w such that $(x; w) \in R$.



Often, R consists of algebraic equations over a finite field \mathbb{F} .

Example: the R1CS relation over \mathbb{F} :

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_1(x, w) = \dots = q_m(x, w) = 0\}$$

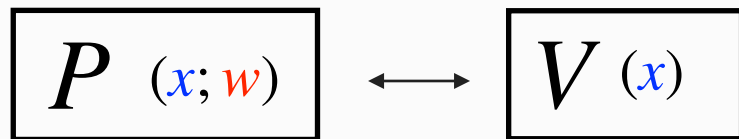
Where q_i are public quadratic polynomials.

Arguments of knowledge

Fix a relation R . Consists of pairs $(x; w)$.

x is a public **instance**, w is a **witness**

In a **SNARK for R** , given x , P convinces V that they know w such that $(x; w) \in R$.



Often, R consists of algebraic equations over a finite field \mathbb{F} .

Example: the R1CS relation over \mathbb{F} :

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_1(x, w) = \dots = q_m(x, w) = 0\}$$

Where q_i are public quadratic polynomials.

Other examples: AIR, Plonkish constraints, CCS, M3 (Irreducible)

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

*

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

*

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)

*

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)

$$(x, w) \in R_0$$

*

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)

$$(x, w) \in R_0$$

→ Rewrite →

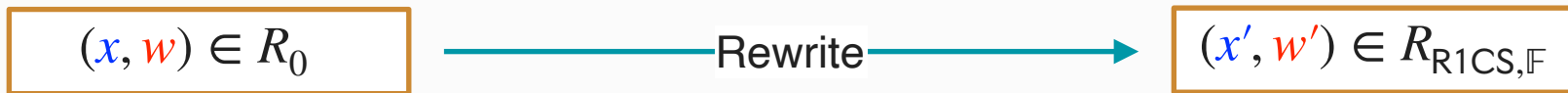
*

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)



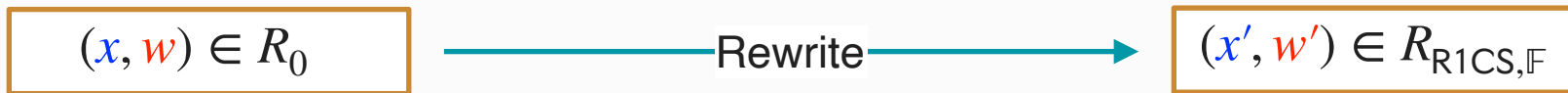
*

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)



We call this rewriting process **arithmetization**.

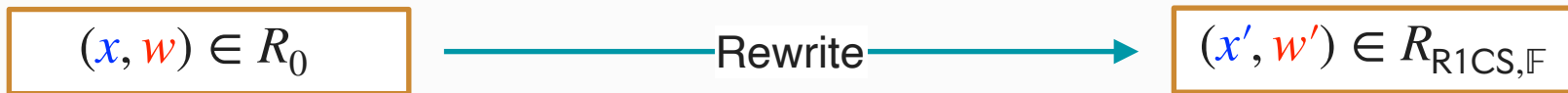
*

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)



We call this rewriting process **arithmetization**.

Arithmetization can create big blowouts.

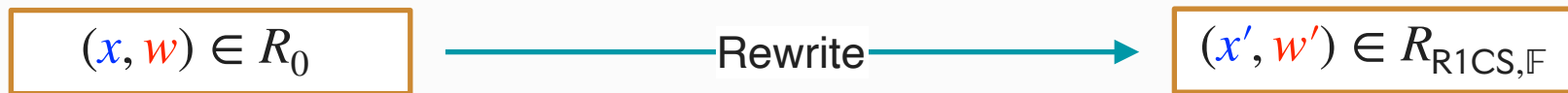
*

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)



We call this rewriting process **arithmetization**.

Arithmetization can create big blowouts.

Example: ECDSA verification proved over a non-native field has 2^{21} R1CS constraints, Vs 2^{16} over native field.* (Paper reports >400x performance improvements)

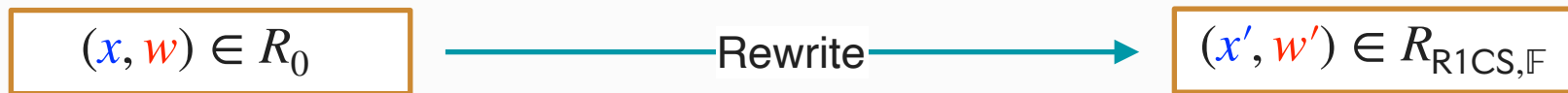
* Block et al. *Field Agnostic SNARKs from EA codes*, Crypto 2024

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)



We call this rewriting process **arithmetization**.

Arithmetization can create big blowouts.

Example: ECDSA verification proved over a non-native field has 2^{21} R1CS constraints, Vs 2^{16} over native field.* (Paper reports >400x performance improvements)

Thesis: There's room for improvements here.

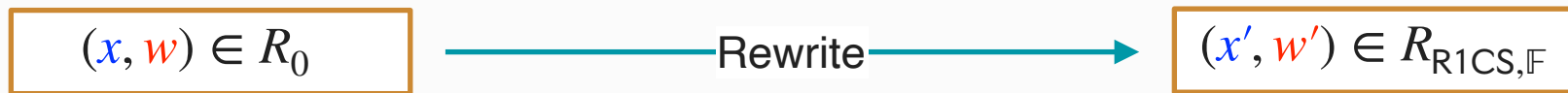
* Block et al. *Field Agnostic SNARKs from EA codes*, Crypto 2024

Arithmetization in arguments

$$R_{\text{R1CS}, \mathbb{F}} = \{(x, w) \in \mathbb{F}^n \mid q_i(x, w) = 0\}$$

What if we want to prove a claim $(x, w) \in R_0$ for a different R_0 ?

Example: CPU operations (add/mult mod 2^{64} , XOR of bitstrings, etc)



We call this rewriting process **arithmetization**.

Arithmetization can create big blowouts.

Example: ECDSA verification proved over a non-native field has 2^{21} R1CS constraints, Vs 2^{16} over native field.* (Paper reports >400x performance improvements)

Thesis: There's room for improvements here.

A lot to be gained (10x speedups? Ease of implementations?)

* Block et al. *Field Agnostic SNARKs from EA codes*, Crypto 2024

Examples

Examples

Computation

Relevant for

Examples

Computation	Relevant for
Rationals, fixed point, floating point	ML, finance, FHE

Examples

Computation	Relevant for
Rationals, fixed point, floating point	ML, finance, FHE
Mult mod 2^n	CPU operations, lattice cryptography,

Examples

Computation	Relevant for
Rationals, fixed point, floating point	ML, finance, FHE
Mult mod 2^n	CPU operations, lattice cryptography,
Mult mod pq	RSA cryptography

Examples

Computation	Relevant for
Rationals, fixed point, floating point	ML, finance, FHE
Mult mod 2^n	CPU operations, lattice cryptography,
Mult mod pq	RSA cryptography
Mult mod non-native q	Recursive proving, IVC, PCD, etc.

Examples

Computation	Relevant for
Rationals, fixed point, floating point	ML, finance, FHE
Mult mod 2^n	CPU operations, lattice cryptography,
Mult mod pq	RSA cryptography
Mult mod non-native q	Recursive proving, IVC, PCD, etc.
XOR, NOT, etc.	Classic hashes, CPU operations, AES encryption

Examples

Computation	Relevant for
Rationals, fixed point, floating point	ML, finance, FHE
Mult mod 2^n	CPU operations, lattice cryptography,
Mult mod pq	RSA cryptography
Mult mod non-native q	Recursive proving, IVC, PCD, etc.
XOR, NOT, etc.	Classic hashes, CPU operations, AES encryption
Lattice operations	Lattice cryptography, FHE

Examples

Computation	Relevant for
Rationals, fixed point, floating point	ML, finance, FHE
Mult mod 2^n	CPU operations, lattice cryptography,
Mult mod pq	RSA cryptography
Mult mod non-native q	Recursive proving, IVC, PCD, etc.
XOR, NOT, etc.	Classic hashes, CPU operations, AES encryption
Lattice operations	Lattice cryptography, FHE
Combinations of above	Classic hashes, legacy cryptography, recursion, zkVM, Ethereum blocks

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Examples

Computation

Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Examples

Computation

Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Then

$$x \cdot y = \sum_{ij} x_i \cdot y_j \cdot 2^{8 \cdot (i+j)}$$

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Then

$$x \cdot y = \sum_{ij} x_i \cdot y_j \cdot 2^{8 \cdot (i+j)}$$

Use lookups for $x_i \cdot y_j$ (or degree-2 constraints), 16 of them (or less with tricks)

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Then

$$x \cdot y = \sum_{ij} x_i \cdot y_j \cdot 2^{8 \cdot (i+j)}$$

Use lookups for $x_i \cdot y_j$ (or degree-2 constraints), 16 of them (or less with tricks)

Combine the $x_i \cdot y_j$ into the result z (requires addition and scalar mult mod 2^{32})

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Then

$$x \cdot y = \sum_{ij} x_i \cdot y_j \cdot 2^{8 \cdot (i+j)}$$

Use lookups for $x_i \cdot y_j$ (or degree-2 constraints), 16 of them (or less with tricks)

Combine the $x_i \cdot y_j$ into the result z (requires addition and scalar mult mod 2^{32})

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Then

$$x \cdot y = \sum_{ij} x_i \cdot y_j \cdot 2^{8 \cdot (i+j)}$$

Use lookups for $x_i \cdot y_j$ (or degree-2 constraints), 16 of them (or less with tricks)

Complex arithmetizations have indirect costs as: mult z (requires addition and scalar mult

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Then

$$x \cdot y = \sum_{ij} x_i \cdot y_j \cdot 2^{8 \cdot (i+j)}$$

Use lookups for $x_i \cdot y_j$ (or degree-2 constraints), 16 of them (or less with tricks)

Complex arithmetizations have indirect costs as: mult z (requires addition and scalar mult

- Engineering effort,

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Then

$$x \cdot y = \sum_{ij} x_i \cdot y_j \cdot 2^{8 \cdot (i+j)}$$

Use lookups for $x_i \cdot y_j$ (or degree-2 constraints), 16 of them (or less with tricks)

Complex arithmetizations have indirect costs as: mult z (requires addition and scalar mult

- Engineering effort,
- Security risks,

Examples

Computation	
Rationals, fixed point, floating point	
Mult mod 2^n	
Mult mod pq	
Mult mod non-native q	
XOR, NOT, etc.	C
Lattice operations	
Combinations of above	C

Arithmetizing $x \cdot y = z \pmod{2^{32}}$ over 32 bit field \mathbb{F}

Binius' benchmarks
(over $\text{GF}(2^{32})$)

2^{20} mults	2^{20} mult mod 2^{32}
80ms	2000ms

Decompose x, y into 4 limbs of 8 bits.

$$x = \sum_i x_i 2^{8 \cdot i}, \quad y = \sum_i y_i 2^{8 \cdot i}, \quad x_i, y_i \in [0, 2^8 - 1]$$

Then

$$x \cdot y = \sum_{ij} x_i \cdot y_j \cdot 2^{8 \cdot (i+j)}$$

Use lookups for $x_i \cdot y_j$ (or degree-2 constraints), 16 of them (or less with tricks)

Complex arithmetizations have indirect costs as: mult z (requires addition and scalar mult

- Engineering effort,
- Security risks,
- Audits, FV, etc.

Examples

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Proving 33 SHA-256 compressions

	Plonkv3*	Stwo*	Binius (2^{13})**
Prover (s)	12	21	0.33
Size (MB)	1.7	39	0.38
Verifier (s)	0.18	N/A	0.14

* 33 compressions <https://hackmd.io/@clientsideproving/zkIDBenchmarks>

** 2^{13} compressions, ran on my MacBook Air

Binius is not necessarily better than the others:

- SHA-256 is native to it (not the AND and add mod 2^{32})

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Goals: Design a proof system with:

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Goals: Design a proof system with:

Universal arithmetization

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Similar to SOTA proof systems

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Similar to SOTA proof systems

Runs over finite fields

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Similar to SOTA proof systems

Runs over finite fields

Error correcting codes

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Similar to SOTA proof systems

Runs over finite fields

Error correcting codes

Hash functions

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Similar to SOTA proof systems

Runs over finite fields

Error correcting codes

Hash functions

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Similar to SOTA proof systems

Runs over finite fields

Error correcting codes

Hash functions

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	?
Mult mod 2^n	?
Mult mod pq	?
Mult mod non-native q	?
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Later: extend these to (almost) all the above

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Similar to SOTA proof systems

Runs over finite fields

Error correcting codes

Hash functions

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	Zinc
Mult mod 2^n	Zinc
Mult mod pq	Zinc
Mult mod non-native q	Zinc
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Later: extend these to (almost) all the above

Goals: Design a proof system with:

Universal arithmetization

All computations in the table are native

Similar to SOTA proof systems

Runs over finite fields

Error correcting codes

Hash functions

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	Zinc
Mult mod 2^n	Zinc
Mult mod pq	Zinc
Mult mod non-native q	Zinc
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Later: extend these to (almost) all the above

Goals: Design a proof system with:

Universal arithmetization

Some use-cases involve a mix of computation types.

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	Zinc
Mult mod 2^n	Zinc
Mult mod pq	Zinc
Mult mod non-native q	Zinc
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Later: extend these to (almost) all the above

Goals: Design a proof system with:

Universal arithmetization

Some use-cases involve a mix of computation types.

Examples:

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	Zinc
Mult mod 2^n	Zinc
Mult mod pq	Zinc
Mult mod non-native q	Zinc
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Later: extend these to (almost) all the above

Goals: Design a proof system with:

Universal arithmetization

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	Zinc
Mult mod 2^n	Zinc
Mult mod pq	Zinc
Mult mod non-native q	Zinc
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Later: extend these to (almost) all the above

Goals: Design a proof system with:

Universal arithmetization

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	Zinc
Mult mod 2^n	Zinc
Mult mod pq	Zinc
Mult mod non-native q	Zinc
XOR, NOT, etc.	Binius
Lattices	Dedicated SNARK
Combinations of above	?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Later: extend these to (almost) all the above

Goals: Design a proof system with:

Universal arithmetization

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS

STARKs, Plonky, Ligerio, etc.

Examples

Computation	Hash-based native proof system
Rationals, fixed point, floating point	Zinc
Mult mod 2^n	Zinc
Mult mod pq	Zinc
Mult mod non-native q	Zinc
XOR, NOT, etc.	Binius, Zinc?
Lattices	Dedicated SNARK, Zinc?
Combinations of above	Zinc?

As first step, we limit ourselves to constraints over $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q} (n arbitrary)

Later: extend these to (almost) all the above

Goals: Design a proof system with:

Universal arithmetization

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS

WIP: extension of Zinc to these

STARKs, Plonky, Ligerio, etc.

Zinc (Crypto '25)

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc =

Spartan + Ligero-like PCS,
mod random prime

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc =

Spartan + Ligero-like PCS,
mod random prime

(In practical terms.
Formally, it is
nuanced)

* Can be essentially any PIOP

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc =

Spartan + Ligero-like PCS,
mod random prime

(In practical terms.
Formally, it is
nuanced)

Performance

WIP Implementation

* Can be essentially any PIOP

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc =

Spartan + Ligero-like PCS,
mod random prime

(In practical terms.
Formally, it is
nuanced)

Performance

Zinc PCS commit & open
(16 vars, 64bit witness entries)

36ms (20 + 16ms),
<250KB**

WIP Implementation

* Can be essentially any PIOP

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc =

Spartan + Ligero-like PCS,
mod random prime

(In practical terms.
Formally, it is
nuanced)

Performance

Zinc PCS commit & open (16 vars, 64bit witness entries)	36ms (20 + 16ms), <250KB**
Spartan PIOP: mod random prime/ mod fixed prime	~ 1

WIP Implementation

** Largish bc of Ligero, not bc Zinc overheads

* Can be essentially any PIOP

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc =

Spartan + Ligero-like PCS,
mod random prime

(In practical terms.
Formally, it is
nuanced)

IOP of proximity to \mathbb{Z}

Performance

Zinc PCS commit & open (16 vars, 64bit witness entries)	36ms (20 + 16ms), <250KB**
Spartan PIOP: mod random prime/ mod fixed prime	~ 1

WIP Implementation

** Largish bc of Ligero, not bc Zinc overheads

* Can be essentially any PIOP

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc =

Spartan + Ligerio-like PCS,
mod random prime

(In practical terms.
Formally, it is
nuanced)

IOP of proximity to \mathbb{Z}

New primitive

PCS guarantees integer/rational
coefficients of bounded bit-size

Performance

Zinc PCS commit & open (16 vars, 64bit witness entries)	36ms (20 + 16ms), <250KB**
Spartan PIOP: mod random prime/ mod fixed prime	~ 1

WIP Implementation

** Largish bc of Ligerio, not bc Zinc overheads

* Can be essentially any PIOP

Zinc (Crypto '25)

Framework and family of succinct arguments with:

Native for $\mathbb{Z}/n\mathbb{Z}$, \mathbb{Z} , \mathbb{Q}

Can handle different constraint types at once.

Zinc =

Spartan + Ligerio-like PCS,
mod random prime

(In practical terms.
Formally, it is
nuanced)

IOP of proximity to \mathbb{Z}

New primitive

PCS guarantees integer/rational
coefficients of bounded bit-size

Performance

Zinc PCS commit & open (16 vars, 64bit witness entries)	36ms (20 + 16ms), <250KB**
Spartan PIOP: mod random prime/ mod fixed prime	~ 1

WIP Implementation

** Largish bc of Ligerio, not bc Zinc overheads

* Can be essentially any PIOP

A journey

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

$$A z \circ B z = C z \text{ over } \mathbb{Z}_n$$

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

$$A z \circ B z = C z \text{ over } \mathbb{Z}_n$$



A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

$$A z \circ B z = C z \text{ over } \mathbb{Z}_n$$



There exists $u \in \mathbb{Z}^{m+k+1}$ such that

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

$$A z \circ B z = C z \text{ over } \mathbb{Z}_n$$



There exists $u \in \mathbb{Z}^{m+k+1}$ such that
 $A z \circ B z = C z + u \cdot n$ as integers

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

$$A z \circ B z = C z \text{ over } \mathbb{Z}_n$$



There exists $u \in \mathbb{Z}^{m+k+1}$ such that
 $A z \circ B z = C z + u \cdot n$ as integers

So, build a proof system for:

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

$$A z \circ B z = C z \text{ over } \mathbb{Z}_n$$



There exists $u \in \mathbb{Z}^{m+k+1}$ such that
 $A z \circ B z = C z + u \cdot n$ as integers

So, build a proof system for:

$$R_{\text{R1CS}^\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A z \circ B z = C z + u \cdot n \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

$$A z \circ B z = C z \text{ over } \mathbb{Z}_n$$



There exists $u \in \mathbb{Z}^{m+k+1}$ such that
 $A z \circ B z = C z + u \cdot n$ as integers

So, build a proof system for:

$$R_{\text{R1CS}_{\ell}, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A z \circ B z = C z + u \cdot \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Arbitrary moduli at the same time!

A journey

Let's first try to build a proof system for R1CS over \mathbb{Z}_n .

$$R_{\text{R1CS}, \mathbb{Z}_n} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}_n^m, w \in \mathbb{Z}_n^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}_n, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Observe: $x = y$ over \mathbb{Z}_n if and only if $x = y + n \cdot \mu$ over \mathbb{Z} , for some $\mu \in \mathbb{Z}$.

$$A z \circ B z = C z \text{ over } \mathbb{Z}_n$$



There exists $u \in \mathbb{Z}^{m+k+1}$ such that
 $A z \circ B z = C z + u \cdot n$ as integers

So, build a proof system for:

$$R_{\text{R1CS}^\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A z \circ B z = C z + u \cdot \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

$\mathbb{Z}_{\leq B}$ = integers of $\leq B$ bits

Arbitrary moduli at the same time!

Zinc in a nutshell

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against
 P^* that use polys over $\mathbb{Z}_{\leq B}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against
 P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against
 P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Issues:

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Issues:

- (1) can require operating with integers of thousands of bits.

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Issues:

- (1) can require operating with integers of thousands of bits.
(2) is complicated to build efficiently.

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Issues:

- (1) can require operating with integers of thousands of bits.
(2) is complicated to build efficiently.

None available comparable to hash-based PCS's over \mathbb{F}

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Issues:

- (1) can require operating with integers of thousands of bits.
(2) is complicated to build efficiently.

None available comparable to hash-based PCS's over \mathbb{F}

- (1) **Solution:**[CH2024]: V samples a random prime q .
Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z} .

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \mid \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right\}$$

- (2) Work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$. Makes extraction simple.

Issues:

- (1) can require operating with integers of thousands of bits.
- (2) is complicated to build efficiently.

None available comparable to hash-based PCS's over \mathbb{F}

- (1) **Solution:** [CH2024]: V samples a random prime q .
Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z} .

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \mid \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right\}$$

(2) Work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$. Makes extraction simple.

Zip: PCS for polynomials over $\mathbb{Q}_{\leq B}$. Similar to Ligerio/Brakedown.

Issues:

(1) can require operating with integers of thousands of bits.

(2) is complicated to build efficiently.

None available comparable to hash-based PCS's over \mathbb{F}

(1) **Solution:**[CH2024]: V samples a random prime q .
Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z} .

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \mid \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right\}$$

(2) Work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$. Makes extraction simple.

Zip: PCS for polynomials over $\mathbb{Q}_{\leq B}$. Similar to Liger/Brakedown.

Allows to compile a PIOP for $R_{\text{R1CS}\ell, \mathbb{Q}}$ into a SNARK Π for $R_{\text{R1CS}\ell, \mathbb{Q}}$

Issues:

(1) can require operating with integers of thousands of bits.

(2) is complicated to build efficiently.

None available comparable to hash-based PCS's over \mathbb{F}

(1) Solution: [CH2024]: V samples a random prime q .
Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z} .

Zinc in a nutshell

$$R_{R1CS\ell, \mathbb{Z}} = \left\{ (x; w, u) \mid \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right\}$$

(2) Work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$. Makes extraction simple.

Zip: PCS for polynomials over $\mathbb{Q}_{\leq B}$. Similar to Ligerio/Brakedown.

Allows to compile a PIOP for $R_{R1CS\ell, \mathbb{Q}}$ into a SNARK Π for $R_{R1CS\ell, \mathbb{Q}}$

Works for any algebraic relation over \mathbb{Q} .

Issues:

(1) can require operating with integers of thousands of bits.

(2) is complicated to build efficiently.

None available comparable to hash-based PCS's over \mathbb{F}

(1) **Solution:**[CH2024]: V samples a random prime q .
Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z} .

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \mid \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right\}$$

(2) Work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$. Makes extraction simple.

Zip: PCS for polynomials over $\mathbb{Q}_{\leq B}$. Similar to Liger/Brakedown.

Allows to compile a PIOP for $R_{\text{R1CS}\ell, \mathbb{Q}}$ into a SNARK Π for $R_{\text{R1CS}\ell, \mathbb{Q}}$

Works for any algebraic relation over \mathbb{Q} .

Yields lookups Π_{look} for $\mathbb{Z}_{\leq B} \subseteq \mathbb{Q}_{\leq B}$

Issues:

(1) can require operating with integers of thousands of bits.

(2) is complicated to build efficiently.

None available comparable to hash-based PCS's over \mathbb{F}

(1) Solution: [CH2024]: V samples a random prime q .
Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z} .

Zinc in a nutshell

$$R_{R1CS\ell, \mathbb{Z}} = \left\{ (x; w, u) \mid \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right\}$$

(2) Work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$. Makes extraction simple.

Zip: PCS for polynomials over $\mathbb{Q}_{\leq B}$. Similar to Ligerio/Brakedown.

Allows to compile a PIOP for $R_{R1CS\ell, \mathbb{Q}}$ into a SNARK Π for $R_{R1CS\ell, \mathbb{Q}}$

Works for any algebraic relation over \mathbb{Q} .

Yields lookups Π_{look} for $\mathbb{Z}_{\leq B} \subseteq \mathbb{Q}_{\leq B}$

Combining Π and Π_{look} yields SNARK for $R_{R1CS\ell, \mathbb{Z}}$.

Issues:

(1) can require operating with integers of thousands of bits.

(2) is complicated to build efficiently.

None available comparable to hash-based PCS's over \mathbb{F}

(1) **Solution:**[CH2024]: V samples a random prime q .
Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z} .

Thanks

Collaborators wanted!

Research (fundamental or applications), engineering, use-cases, etc.



Paper



Implementation
(WIP)



Technical talk
at zkproof 6

Zinc in a nutshell

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against
 P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against
 P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against
 P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2). [CH2024] propose using the PCS over $\mathbb{Z}_{\leq B}$ of Block et al.

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2). [CH2024] propose using the PCS over $\mathbb{Z}_{\leq B}$ of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2). [CH2024] propose using the PCS over $\mathbb{Z}_{\leq B}$ of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

- When trying to extract the committed polynomial f , one has to solve a system of linear equations over \mathbb{Z} .

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2). [CH2024] propose using the PCS over $\mathbb{Z}_{\leq B}$ of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

- When trying to extract the committed polynomial f , one has to solve a system of linear equations over \mathbb{Z} .
- The solution determines f .

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2). [CH2024] propose using the PCS over $\mathbb{Z}_{\leq B}$ of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

- When trying to extract the committed polynomial f , one has to solve a system of linear equations over \mathbb{Z} .
- The solution determines f .
- However, in general, the solution consists of rational numbers.

Zinc in a nutshell

- (1) Build a PIOP for $R_{R1CS\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against
 P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{R1CS\ell, \mathbb{Z}}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against
 P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct
argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2).

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2).

We decided to work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$. Makes extraction simple.

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2).

We decided to work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$. Makes extraction simple.

Zip: PCS for multilinear polynomials with coeffs in $\mathbb{Q}_{\leq B}$. Similar to Ligero/Brakedown.

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2).

We decided to **work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$** . Makes extraction simple.

Zip: PCS for multilinear polynomials with coeffs in $\mathbb{Q}_{\leq B}$. Similar to Ligero/Brakedown.

Allows to compile a PIOP for $R_{\text{R1CS}\ell, \mathbb{Q}}$ into a succinct argument Π for $R_{\text{R1CS}\ell, \mathbb{Q}}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \mathbf{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2).

We decided to **work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$** . Makes extraction simple.

Zip: PCS for multilinear polynomials with coeffs in $\mathbb{Q}_{\leq B}$. Similar to Ligero/Brakedown.

Allows to compile a PIOP for $R_{\text{R1CS}\ell, \mathbb{Q}}$ into a succinct argument Π for $R_{\text{R1CS}\ell, \mathbb{Q}}$

This actually works for any algebraic relation over \mathbb{Q} .

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2).

We decided to **work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$** . Makes extraction simple.

Zip: PCS for multilinear polynomials with coeffs in $\mathbb{Q}_{\leq B}$. Similar to Ligero/Brakedown.

Allows to compile a PIOP for $R_{\text{R1CS}\ell, \mathbb{Q}}$ into a succinct argument Π for $R_{\text{R1CS}\ell, \mathbb{Q}}$

This actually works for any algebraic relation over \mathbb{Q} .

In particular, we obtain succinct lookups Π_{look} for $\mathbb{Z}_{\leq B} \subseteq \mathbb{Q}_{\leq B}$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

- (1) Build a PIOP for $R_{\text{R1CS}\ell, \mathbb{Z}}$
(e.g. Spartan)

With soundness holding against P^* that use polys over $\mathbb{Z}_{\leq B}$

- (2) Design a PCS for polynomials over $\mathbb{Z}_{\leq B}$

Compile into succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$

Building (2).

We decided to **work over $\mathbb{Q}_{\leq B}$ instead of $\mathbb{Z}_{\leq B}$** . Makes extraction simple.

Zip: PCS for multilinear polynomials with coeffs in $\mathbb{Q}_{\leq B}$. Similar to Ligero/Brakedown.

Allows to compile a PIOP for $R_{\text{R1CS}\ell, \mathbb{Q}}$ into a succinct argument Π for $R_{\text{R1CS}\ell, \mathbb{Q}}$

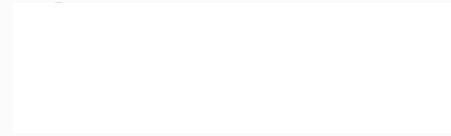
This actually works for any algebraic relation over \mathbb{Q} .

In particular, we obtain succinct lookups Π_{lookup} for $\mathbb{Z}_{\leq B} \subseteq \mathbb{Q}_{\leq B}$

Combining Π and Π_{lookup} we obtain a succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}}$.

Based on Liger/Brakedown with commitment on \mathbb{Q} , but runs modulo a prime.
Incorporates a costless bit-size check to make sure elements have less than B bits.
Commitment is over \mathbb{Q} . We use EA or RAA codes to keep the codeword entries small.

Zinc in a nutshell



Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

$$\begin{array}{c} P \\ (x; w, u) \end{array}$$

$$\begin{array}{c} V \\ (x) \end{array}$$

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

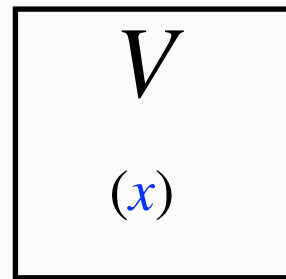
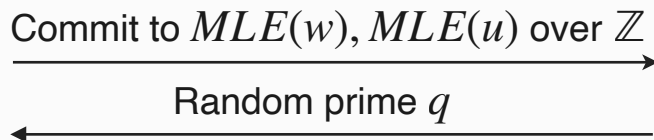
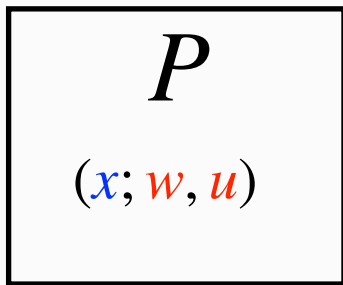
$$\boxed{\begin{array}{c} P \\ (x; w, u) \end{array}}$$

Commit to $MLE(w), MLE(u)$ over \mathbb{Z} \longrightarrow

$$\boxed{\begin{array}{c} V \\ (x) \end{array}}$$

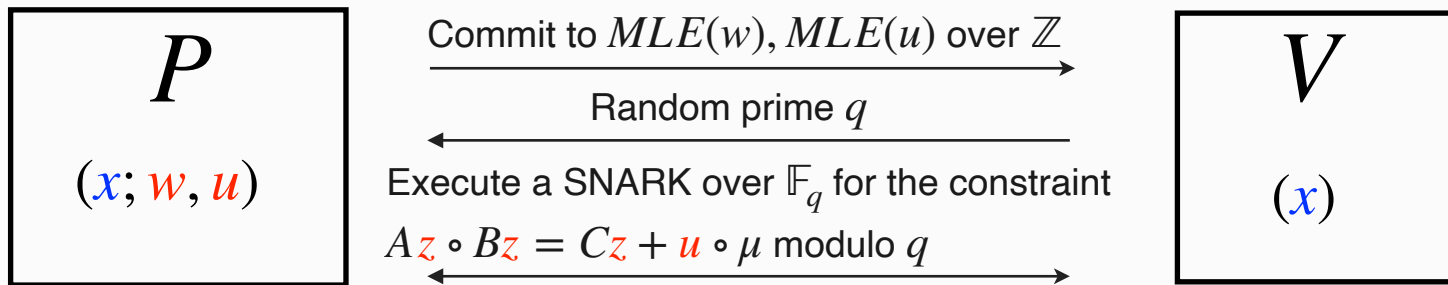
Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



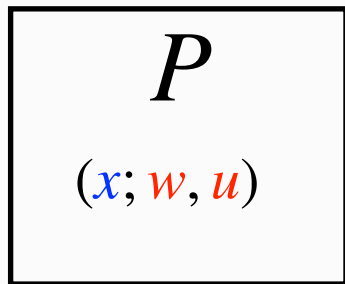
Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



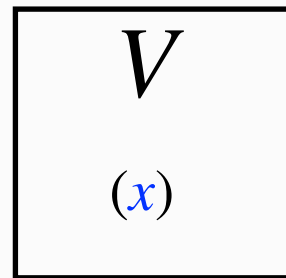
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

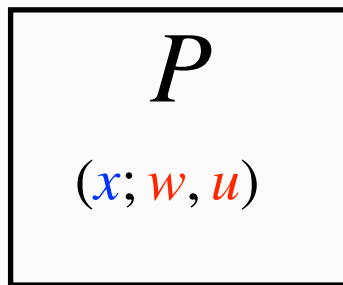
$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .



Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



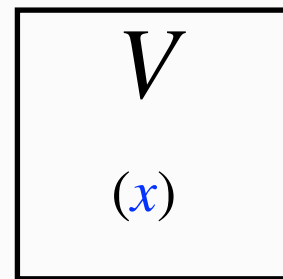
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .

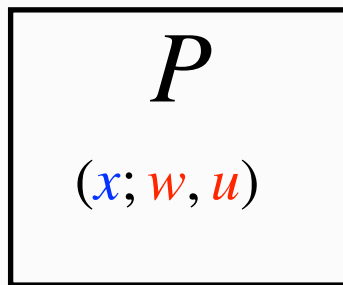


Ingredients

+

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



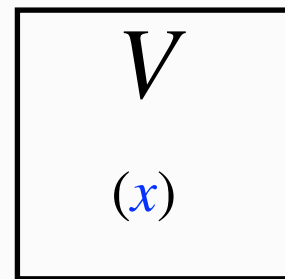
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .



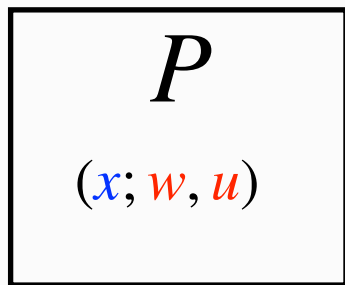
Ingredients

PIOP over \mathbb{F}_q

+

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



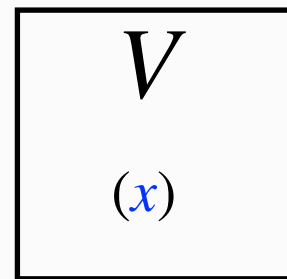
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .



Ingredients

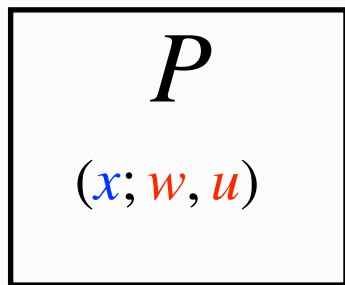
PIOP over \mathbb{F}_q

+

PCS over \mathbb{Z}

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



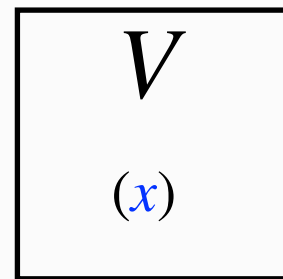
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .



Ingredients

PIOP over \mathbb{F}_q

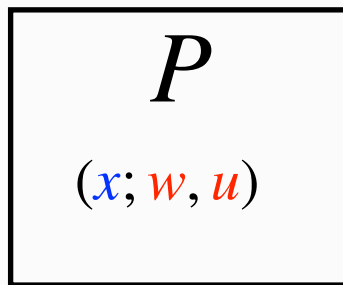
+

PCS over \mathbb{Z}

Similar to Ligero/Brakedown

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



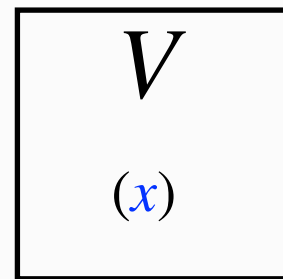
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .



Ingredients

PIOP over \mathbb{F}_q

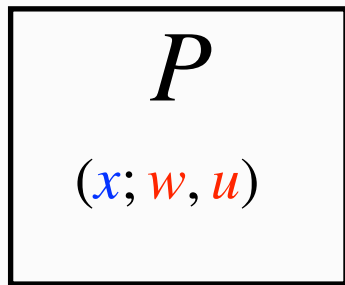
+

PCS over \mathbb{Z}

Similar to Ligero/Brakedown
But some parts run on \mathbb{Q}

Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



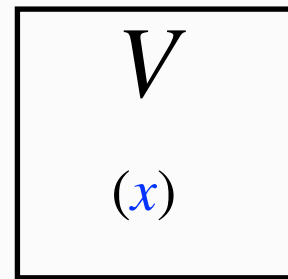
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .



Ingredients

PIOP over \mathbb{F}_q

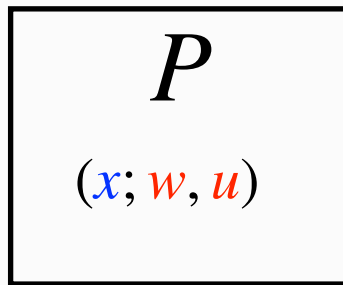
+

PCS over \mathbb{Z}

Similar to Ligero/Brakedown
But some parts run on \mathbb{Q}
Others on \mathbb{F}_q

Zinc in a nutshell \mathbb{Q}^m

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



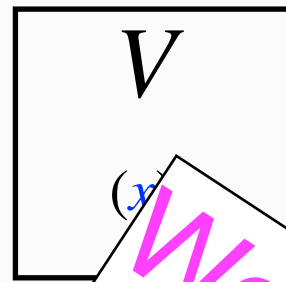
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .



Ingredients

PIOP over \mathbb{F}_q

+

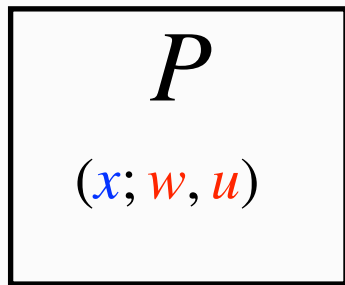
PCS over \mathbb{Q}

Sim
Be
Others

Works over \mathbb{Q}
(In fact \mathbb{Q} is the true field)

Zinc in a nutshell \mathbb{Q}^m

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_{\leq B}^m, w \in \mathbb{Z}_{\leq B}^k, u \in \mathbb{Z}_{\leq B}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$



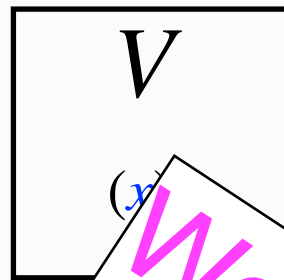
Commit to $MLE(w), MLE(u)$ over \mathbb{Z}

Random prime q

Execute a SNARK over \mathbb{F}_q for the constraint

$$Az \circ Bz = Cz + u \circ \mu \text{ modulo } q$$

When opening $MLE(w), MLE(u)$,
reduce mod q .



Ingredients

PIOP over \mathbb{F}_q

+

PCS over \mathbb{Q}

Sim
Be
Others

Works over \mathbb{Q}

(In fact \mathbb{Q} is the true field)

Also: works for any \mathbb{Z}

Initial timid attempts

Initial timid attempts

By replacing n with an arbitrary vector $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$, we capture modular arithmetic for different moduli, at the same time.

Initial timid attempts

By replacing n with an arbitrary vector $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$, we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Initial timid attempts

By replacing n with an arbitrary vector $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$, we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

R looks like a CCS relation over \mathbb{Z} . We could try to build a SNARK for it.

Initial timid attempts

By replacing n with an arbitrary vector $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$, we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

R looks like a CCS relation over \mathbb{Z} . We could try to build a SNARK for it.

Let's limit to $R_{\text{R1CS}\ell, \mathbb{Z}}$ for simplicity.

Initial timid attempts

By replacing n with an arbitrary vector $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$, we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \mu \text{ over } \mathbb{Z}, \quad \mathbf{z} = (w, x, 1) \end{array} \right. \right\}$$

R looks like a CCS relation over \mathbb{Z} . We could try to build a SNARK for it.

Let's limit to $R_{\text{R1CS}\ell, \mathbb{Z}}$ for simplicity.

First off. For technical reasons we set a bound B on the bit-size of witnesses.

Initial timid attempts

By replacing n with an arbitrary vector $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$, we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

R looks like a CCS relation over \mathbb{Z} . We could try to build a SNARK for it.

Let's limit to $R_{\text{R1CS}\ell, \mathbb{Z}}$ for simplicity.

First off. For technical reasons we set a bound B on the bit-size of witnesses.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, \\ z = (w, x, 1) \end{array} \right. \right\}$$

Where \mathbb{Z}_B is the set of integers with bit-size less than B .

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against P^* that use polys over \mathbb{Z}_B

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against P^* that use polys over \mathbb{Z}_B

- (2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against P^* that use polys over \mathbb{Z}_B

- (2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Solution: Campanelli and Hall-Andersen [CH2024]: have V sample a random prime q .

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against P^* that use polys over \mathbb{Z}_B

- (2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Solution: Campanelli and Hall-Andersen [CH2024]: have V sample a random prime q .

Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z}_B .

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against P^* that use polys over \mathbb{Z}_B

- (2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Solution: Campanelli and Hall-Andersen [CH2024]: have V sample a random prime q .

Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z}_B .

They call the resulting PIOP a *mod-PIOP* (or *mod-AHP*).

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against P^* that use polys over \mathbb{Z}_B

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Solution: Campanelli and Hall-Andersen [CH2024]: have V sample a random prime q .

Then execute Spartan over \mathbb{F}_q , rather than over \mathbb{Z}_B .

They call the resulting PIOP a *mod-PIOP* (or *mod-AHP*).

[CH2024] compile mod-PIOPs with (2) into a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

New program:

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

New program:

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

New program:

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$

Soundness holds against P^* that use polys over \mathbb{Q}_B

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

New program:

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$

Design a PCS for polynomials over \mathbb{Q}_B

Soundness holds against P^* that use polys over \mathbb{Q}_B

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over \mathbb{Z}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

New program:

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$

Design a PCS for polynomials over \mathbb{Q}_B

Soundness holds against P^* that use polys over \mathbb{Q}_B

Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for $R_{R1CS\ell, \mathbb{Z}, B}$
- (2) Design a PCS for polynomials over \mathbb{Z}_B
- Compile into succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$
-
- ```
graph LR; A["(1) Adapt Spartan PIOP (or SuperSpartan) for R_{R1CS\ell, \mathbb{Z}, B}"] --> D["Compile into succinct argument for R_{R1CS\ell, \mathbb{Z}, B}"]; B["(2) Design a PCS for polynomials over \mathbb{Z}_B"] --> D;
```

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

New program:

PIOP for  $R_{R1CS\ell, \mathbb{Q}, B}$

Design a PCS for polynomials over  $\mathbb{Q}_B$

Soundness holds against  $P^*$  that use polys over  $\mathbb{Q}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Q}, B}$

```
graph LR; C["PIOP for R_{R1CS\ell, \mathbb{Q}, B}"] --> E["Compile into succinct argument for R_{R1CS\ell, \mathbb{Q}, B}"]; D["Design a PCS for polynomials over \mathbb{Q}_B"] --> E;
```

# Moving to the field of rational numbers



# Moving to the field of rational numbers

Designing a PCS over  $\mathbb{Z}_B$  is hard because  $\mathbb{Z}$  is not a field.

# Moving to the field of rational numbers

Designing a PCS over  $\mathbb{Z}_B$  is hard because  $\mathbb{Z}$  is not a field.

We make the following **key design choice**:

# Moving to the field of rational numbers

Designing a PCS over  $\mathbb{Z}_B$  is hard because  $\mathbb{Z}$  is not a field.

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

# Moving to the field of rational numbers

Designing a PCS over  $\mathbb{Z}_B$  is hard because  $\mathbb{Z}$  is not a field.

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Q}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

# Moving to the field of rational numbers

Designing a PCS over  $\mathbb{Z}_B$  is hard because  $\mathbb{Z}$  is not a field.

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Q}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

**Note:** Now  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$  no longer captures arbitrary modular arithmetic.

# Moving to the field of rational numbers

Designing a PCS over  $\mathbb{Z}_B$  is hard because  $\mathbb{Z}$  is not a field.

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Q}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

**Note:** Now  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$  no longer captures arbitrary modular arithmetic.

Because of this, we also design a **lookup argument over  $\mathbb{Q}_B$** . I.e. an argument for

# Moving to the field of rational numbers

Designing a PCS over  $\mathbb{Z}_B$  is hard because  $\mathbb{Z}$  is not a field.

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Q}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

**Note:** Now  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$  no longer captures arbitrary modular arithmetic.

Because of this, we also design a **lookup argument over  $\mathbb{Q}_B$** . I.e. an argument for

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

# Lookup arguments over the rational numbers



# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \mid \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right\}$$

# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

Then an argument for  $R_{\text{Look}, \mathbb{Q}, B}$  enforces  $a$  to contain entries from  $\mathbb{Z}_B$ .

# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

Then an argument for  $R_{\text{Look}, \mathbb{Q}, B}$  enforces  $a$  to contain entries from  $\mathbb{Z}_B$ .

Argument  
for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$

# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

Then an argument for  $R_{\text{Look}, \mathbb{Q}, B}$  enforces  $a$  to contain entries from  $\mathbb{Z}_B$ .

Argument  
for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$  +

# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

Then an argument for  $R_{\text{Look}, \mathbb{Q}, B}$  enforces  $a$  to contain entries from  $\mathbb{Z}_B$ .

Argument  
for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$

+

Argument  
for  $R_{\text{Look}, \mathbb{Q}, B}$

# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

Then an argument for  $R_{\text{Look}, \mathbb{Q}, B}$  enforces  $a$  to contain entries from  $\mathbb{Z}_B$ .

$$\boxed{\begin{array}{c} \text{Argument} \\ \text{for } R_{\text{R1CS}\ell, \mathbb{Q}, B} \end{array}} + \boxed{\begin{array}{c} \text{Argument} \\ \text{for } R_{\text{Look}, \mathbb{Q}, B} \end{array}} =$$

# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

Then an argument for  $R_{\text{Look}, \mathbb{Q}, B}$  enforces  $a$  to contain entries from  $\mathbb{Z}_B$ .

|                                                      |   |                                                  |   |                                                  |
|------------------------------------------------------|---|--------------------------------------------------|---|--------------------------------------------------|
| Argument<br>for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ | + | Argument<br>for $R_{\text{Look}, \mathbb{Q}, B}$ | = | Argument<br>for $R_{\text{R1CS}, \mathbb{Z}, B}$ |
|------------------------------------------------------|---|--------------------------------------------------|---|--------------------------------------------------|



# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

Then an argument for  $R_{\text{Look}, \mathbb{Q}, B}$  enforces  $a$  to contain entries from  $\mathbb{Z}_B$ .

$$\boxed{\begin{array}{c} \text{Argument} \\ \text{for } R_{\text{R1CS}\ell, \mathbb{Q}, B} \end{array}} + \boxed{\begin{array}{c} \text{Argument} \\ \text{for } R_{\text{Look}, \mathbb{Q}, B} \end{array}} = \boxed{\begin{array}{c} \text{Argument} \\ \text{for } R_{\text{R1CS}, \mathbb{Z}, B} \end{array}}$$

In our work we are general and describe an argument for any relation over  $\mathbb{Q}_B$  that can be expressed algebraically.

# Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set  $t = [-2^B, 2^B] \cap \mathbb{Z}$ .

Then an argument for  $R_{\text{Look}, \mathbb{Q}, B}$  enforces  $a$  to contain entries from  $\mathbb{Z}_B$ .

$$\boxed{\text{Argument for } R_{\text{R1CS}\ell, \mathbb{Q}, B}} + \boxed{\text{Argument for } R_{\text{Look}, \mathbb{Q}, B}} = \boxed{\text{Argument for } R_{\text{R1CS}, \mathbb{Z}, B}}$$

In our work we are general and describe an argument for any relation over  $\mathbb{Q}_B$  that can be expressed algebraically.

This provides arguments for both  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$  and  $R_{\text{Look}, \mathbb{Q}, B}$ .

# The mod-PIOP technique

# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.  
First, let's see how [CH2024] does that over the integers.

# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

$$\boxed{\begin{array}{c} P \\ (x; w, u) \end{array}}$$

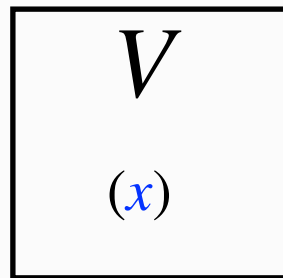
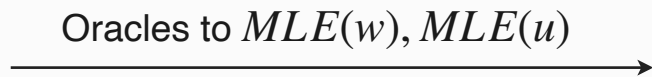
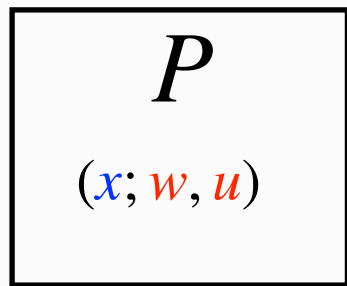
$$\boxed{\begin{array}{c} V \\ (x) \end{array}}$$

# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



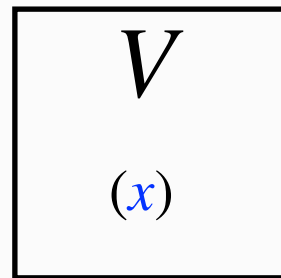
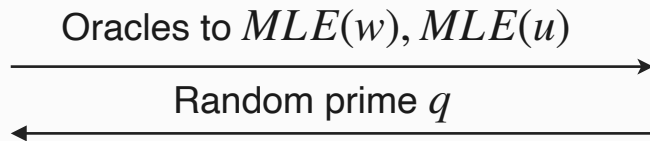
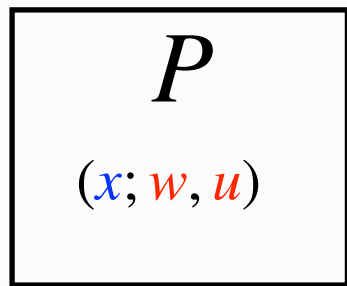


# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

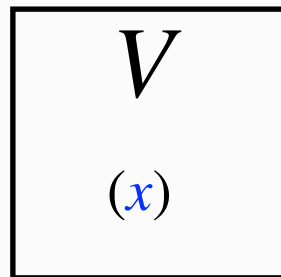
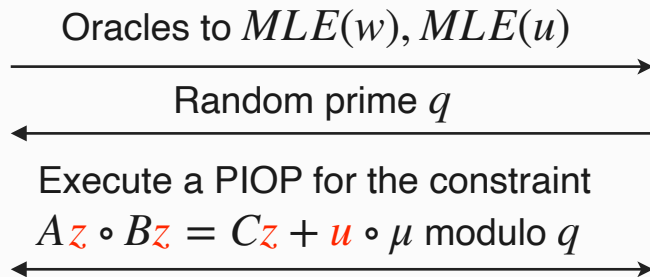
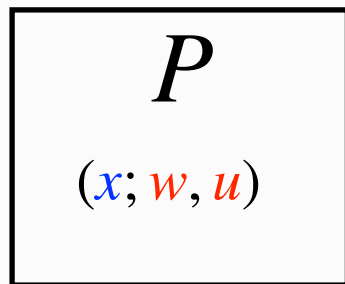


# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

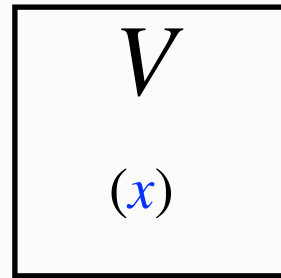
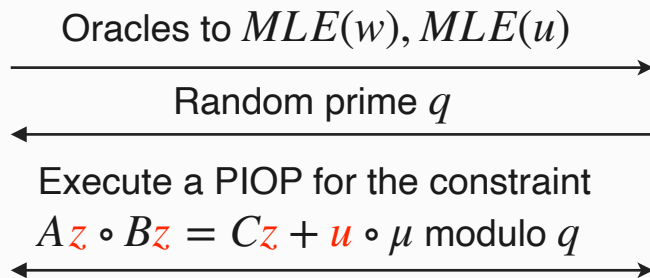
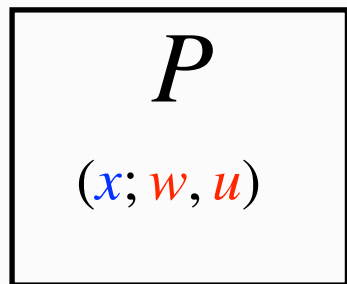


# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



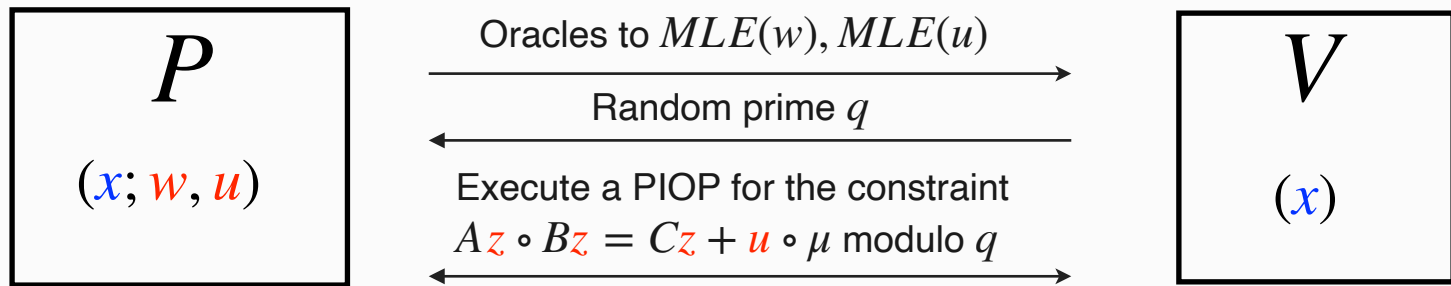
For the latter, it suffices to use the usual version of (Super)Spartan.

# The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

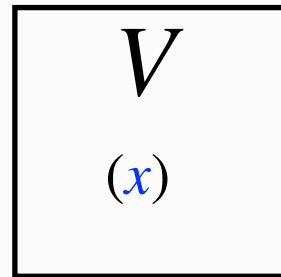
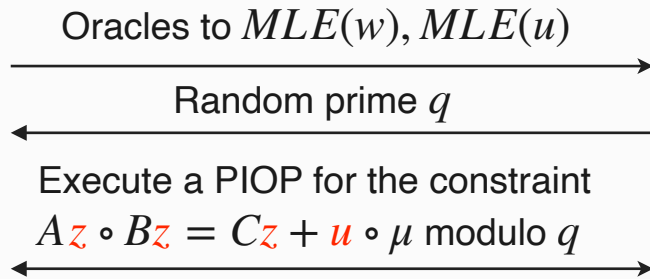
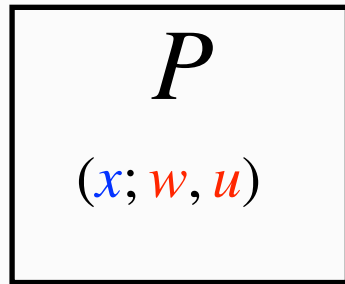


For the latter, it suffices to use the usual version of (Super)Spartan.

When  $MLE(w), MLE(u)$  are queried,  $V$  receives a value in  $\mathbb{Z}$  and reduces it mod  $q$

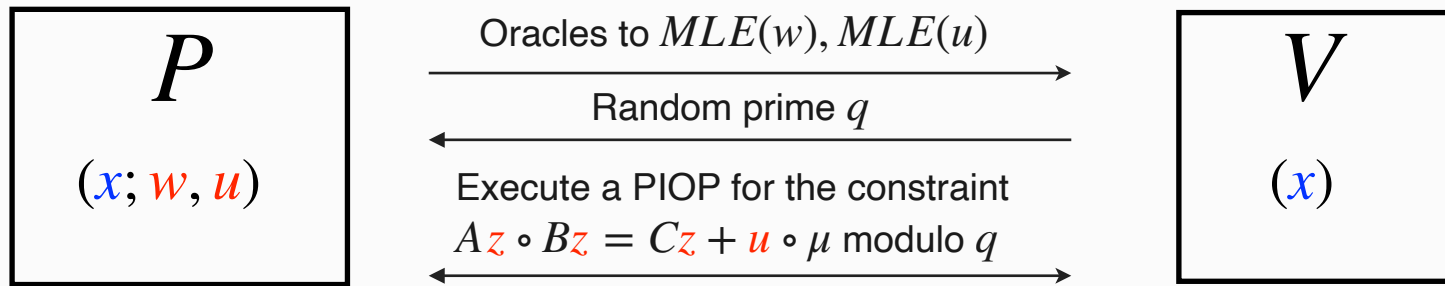
# The mod-PIOP technique

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



# The mod-PIOP technique

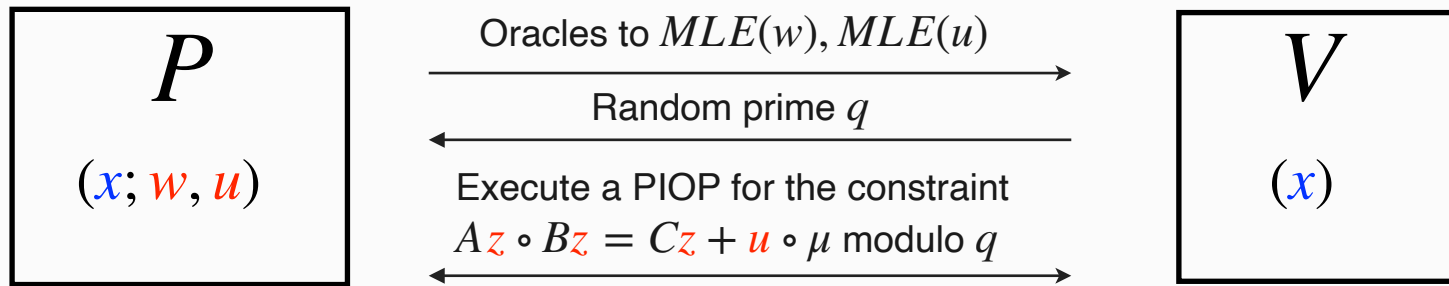
$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



This PIOP is sound against  $P^*$  that send oracles to  $MLE(w), MLE(u)$  with entries in  $\mathbb{Z}_B$ .

# The mod-PIOP technique

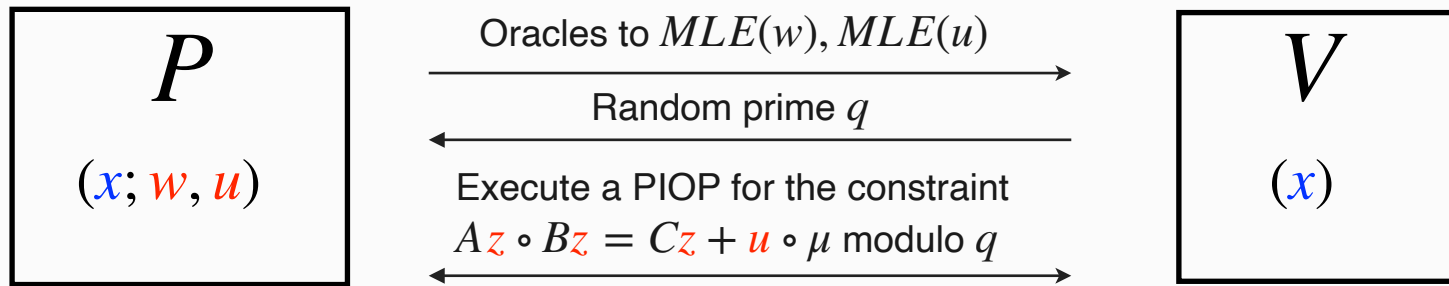
$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



This PIOP is sound against  $P^*$  that send oracles to  $MLE(w), MLE(u)$  with entries in  $\mathbb{Z}_B$ .  
(Assuming the PIOP in the last step is sound).

# The mod-PIOP technique

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



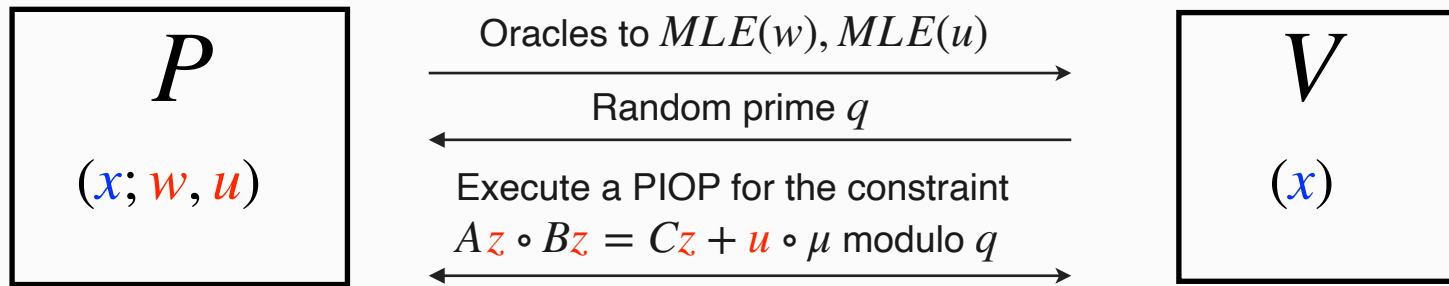
This PIOP is sound against  $P^*$  that send oracles to  $MLE(w), MLE(u)$  with entries in  $\mathbb{Z}_B$ . (Assuming the PIOP in the last step is sound).

**Why?** Suppose  $Az \circ Bz \neq Cz + u \circ \mu$  but that  $Az \circ Bz = Cz + u \circ \mu$  modulo  $q$  for many primes  $q$ . Then one can see that some entry in  $z, u$  is divisible by many primes.



# The mod-PIOP technique

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



This PIOP is sound against  $P^*$  that send oracles to  $MLE(w), MLE(u)$  with entries in  $\mathbb{Z}_B$ . (Assuming the PIOP in the last step is sound).

**Why?** Suppose  $Az \circ Bz \neq Cz + u \circ \mu$  but that  $Az \circ Bz = Cz + u \circ \mu$  modulo  $q$  for many primes  $q$ . Then one can see that some entry in  $z, u$  is divisible by many primes.

But the entries of  $z, u$  have at most  $B$  bits.

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

We focus on  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ , but the techniques apply to any algebraic relation over  $\mathbb{Q}_B$

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

We focus on  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ , but the techniques apply to any algebraic relation over  $\mathbb{Q}_B$

We will obtain a succinct argument for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ .

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

We focus on  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ , but the techniques apply to any algebraic relation over  $\mathbb{Q}_B$

We will obtain a succinct argument for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ .

We start by designing a PIOP over  $\mathbb{Q}_B$  for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ . Recall:

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

We focus on  $R_{R1CS\ell, \mathbb{Q}, B}$ , but the techniques apply to any algebraic relation over  $\mathbb{Q}_B$

We will obtain a succinct argument for  $R_{R1CS\ell, \mathbb{Q}, B}$ .

We start by designing a PIOP over  $\mathbb{Q}_B$  for  $R_{R1CS\ell, \mathbb{Q}, B}$ . Recall:

$$R_{R1CS\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \mu \text{ over } \mathbb{Q}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

We focus on  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ , but the techniques apply to any algebraic relation over  $\mathbb{Q}_B$

We will obtain a succinct argument for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ .

We start by designing a PIOP over  $\mathbb{Q}_B$  for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ . Recall:

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \mu \text{ over } \mathbb{Q}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

We would like to use the mod-PIOP idea from Campanelli and Hall-Andersen.

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

We focus on  $R_{R1CS\ell, \mathbb{Q}, B}$ , but the techniques apply to any algebraic relation over  $\mathbb{Q}_B$

We will obtain a succinct argument for  $R_{R1CS\ell, \mathbb{Q}, B}$ .

We start by designing a PIOP over  $\mathbb{Q}_B$  for  $R_{R1CS\ell, \mathbb{Q}, B}$ . Recall:

$$R_{R1CS\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \mu \text{ over } \mathbb{Q}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

We would like to use the mod-PIOP idea from Campanelli and Hall-Andersen.

But reduction modulo a prime is not well-defined:  $w, u$  can contain rational entries.



# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

We focus on  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ , but the techniques apply to any algebraic relation over  $\mathbb{Q}_B$

We will obtain a succinct argument for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ .

We start by designing a PIOP over  $\mathbb{Q}_B$  for  $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ . Recall:

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$

We would like to use the mod-PIOP idea from Campanelli and Hall-Andersen.

But reduction modulo a prime is not well-defined:  $w, u$  can contain rational entries.

We use the concept of **local subrings** of  $\mathbb{Q}$ .

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

Given a prime  $q$ , define

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

Given a prime  $q$ , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

Given a prime  $q$ , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$  is a subring of  $\mathbb{Q}$ , called the **localization of  $\mathbb{Q}$  on  $q$** .

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

Given a prime  $q$ , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$  is a subring of  $\mathbb{Q}$ , called the **localization of  $\mathbb{Q}$  on  $q$** .

There is a ring homomorphism.

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

Given a prime  $q$ , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$  is a subring of  $\mathbb{Q}$ , called the **localization of  $\mathbb{Q}$  on  $q$** .

There is a ring homomorphism.

$$\phi_q : \mathbb{Z}_{(q)} \rightarrow \mathbb{F}_q \qquad a/b \mapsto a \cdot b^{-1} \bmod q$$

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

Given a prime  $q$ , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$  is a subring of  $\mathbb{Q}$ , called the **localization of  $\mathbb{Q}$  on  $q$** .

There is a ring homomorphism.

$$\phi_q : \mathbb{Z}_{(q)} \rightarrow \mathbb{F}_q \quad a/b \mapsto a \cdot b^{-1} \bmod q$$

where  $b^{-1}$  denotes an inverse of  $b \bmod q$ .



# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

Given a prime  $q$ , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$  is a subring of  $\mathbb{Q}$ , called the **localization of  $\mathbb{Q}$  on  $q$** .

There is a ring homomorphism.

$$\phi_q : \mathbb{Z}_{(q)} \rightarrow \mathbb{F}_q \quad a/b \mapsto a \cdot b^{-1} \bmod q$$

where  $b^{-1}$  denotes an inverse of  $b \bmod q$ .

So, reduction mod  $q$  has a natural meaning for most rational numbers.

# Zinc-PIOP: A PIOP for relations over $\mathbb{Q}_B$

Given a prime  $q$ , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$  is a subring of  $\mathbb{Q}$ , called the **localization of  $\mathbb{Q}$  on  $q$** .

There is a ring homomorphism.

$$\phi_q : \mathbb{Z}_{(q)} \rightarrow \mathbb{F}_q \quad a/b \mapsto a \cdot b^{-1} \bmod q$$

where  $b^{-1}$  denotes an inverse of  $b \bmod q$ .

So, reduction mod  $q$  has a natural meaning for most rational numbers.

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod $pq$                          | Zinc                |
| Mult mod non-native $q$                | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

# Towards universal arithmetization

Some use-cases involve a mix of computation types.

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod $pq$                          | Zinc                |
| Mult mod non-native $q$                | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod pq                            | Zinc                |
| Mult mod non-native q                  | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

Some use-cases involve a mix of computation types.

Examples:

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod pq                            | Zinc                |
| Mult mod non-native q                  | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod $pq$                          | Zinc                |
| Mult mod non-native $q$                | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod pq                            | Zinc                |
| Mult mod non-native q                  | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

Some use-cases involve a mix of computation types.

## Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS



# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod pq                            | Zinc                |
| Mult mod non-native q                  | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS

What to do then?

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod $pq$                          | Zinc                |
| Mult mod non-native $q$                | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS

What to do then?

- Zinc handles multiple moduli,  $Z$  and  $Q$  at once

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod $pq$                          | Zinc                |
| Mult mod non-native $q$                | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS

What to do then?

- Zinc handles multiple moduli,  $Z$  and  $Q$  at once
- But how about bitwise ops and modular arithmetic?

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod $pq$                          | Zinc                |
| Mult mod non-native $q$                | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

- If types of computation is clearly separated (e.g. hash+sign):

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS

What to do then?

- Zinc handles multiple moduli,  $Z$  and  $Q$  at once
- But how about bitwise ops and modular arithmetic?

# Towards universal arithmetization

| Computation                            | Native proof system |
|----------------------------------------|---------------------|
| Rationals, fixed point, floating point | Zinc                |
| Mult mod $2^n$                         | Zinc                |
| Mult mod pq                            | Zinc                |
| Mult mod non-native q                  | Zinc                |
| XOR, NOT, etc.                         | Binius              |
| Lattices                               | Dedicated SNARK     |
| Classic hash (SHA)                     | Binius (partially)  |

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS

What to do then?

- Zinc handles multiple moduli,  $Z$  and  $Q$  at once
- But how about bitwise ops and modular arithmetic?

- If types of computation is clearly separated (e.g. hash+sign):
  - Use two proof systems: binary field (SHA256), prime field (ECDSA) (e.g. Frigo and Shelat, Anonymous credentials for ECDSA, 2025)

# Towards universal arithmetization

| Computation                            | Native proof system               |
|----------------------------------------|-----------------------------------|
| Rationals, fixed point, floating point | Zinc                              |
| Mult mod $2^n$                         | Zinc                              |
| Mult mod pq                            | Zinc                              |
| Mult mod non-native q                  | Zinc                              |
| XOR, NOT, etc.                         | Binius, Zinc?                     |
| Lattices                               | Dedicated SNARK, Zinc?            |
| Classic hash (SHA)                     | Binius, Zinc?<br>(Both partially) |

Some use-cases involve a mix of computation types.

Examples:

- zkVM instruction sequences
- zkID: SHA-256 + RSA/ECDSA signature
- AES encryption, TLS

What to do then?

- Zinc handles multiple moduli, Z and Q at once
- But how about bitwise ops and modular arithmetic?

- If types of computation is clearly separated (e.g. hash+sign):
  - Use two proof systems: binary field (SHA256), prime field (ECDSA) (e.g. Frigo and Shelat, Anonymous credentials for ECDSA, 2025)
  - Something else.

# Universal arithmetization

# Universal arithmetization

Case study: bitwise operations (XOR)



# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$Az \circ Bz = Cz \text{ over } \mathbb{Z}/n\mathbb{Z}$$

# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$Az \circ Bz = Cz \text{ over } \mathbb{Z}/n\mathbb{Z}$$



# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $\mathbf{u} \in \mathbb{Z}^{m+k+1}$  such that

# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $\mathbf{u} \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + \mathbf{u} \cdot n$  as integers

# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $\mathbf{u} \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + \mathbf{u} \cdot n$  as integers

XOR operations = addition in binary fields:

# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $\mathbf{u} \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + \mathbf{u} \cdot n$  as integers

XOR operations = addition in binary fields:

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $\mathbf{u} \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + \mathbf{u} \cdot n$  as integers

XOR operations = addition in binary fields:

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

Elements of  $GF(2^n)$  are polynomials of degree  $< n$ .



# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbb{Z} \circ B\mathbb{Z} = C\mathbb{Z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $u \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbb{Z} \circ B\mathbb{Z} = C\mathbb{Z} + u \cdot n$  as integers

XOR operations = addition in binary fields:

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

Elements of  $GF(2^n)$  are polynomials of degree  $< n$ .

$$A\mathbb{Z} \circ B\mathbb{Z} = C\mathbb{Z} \text{ over } GF(2^n)$$

# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $\mathbf{u} \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + \mathbf{u} \cdot n$  as integers

XOR operations = addition in binary fields:

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

Elements of  $GF(2^n)$  are polynomials of degree  $< n$ .

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $\mathbf{u} \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + \mathbf{u} \cdot n$  as integers

XOR operations = addition in binary fields:

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

Elements of  $GF(2^n)$  are polynomials of degree  $< n$ .

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $\mathbf{u}, \mathbf{v} \in \mathbb{Z}[X]^{m+k+1}$   
such that

# Universal arithmetization

Case study: bitwise operations (XOR)

Recall:

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $\mathbf{u} \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + \mathbf{u} \cdot n$  as integers

XOR operations = addition in binary fields:

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

Elements of  $GF(2^n)$  are polynomials of degree  $< n$ .

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $\mathbf{u}, \mathbf{v} \in \mathbb{Z}[X]^{m+k+1}$   
such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + \mathbf{u} \cdot 2 + \mathbf{v} \cdot f$   
as integer polynomials

# Universal arithmetization

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $u, v \in \mathbb{Z}[X]^{m+k+1}$   
such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot 2 + v \cdot f(X)$   
as integer polynomials

# Universal arithmetization

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $u, v \in \mathbb{Z}[X]^{m+k+1}$   
such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot 2 + v \cdot f(X)$   
as integer polynomials

We can mix in operations mod  $2^n$  (and others!).

# Universal arithmetization

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $u, v \in \mathbb{Z}[X]^{m+k+1}$   
such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot 2 + v \cdot f(X)$   
as integer polynomials

We can mix in operations mod  $2^n$  (and others!).

Let  $\mu, \eta$  be public vectors with entries in  $\mathbb{Z}[X]$ .

# Universal arithmetization

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $u, v \in \mathbb{Z}[X]^{m+k+1}$   
such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot 2 + v \cdot f(X)$   
as integer polynomials

We can mix in operations mod  $2^n$  (and others!).

Let  $\mu, \eta$  be public vectors with entries in  $\mathbb{Z}[X]$ .

Then  $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \vec{\mu} + v \circ \vec{\eta}$  over  $\mathbb{Z}[X]$  can encode “everything from everywhere all at once”



# Universal arithmetization

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $u, v \in \mathbb{Z}[X]^{m+k+1}$   
such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot 2 + v \cdot f(X)$   
as integer polynomials

We can mix in operations mod  $2^n$  (and others!).

Let  $\mu, \eta$  be public vectors with entries in  $\mathbb{Z}[X]$ .

Then  $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \vec{\mu} + v \circ \vec{\eta}$  over  $\mathbb{Z}[X]$  can encode “everything from everywhere all at once”

I.e. each row can encode constraints over any of:

# Universal arithmetization

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $u, v \in \mathbb{Z}[X]^{m+k+1}$   
such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot 2 + v \cdot f(X)$   
as integer polynomials

We can mix in operations mod  $2^n$  (and others!).

Let  $\mu, \eta$  be public vectors with entries in  $\mathbb{Z}[X]$ .

Then  $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \vec{\mu} + v \circ \vec{\eta}$  over  $\mathbb{Z}[X]$  can encode “everything from everywhere all at once”

I.e. each row can encode constraints over any of:

$GF(2^n)$ ,  $\mathbb{Z}/2^n\mathbb{Z}$ , Cyclotomic rings, Galois rings, any finite field,  $\mathbb{Z}/n\mathbb{Z}$ .

# Universal arithmetization

$$GF(2^n) = \mathbb{F}_2[X]/(f(X)) = \mathbb{Z}[X]/(2, f(X))$$

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } GF(2^n)$$



There exists  $u, v \in \mathbb{Z}[X]^{m+k+1}$   
such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot 2 + v \cdot f(X)$   
as integer polynomials

We can mix in operations mod  $2^n$  (and others!).

Let  $\mu, \eta$  be public vectors with entries in  $\mathbb{Z}[X]$ .

Then  $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot \vec{\mu} + v \cdot \vec{\eta}$  over  $\mathbb{Z}[X]$  can encode “everything from everywhere all at once”

I.e. each row can encode constraints over any of:

$GF(2^n)$ ,  $\mathbb{Z}/2^n\mathbb{Z}$ , Cyclotomic rings, Galois rings, any finite field,  $\mathbb{Z}/n\mathbb{Z}$ .

So: a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$  has universal arithmetization properties

# Zinc+

# Zinc+

$$R_{\text{R1CS}\ell, \mathbb{Z}[X]} = \left\{ (x; w, u, v) \left| \begin{array}{l} x \in \mathbb{Z}[X]^m, w \in \mathbb{Z}[X]^k, u, v \in \mathbb{Z}[X]^{m+k+1} \\ A\vec{z} \circ B\vec{z} = C\vec{z} + u \circ \vec{\mu} + v \cdot \vec{\eta} \text{ over } \mathbb{Z}[X], \quad \vec{z} = (w, x, 1) \end{array} \right. \right\}$$

# Zinc+

$$R_{\text{R1CS}\ell, \mathbb{Z}[X]} = \left\{ (x; w, u, v) \left| \begin{array}{l} x \in \mathbb{Z}[X]^m, w \in \mathbb{Z}[X]^k, u, v \in \mathbb{Z}[X]^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{\mu} + v \cdot \vec{\eta} \text{ over } \mathbb{Z}[X], \quad \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

$$\boxed{\begin{array}{c} P \\ (x; w, u) \end{array}}$$

$$\boxed{\begin{array}{c} V \\ (x) \end{array}}$$

# Zinc+

$$R_{\text{R1CS}\ell, \mathbb{Z}[X]} = \left\{ (x; w, u, v) \left| \begin{array}{l} x \in \mathbb{Z}[X]^m, w \in \mathbb{Z}[X]^k, u, v \in \mathbb{Z}[X]^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{\mu} + v \cdot \vec{\eta} \text{ over } \mathbb{Z}[X], \quad \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

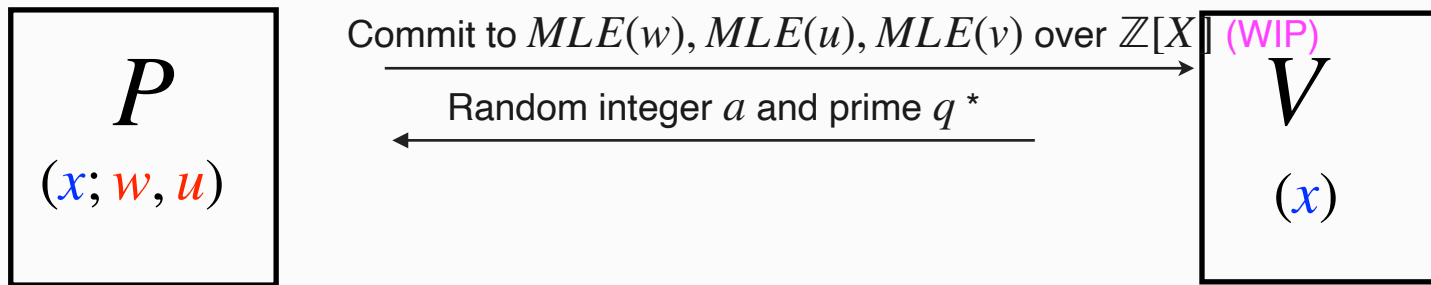
$$\boxed{\begin{array}{c} P \\ (x; w, u) \end{array}}$$

Commit to  $MLE(w), MLE(u), MLE(v)$  over  $\mathbb{Z}[X]$

$$\xrightarrow{\quad} \boxed{\begin{array}{c} \text{(WIP)} \\ V \\ (x) \end{array}}$$

# Zinc+

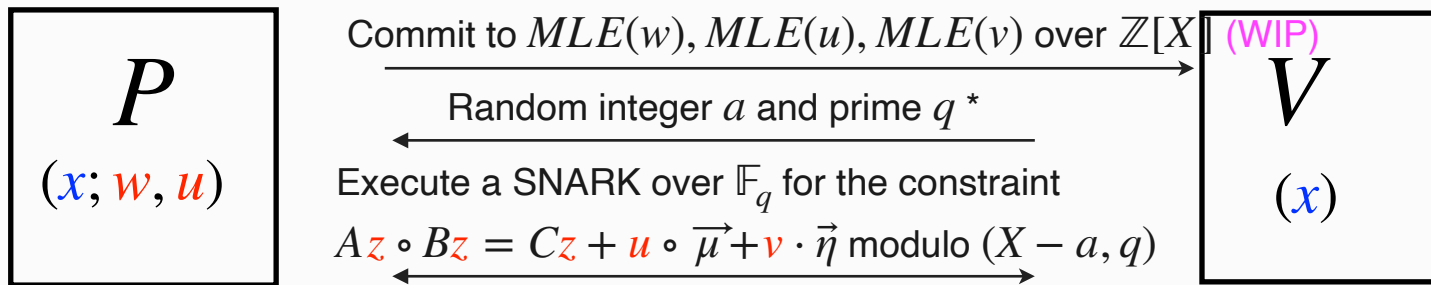
$$R_{\text{R1CS}\ell, \mathbb{Z}[X]} = \left\{ (x; w, u, v) \left| \begin{array}{l} x \in \mathbb{Z}[X]^m, w \in \mathbb{Z}[X]^k, u, v \in \mathbb{Z}[X]^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{\mu} + v \cdot \vec{\eta} \text{ over } \mathbb{Z}[X], \quad \bar{z} = (w, x, 1) \end{array} \right. \right\}$$





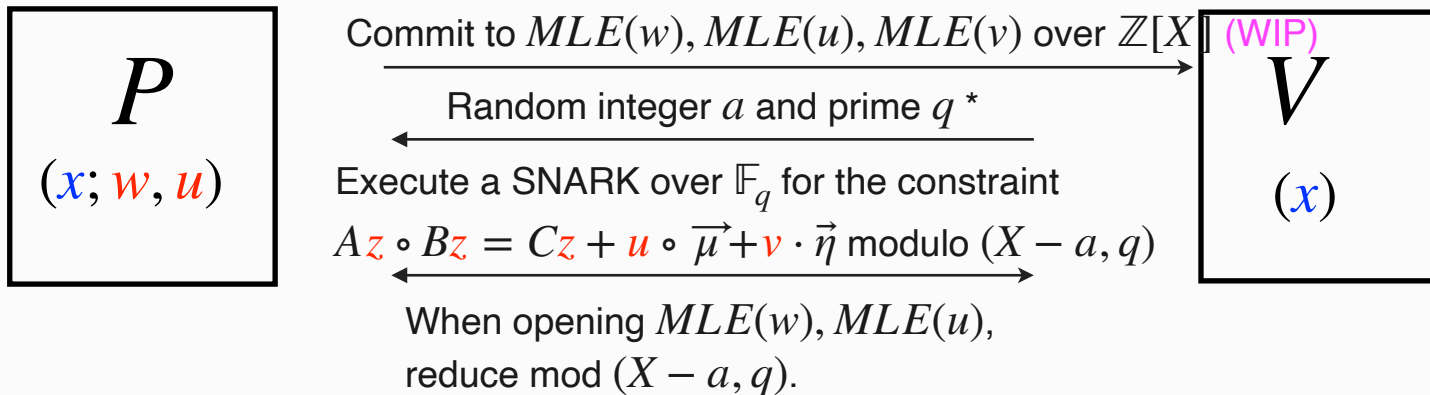
# Zinc+

$$R_{\text{R1CS}\ell, \mathbb{Z}[X]} = \left\{ (x; w, u, v) \left| \begin{array}{l} x \in \mathbb{Z}[X]^m, w \in \mathbb{Z}[X]^k, u, v \in \mathbb{Z}[X]^{m+k+1} \\ A z \circ B z = C z + u \circ \vec{\mu} + v \cdot \vec{\eta} \text{ over } \mathbb{Z}[X], \quad z = (w, x, 1) \end{array} \right. \right\}$$



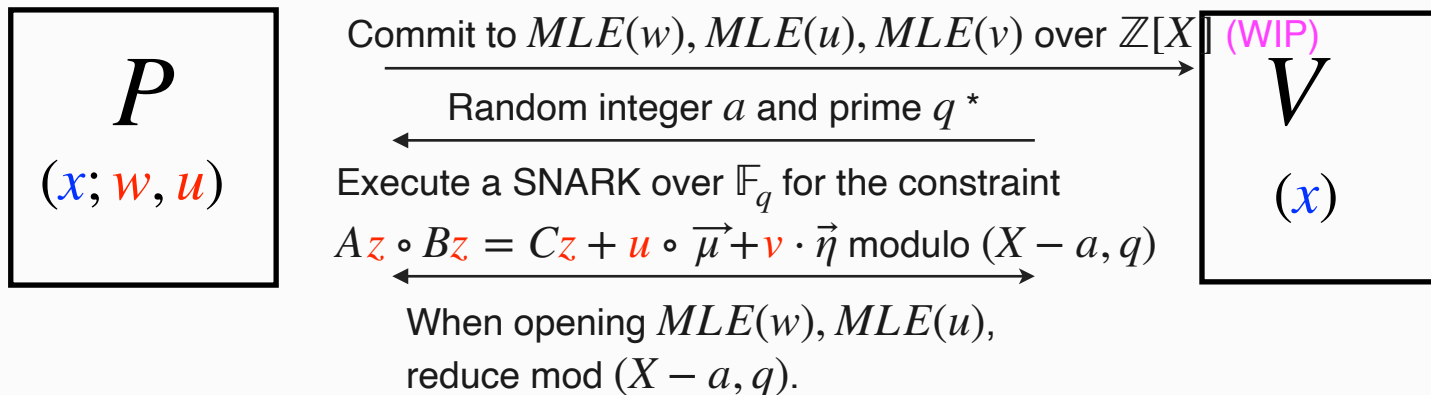
# Zinc+

$$R_{\text{R1CS}\ell, \mathbb{Z}[X]} = \left\{ (x; w, u, v) \left| \begin{array}{l} x \in \mathbb{Z}[X]^m, w \in \mathbb{Z}[X]^k, u, v \in \mathbb{Z}[X]^{m+k+1} \\ A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \vec{\mu} + v \cdot \vec{\eta} \text{ over } \mathbb{Z}[X], \quad \mathbf{z} = (w, x, 1) \end{array} \right. \right\}$$



# Zinc+

$$R_{\text{R1CS}\ell, \mathbb{Z}[X]} = \left\{ (x; w, u, v) \left| \begin{array}{l} x \in \mathbb{Z}[X]^m, w \in \mathbb{Z}[X]^k, u, v \in \mathbb{Z}[X]^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \vec{\mu} + v \cdot \vec{\eta} \text{ over } \mathbb{Z}[X], \quad \bar{z} = (w, x, 1) \end{array} \right. \right\}$$



Reduction mod  $(X - a, q)$  is the same as replacing  $X$  by  $a$  and reducing mod  $q$ .  
This gets you to  $\mathbb{F}_q$

$$= \left\{ (\textcolor{blue}{x}; \textcolor{red}{w}) \mid \begin{aligned} & \textcolor{blue}{x} \in \mathbb{F}^n, \textcolor{red}{w} \in \mathbb{F}^m \\ & \textcolor{red}{z} \in A \textcolor{red}{z} \circ B \textcolor{red}{z} = C \textcolor{red}{z}, \textcolor{red}{z} = (\textcolor{red}{w}, \textcolor{blue}{x}, 1) \end{aligned} \right\}$$

# Zinc+

# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .

# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .
- Again we can run our schemes by making a random projection

# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .
- Again we can run our schemes by making a random projection

$$\mathbb{Z}[X] \rightarrow \mathbb{F}$$



# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .
- Again we can run our schemes by making a random projection

$$\mathbb{Z}[X] \rightarrow \mathbb{F}$$

where  $\mathbb{F}$  is a prime field.

# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .
- Again we can run our schemes by making a random projection

$$\mathbb{Z}[X] \rightarrow \mathbb{F}$$

where  $\mathbb{F}$  is a prime field.

- Technically, we mod  $\mathbb{Z}[X]$  out by an ideal  $(q, X - a)$  generated by a random prime  $q$  and  $X - a$  where  $a$  is a random integer.

# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .
- Again we can run our schemes by making a random projection

$$\mathbb{Z}[X] \rightarrow \mathbb{F}$$

where  $\mathbb{F}$  is a prime field.

- Technically, we mod  $\mathbb{Z}[X]$  out by an ideal  $(q, X - a)$  generated by a random prime  $q$  and  $X - a$  where  $a$  is a random integer.
- In other words, replace  $X$  by  $a$  and reduce mod  $q$ .

# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .
- Again we can run our schemes by making a random projection

$$\mathbb{Z}[X] \rightarrow \mathbb{F}$$

where  $\mathbb{F}$  is a prime field.

- Technically, we mod  $\mathbb{Z}[X]$  out by an ideal  $(q, X - a)$  generated by a random prime  $q$  and  $X - a$  where  $a$  is a random integer.
- In other words, replace  $X$  by  $a$  and reduce mod  $q$ .
- The parts of the scheme that occur in  $\mathbb{Z}[X]$  need to be handled with care.

# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .
- Again we can run our schemes by making a random projection

$$\mathbb{Z}[X] \rightarrow \mathbb{F}$$

where  $\mathbb{F}$  is a prime field.

- Technically, we mod  $\mathbb{Z}[X]$  out by an ideal  $(q, X - a)$  generated by a random prime  $q$  and  $X - a$  where  $a$  is a random integer.
- In other words, replace  $X$  by  $a$  and reduce mod  $q$ .
- The parts of the scheme that occur in  $\mathbb{Z}[X]$  need to be handled with care.
- But it seems that we don't have blowouts on the witness size (i.e. costs stay always close to the witness bit-size, with a small constant).

# Zinc+

- Zinc+ will be a SNARK for constraints over  $\mathbb{Z}[X]$  or  $\mathbb{Q}[X]$ .
- Again we can run our schemes by making a random projection

$$\mathbb{Z}[X] \rightarrow \mathbb{F}$$

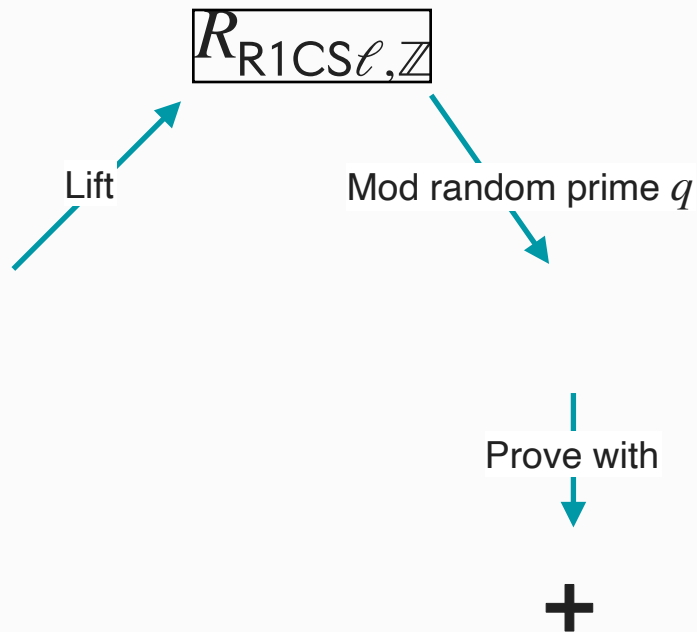
where  $\mathbb{F}$  is a prime field.

- Technically, we mod  $\mathbb{Z}[X]$  out by an ideal  $(q, X - a)$  generated by a random prime  $q$  and  $X - a$  where  $a$  is a random integer.
- In other words, replace  $X$  by  $a$  and reduce mod  $q$ .
- The parts of the scheme that occur in  $\mathbb{Z}[X]$  need to be handled with care.
- But it seems that we don't have blowouts on the witness size (i.e. costs stay always close to the witness bit-size, with a small constant).
- We hope to have a PoC implementation of Zinc+ in about 3 months.

# WIP: More improvements

Avoid  $V$  sampling a prime. Sample just an integer

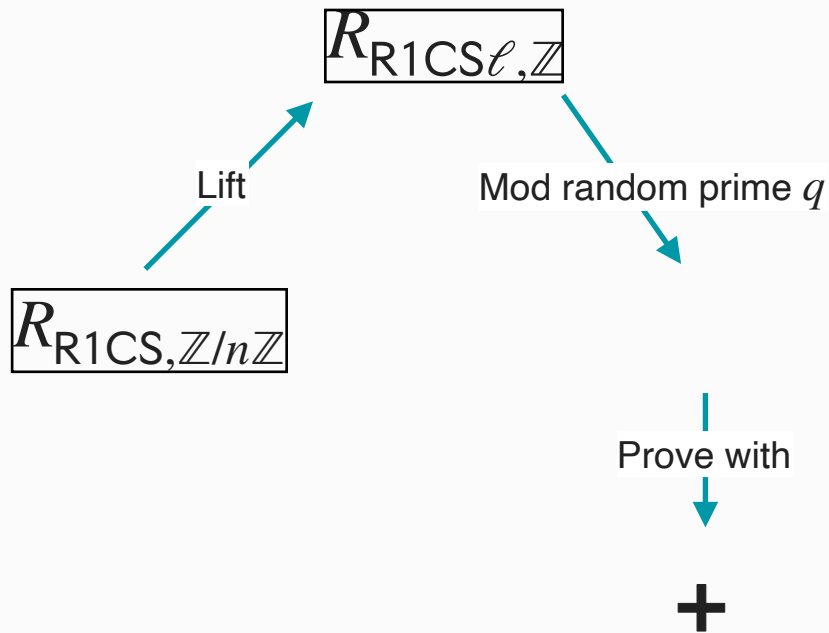
# Zinc in a nutshell



Similar to Ligero/Brakedown  
But some parts run on  $\mathbb{Q}$   
Others on  $\mathbb{F}_q$

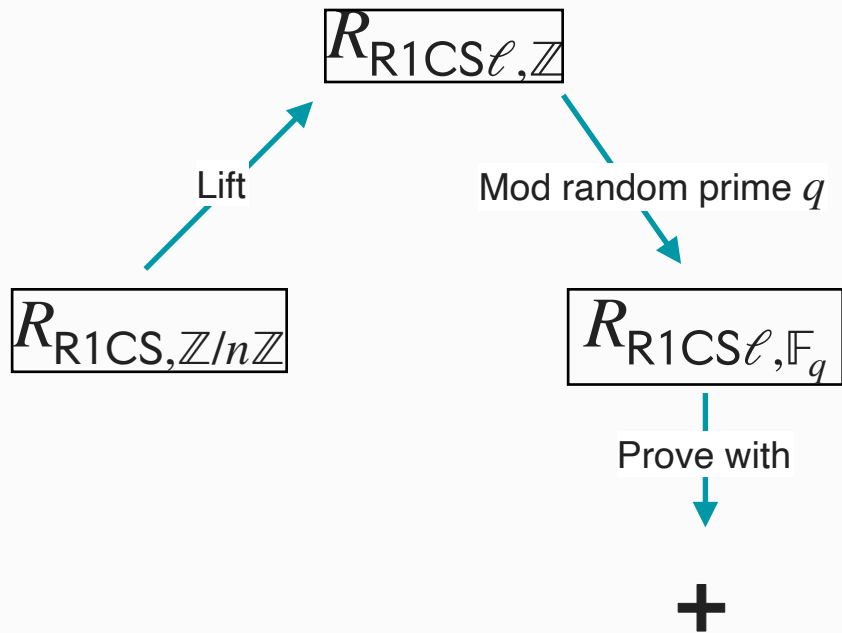


# Zinc in a nutshell



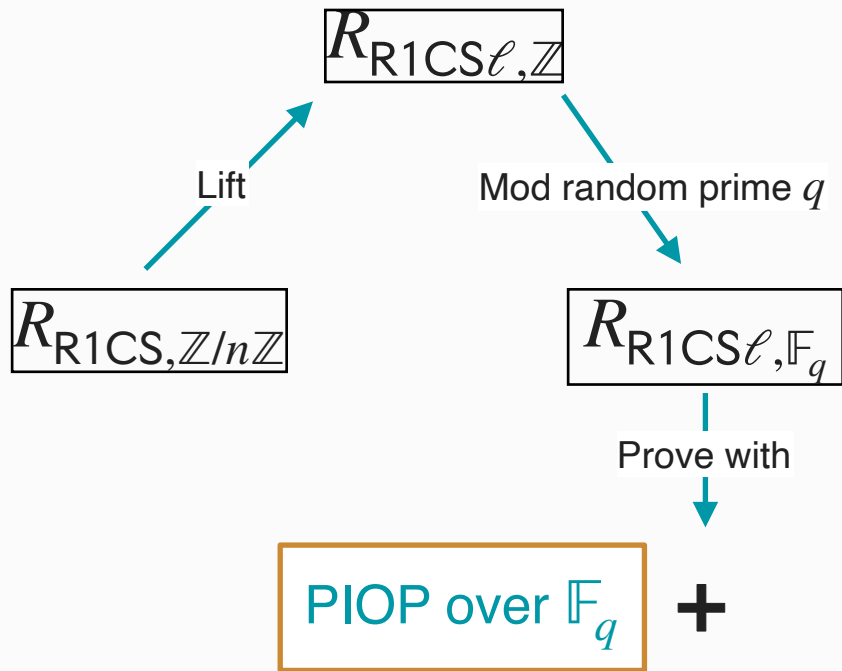
Similar to Ligero/Brakedown  
But some parts run on  $\mathbb{Q}$   
Others on  $\mathbb{F}_q$

# Zinc in a nutshell



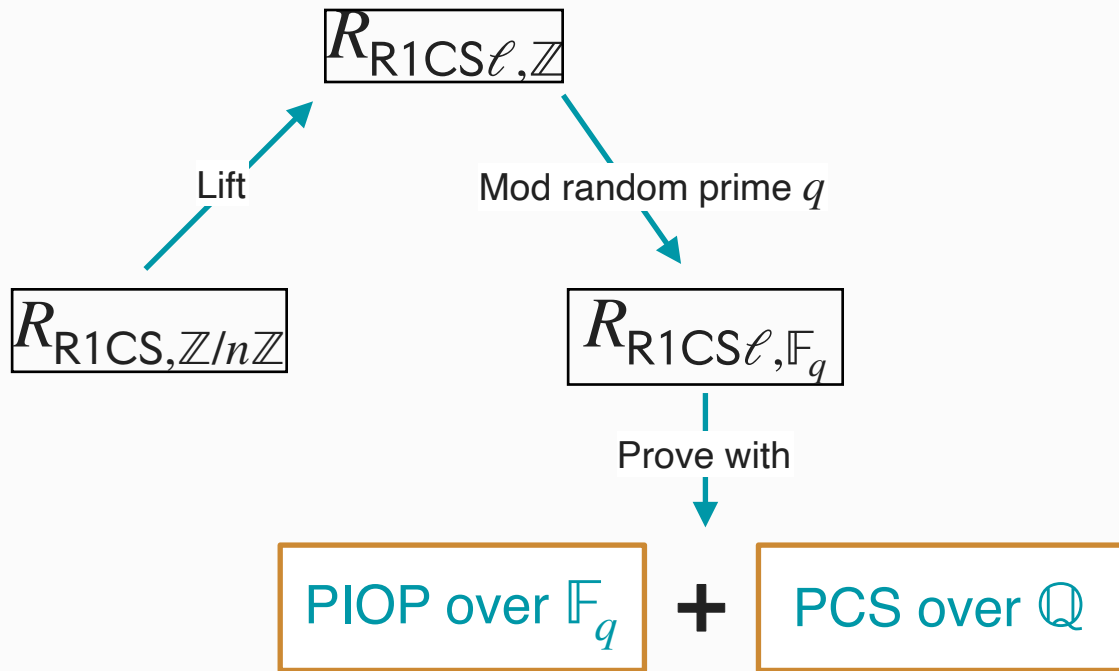
Similar to Liger/Brakedown  
But some parts run on  $\mathbb{Q}$   
Others on  $\mathbb{F}_q$

# Zinc in a nutshell



Similar to Ligero/Brakedown  
But some parts run on  $\mathbb{Q}$   
Others on  $\mathbb{F}_q$

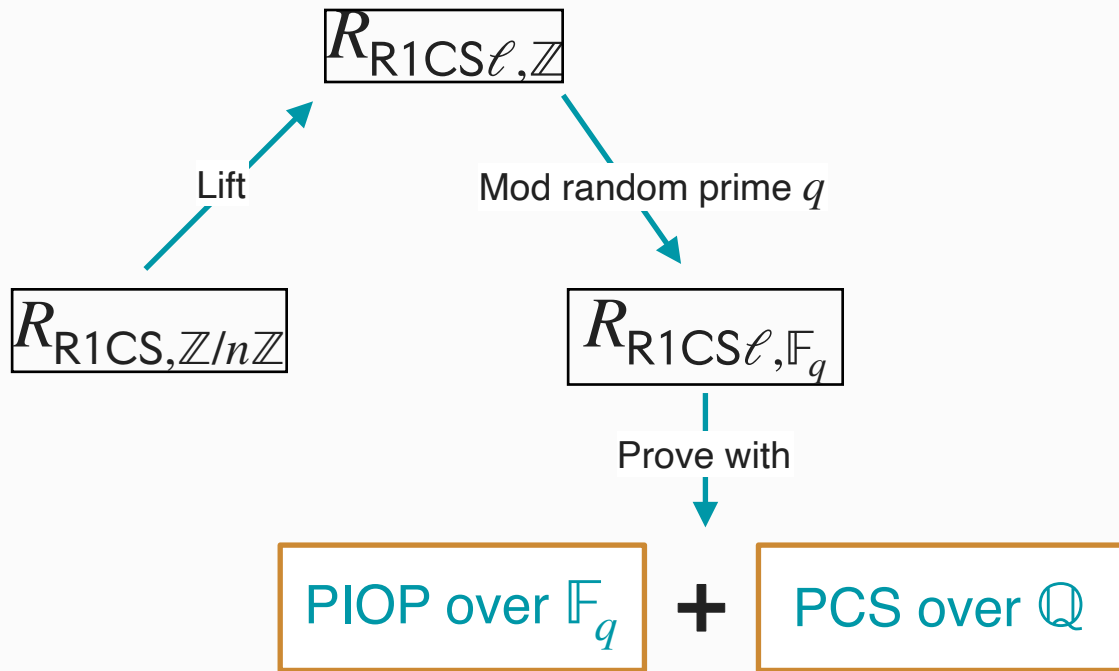
# Zinc in a nutshell



Similar to Ligero/Brakedown  
But some parts run on  $\mathbb{Q}$   
Others on  $\mathbb{F}_q$

# Zinc in a nutshell

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A\vec{z} \circ B\vec{z} = C\vec{z} + u \circ \vec{n} \text{ over } \mathbb{Z}, \quad \vec{z} = (w, x, 1) \end{array} \right. \right\}$$



Similar to Ligero/Brakedown  
But some parts run on  $\mathbb{Q}$   
Others on  $\mathbb{F}_q$

# Bit-sizes

# Bit-sizes

Pippenger: small entries vs large entries

# Bit-sizes

Pippenger: small entries vs large entries

Hashes: not affected



# Bit-sizes

Pippenger: small entries vs large entries

Hashes: not affected

Arithmetization has typically been delegated to engineers, and neglected in research (IMO)

# Bit-sizes

Pippenger: small entries vs large entries

Hashes: not affected

Call for primitive: efficient hash-based pay-per-bit vector commitments

Arithmetization has typically been delegated to engineers, and neglected in research (IMO)

# Bit-sizes

Pippenger: small entries vs large entries

Hashes: not affected

Call for primitive: efficient hash-based pay-per-bit vector commitments

Use with RAA codes so the codeword is small

Arithmetization has typically been delegated to engineers, and neglected in research (IMO)

# Bit-sizes

Pippenger: small entries vs large entries

Hashes: not affected

Call for primitive: efficient hash-based pay-per-bit vector commitments

Use with RAA codes so the codeword is small

Arithmetization has typically been delegated to engineers, and neglected in research (IMO)

Table 1

| Polygon Zisk  |      |       |                               |
|---------------|------|-------|-------------------------------|
|               |      |       |                               |
| Using lookups | to   | avoid | arithmetization               |
|               |      |       |                               |
| Check Binius  | mult | use   | case for the above            |
|               |      |       |                               |
| Tessel        |      |       |                               |
|               |      |       |                               |
| Invite people | to   | colla | b in research and engineering |

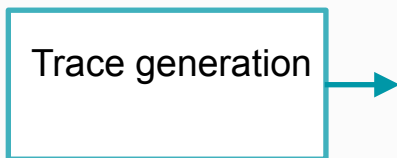
# STARK proof cost breakdown

# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:

# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:
  1. Compute the trace (i.e. the witness)





# STARK proof cost breakdown

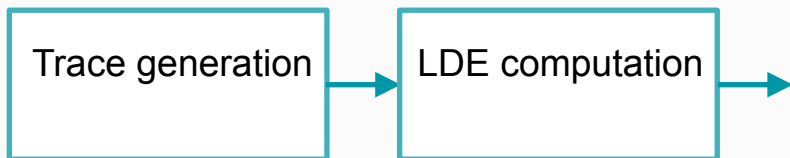
- Anatomy of a FRI-based SNARK:

1. Compute the trace (i.e. the witness)

2. Encode the trace.

AKA compute the **Low Degree Extension (LDE)** of the trace.

3. Commit to the LDE with a Merkle tree.



# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:

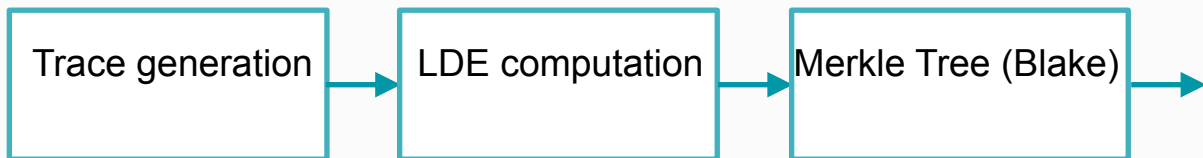
1. Compute the trace (i.e. the witness)

2. Encode the trace.

AKA compute the **Low Degree Extension (LDE)** of the trace.

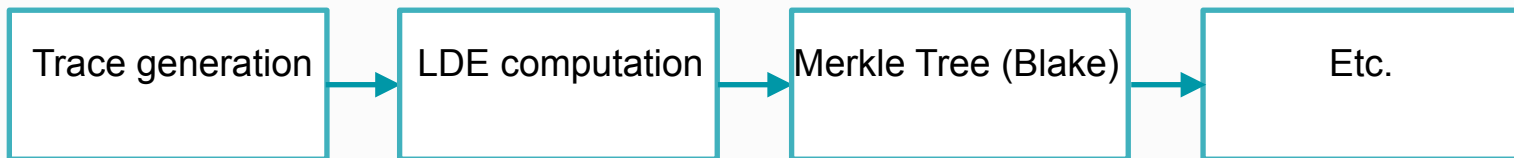
3. Commit to the LDE with a Merkle tree.

4. Etc. (batch polynomial constraints, compute quotient polynomials, apply FRI, ...)



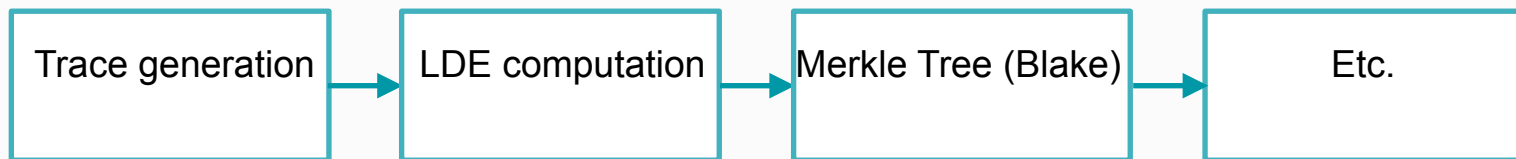
# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:
  1. Compute the trace (i.e. the witness)
  2. Encode the trace.  
AKA compute the **Low Degree Extension (LDE)** of the trace.
  3. Commit to the LDE with a Merkle tree.
  4. Etc. (batch polynomial constraints, compute quotient polynomials, apply FRI, ...)
- STWO's prover cost breakdown, when proving a Blake hash computation is:



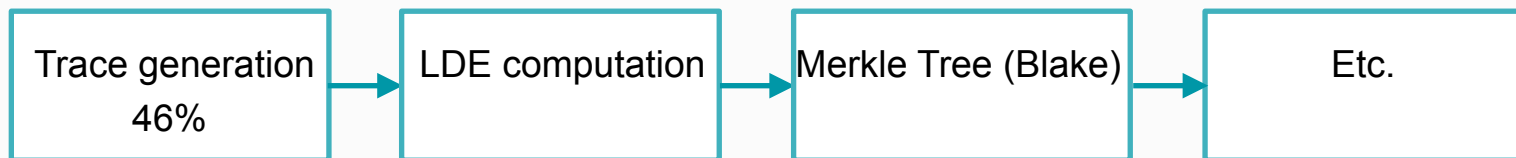
# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:
  1. Compute the trace (i.e. the witness)
  2. Encode the trace.  
AKA compute the **Low Degree Extension (LDE)** of the trace.
  3. Commit to the LDE with a Merkle tree.
  4. Etc. (batch polynomial constraints, compute quotient polynomials, apply FRI, ...)
- STWO's prover cost breakdown, when proving a Blake hash computation is:  
(Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)



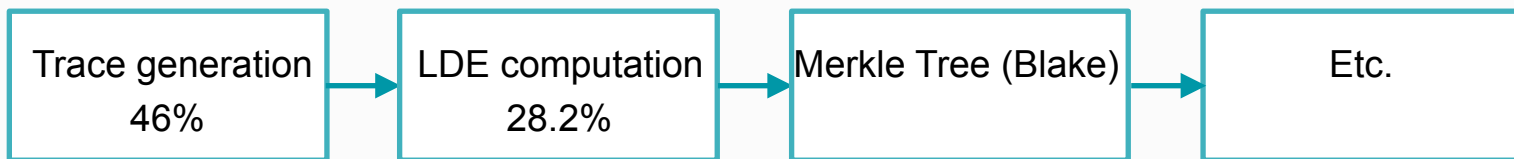
# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:
  1. Compute the trace (i.e. the witness)
  2. Encode the trace.  
AKA compute the **Low Degree Extension (LDE)** of the trace.
  3. Commit to the LDE with a Merkle tree.
  4. Etc. (batch polynomial constraints, compute quotient polynomials, apply FRI, ...)
- STWO's prover cost breakdown, when proving a Blake hash computation is:  
(Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)



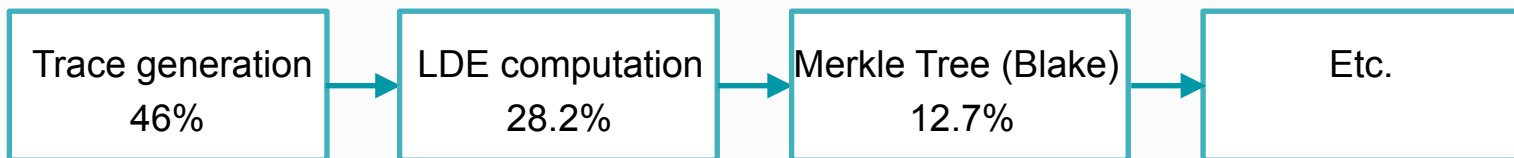
# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:
  1. Compute the trace (i.e. the witness)
  2. Encode the trace.  
AKA compute the **Low Degree Extension (LDE)** of the trace.
  3. Commit to the LDE with a Merkle tree.
  4. Etc. (batch polynomial constraints, compute quotient polynomials, apply FRI, ...)
- STWO's prover cost breakdown, when proving a Blake hash computation is:  
(Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)



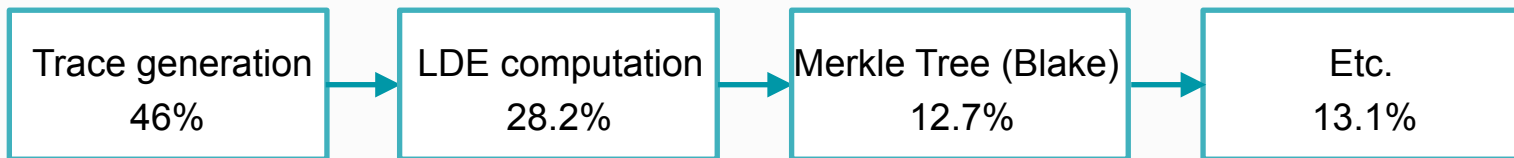
# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:
  1. Compute the trace (i.e. the witness)
  2. Encode the trace.  
AKA compute the **Low Degree Extension (LDE)** of the trace.
  3. Commit to the LDE with a Merkle tree.
  4. Etc. (batch polynomial constraints, compute quotient polynomials, apply FRI, ...)
- STWO's prover cost breakdown, when proving a Blake hash computation is:  
(Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)



# STARK proof cost breakdown

- Anatomy of a FRI-based SNARK:
  1. Compute the trace (i.e. the witness)
  2. Encode the trace.  
AKA compute the **Low Degree Extension (LDE)** of the trace.
  3. Commit to the LDE with a Merkle tree.
  4. Etc. (batch polynomial constraints, compute quotient polynomials, apply FRI, ...)
- STWO's prover cost breakdown, when proving a Blake hash computation is:  
(Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)





# Initial timid attempts

# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

$$R_{\text{R1CS}, \mathbb{Z}/n\mathbb{Z}} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}/n\mathbb{Z}^m, w \in \mathbb{Z}/n\mathbb{Z}^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

$$R_{\text{R1CS}, \mathbb{Z}/n\mathbb{Z}} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}/n\mathbb{Z}^m, w \in \mathbb{Z}/n\mathbb{Z}^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

**Observe:**  $x = y$  over  $\mathbb{Z}/n\mathbb{Z}$  if and only if  $x = y + n \cdot \mu$  over  $\mathbb{Z}$ , or some  $\mu \in \mathbb{Z}$ .

# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

$$R_{\text{R1CS}, \mathbb{Z}/n\mathbb{Z}} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}/n\mathbb{Z}^m, w \in \mathbb{Z}/n\mathbb{Z}^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

**Observe:**  $x = y$  over  $\mathbb{Z}/n\mathbb{Z}$  if and only if  $x = y + n \cdot \mu$  over  $\mathbb{Z}$ , or some  $\mu \in \mathbb{Z}$ .

$$A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}$$

# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

$$R_{\text{R1CS}, \mathbb{Z}/n\mathbb{Z}} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}/n\mathbb{Z}^m, w \in \mathbb{Z}/n\mathbb{Z}^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

**Observe:**  $x = y$  over  $\mathbb{Z}/n\mathbb{Z}$  if and only if  $x = y + n \cdot \mu$  over  $\mathbb{Z}$ , or some  $\mu \in \mathbb{Z}$ .

$$A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}$$



# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

$$R_{\text{R1CS}, \mathbb{Z}/n\mathbb{Z}} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}/n\mathbb{Z}^m, w \in \mathbb{Z}/n\mathbb{Z}^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

**Observe:**  $x = y$  over  $\mathbb{Z}/n\mathbb{Z}$  if and only if  $x = y + n \cdot \mu$  over  $\mathbb{Z}$ , or some  $\mu \in \mathbb{Z}$ .

$$A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $u \in \mathbb{Z}^{m+k+1}$  such that

# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

$$R_{\text{R1CS}, \mathbb{Z}/n\mathbb{Z}} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}/n\mathbb{Z}^m, w \in \mathbb{Z}/n\mathbb{Z}^k \\ A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}, \mathbf{z} = (w, x, 1) \end{array} \right. \right\}$$

**Observe:**  $x = y$  over  $\mathbb{Z}/n\mathbb{Z}$  if and only if  $x = y + n \cdot \mu$  over  $\mathbb{Z}$ , or some  $\mu \in \mathbb{Z}$ .

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $u \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot n$  as integers



# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

$$R_{\text{R1CS}, \mathbb{Z}/n\mathbb{Z}} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}/n\mathbb{Z}^m, w \in \mathbb{Z}/n\mathbb{Z}^k \\ A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}, \mathbf{z} = (w, x, 1) \end{array} \right. \right\}$$

**Observe:**  $x = y$  over  $\mathbb{Z}/n\mathbb{Z}$  if and only if  $x = y + n \cdot \mu$  over  $\mathbb{Z}$ , or some  $\mu \in \mathbb{Z}$ .

$$A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $u \in \mathbb{Z}^{m+k+1}$  such that  
 $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \cdot n$  as integers

So, let's try to build a proof system for:

# Initial timid attempts

Let's start by trying to build a proof system for R1CS over  $\mathbb{Z}/n\mathbb{Z}$ .

$$R_{\text{R1CS}, \mathbb{Z}/n\mathbb{Z}} = \left\{ (x; w) \left| \begin{array}{l} x \in \mathbb{Z}/n\mathbb{Z}^m, w \in \mathbb{Z}/n\mathbb{Z}^k \\ A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

**Observe:**  $x = y$  over  $\mathbb{Z}/n\mathbb{Z}$  if and only if  $x = y + n \cdot \mu$  over  $\mathbb{Z}$ , or some  $\mu \in \mathbb{Z}$ .

$$A z \circ B z = C z \text{ over } \mathbb{Z}/n\mathbb{Z}$$



There exists  $u \in \mathbb{Z}^{m+k+1}$  such that  
 $A z \circ B z = C z + u \cdot n$  as integers

So, let's try to build a proof system for:

$$R_{\text{R1CS}^\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A z \circ B z = C z + u \cdot n \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

# Initial timid attempts

# Initial timid attempts

By replacing  $n$  with an arbitrary vector  $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$ , we capture modular arithmetic for different moduli, at the same time.

# Initial timid attempts

By replacing  $n$  with an arbitrary vector  $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$ , we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

# Initial timid attempts

By replacing  $n$  with an arbitrary vector  $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$ , we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

$R$  looks like a CCS relation over  $\mathbb{Z}$ . We could try to build a SNARK for it.

# Initial timid attempts

By replacing  $n$  with an arbitrary vector  $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$ , we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \mu \text{ over } \mathbb{Z}, \quad \mathbf{z} = (w, x, 1) \end{array} \right. \right\}$$

$R$  looks like a CCS relation over  $\mathbb{Z}$ . We could try to build a SNARK for it.

Let's limit to  $R_{\text{R1CS}\ell, \mathbb{Z}}$  for simplicity.

# Initial timid attempts

By replacing  $n$  with an arbitrary vector  $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$ , we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \mu \text{ over } \mathbb{Z}, \quad \mathbf{z} = (w, x, 1) \end{array} \right. \right\}$$

$R$  looks like a CCS relation over  $\mathbb{Z}$ . We could try to build a SNARK for it.

Let's limit to  $R_{\text{R1CS}\ell, \mathbb{Z}}$  for simplicity.

First off. For technical reasons we set a bound  $B$  on the bit-size of witnesses.



# Initial timid attempts

By replacing  $n$  with an arbitrary vector  $\mu \in \mathbb{Z}_{\geq 1}^{m+k+1}$ , we capture modular arithmetic for different moduli, at the same time.

$$R_{\text{R1CS}\ell, \mathbb{Z}} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}^m, w \in \mathbb{Z}^k, u \in \mathbb{Z}^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

$R$  looks like a CCS relation over  $\mathbb{Z}$ . We could try to build a SNARK for it.

Let's limit to  $R_{\text{R1CS}\ell, \mathbb{Z}}$  for simplicity.

First off. For technical reasons we set a bound  $B$  on the bit-size of witnesses.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, \\ z = (w, x, 1) \end{array} \right. \right\}$$

Where  $\mathbb{Z}_B$  is the set of integers with bit-size less than  $B$ .

Naïve attempt: A succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

# Naïve attempt: A succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

Let's build a succinct argument  $\Pi$  for the following relation:

# Naïve attempt: A succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}, B}$

Let's build a succinct argument  $\Pi$  for the following relation:

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

# Naïve attempt: A succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

Let's build a succinct argument  $\Pi$  for the following relation:

$$R_{R1CS\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} + u \circ \mu \text{ over } \mathbb{Z}, \quad \mathbf{z} = (w, x, 1) \end{array} \right. \right\}$$

Adapt Spartan PIOP (or  
SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

# Naïve attempt: A succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

Let's build a succinct argument  $\Pi$  for the following relation:

$$R_{R1CS\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \mu \text{ over } \mathbb{Z}, \quad \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

Prove it is sound against  $P^*$   
that use polys over  $\mathbb{Z}_B$

# Naïve attempt: A succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}, B}$

Let's build a succinct argument  $\Pi$  for the following relation:

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Adapt Spartan PIOP (or SuperSpartan) for  $R_{\text{R1CS}\ell, \mathbb{Z}, B}$

Prove it is sound against  $P^*$   
that use polys over  $\mathbb{Z}_B$

Design a PCS for polynomials  
over  $\mathbb{Z}_B$

# Naïve attempt: A succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

Let's build a succinct argument  $\Pi$  for the following relation:

$$R_{R1CS\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

Prove it is sound against  $P^*$   
that use polys over  $\mathbb{Z}_B$

Design a PCS for polynomials  
over  $\mathbb{Z}_B$



# Naïve attempt: A succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}, B}$

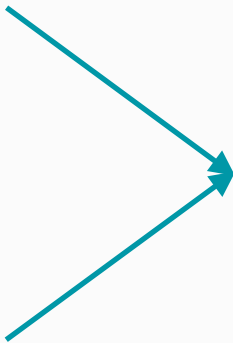
Let's build a succinct argument  $\Pi$  for the following relation:

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Adapt Spartan PIOP (or SuperSpartan) for  $R_{\text{R1CS}\ell, \mathbb{Z}, B}$

Prove it is sound against  $P^*$   
that use polys over  $\mathbb{Z}_B$

Design a PCS for polynomials  
over  $\mathbb{Z}_B$



# Naïve attempt: A succinct argument for $R_{\text{R1CS}\ell, \mathbb{Z}, B}$

Let's build a succinct argument  $\Pi$  for the following relation:

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Adapt Spartan PIOP (or SuperSpartan) for  $R_{\text{R1CS}\ell, \mathbb{Z}, B}$

Prove it is sound against  $P^*$  that use polys over  $\mathbb{Z}_B$

Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{\text{R1CS}\ell, \mathbb{Z}, B}$

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

Issues:

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

Issues:

- (1) requires operating with integers of thousands of bits.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

Issues:

- (1) requires operating with integers of thousands of bits.
- (2) is a very strong primitive.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

Issues:

- (1) requires operating with integers of thousands of bits.
- (2) is a very strong primitive.

The only one we are aware of is due to Block et al.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

Issues:

- (1) requires operating with integers of thousands of bits.
- (2) is a very strong primitive.

The only one we are aware of is due to Block et al.

Based on the DARK scheme. Uses hidden order groups. Is very slow in practice.



# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

Issues:

(1) requires operating with integers of thousands of bits.

(2) is a very strong primitive.

The only one we are aware of is due to Block et al.

Based on the DARK scheme. Uses hidden order groups. Is very slow in practice.

Let's modify our naïve attempt so as to address these issues.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Solution: Campanelli and Hall-Andersen [CH2024]: have  $V$  sample a random prime  $q$ .

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Solution: Campanelli and Hall-Andersen [CH2024]: have  $V$  sample a random prime  $q$ .

Then execute Spartan over  $\mathbb{F}_q$ , rather than over  $\mathbb{Z}_B$ .

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Solution: Campanelli and Hall-Andersen [CH2024]: have  $V$  sample a random prime  $q$ .

Then execute Spartan over  $\mathbb{F}_q$ , rather than over  $\mathbb{Z}_B$ .

They call the resulting PIOP a *mod-PIOP* (or *mod-AHP*).

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

With soundness holding against  $P^*$  that use polys over  $\mathbb{Z}_B$

- (2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(1) requires operating with integers of thousands of bits.

Solution: Campanelli and Hall-Andersen [CH2024]: have  $V$  sample a random prime  $q$ .

Then execute Spartan over  $\mathbb{F}_q$ , rather than over  $\mathbb{Z}_B$ .

They call the resulting PIOP a *mod-PIOP* (or *mod-AHP*).

[CH2024] compile mod-PIOPs with (2) into a succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$



# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) is a very strong primitive.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) is a very strong primitive.

[CH2024] propose using the PCS over  $\mathbb{Z}_B$  of Block et al.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) is a very strong primitive.

[CH2024] propose using the PCS over  $\mathbb{Z}_B$  of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) is a very strong primitive.

[CH2024] propose using the PCS over  $\mathbb{Z}_B$  of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

- In a nutshell, when trying to extract the committed polynomial  $f$ , one has to solve a system of linear equations over  $\mathbb{Z}$ .

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) is a very strong primitive.

[CH2024] propose using the PCS over  $\mathbb{Z}_B$  of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

- In a nutshell, when trying to extract the committed polynomial  $f$ , one has to solve a system of linear equations over  $\mathbb{Z}$ .
- The solution determines  $f$ .

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) is a very strong primitive.

[CH2024] propose using the PCS over  $\mathbb{Z}_B$  of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

- In a nutshell, when trying to extract the committed polynomial  $f$ , one has to solve a system of linear equations over  $\mathbb{Z}$ .
- The solution determines  $f$ .
- However, in general, the solution consists of rational numbers.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) is a very strong primitive.

[CH2024] propose using the PCS over  $\mathbb{Z}_B$  of Block et al.

It is not clear how to build efficient PCS's for integral polynomials. Why?

- In a nutshell, when trying to extract the committed polynomial  $f$ , one has to solve a system of linear equations over  $\mathbb{Z}$ .
- The solution determines  $f$ .
- However, in general, the solution consists of rational numbers.
- I.e. extraction often requires inversion, but  $\mathbb{Z}$  is not closed under inversion.

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$



# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

New program:

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

New program:

PIOP for  $R_{R1CS\ell, \mathbb{Q}, B}$

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

New program:

PIOP for  $R_{R1CS\ell, \mathbb{Q}, B}$

Soundness holds against  $P^*$  that use polys over  $\mathbb{Q}_B$

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

New program:

PIOP for  $R_{R1CS\ell, \mathbb{Q}, B}$

Design a PCS for polynomials over  $\mathbb{Q}_B$

Soundness holds against  $P^*$  that use polys over  $\mathbb{Q}_B$

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

(1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$

(2) Design a PCS for polynomials over  $\mathbb{Z}_B$

Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$

We make the following **key design choice**:

We work over  $\mathbb{Q}_B$  instead of  $\mathbb{Z}_B$ .

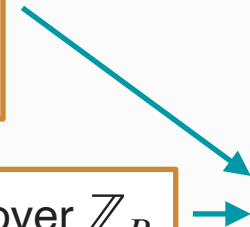
New program:

PIOP for  $R_{R1CS\ell, \mathbb{Q}, B}$

Design a PCS for polynomials over  $\mathbb{Q}_B$

Soundness holds against  $P^*$  that use polys over  $\mathbb{Q}_B$

# Designing a succinct argument for $R_{R1CS\ell, \mathbb{Z}, B}$

- (1) Adapt Spartan PIOP (or SuperSpartan) for  $R_{R1CS\ell, \mathbb{Z}, B}$
  - (2) Design a PCS for polynomials over  $\mathbb{Z}_B$
- Compile into succinct argument for  $R_{R1CS\ell, \mathbb{Z}, B}$
- 
- ```
graph LR; A["(1) Adapt Spartan PIOP (or SuperSpartan) for R_{R1CS\ell, \mathbb{Z}, B}"] --> D["Compile into succinct argument for R_{R1CS\ell, \mathbb{Z}, B}"]; B["(2) Design a PCS for polynomials over \mathbb{Z}_B"] --> D;
```

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

New program:

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$

Design a PCS for polynomials over \mathbb{Q}_B

Soundness holds against P^* that use polys over \mathbb{Q}_B

Compile into succinct argument for $R_{R1CS\ell, \mathbb{Q}, B}$



```
graph LR; C["PIOP for R_{R1CS\ell, \mathbb{Q}, B}"] --> E["Compile into succinct argument for R_{R1CS\ell, \mathbb{Q}, B}"]; D["Design a PCS for polynomials over \mathbb{Q}_B"] --> E;
```


Moving to the field of rational numbers

Moving to the field of rational numbers

Designing a PCS over \mathbb{Z}_B is hard because \mathbb{Z} is not a field.

Moving to the field of rational numbers

Designing a PCS over \mathbb{Z}_B is hard because \mathbb{Z} is not a field.

We make the following **key design choice**:

Moving to the field of rational numbers

Designing a PCS over \mathbb{Z}_B is hard because \mathbb{Z} is not a field.

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

Moving to the field of rational numbers

Designing a PCS over \mathbb{Z}_B is hard because \mathbb{Z} is not a field.

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Q}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Moving to the field of rational numbers

Designing a PCS over \mathbb{Z}_B is hard because \mathbb{Z} is not a field.

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Q}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Note: Now $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ no longer captures arbitrary modular arithmetic.

Moving to the field of rational numbers

Designing a PCS over \mathbb{Z}_B is hard because \mathbb{Z} is not a field.

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Q}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Note: Now $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ no longer captures arbitrary modular arithmetic.

Because of this, we also design a **lookup argument over \mathbb{Q}_B** . I.e. an argument for

Moving to the field of rational numbers

Designing a PCS over \mathbb{Z}_B is hard because \mathbb{Z} is not a field.

We make the following **key design choice**:

We work over \mathbb{Q}_B instead of \mathbb{Z}_B .

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, \quad z = (w, x, 1) \end{array} \right. \right\}$$

Note: Now $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ no longer captures arbitrary modular arithmetic.

Because of this, we also design a **lookup argument over \mathbb{Q}_B** . I.e. an argument for

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Lookup arguments over the rational numbers

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \mid \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right\}$$

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Then an argument for $R_{\text{Look}, \mathbb{Q}, B}$ enforces a to contain entries from \mathbb{Z}_B .

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Then an argument for $R_{\text{Look}, \mathbb{Q}, B}$ enforces a to contain entries from \mathbb{Z}_B .

Argument
for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Then an argument for $R_{\text{Look}, \mathbb{Q}, B}$ enforces a to contain entries from \mathbb{Z}_B .

Argument
for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ +

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Then an argument for $R_{\text{Look}, \mathbb{Q}, B}$ enforces a to contain entries from \mathbb{Z}_B .

Argument
for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$

+

Argument
for $R_{\text{Look}, \mathbb{Q}, B}$

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Then an argument for $R_{\text{Look}, \mathbb{Q}, B}$ enforces a to contain entries from \mathbb{Z}_B .

$$\boxed{\begin{array}{c} \text{Argument} \\ \text{for } R_{\text{R1CS}\ell, \mathbb{Q}, B} \end{array}} + \boxed{\begin{array}{c} \text{Argument} \\ \text{for } R_{\text{Look}, \mathbb{Q}, B} \end{array}} =$$

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Then an argument for $R_{\text{Look}, \mathbb{Q}, B}$ enforces a to contain entries from \mathbb{Z}_B .

Argument for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$	+	Argument for $R_{\text{Look}, \mathbb{Q}, B}$	=	Argument for $R_{\text{R1CS}, \mathbb{Z}, B}$
--	---	--	---	--

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Then an argument for $R_{\text{Look}, \mathbb{Q}, B}$ enforces a to contain entries from \mathbb{Z}_B .

$$\boxed{\text{Argument for } R_{\text{R1CS}, \ell, \mathbb{Q}, B}} + \boxed{\text{Argument for } R_{\text{Look}, \mathbb{Q}, B}} = \boxed{\text{Argument for } R_{\text{R1CS}, \mathbb{Z}, B}}$$

In our work we are general and describe an argument for any relation over \mathbb{Q}_B that can be expressed algebraically.

Lookup arguments over the rational numbers

$$R_{\text{Look}, \mathbb{Q}, B} = \left\{ (t; a) \left| \begin{array}{l} t \in \mathbb{Q}_B^n, a \in \mathbb{Q}_B^m, \\ \{a_i \mid i \in [m]\} \subseteq \{t_i \mid i \in [n]\} \end{array} \right. \right\}$$

Set $t = [-2^B, 2^B] \cap \mathbb{Z}$.

Then an argument for $R_{\text{Look}, \mathbb{Q}, B}$ enforces a to contain entries from \mathbb{Z}_B .

$$\boxed{\text{Argument for } R_{\text{R1CS}\ell, \mathbb{Q}, B}} + \boxed{\text{Argument for } R_{\text{Look}, \mathbb{Q}, B}} = \boxed{\text{Argument for } R_{\text{R1CS}, \mathbb{Z}, B}}$$

In our work we are general and describe an argument for any relation over \mathbb{Q}_B that can be expressed algebraically.

This provides arguments for both $R_{\text{R1CS}\ell, \mathbb{Q}, B}$ and $R_{\text{Look}, \mathbb{Q}, B}$.

The mod-PIOP technique

The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.
First, let's see how [CH2024] does that over the integers.

The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

$$\boxed{\begin{array}{c} P \\ (x; w, u) \end{array}}$$

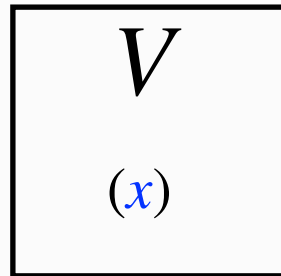
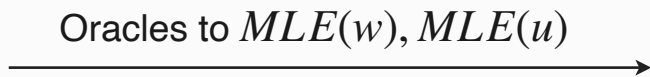
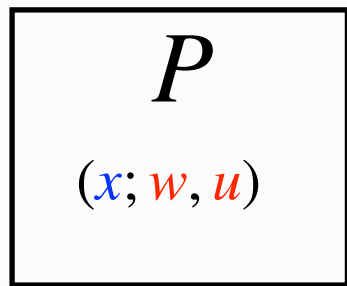
$$\boxed{\begin{array}{c} V \\ (x) \end{array}}$$

The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

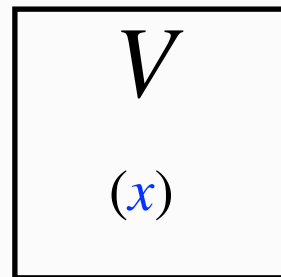
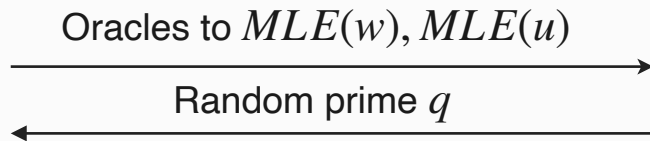
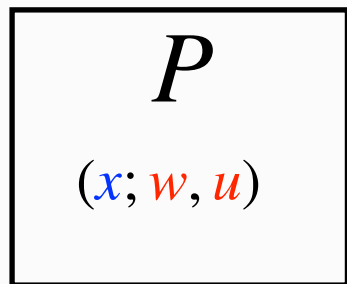


The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

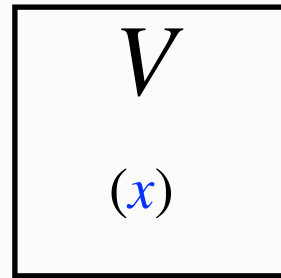
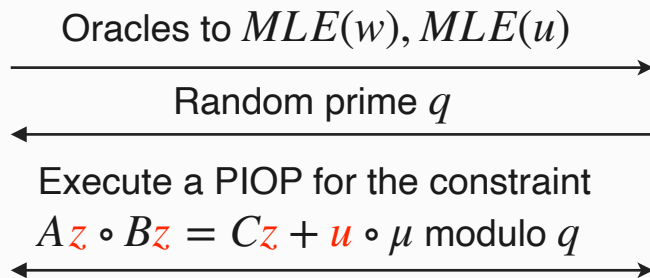
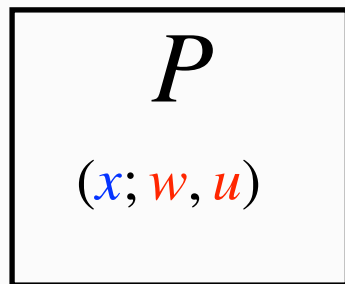


The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

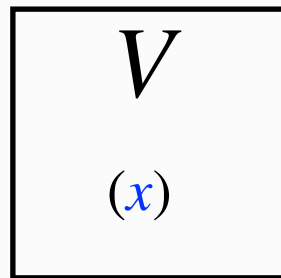
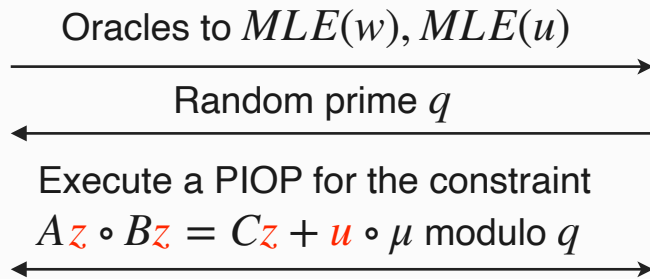
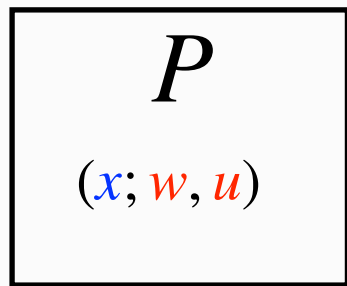


The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



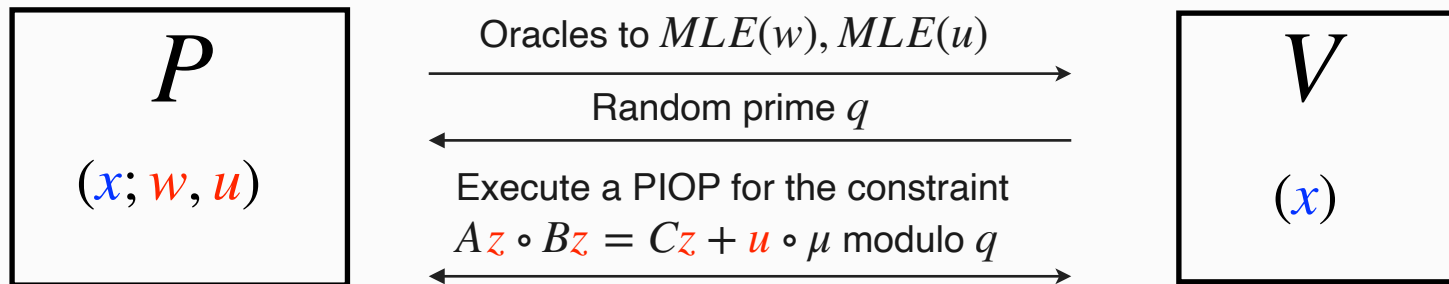
For the latter, it suffices to use the usual version of (Super)Spartan.

The mod-PIOP technique

We will use the idea of [CH2024] of reducing modulo a random prime.

First, let's see how [CH2024] does that over the integers.

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

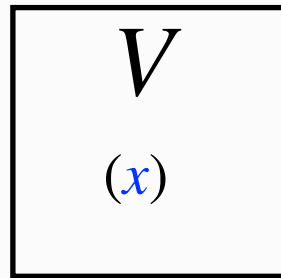
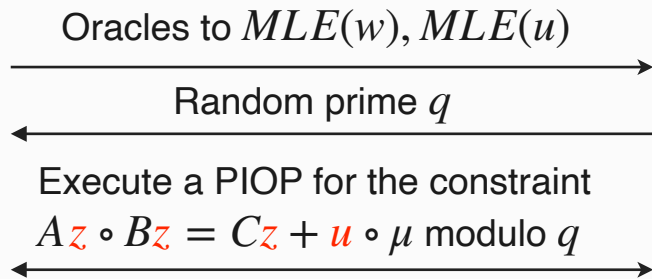
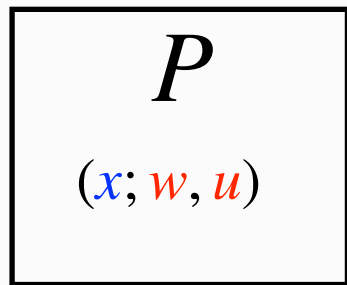


For the latter, it suffices to use the usual version of (Super)Spartan.

When $MLE(w), MLE(u)$ are queried, V receives a value in \mathbb{Z} and reduces it mod q

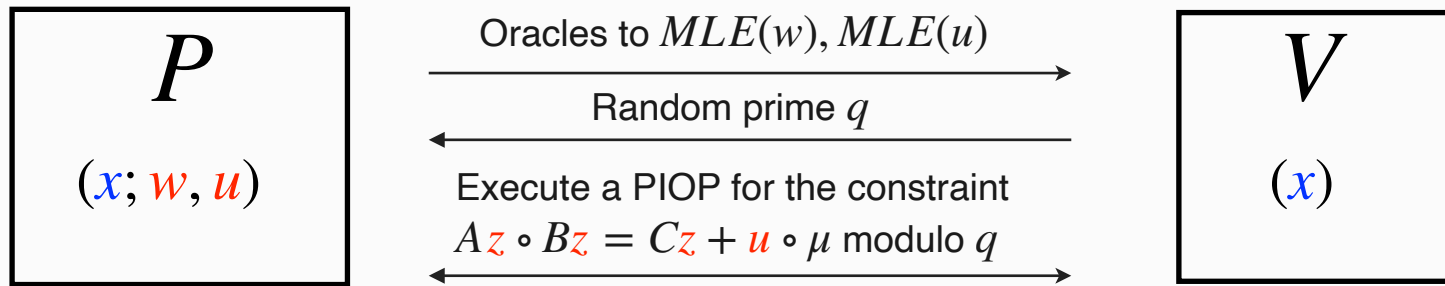
The mod-PIOP technique

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



The mod-PIOP technique

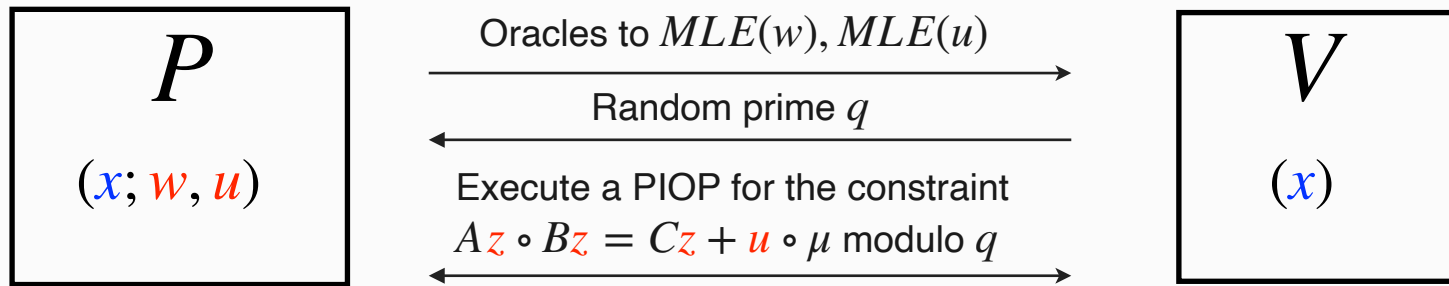
$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



This PIOP is sound against P^* that send oracles to $MLE(w), MLE(u)$ with entries in \mathbb{Z}_B .

The mod-PIOP technique

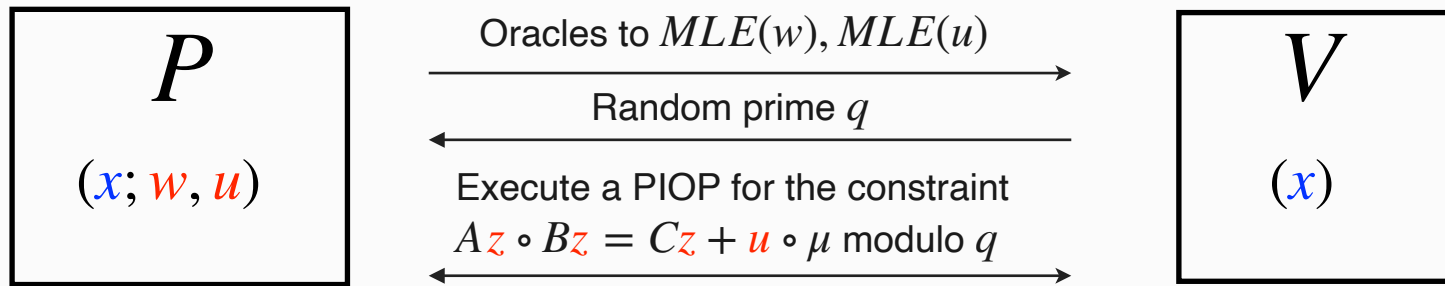
$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



This PIOP is sound against P^* that send oracles to $MLE(w), MLE(u)$ with entries in \mathbb{Z}_B .
(Assuming the PIOP in the last step is sound).

The mod-PIOP technique

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$

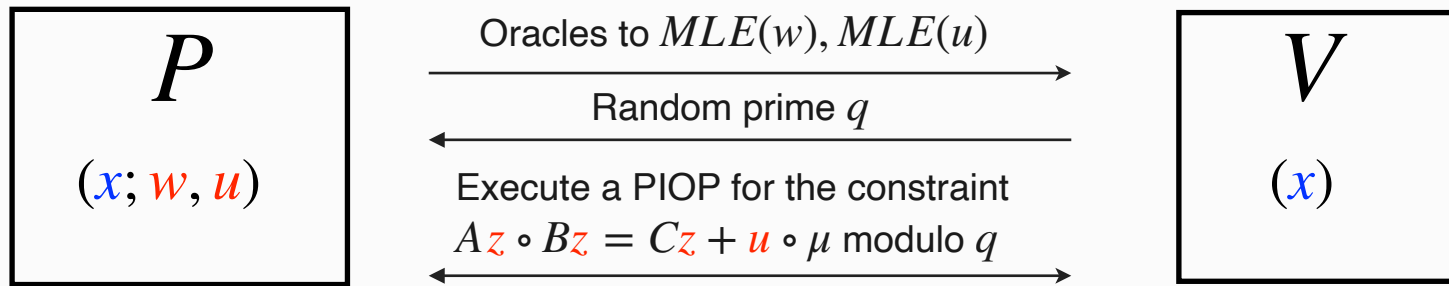


This PIOP is sound against P^* that send oracles to $MLE(w), MLE(u)$ with entries in \mathbb{Z}_B . (Assuming the PIOP in the last step is sound).

Why? Suppose $Az \circ Bz \neq Cz + u \circ \mu$ but that $Az \circ Bz = Cz + u \circ \mu$ modulo q for many primes q . Then one can see that some entry in z, u is divisible by many primes.

The mod-PIOP technique

$$R_{\text{R1CS}\ell, \mathbb{Z}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Z}_B^k, u \in \mathbb{Z}_B^{m+k+1} \\ Az \circ Bz = Cz + u \circ \mu \text{ over } \mathbb{Z}, z = (w, x, 1) \end{array} \right. \right\}$$



This PIOP is sound against P^* that send oracles to $MLE(w), MLE(u)$ with entries in \mathbb{Z}_B . (Assuming the PIOP in the last step is sound).

Why? Suppose $Az \circ Bz \neq Cz + u \circ \mu$ but that $Az \circ Bz = Cz + u \circ \mu$ modulo q for many primes q . Then one can see that some entry in z, u is divisible by many primes.

But the entries of z, u have at most B bits.

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

We focus on $R_{\text{R1CS}\ell, \mathbb{Q}, B}$, but the techniques apply to any algebraic relation over \mathbb{Q}_B

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

We focus on $R_{\text{R1CS}\ell, \mathbb{Q}, B}$, but the techniques apply to any algebraic relation over \mathbb{Q}_B

We will obtain a succinct argument for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$.

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

We focus on $R_{\text{R1CS}\ell, \mathbb{Q}, B}$, but the techniques apply to any algebraic relation over \mathbb{Q}_B

We will obtain a succinct argument for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$.

We start by designing a PIOP over \mathbb{Q}_B for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$. Recall:

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

We focus on $R_{R1CS\ell, \mathbb{Q}, B}$, but the techniques apply to any algebraic relation over \mathbb{Q}_B

We will obtain a succinct argument for $R_{R1CS\ell, \mathbb{Q}, B}$.

We start by designing a PIOP over \mathbb{Q}_B for $R_{R1CS\ell, \mathbb{Q}, B}$. Recall:

$$R_{R1CS\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \mu \text{ over } \mathbb{Q}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

We focus on $R_{R1CS\ell, \mathbb{Q}, B}$, but the techniques apply to any algebraic relation over \mathbb{Q}_B

We will obtain a succinct argument for $R_{R1CS\ell, \mathbb{Q}, B}$.

We start by designing a PIOP over \mathbb{Q}_B for $R_{R1CS\ell, \mathbb{Q}, B}$. Recall:

$$R_{R1CS\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \mu \text{ over } \mathbb{Q}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

We would like to use the mod-PIOP idea from Campanelli and Hall-Andersen.

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

We focus on $R_{\text{R1CS}\ell, \mathbb{Q}, B}$, but the techniques apply to any algebraic relation over \mathbb{Q}_B

We will obtain a succinct argument for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$.

We start by designing a PIOP over \mathbb{Q}_B for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$. Recall:

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A\bar{z} \circ B\bar{z} = C\bar{z} + u \circ \mu \text{ over } \mathbb{Q}, \bar{z} = (w, x, 1) \end{array} \right. \right\}$$

We would like to use the mod-PIOP idea from Campanelli and Hall-Andersen.

But reduction modulo a prime is not well-defined: w, u can contain rational entries.

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

We focus on $R_{\text{R1CS}\ell, \mathbb{Q}, B}$, but the techniques apply to any algebraic relation over \mathbb{Q}_B

We will obtain a succinct argument for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$.

We start by designing a PIOP over \mathbb{Q}_B for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$. Recall:

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$

We would like to use the mod-PIOP idea from Campanelli and Hall-Andersen.

But reduction modulo a prime is not well-defined: w, u can contain rational entries.

We use the concept of **local subrings** of \mathbb{Q} .

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Given a prime q , define

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Given a prime q , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Given a prime q , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$ is a subring of \mathbb{Q} , called the **localization of \mathbb{Q} on q** .

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Given a prime q , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$ is a subring of \mathbb{Q} , called the **localization of \mathbb{Q} on q** .

There is a ring homomorphism.

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Given a prime q , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$ is a subring of \mathbb{Q} , called the **localization of \mathbb{Q} on q** .

There is a ring homomorphism.

$$\phi_q : \mathbb{Z}_{(q)} \rightarrow \mathbb{F}_q \qquad a/b \mapsto a \cdot b^{-1} \bmod q$$

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Given a prime q , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$ is a subring of \mathbb{Q} , called the **localization of \mathbb{Q} on q** .

There is a ring homomorphism.

$$\phi_q : \mathbb{Z}_{(q)} \rightarrow \mathbb{F}_q \quad a/b \mapsto a \cdot b^{-1} \bmod q$$

where b^{-1} denotes an inverse of $b \bmod q$.

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Given a prime q , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$ is a subring of \mathbb{Q} , called the **localization of \mathbb{Q} on q** .

There is a ring homomorphism.

$$\phi_q : \mathbb{Z}_{(q)} \rightarrow \mathbb{F}_q \quad a/b \mapsto a \cdot b^{-1} \bmod q$$

where b^{-1} denotes an inverse of $b \bmod q$.

So, reduction mod q has a natural meaning for most rational numbers.

Zinc-PIOP: A PIOP for relations over \mathbb{Q}_B

Given a prime q , define

$$\mathbb{Z}_{(q)} = \{a/b \in \mathbb{Q} \mid q \text{ does not divide } b\}.$$

$\mathbb{Z}_{(q)}$ is a subring of \mathbb{Q} , called the **localization of \mathbb{Q} on q** .

There is a ring homomorphism.

$$\phi_q : \mathbb{Z}_{(q)} \rightarrow \mathbb{F}_q \quad a/b \mapsto a \cdot b^{-1} \bmod q$$

where b^{-1} denotes an inverse of $b \bmod q$.

So, reduction mod q has a natural meaning for most rational numbers.

The mod-PIOP technique over the rationals

The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$

The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$

P

$(x; w, u)$

V

(x)

The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$

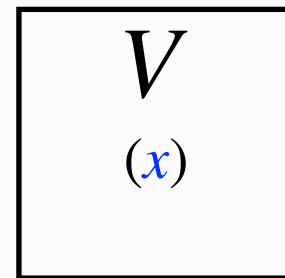
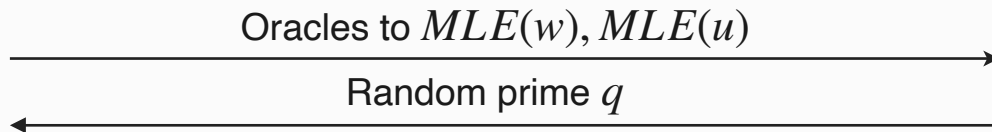
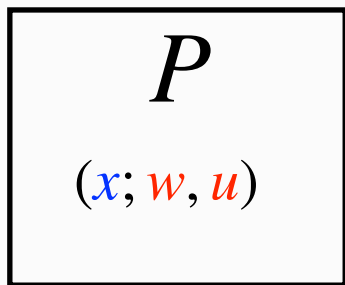
$$\boxed{\begin{array}{c} P \\ (x; w, u) \end{array}}$$

Oracles to $MLE(w), MLE(u)$

$$\boxed{\begin{array}{c} V \\ (x) \end{array}}$$

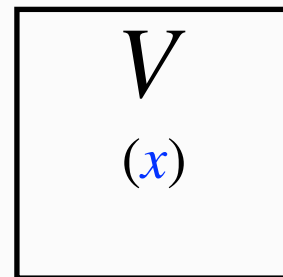
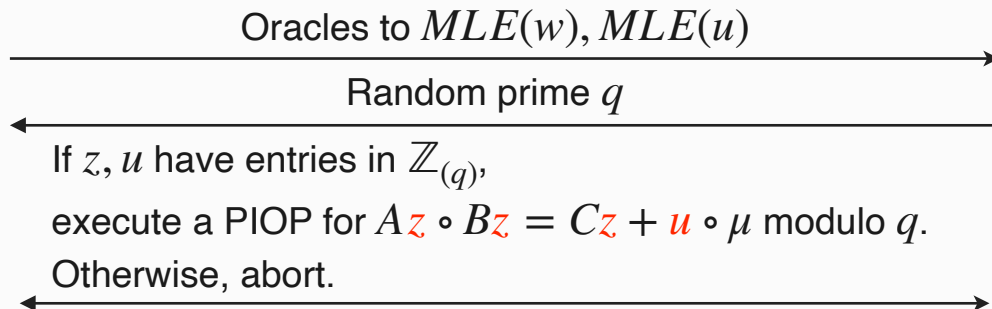
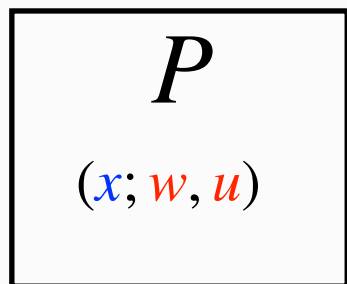
The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$



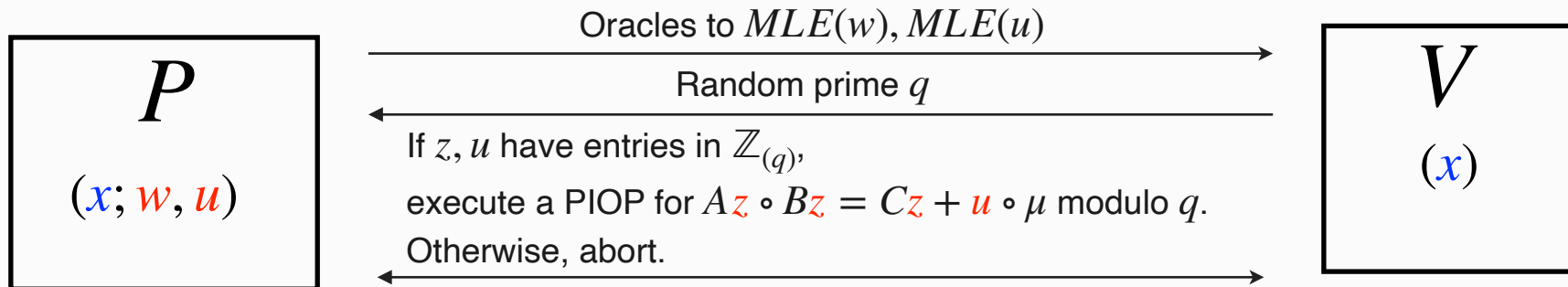
The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$



The mod-PIOP technique over the rationals

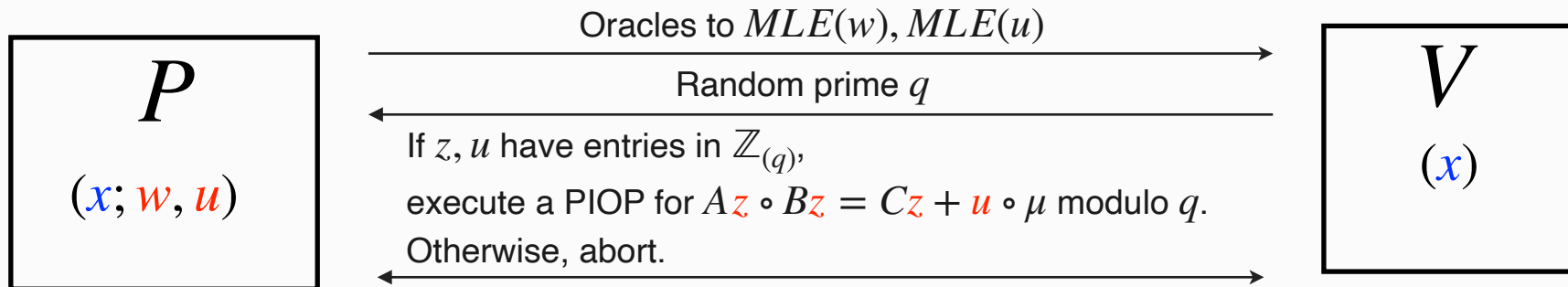
$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$



Completeness: Completeness only fails when z, u don't have all entries in $\mathbb{Z}_{(q)}$.

The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$

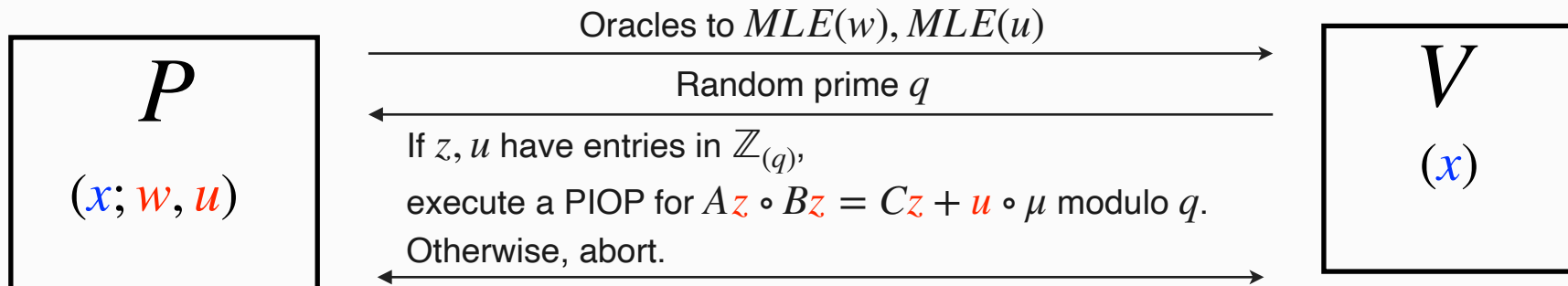


Completeness: Completeness only fails when z, u don't have all entries in $\mathbb{Z}_{(q)}$.

If this happens for many primes, then some entry in z, u has huge size.

The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$



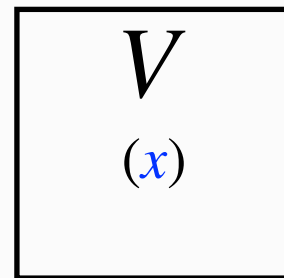
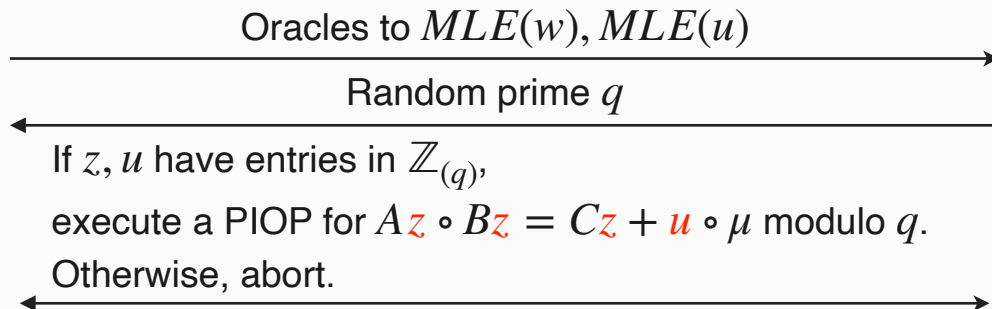
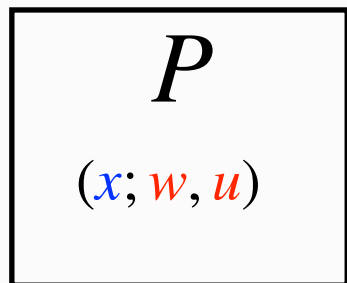
Completeness: Completeness only fails when z, u don't have all entries in $\mathbb{Z}_{(q)}$.

If this happens for many primes, then some entry in z, u has huge size.

But entries in z, u have bit-size $\leq B$.

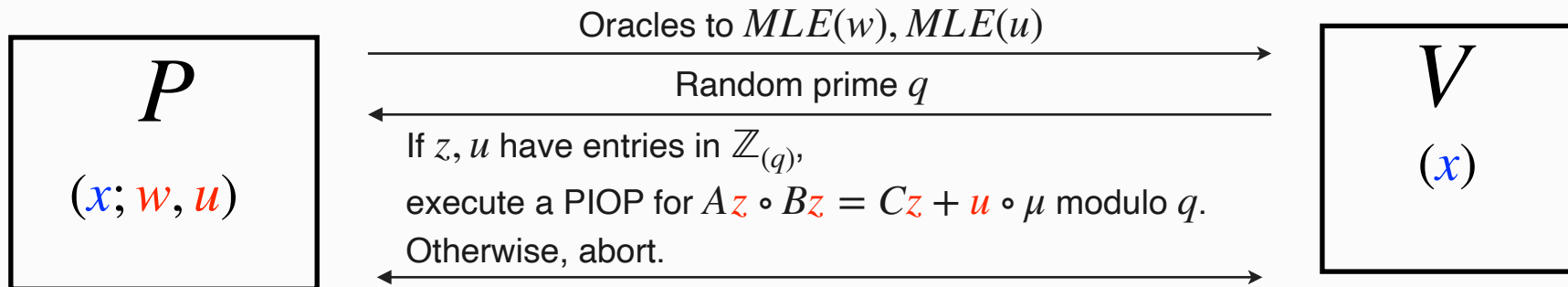
The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$



The mod-PIOP technique over the rationals

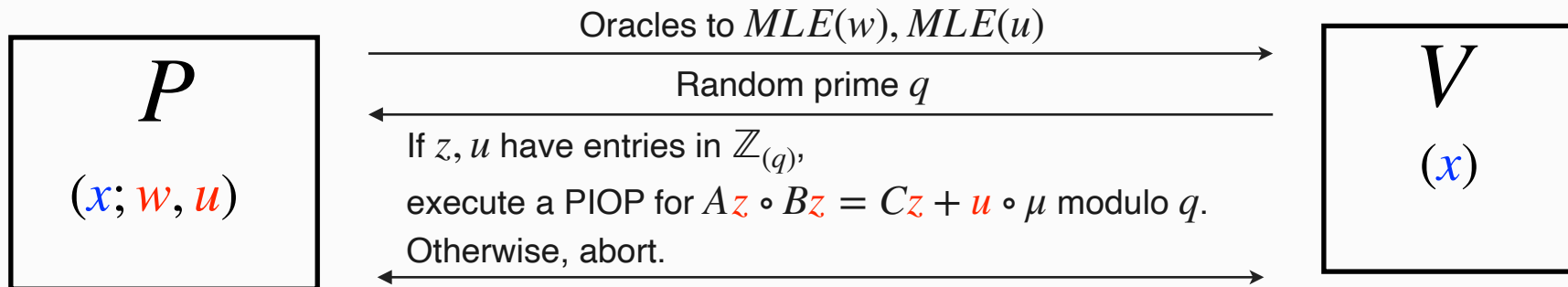
$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$



Soundness: Similarly as in mod-PIOPs over the integers (with some technical subtleties).

The mod-PIOP technique over the rationals

$$R_{\text{R1CS}\ell, \mathbb{Q}, B} = \left\{ (x; w, u) \left| \begin{array}{l} x \in \mathbb{Z}_B^m, w \in \mathbb{Q}_B^k, u \in \mathbb{Q}_B^{m+k+1} \\ A z \circ B z = C z + u \circ \mu \text{ over } \mathbb{Q}, z = (w, x, 1) \end{array} \right. \right\}$$



Soundness: Similarly as in mod-PIOPs over the integers (with some technical subtleties).

If P^* has large success probability, then some witness entry has large bit-size.

Where are we?

Where are we?

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$



Where are we?

PIOP for $R_{\text{R1CS}\ell, \mathbb{Q}, B}$



Soundness holds against P^* that
use polys over \mathbb{Q}_B

Where are we?

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$



Design a PCS for polynomials
over \mathbb{Q}_B



Soundness holds against P^* that
use polys over \mathbb{Q}_B

Where are we?

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$



Design a PCS for polynomials
over \mathbb{Q}_B



Soundness holds against P^* that
use polys over \mathbb{Q}_B



Where are we?

PIOP for $R_{R1CS\ell, \mathbb{Q}, B}$



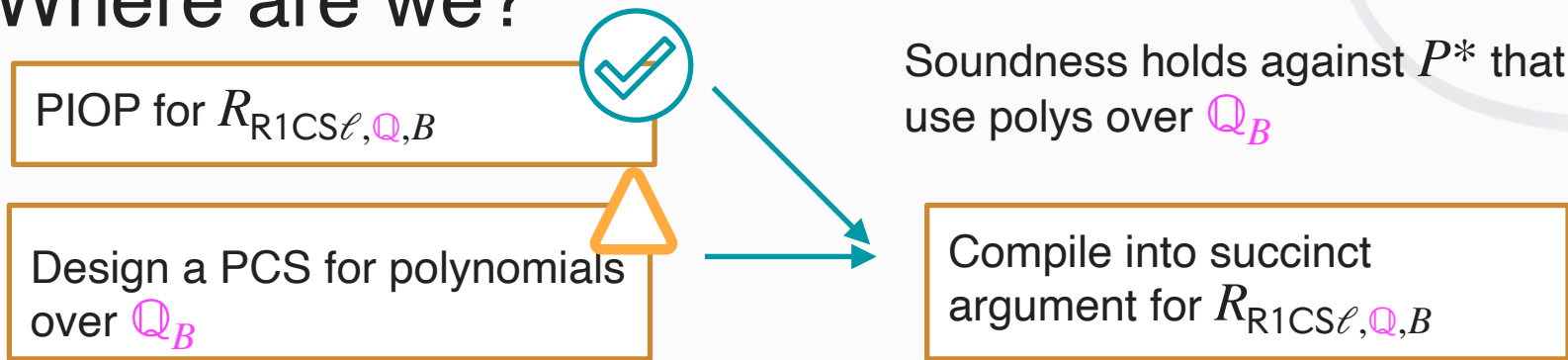
Design a PCS for polynomials
over \mathbb{Q}_B



Soundness holds against P^* that
use polys over \mathbb{Q}_B

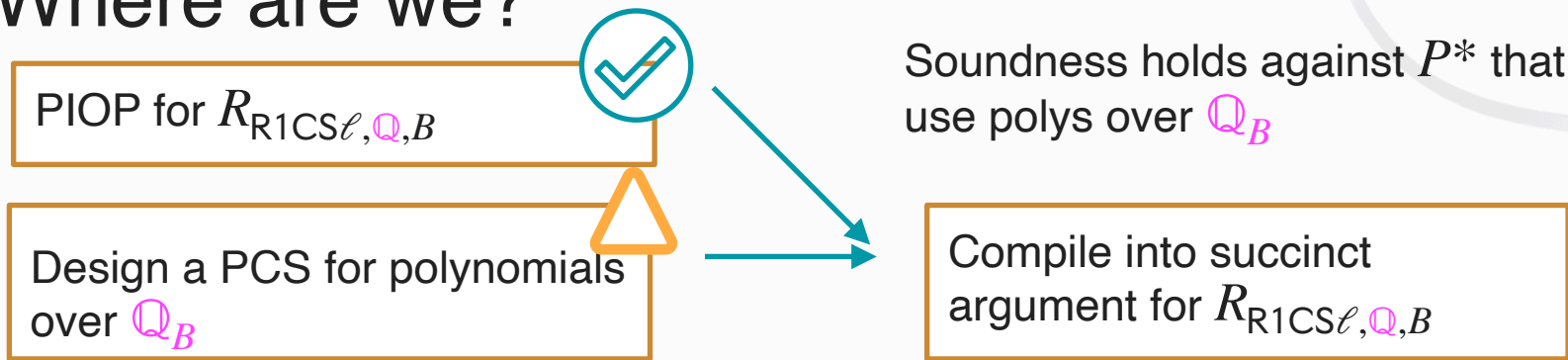
Compile into succinct
argument for $R_{R1CS\ell, \mathbb{Q}, B}$

Where are we?



Next, we design **Zip**, a PCS for polynomials over \mathbb{Q}_B .

Where are we?

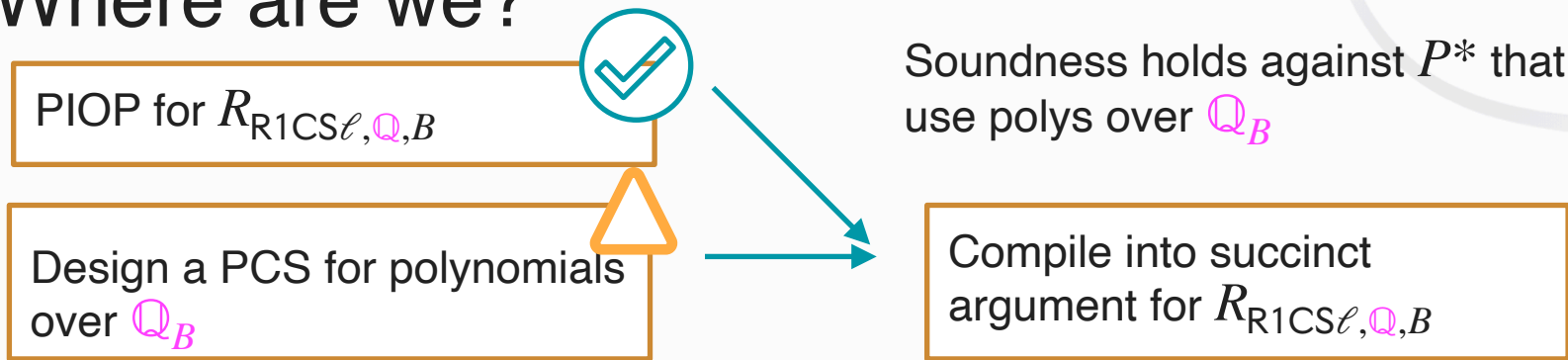


Soundness holds against P^* that use polys over \mathbb{Q}_B

Next, we design **Zip**, a PCS for polynomials over \mathbb{Q}_B .

Zip is based on the Brakedown PCS.

Where are we?

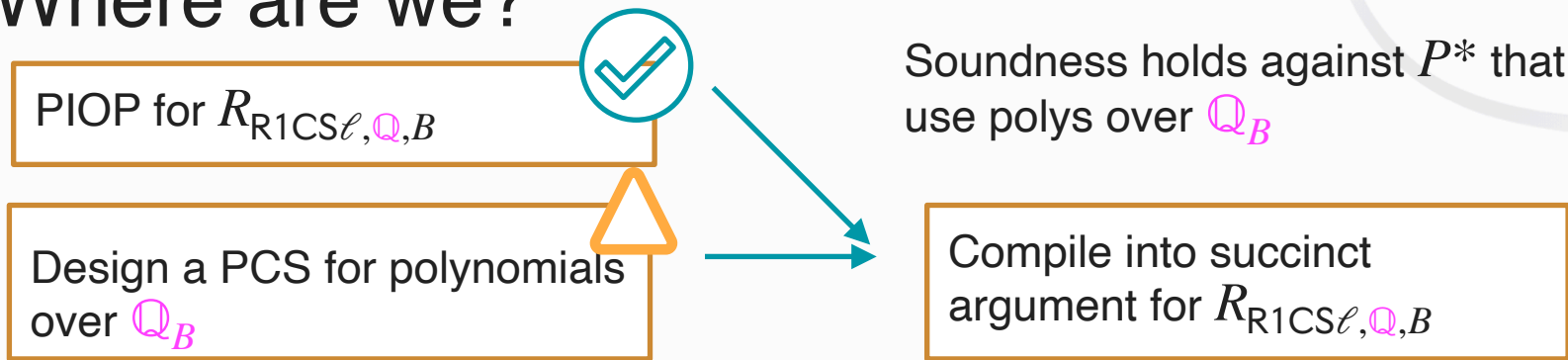


Next, we design **Zip**, a PCS for polynomials over \mathbb{Q}_B .

Zip is based on the Brakedown PCS.

Uses error correcting codes and hash functions.

Where are we?



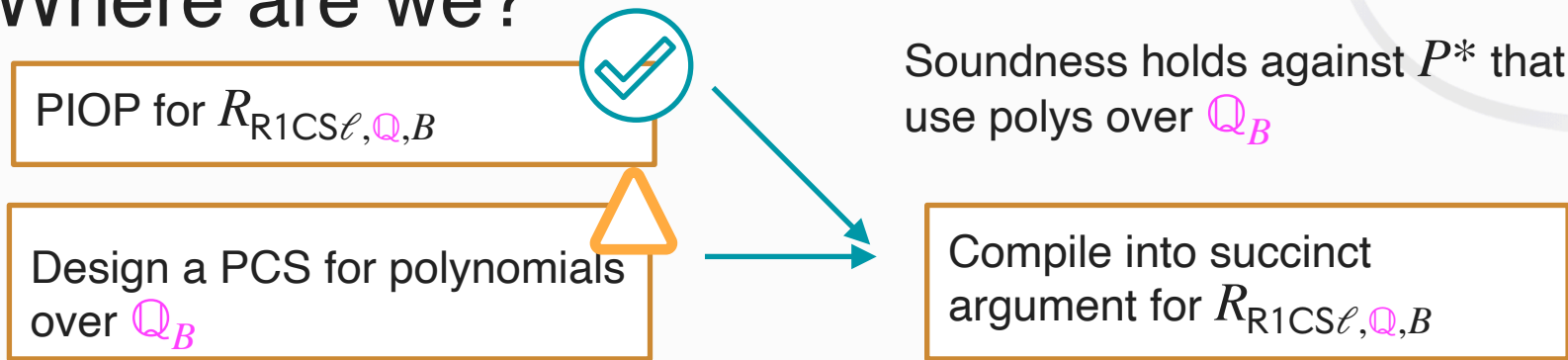
Next, we design **Zip**, a PCS for polynomials over \mathbb{Q}_B .

Zip is based on the Brakedown PCS.

Uses error correcting codes and hash functions.

This is the most involved part of our work.

Where are we?



Next, we design **Zip**, a PCS for polynomials over \mathbb{Q}_B .

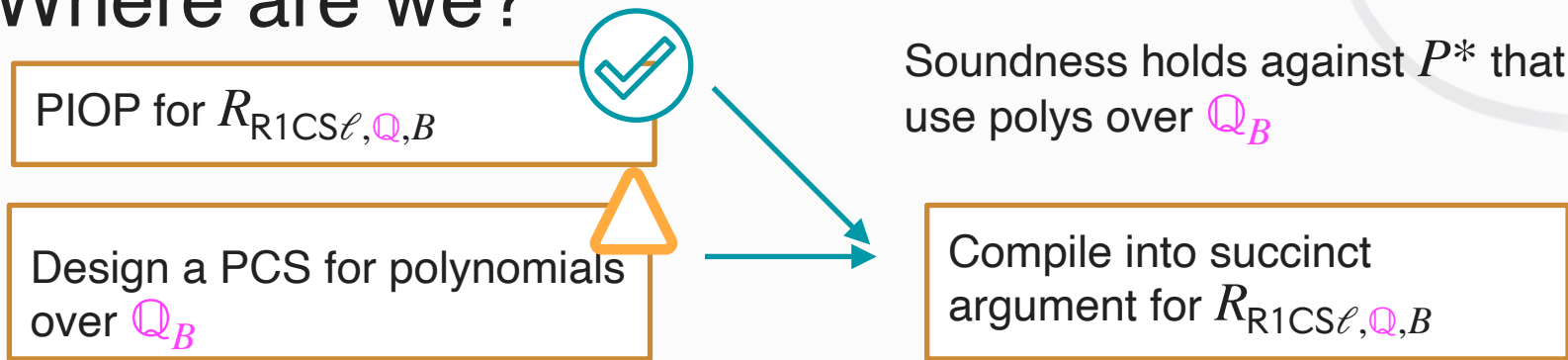
Zip is based on the Brakedown PCS.

Uses error correcting codes and hash functions.

This is the most involved part of our work.

Zip features both

Where are we?



Next, we design **Zip**, a PCS for polynomials over \mathbb{Q}_B .

Zip is based on the Brakedown PCS.

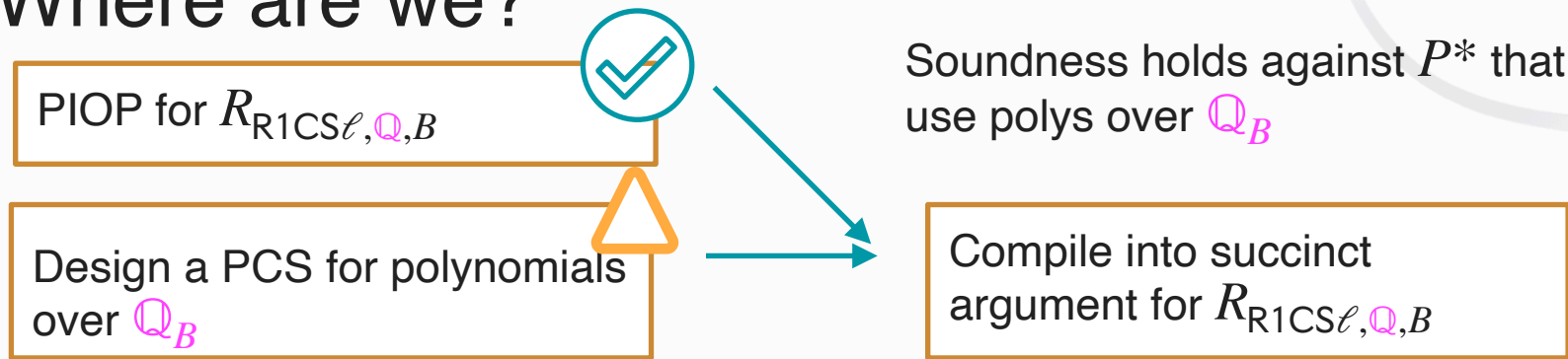
Uses error correcting codes and hash functions.

This is the most involved part of our work.

Zip features both

- IOP of proximity to a linear code

Where are we?



Next, we design **Zip**, a PCS for polynomials over \mathbb{Q}_B .

Zip is based on the Brakedown PCS.

Uses error correcting codes and hash functions.

This is the most involved part of our work.

Zip features both

- IOP of proximity to a linear code
- **IOP of proximity to the integers**

← **New!**

Zip

Zip

Soundness



Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

But we expect the honest P to commit to multilinear $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$,

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

But we expect the honest P to commit to multilinear $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$,

Here B' is certain bound determined by B, μ , and other parameters.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

But we expect the honest P to commit to multilinear $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$,

Here B' is certain bound determined by B, μ , and other parameters.

If P doesn't use $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$, completeness may fail.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

But we expect the honest P to commit to multilinear $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$,

Here B' is certain bound determined by B, μ , and other parameters.

If P doesn't use $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$, completeness may fail.

Analogous to IOP of proximity (IOPP) to a code:

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

But we expect the honest P to commit to multilinear $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$,

Here B' is certain bound determined by B, μ , and other parameters.

If P doesn't use $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$, completeness may fail.

Analogous to IOP of proximity (IOPP) to a code:

An IOPP guarantees P committed to words close to codewords,

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

But we expect the honest P to commit to multilinear $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$,

Here B' is certain bound determined by B, μ , and other parameters.

If P doesn't use $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$, completeness may fail.

Analogous to IOP of proximity (IOPP) to a code:

An IOPP guarantees P committed to words close to codewords,
But completeness is only guaranteed if P actually used codewords

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

But we expect the honest P to commit to multilinear $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$,

Here B' is certain bound determined by B, μ , and other parameters.

If P doesn't use $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$, completeness may fail.

Analogous to IOP of proximity (IOPP) to a code:

An IOPP guarantees P committed to words close to codewords,

But completeness is only guaranteed if P actually used codewords

In our use cases, honest P always uses integral polys.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip guarantees P committed to multilinear $f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$.

But we expect the honest P to commit to multilinear $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$,

Here B' is certain bound determined by B, μ , and other parameters.

If P doesn't use $f \in \mathbb{Z}_B[X_1, \dots, X_\mu]$, completeness may fail.

Analogous to IOP of proximity (IOPP) to a code:

An IOPP guarantees P committed to words close to codewords,

But completeness is only guaranteed if P actually used codewords

In our use cases, honest P always uses integral polys.

(We can extend Zip to enable completeness for $f \in \mathbb{Q}_B[X_1, \dots, X_\mu]$.)



NETHERMIND

RESEARCH

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip is essentially Brakedown over \mathbb{Q} , executed modulo a random prime at times.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip is essentially Brakedown over \mathbb{Q} , executed modulo a random prime at times.
The commitment happens over \mathbb{Q} .

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip is essentially Brakedown over \mathbb{Q} , executed modulo a random prime at times.

The commitment happens over \mathbb{Q} .

Hence, we use a linear code over \mathbb{Q} (with integral generator matrix M_{Gen}).

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip is essentially Brakedown over \mathbb{Q} , executed modulo a random prime at times.

The commitment happens over \mathbb{Q} .

Hence, we use a linear code over \mathbb{Q} (with integral generator matrix M_{Gen}).

We use Expand-Accumulate codes to make sure M_{Gen} has “small” entries.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip is essentially Brakedown over \mathbb{Q} , executed modulo a random prime at times.

The commitment happens over \mathbb{Q} .

Hence, we use a linear code over \mathbb{Q} (with integral generator matrix M_{Gen}).

We use Expand-Accumulate codes to make sure M_{Gen} has “small” entries.

We need to add extra (but costless) Verifier checks to make sure P committed to a polynomial with bounded coefficients.

Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip is essentially Brakedown over \mathbb{Q} , executed modulo a random prime at times.

The commitment happens over \mathbb{Q} .

Hence, we use a linear code over \mathbb{Q} (with integral generator matrix M_{Gen}).

We use Expand-Accumulate codes to make sure M_{Gen} has “small” entries.

We need to add extra (but costless) Verifier checks to make sure P committed to a polynomial with bounded coefficients.



Zip

Soundness

Guaranteed: P committed to

$$f \in \mathbb{Q}_{B'}[X_1, \dots, X_\mu]$$

Completeness

To be used only for

$$f \in \mathbb{Z}_B[X_1, \dots, X_\mu], B < B'$$

Zip is essentially Brakedown over \mathbb{Q} , executed modulo a random prime at times.

The commitment happens over \mathbb{Q} .

Hence, we use a linear code over \mathbb{Q} (with integral generator matrix M_{Gen}).

We use Expand-Accumulate codes to make sure M_{Gen} has “small” entries.

We need to add extra (but costless) Verifier checks to make sure P committed to a polynomial with bounded coefficients.

Key technical lemma: A random linear combination of rational numbers with large bit-size has large bit-size, e.w.n.p.