

PLONK!

Privacy in a world of Universal Snarks

Zachary Williamson
AZTEC Protocol

The three flavours of SNARK

Non-Universal

Circuit-specific trusted setup

Large CRS required

Universal

Requires one trusted setup
Circuit 'preprocessing' is
transparent

Smaller SRS than non-universal

Transparent

No trusted setup

Small CRS
Larger proof sizes

The 'new wave' SNARKs: Marlin and PLONK

- Circuit construction in 2 phases
 - 1: SRS generation (requires a trusted setup)
 - 2: Untrusted circuit specialisation
- Specialisation phase is transparent and independently verifiable
- SRS sizes smaller than non-succinct SNARKs (but still linear)

PLONK!

- Developed by AZTEC and Protocol Labs
- Circuit's described with + and * gates

- Addition gates aren't free



- ...but 'custom' gates are*



Efficiency Analysis

We are currently
implementing optimisations
to improve these numbers

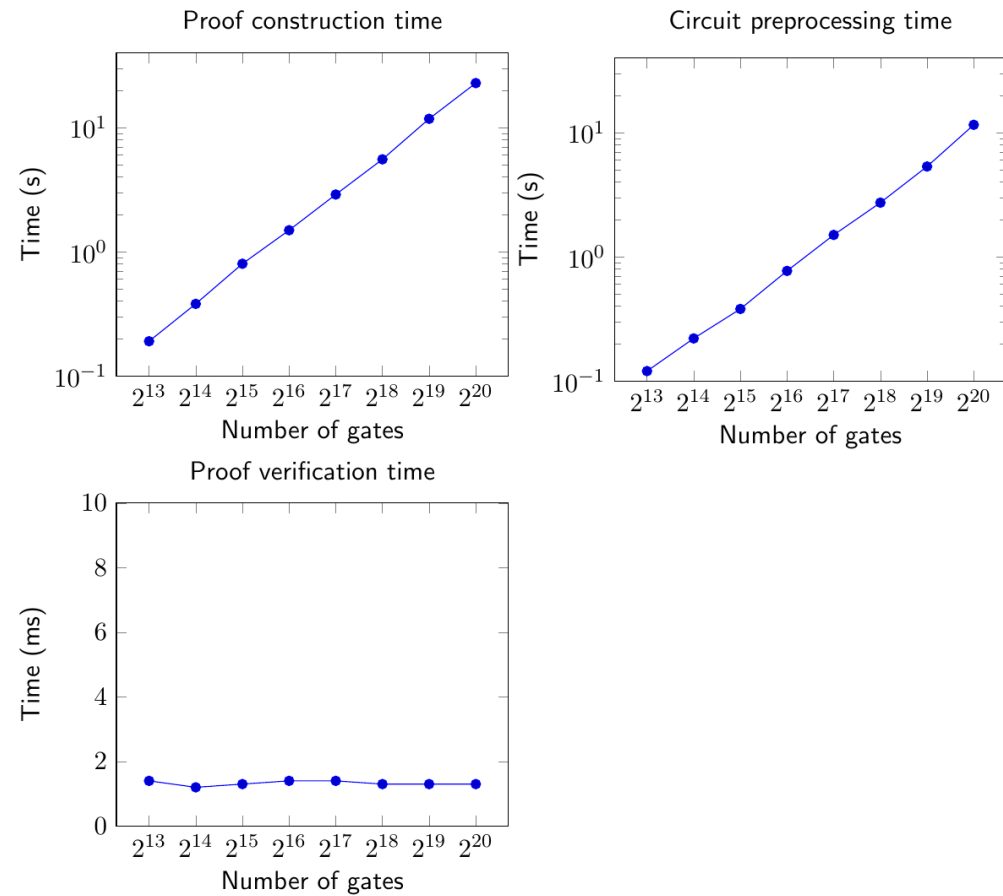


Figure 1: Benchmarks for test **Plonk** circuits using the BN254 curve. Does not include witness generation. Tests performed on a Surface pro 6 with 16GB RAM and a core i7-8650U CPU, utilizing all 8 logical/4 physical cores.

- Prover time <23s for 1 million gates
- (includes addition gates)
- Uses BN254 curve

Efficiency Analysis

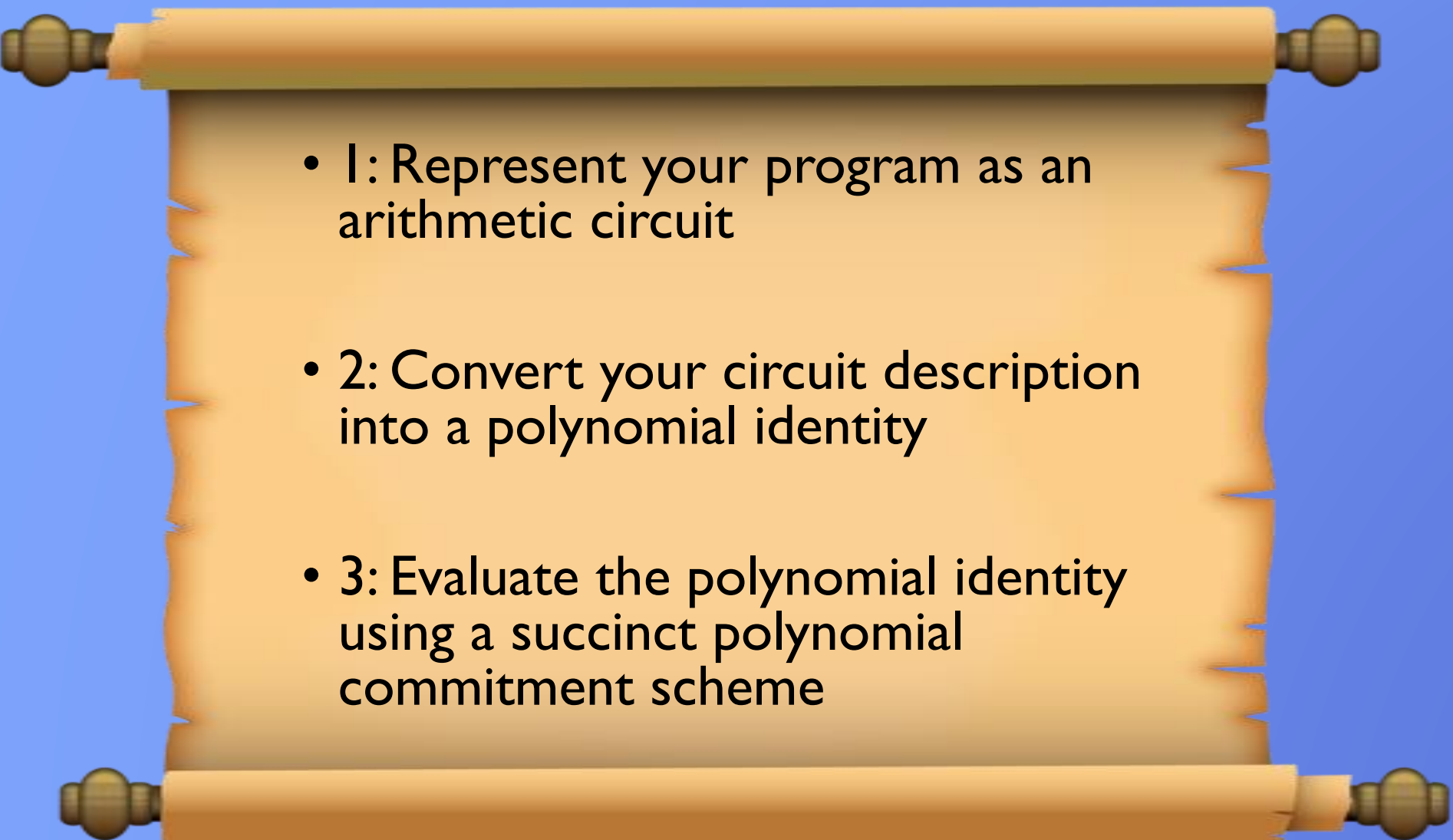
- Good at
 - Binary decomposition
 - Sequential hash algorithms (MiMC)
 - Elliptic curve primitives
- Not so good at
 - High fan-in, non-sequential linear relations (sponge hashes e.g. Poseidon)
 - Matrix addition
 - Matrix multiplication

Efficiency Analysis

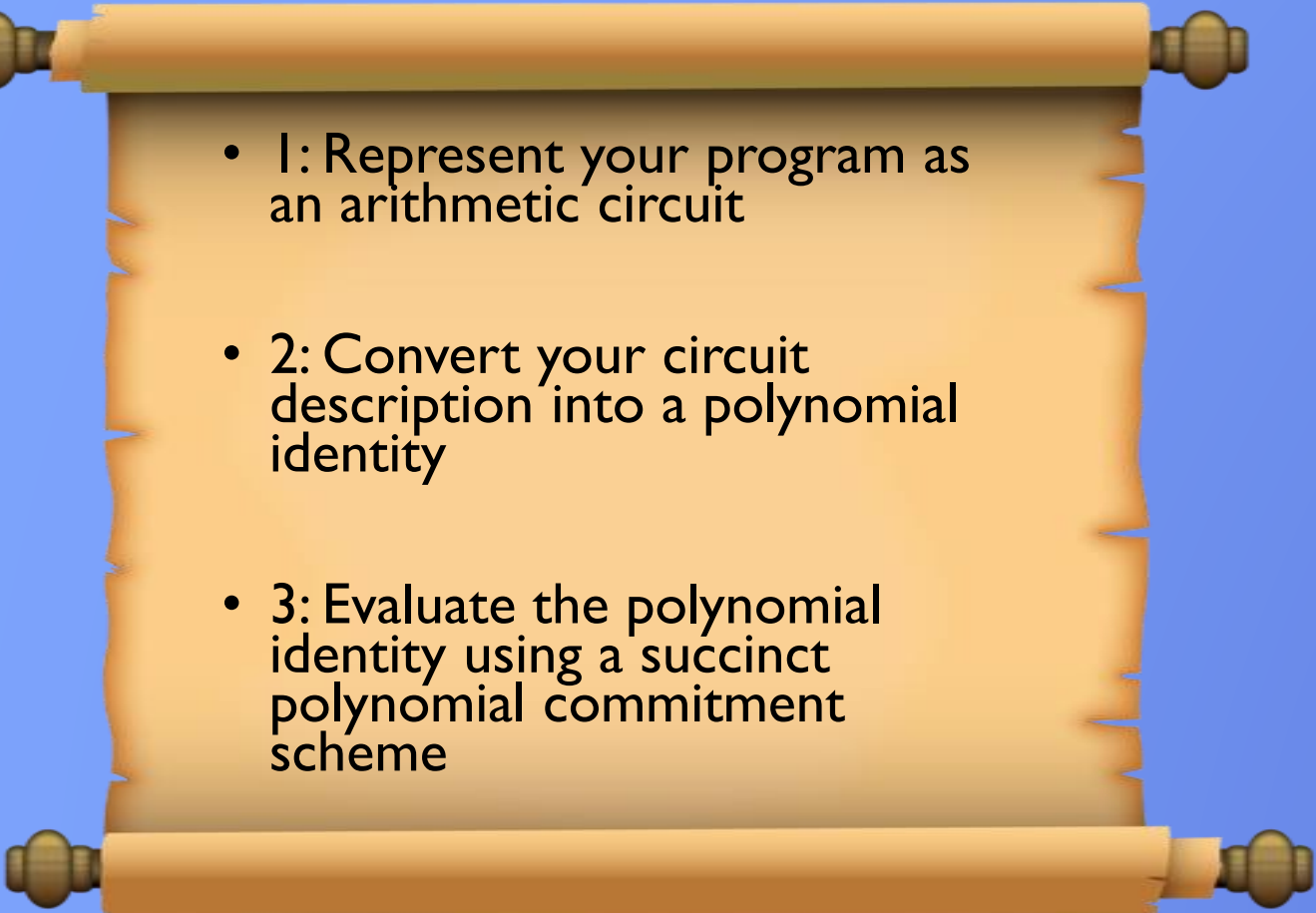
Bilinear Curve	Proof Size
BN-256	512 bytes
BLS12-381	656 bytes

Operation	Basic PLONK constraint count	Turbo PLONK constraint count	Groth16 constraint count
N-bit binary decomposition	$2n$	$n/2$	n
Bool constraint	1	0	1
MiMC Hash (129 bit security)	240	48	192
ECC Point Addition	8	2	3

Cooking with universal SNARKs: a recipe

- 
- A scroll with three steps of a recipe for universal SNARKs. The scroll is yellow with a brown border and is held by four wooden rollers. The text is written in a black, serif font.
- 1: Represent your program as an arithmetic circuit
 - 2: Convert your circuit description into a polynomial identity
 - 3: Evaluate the polynomial identity using a succinct polynomial commitment scheme

Cooking with universal SNARKs: a recipe (2/2)

- 
- 1: Represent your program as an arithmetic circuit
 - 2: Convert your circuit description into a polynomial identity
 - 3: Evaluate the polynomial identity using a succinct polynomial commitment scheme

- Can be instantiated over any poly commitment scheme e.g.
- IOP-based (Vlasov, Fractal)
- Pairing-based (Kate)
- Groups of unknown order (DARK)

PLONK and Lagrange Bases

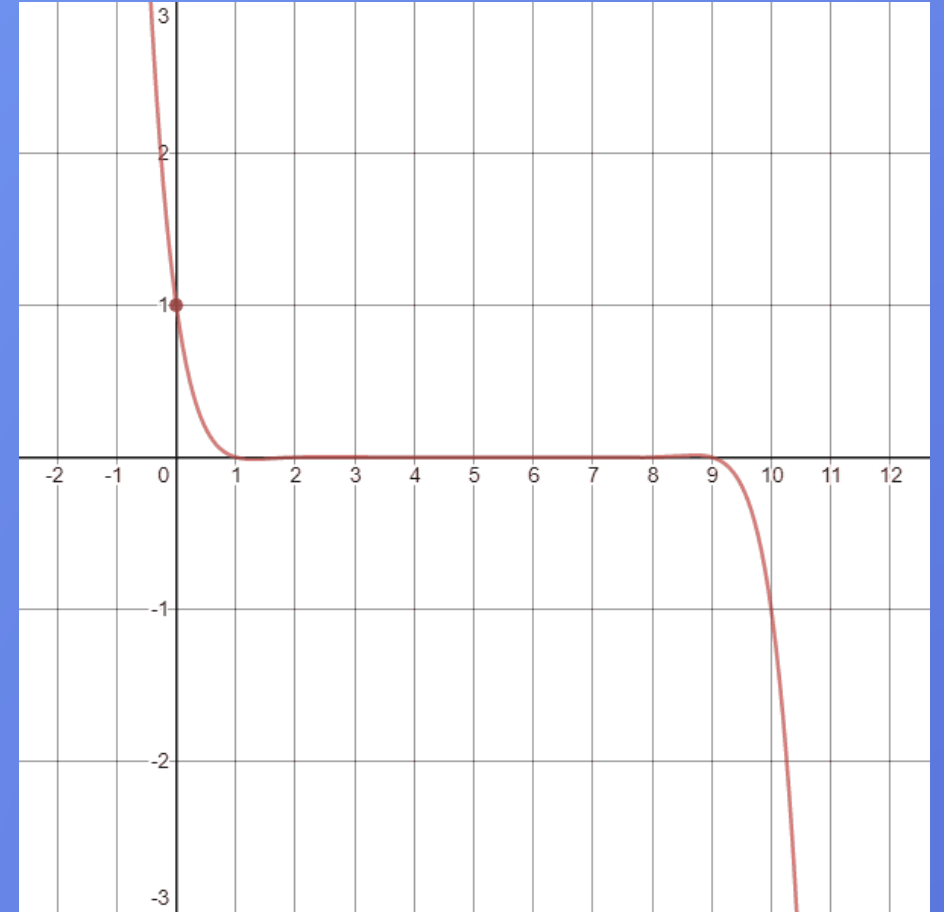
Goal: Represent a vector of n elements (w_1, \dots, w_n) , as a polynomial

Solution: use a **multiplicative subgroup** to define a delta function

$$L_i(X) = \prod_{\substack{j=1 \\ j \neq i}}^n \left(\frac{X - \omega^j}{\omega^i - \omega^j} \right), L_i(\omega^j) = \delta_{i,j}$$

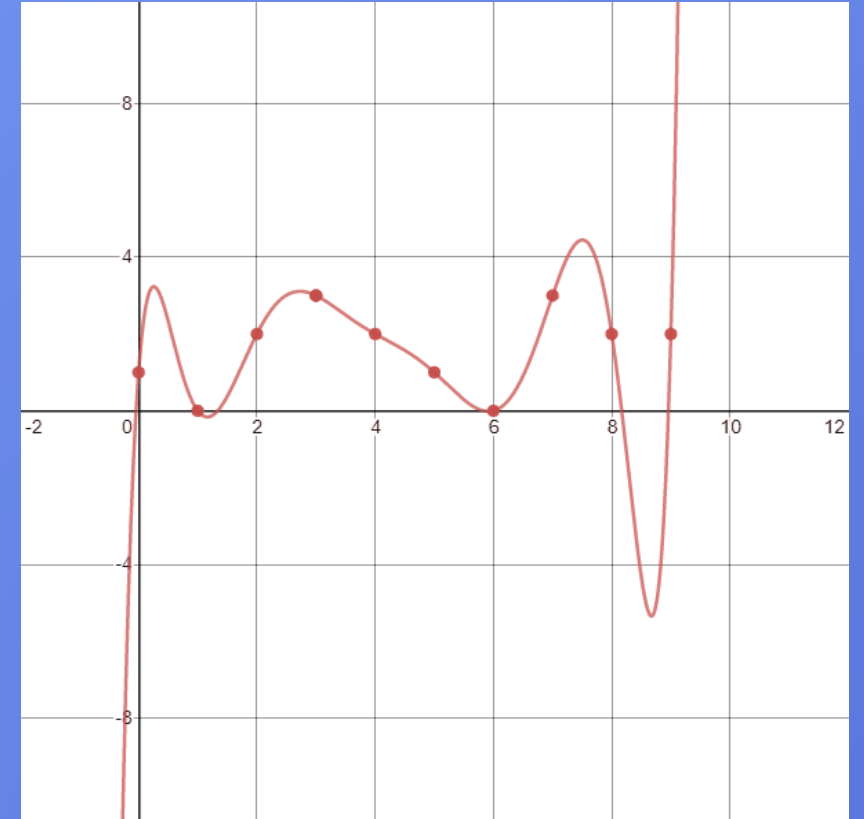
Represent vector as linear sum of Lagrange polynomials

$$(w_1, \dots, w_n) \rightarrow P(X) = \sum_{i=1}^n w_i L_i(X)$$



PLONK and Lagrange Bases (2)

- Vector arithmetic replaced with polynomial arithmetic modulo the *vanishing polynomial* of H ; $Z_H(X)$
- $Z_H(X) = \prod_{i \in [H]} (X - i)$
- $(a \circ b + c = 0) \equiv (A(X)B(X) + C(X) = 0 \bmod Z_H(X))$
- Any arithmetic circuit described via vectors can be transformed into a Lagrange-base representation



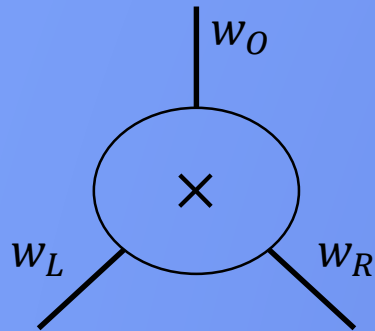
$$f_1 = A(X) \quad f_2 = B(X) \quad f_3 = C(X) \quad f_4 = A(X)B(X) + C(X) \quad f_{5(X)} = Z_H(X)$$

PLONK Circuit Arithmetisation

- We need **addition** and **multiplication** gates!

Wire values: w_L, w_R, w_O

Gate constants: q_M, q_O, q_C

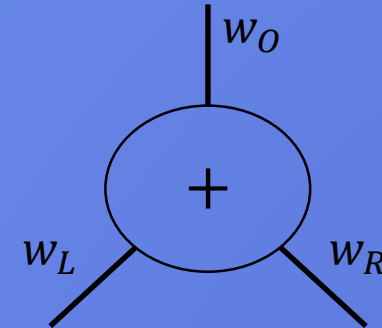


$$q_M \circ w_L \circ w_R + q_O \circ w_O + q_C = 0$$

- Start with vector representation, then map to polynomials

Wire values: w_L, w_R, w_O

Gate constants: q_L, q_R, q_O, q_C



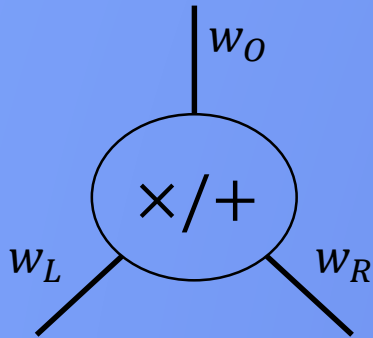
$$q_L \circ w_L + q_R \circ w_R + q_O \circ w_O + q_C = 0$$

Gate constants can be zero...what about combining them?

The PLONK Arithmetic Gate

Combined MUL/ADD gate

- Wire values: w_L, w_R, w_O
- Gate constants: q_M, q_L, q_R, q_O, q_C



$$q_M \circ w_L \circ w_R + q_L \circ w_L + q_R \circ w_R + q_O \circ w_O + q_C = 0$$

Converting into Lagrange-base form

Wire value polynomials

- $w_L(X) = \sum_{i=1}^n w_{L,i} L_i(X)$
- $w_R(X) = \sum_{i=1}^n w_{R,i} L_i(X)$
- $w_O(X) = \sum_{i=1}^n w_{O,i} L_i(X)$

Circuit specific polynomials

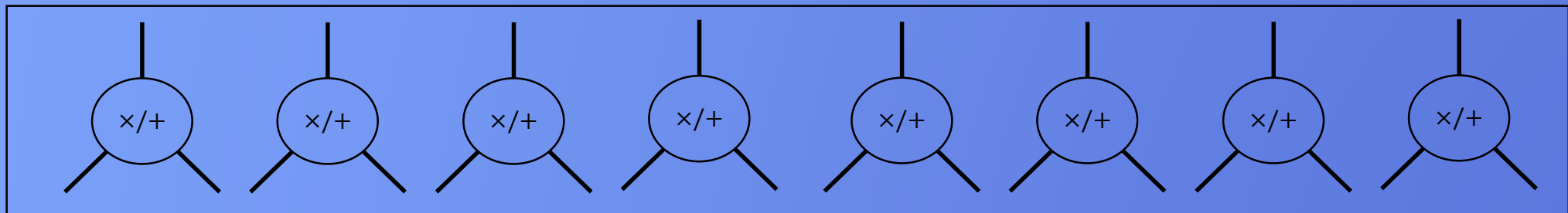
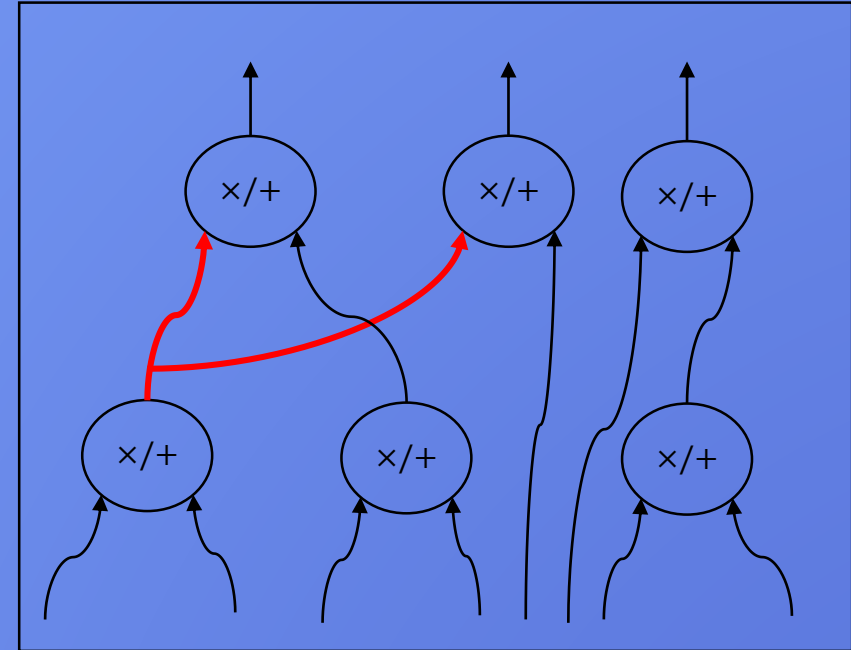
- $q_M(X) = \sum_{i=1}^n q_{M,i} L_i(X)$
- $q_L(X) = \sum_{i=1}^n q_{L,i} L_i(X)$
- $q_R(X) = \sum_{i=1}^n q_{R,i} L_i(X)$
- $q_O(X) = \sum_{i=1}^n q_{O,i} L_i(X)$
- $q_C(X) = \sum_{i=1}^n q_{C,i} L_i(X)$

$$q_M(X)w_L(X)w_R(X) + q_L(X)w_L(X) + q_R(X)w_R(X) + q_O(X)w_O(X) + q_C(X) = 0 \bmod Z_H(X)$$

Taking Stock – what's missing?

- Can represent program as sequence of $\times/+$ gates
- Can efficiently create polynomial representation of
 - Each gate's wire values
 - Each gate's constant coefficients
- But...we haven't connected any gate wires together!

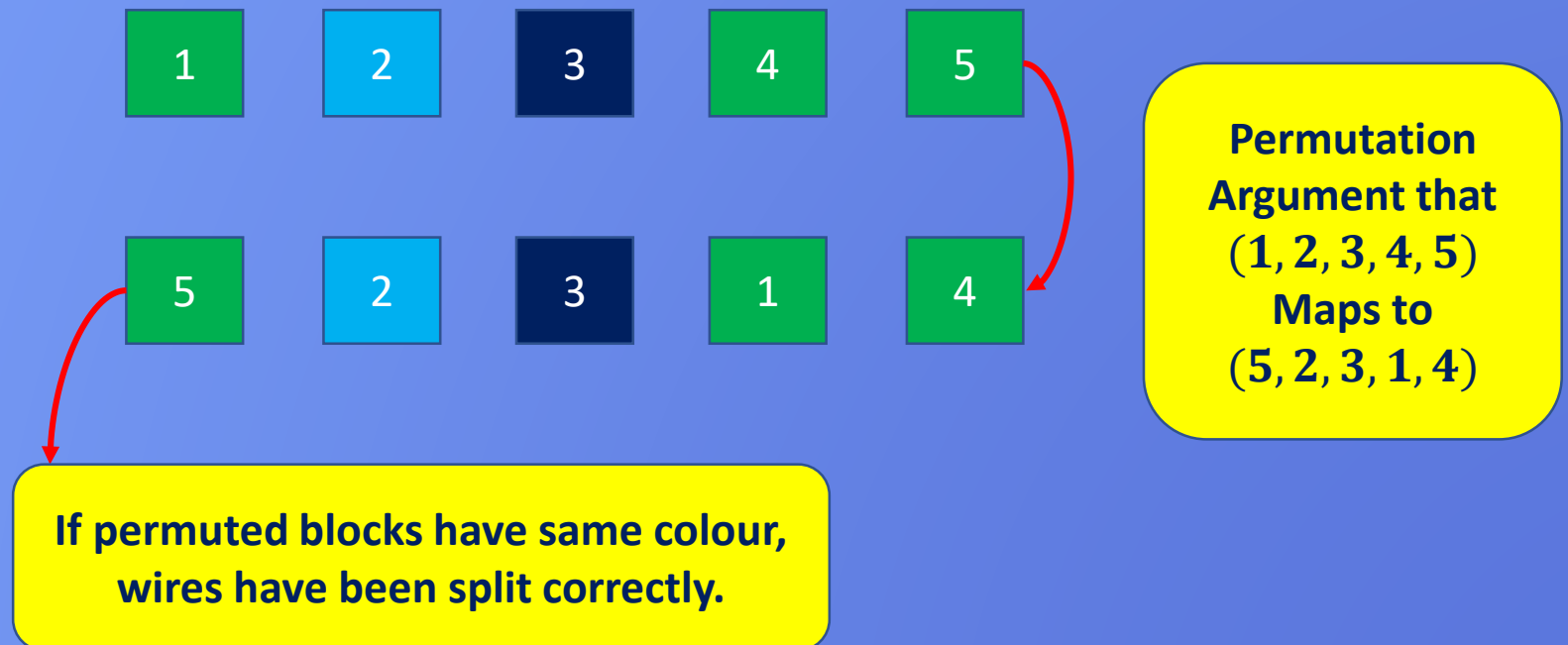
We want something like this →
...But we currently have **this!** ↓



Difficult bit: proving “split” wires contain same value

Checking copies with permutations [Bayer, Groth 12]

- Use copy argument to prove “split” wires contain same value
- Use permutation argument to prove copy argument.



The PLONK Permutation Check

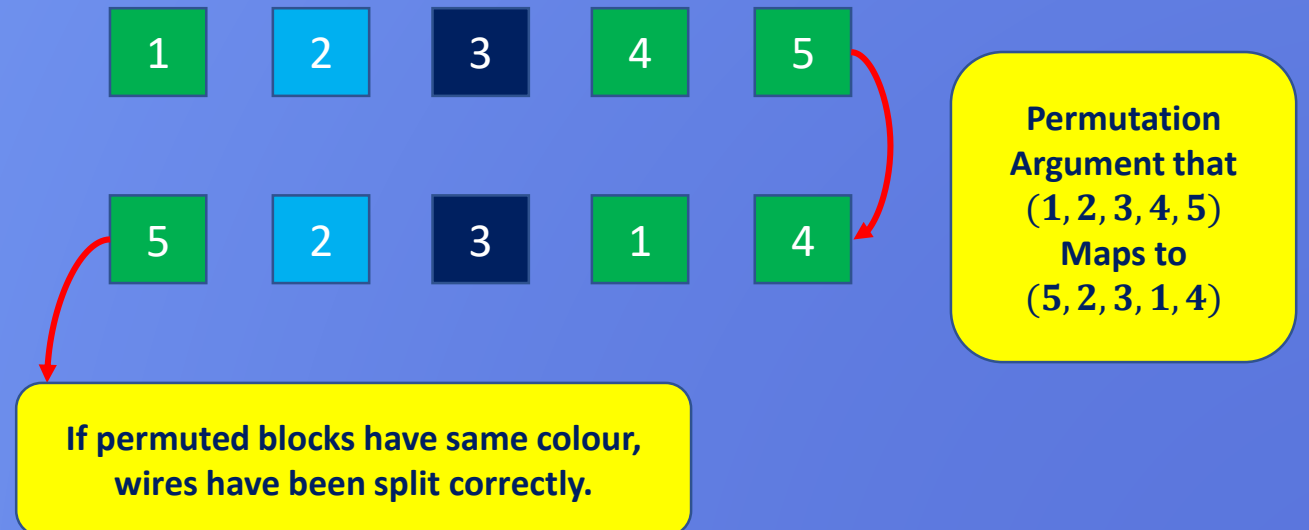
- Use random β, γ to combine each **wire value** w_i with **index position** i

$$z_{1,i} = w_i + \beta i + \gamma$$

- Repeat, this time using **permuted index position** σ_i

$$z_{2,i} = w_i + \beta \sigma_i + \gamma$$

- Check that $\prod_{i=1}^n \frac{z_{1,i}}{z_{2,i}} = 1$



The PLONK Permutation Check (example)

-  = 5,  = 10,  = 2

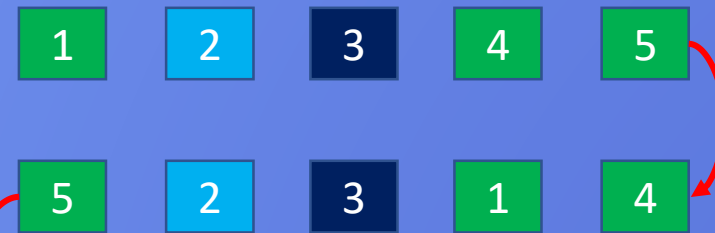
- $\sigma = \{5, 2, 3, 1, 4\}$

- $z_1 = [(5 + \beta + \gamma), (10 + 2\beta + \gamma), (2 + 3\beta + \gamma), (5 + 4\beta + \gamma), (5 + 5\beta + \gamma)]$

- $z_2 = [(5 + 5\beta + \gamma), (10 + 2\beta + \gamma), (2 + 3\beta + \gamma), (5 + \beta + \gamma), (5 + 4\beta + \gamma)]$

$$z_{1,i} = w_i + \beta i + \gamma$$

$$z_{2,i} = w_i + \beta \sigma_i + \gamma$$



Permutation
Argument that
(1, 2, 3, 4, 5)
Maps to
(5, 2, 3, 1, 4)

If permuted blocks have same
colour, wires have been split
correctly.

Permutations and Grand Products (1/2)

- We can encode permutation vector as a polynomial:

$$Z(X) = L_1(X) + \prod_{i=1}^n \frac{(w_i + \beta i + \gamma)}{(w_i + \beta \sigma_i + \gamma)} L_{i+1}(X)$$

- To compute **grand product** of permutation vector elements though....
- We need to check that $\frac{z_{1,i+1}}{z_{2,i+1}} = \frac{z_{1,i}}{z_{2,i}} \cdot \frac{(w_i + \beta i + \gamma)}{(w_i + \beta \sigma_i + \gamma)}$
- ...how do we evaluate a **right shift** of $Z(X)$???????

Permutations and Grand Products (2/2)

- If $Z(X)$ evaluates to $\frac{Z_{1,i}}{Z_{2,i}}$ at $X = \omega^i$...
- $Z(Y)$ will evaluate to $\frac{Z_{1,i+1}}{Z_{2,i+1}}$ at $Y = \omega^{i+1}$
- i.e. $Z(X\omega)$ is the right-shifted form of $Z(X)$
- i.e. iff permutation satisfied:
- 1: $Z(X\omega)(w_i + \beta\sigma_i(X) + \gamma) - Z(X)(w_i + \beta i + \gamma) = 0 \bmod Z_H(X)$
- 2: $(Z(X) - 1)L_1(X) = 0 \bmod Z_H(X)$

Bottling it – complete PLONK

- We have 3 verification equations: 2 for permutation, 1 for arithmetisation
- Can batch these together into a single equation (with random r), by computing a single *quotient polynomial*

$$T(X) = \frac{(q_M(X)w_L(X)w_R(X) + q_L(X)w_L(X) + q_R(X)w_R(X) + q_O(X)w_O(X) + q_C(X))r}{Z_{H^*}(X)} +$$

$$\frac{((w_L(X) + S_{ID}(X)\beta + \gamma)(w_R(X) + S_{ID}(X)(n + \beta) + \gamma)(w_O(X) + S_{ID}(X)(2n + \beta) + \gamma)Z(X) - (w_L(X) + S_{\sigma_1}(X)\beta + \gamma)(w_R(X) + S_{\sigma_2}(X)\beta + \gamma)(w_O(X) + S_{\sigma_3}(X)\beta + \gamma)Z(X\omega))r^2}{Z_{H^*}(X)} +$$

$$\frac{((Z(X) - 1)L_1(X))r^3}{Z_{H^*}(X)}$$

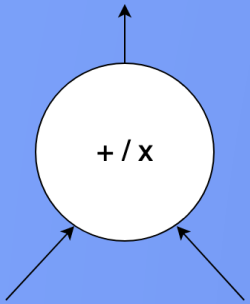
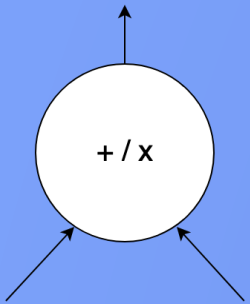
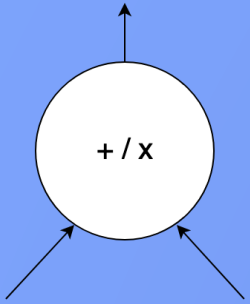
- Prover must compute Kate polynomial commitments to 5 polynomials: $w_L(X), w_R(X), w_O(X), Z(X), T(X)$
- Prover must additionally compute 2 group elements for Kate opening proofs
- Overall prover cost (should be) dominated by $9n$ group exponentiations ($T(X)$ is split into three degree- n commitments)

...the end?

...the end?

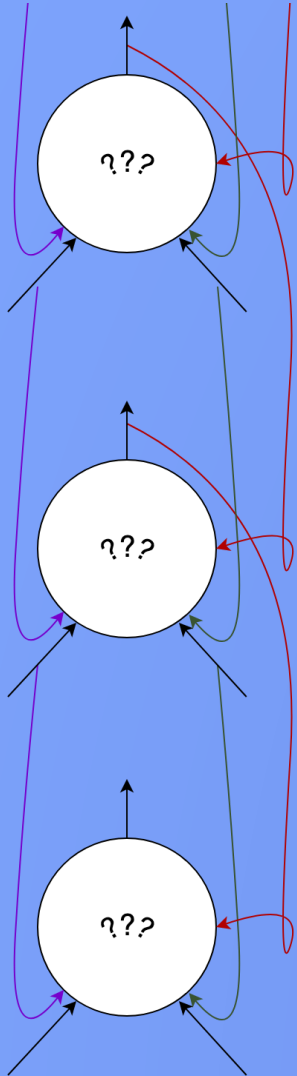
- No!

...the end?



- No!
- Can we do more than add / mul?
- And can we get more than 3 wires?
- ← ...what about turning this?

...the end?

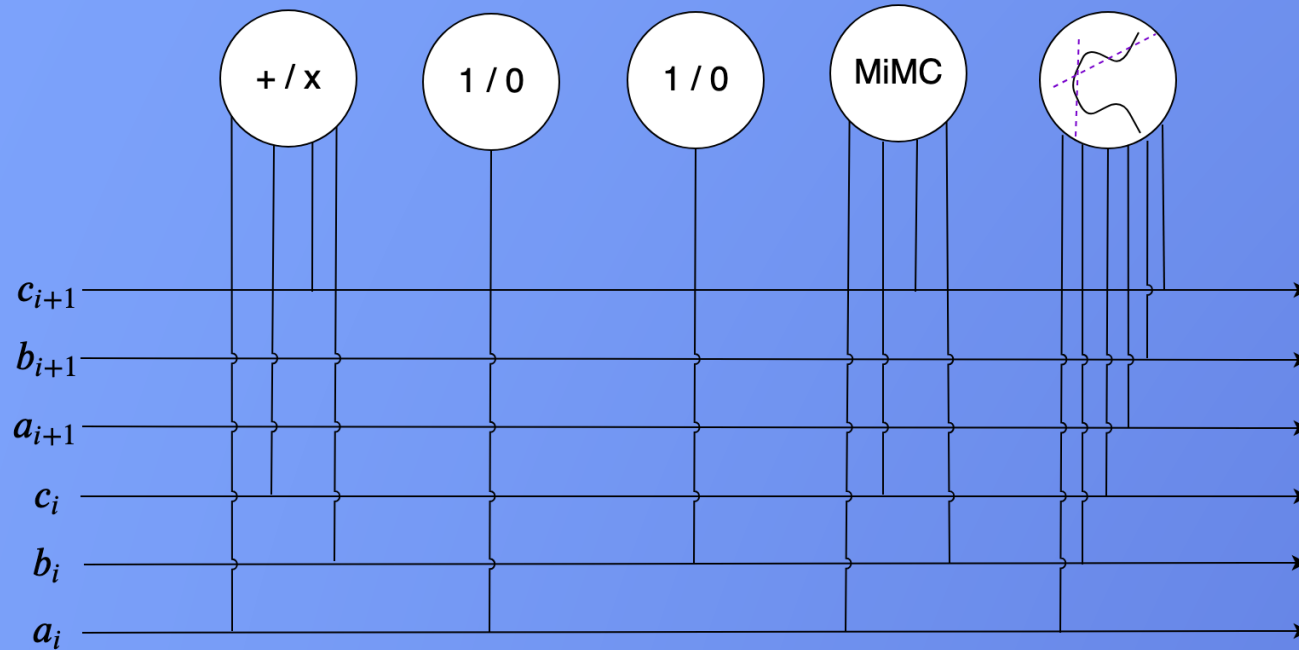


- No!
- Can we do more than add / mul?
- And can we get more than 3 wires?
- ← into this?

*Accessing the *next* gate's wire values is free*

Can evaluate complex 'mini-circuits' by using extra selector polynomials

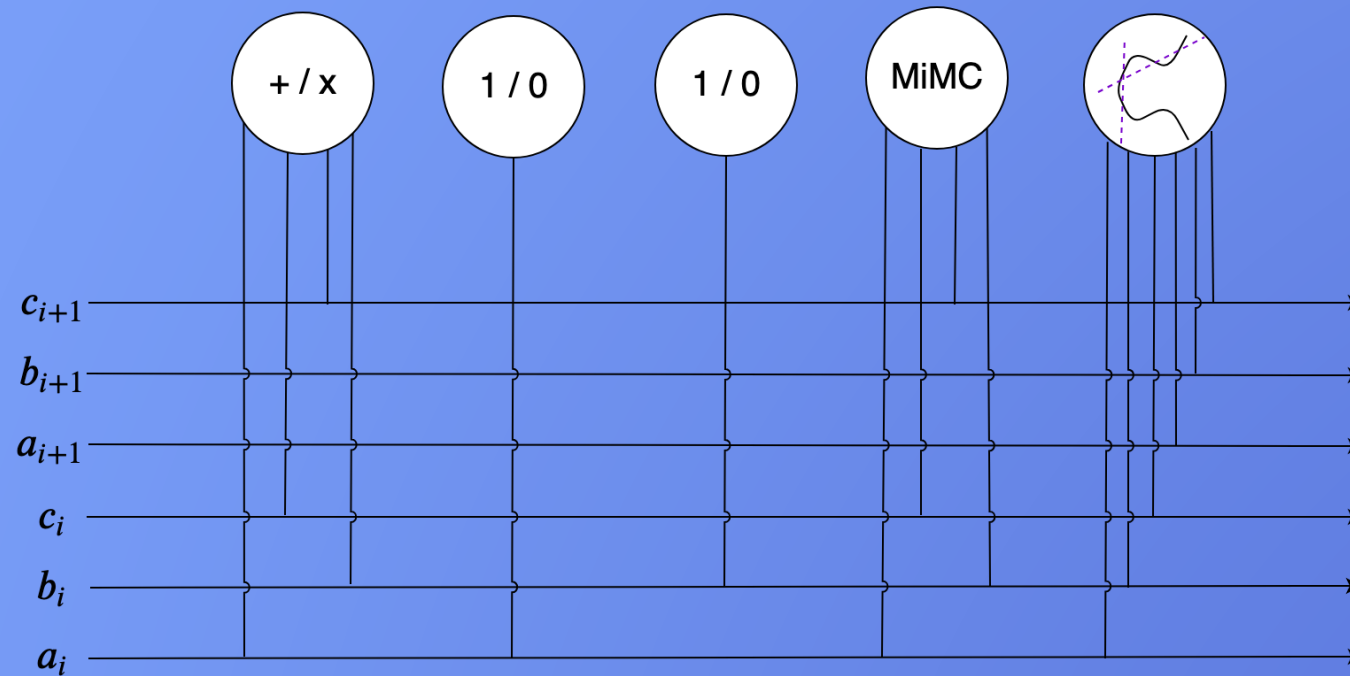
The Turbo-PLONK Polynomial Logic Unit



- PLU gate types:
- Extended add/mul gate
- Bool check gates
- MiMC hash round gate
- ECC point addition gate

- A Turbo-PLONK ‘constraint’ consists of 5 independent mini-gates
- Each ‘mini gate’ can be toggled on/off via selector polynomials

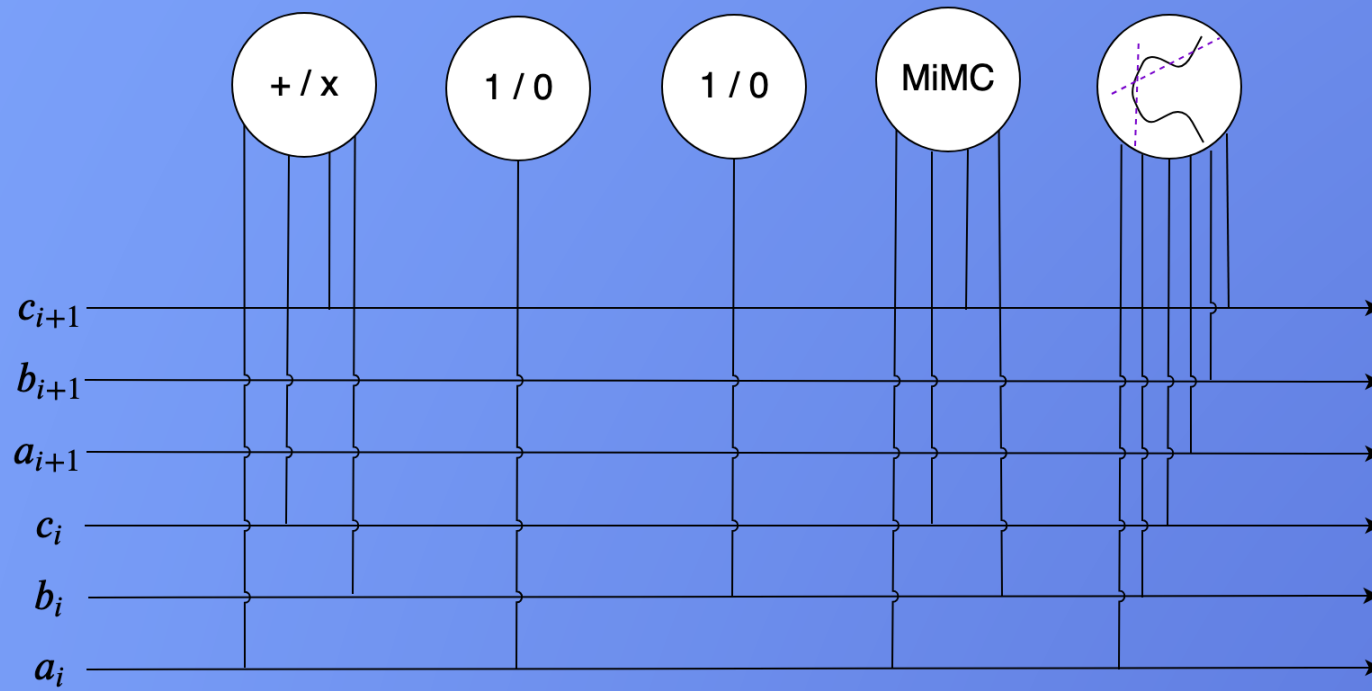
The Turbo-PLONK Polynomial Logic Unit



- Extended + / x gate:

- $q_m(X)A(X)B(X) + q_l(X)A(X) + q_r(X)B(X) + q_o(X)C(X) + q_{oo}(X)C(X\omega) + q_c(X) = 0 \bmod Z_H(X)$

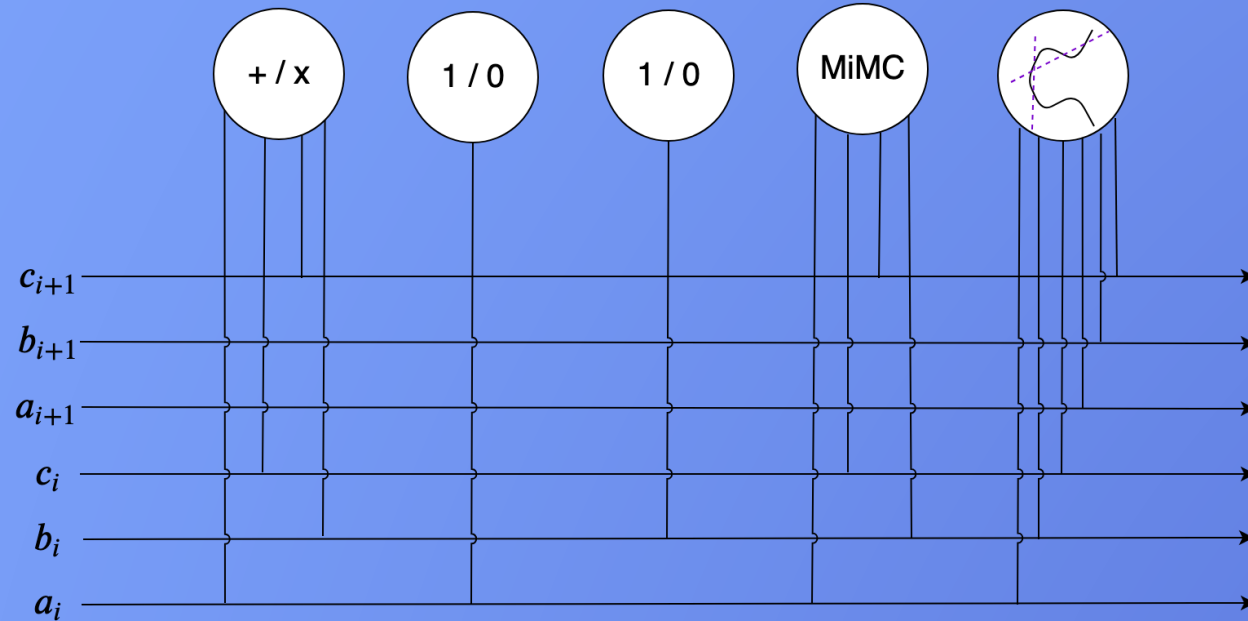
The Turbo-PLONK Polynomial Logic Unit



- Bool gate:

- $q_{bl}(X)(A(X)A(X) - A(X)) = 0 \mod Z_H(X)$

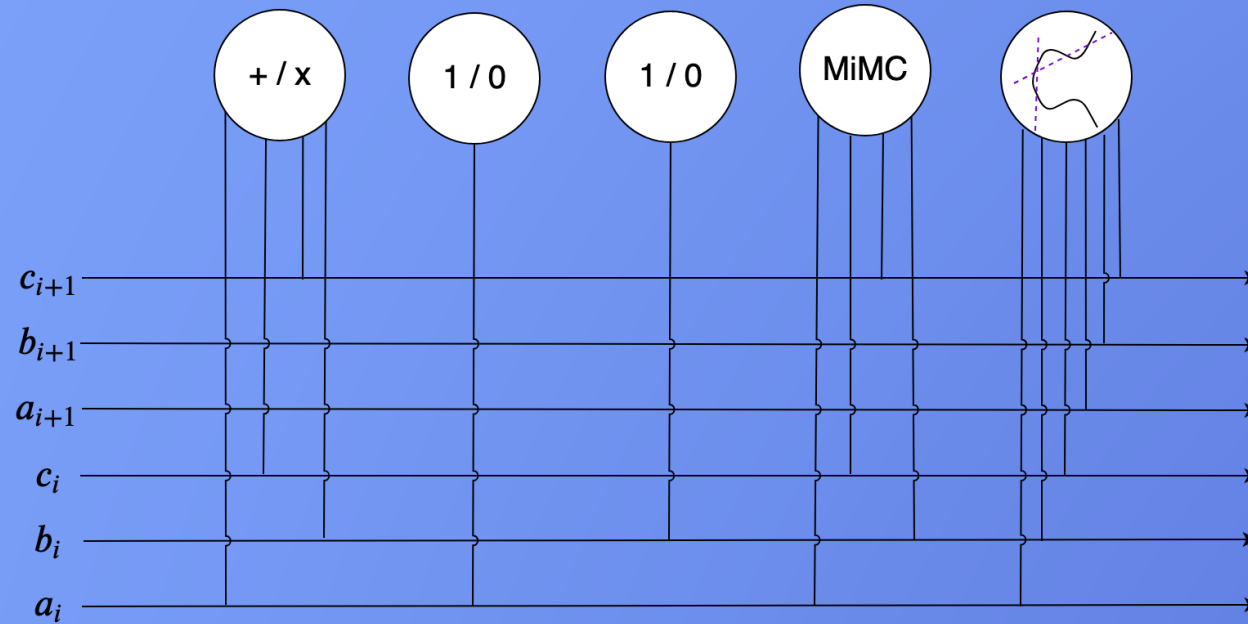
The Turbo-PLONK Polynomial Logic Unit



MiMC Gate

- $q_{MiMC_{sel}}(X) \left(\left(C(X) + A(X) + q_{MiMC_{coeff}} \right)^3 - B(X) \right) = 0 \mod Z_H(X)$
- $q_{MiMC_{sel}}(X) \left(B(X) B(X) \left(C(X) + A(X) + q_{MiMC_{coeff}} \right) - C(X) \omega \right) = 0 \mod Z_H(X)$

The Turbo-PLONK Polynomial Logic Unit



ECC addition gate

- $q_{ECC}(X) \left((A(X\omega) + C(X) + A(X))(C(X) - A(X))(C(X) - A(X)) - (C(X\omega) - B(X))(C(X\omega) - B(X)) \right) = 0 \text{ mod } Z_H(X)$
- $q_{ECC}(X) \left((B(X\omega) + B(X))(C(X) - A(X)) - (C(X\omega) - B(X))(A(X) - A(X\omega)) \right) = 0 \text{ mod } Z_H(X)$



[TRUSTED SETUP MULTI PARTY COMPUTATION]



Coordination Server Status

Ignition : 2019-09-05 18:04:40.35

Time : 2019-09-10 12:12:11.52

Status : AWAITING PARTICIPANT

Participants : (total: 10) (complete: 4) (invalid: 0)

Network : (online: 0) (offline: 10)

Participant Queue

- 03. 0xc6B1C9E9961d2cf7055b35dFBb09DB3978e61419 (46s)
- 04. 0x5e5AaB22d5E22D47efc84f99d22B864A129A7caE (320s)
- 05. 0x3a548c928408762Bfe12267246d4d1B6BC58a150 (3)
- 06. 0x6Bd7Ea43FB9E05F551ad4128dD8E412B15B6a770 (5)
- 07. 0xaf48021C027FA9dE7F915D1899D5372De0270e9f (6)