

## 0.1 Plonk Arithmetization

Consider we have a certain relation  $\mathcal{R}$ , which we would like to write down into a processing-prone format over the field  $\mathbb{F}$ . Plonk arithmetizes this relation into a set of  $8$  *polynomials*, which are then used to verify the witness knowledge. Let us start with the concrete example.

**Example.** To begin with, observe this fairly simple relation  $\mathcal{R}_{\text{example}}$ : suppose we have a public input  $x \in \mathbb{F}$  and public output  $y \in \mathbb{F}$ , and we want to prove the knowledge of  $e \in \mathbb{F}$  such that  $e \times x + x - 1 = y$ . Formally, we have the following relation:

$$\mathcal{R}_{\text{example}} = \left\{ \begin{array}{l} \text{Public Statement: } x, y \in \mathbb{F} \\ \text{Witness: } e \in \mathbb{F} \end{array} \mid e \times x + x - 1 = y \right\}$$

**Remark.** Note that of course, from  $x$  and  $y$ , it is fairly simple to find  $e$ : simply take  $\frac{1-x+y}{x}$ . However, the Plonk arithmetization is not limited to this simple example, and can be applied to more complex relations, such as hash function pre-image knowledge or any NP statement.

### 0.1.1 Execution Trace

Standard Plonk is defined as a system with two types of gates: **addition** and **multiplication**. We would explain how to build custom gates later. So, let us consider our program in terms of gates with left, right operands and output.

**Example.** We need **three gates** to encode our program:

1. **Gate #1:** left  $e$ , right  $x$ , output  $u = e \times x$
2. **Gate #2:** left  $u$ , right  $x$ , output  $v = u + x$
3. **Gate #3:** left  $v$ , right  $x$ , output  $w = v + (-1)$

You might have glanced the intuitive formation of what is called *execution trace table* — a matrix  $T$  with columns  $L$ ,  $R$  and  $O$  (it is common to denote those as  $A, B, C$  to distinguish from another matrix we will discuss later). Moreover, we will mark columns **A**, **B** and **C** in bold to indicate that they are vectors from  $\mathbb{F}^N$ , where here and hereafter, unless stated otherwise,  $N$  is the number of gates in the program.

**Example.** We might visualize the execution trace table  $T$  for the example program as follows:

<b>A</b>	<b>B</b>	<b>C</b>
2	3	6
6	3	9
9	✗	8

Notice how the last row has no value in **B** column (marked by ✗) — this is reasoned by the fact it is not a variable, but rather a constant, meaning it doesn't depend on execution. Also note that the number of gates in this particular circuit is  $N = 3$ .

**Remark.** As you might notice, in contrast to classic R1CS (which we used for Groth16), the standard Plonk arithmetization as is only allows two input values to be processed at a time. This way, if Groth16 requires only one constraint for verifying  $x_1(x_2 + x_3 + x_4) = x_5$ , Plonk would need three constraints to verify the same statement. Custom gates partially solve this problem as we will see later, but it is important to keep in mind.

### 0.1.2 Encode the program

It is essential to distinguish the definition of the program and its specific evaluation for the sake of simplicity and efficiency — once having established encoding for the program, you might apply it for any reasonable inputs. Therefore, let us at first focus on what defines whether execution trace table will be considered valid for our circuit, because having a table by itself does not tell much, since it can be populated with any values.

For that reason, we would define two matrices —  $Q \in \mathbb{F}^{N \times 5}$  and  $V \in \mathbb{Z}_{\geq 0}^{N \times 3}$  where  $N \in \mathbb{N}$ , again, is the number of gates in the program:

- $Q$  is the **Gate Matrix**, which encodes the values of the gates and stores all the intermediate values computed.
- $V$  is the **Wiring Matrix**, which encodes the wiring of the gates, i.e., how the output of one gate is carried as input to another.

**Definition 0.1.** The **gate matrix**  $Q \in \mathbb{F}^{N \times 5}$  has one row per each gate with columns  $Q_L, Q_R, Q_O, Q_M, Q_C$  from  $\mathbb{F}^N$ . If columns  $A, B$  and  $C \in \mathbb{F}^N$  of the execution trace table form valid evaluation of the circuit, then the following holds:

$$A_i(Q_L)_i + B_i(Q_R)_i + A_i B_i (Q_M)_i + C_i (Q_O)_i + (Q_C)_i = 0, \forall i \in [N]$$

Using Hadamard product notation, this can be concisely rewritten as:

$$A \odot Q_L + B \odot Q_R + A \odot B \odot Q_M + C \odot Q_O + Q_C = 0$$

**Example.** For our program, we would have a following  $Q$  table:

$Q_L$	$Q_R$	$Q_M$	$Q_O$	$Q_C$
0	0	1	-1	0
1	1	0	-1	0
1	0	0	-1	-1

You can verify that our claim holds for aforementioned trace matrix:

$$2 \times 0 + 3 \times 0 + 2 \times 3 \times 1 + 6 \times (-1) + 0 = 0$$

$$6 \times 1 + 3 \times 1 + 6 \times 3 \times 0 + 9 \times (-1) + 0 = 0$$

$$9 \times 1 + 0 \times 0 + 9 \times 0 \times 0 + 8 \times (-1) + (-1) = 0$$

Recall that columns of trace matrix  $T$  are  $A = \begin{bmatrix} 2 \\ 6 \\ 9 \end{bmatrix}$ ,  $B = \begin{bmatrix} 3 \\ 3 \\ \text{X} \end{bmatrix}$ ,  $C = \begin{bmatrix} 6 \\ 9 \\ 8 \end{bmatrix}$ .

Now, we do have a way of encoding gates separately, yet in order to guarantee how result of one gate is carried in as input of the other (*wirings*), we need another matrix —  $V$ .

**Definition 0.2.** The **wiring matrix**  $V \in \mathbb{Z}_{\geq 0}^{N \times 3}$  consists of indices of all inputs and intermediate values, so that if  $T$  is a valid trace,

$$\forall(i, j) \forall(k, \ell) : V_{i,j} = V_{k,\ell} \implies T_{i,j} = T_{k,\ell}$$

Put more simply, if two values are equal in  $V$ , then the corresponding values (corresponding to these indices) in  $T$  must be equal as well.

**Example.** For our program,  $V$  can be defined as follows:

L	R	O
0	1	2
2	1	3
3	✗	4

Here 0 is an index of  $e$ , 1 is an index of  $x$ , 2 — of intermediate value  $u$ , 3 — of  $v$  and finally 4 — of output  $w$ .

### 0.1.3 Custom Gates

In order to reach beyond classical operations such as addition and multiplication, one may consider composing a custom gate. The main streamliner of this functionality is a matrix  $Q$ , using 5 basic columns of which, you already may build custom logic.

**Example.** Our entire program may be encoded as one custom gate.

$$Q = \begin{array}{|c|c|c|c|c|} \hline Q_L & Q_R & Q_M & Q_O & Q_C \\ \hline 0 & 1 & 1 & -1 & -1 \\ \hline \end{array} \quad V = \begin{array}{|c|c|c|} \hline L & R & O \\ \hline 0 & 1 & 2 \\ \hline \end{array} \quad T = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline 2 & 3 & 8 \\ \hline \end{array}$$

$$2 \times 0 + 3 \times 1 + 2 \times 3 \times 1 + 8 \times (-1) + (-1) = 0$$

As you can see, custom gates is a good way to reduce the number of constraints needed for the same program.

**Remark.** Real-world PlonK applications commonly have additional columns in the  $Q$  matrix, enabling an even broader set of custom functionality.

### 0.1.4 Public Inputs

With the current design, we can prove that the computations were done correctly, but we have no restrictions on the values of inputs. For example, when the prover wants to convince the verifier that he knows  $e$  for  $x = 3$  and  $y = 7$ , the verifier does not even check whether  $x$  is 3 (not to mention whether the result of execution  $y = 7$  is correct) in the trace table  $T$ . One way of doing this is by incorporating them in three previously defined matrices  $Q$ ,  $V$ ,  $T$ .

**Proposition 0.3.** One way to solve this is to use the **equality gates**. Introduce two gadgets:

- **Constant Equality Gate:** Suppose we want to check whether the certain variable equals to the constant value  $\alpha \in \mathbb{F}$  at gate with index  $i$ . For  $i$ th gate, set  $(Q_L)_i = -1$ ,  $(Q_C)_i = \alpha$  and other columns to 0. Then, add a row to  $V$  with  $L = i$ ,  $R = \text{X}$  and  $O = \text{X}$ . Then, to satisfy the condition, the  $i$ th left input **must** be equal to  $\alpha$ .
- **Nodes Equality Gate:** Suppose we want to check whether the  $i$ th and  $j$ th gates have equal outputs in the  $k$ th gate. Set  $(Q_L)_k = 1$ ,  $(Q_R)_k = -1$  with other columns to 0. Add a row to  $V$  with  $L = i$ ,  $R = j$  and  $O = \text{X}$ . Then, to satisfy the condition, the  $i$ th and  $j$ th outputs **must** be equal.

**Example.** Suppose the prover wants to prove that he knows  $e$  for the public statement  $(x, y) = (3, 8)$ . We can encode this as follows:

$$Q = \begin{array}{c|ccccc} & Q_L & Q_R & Q_M & Q_O & Q_C \\ \hline & -1 & 0 & 0 & 0 & 3 \\ & -1 & 0 & 0 & 0 & 8 \\ & 1 & 1 & 1 & -1 & 1 \\ & 1 & -1 & 0 & 0 & 0 \end{array} \quad V = \begin{array}{c|ccc} & L & R & O \\ \hline & 0 & \text{X} & \text{X} \\ & 1 & \text{X} & \text{X} \\ & 2 & 0 & 3 \\ & 1 & 3 & \text{X} \end{array} \quad T = \begin{array}{c|ccc} & A & B & C \\ \hline & 3 & \text{X} & \text{X} \\ & 8 & \text{X} & \text{X} \\ & 2 & 3 & 8 \\ & 8 & 8 & \text{X} \end{array}$$

As can be seen, besides the original program gate, inscribed in the third row, we have three additional gates:

- The first two gates “allocate” two nodes with indices 0 and 1 to the values 3 and 8 respectively. This is done through the *constant equality gates*.
- The last gate checks whether the result of the third gate is equal to the index 1, corresponding to the allocated value 8. This is done through the *nodes equality gate*.

The primary problem with this approach, is that now we have lost agnosticism in  $Q$  and  $V$  of concrete evaluations. In other words, our circuit is now “hardcoded” to the specific values of public inputs. In order to resolve this, we would define a separate one-column matrix named  $\Pi \in \mathbb{F}^N$ , in which we would encode the public inputs.

**Example.** With only  $Q$  modified, we now have:

$$\Pi = \begin{array}{c|c} & \Pi \\ \hline & 3 \\ & 8 \\ & 0 \\ & 0 \end{array} \quad Q = \begin{array}{c|ccccc} & Q_L & Q_R & Q_M & Q_O & Q_C \\ \hline & -1 & 0 & 0 & 0 & 0 \\ & -1 & 0 & 0 & 0 & 0 \\ & 1 & 1 & 1 & -1 & 1 \\ & 1 & -1 & 0 & 0 & 0 \end{array}$$

**Proposition 0.4** (Wrap-up). The matrix  $T$  with columns  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C} \in \mathbb{F}^N$  encodes correct execution of the program, if the following two conditions hold:

1.  $\forall i \in [N] : A_i(Q_L)_i + B_i(Q_R)_i + A_i B_i(Q_M)_i + C_i(Q_O)_i + (Q_C)_i + \Pi_i = 0$
2.  $\forall (i, j) \forall (k, \ell) : V_{i,j} = V_{k,\ell} \implies T_{i,j} = T_{k,\ell}$

### 0.1.5 Matrices to Polynomials

**Gates Satisfiability.** Now we can traduce the sets of constraints on matrices to just a few equations on polynomials, as we have already done for Groth16. Let  $\omega$  be a primitive  $N$ -th root of unity<sup>1</sup> and let  $\Omega = \{\omega^j\}_{0 \leq j < N}$ . Although currently the choice of set  $\Omega$  might seem totally random, in the next sections we will see how the usage of Fast-Fourier Transform (FFT) will make this choice convenient.

Let  $a, b, c, q_L, q_R, q_M, q_O, q_C, \pi \in \mathbb{F}^{(\leq N)}[X]$  be polynomials of degree at most  $N$  that interpolate corresponding columns from matrices at the domain  $\Omega$ . In other words, we have  $\forall j \in [N] : a(\omega^j) = A_j$  and the same holds for other polynomials.

Notice that if our trace matrix is correct, then the first condition of [Proposition 0.4](#) can be reduced to the following polynomial equation:

$$a(\omega^j)q_L(\omega^j) + b(\omega^j)q_R(\omega^j) + a(\omega^j)b(\omega^j)q_M(\omega^j) + c(\omega^j)q_O(\omega^j) + q_C(\omega^j) + \pi(\omega^j) = 0, \quad \forall j \in [N]$$

Notice that this essentially means that the left polynomial  $aq_L + bq_R + abq_M + cq_O + q_C + \pi$  has roots at  $\omega^j$  for all  $j \in [N]$ . This is equivalent to stating that the polynomial  $z_\Omega(X) = \prod_{j=0}^{N-1} (X - \omega^j)$  divides the left hand side. Now, the interesting fact. . .

**Lemma 0.5.** It so happens that if  $\Omega$  is a set of  $N$ -th roots of unity, then the polynomial  $z_\Omega(X) = X^N - 1$  is the vanishing polynomial of  $\Omega$ .

**Proof Idea.** If  $\omega$  is the  $N$ th primitive root, then for any  $h \in \Omega$  we have  $h^N = 1$  and therefore all elements of  $\Omega$  are the roots of  $X^N - 1$ . There are precisely  $N$  such roots, so  $X^N - 1$  can be decomposed as a product of linear factors  $c \cdot \prod_{j=0}^{N-1} (X - \omega^j)$ . It is easy to see that  $c = 1$  by comparing the leading coefficient.

Aha! So we have that  $X^N - 1$  must divide the left polynomial. Let us wrap this up in the following proposition.

**Proposition 0.6.** Now we can reduce down our first condition of [Proposition 0.4](#) to checking valid execution trace into the following claim over polynomials:

$$\exists t \in \mathbb{F}^{(\leq 3N)}[X] : aq_L + bq_R + abq_M + cq_O + q_C + \pi = z_\Omega t,$$

where  $z_\Omega(X)$  is the vanishing polynomial  $X^N - 1$ .

**Wiring Satisfiability.** The next step is to shrink the second condition imposed by the  $V$  matrix. This may be achieved by introducing the concept of permutation.

**Remark.** Permutation of the set  $S$  is commonly denoted as  $\sigma : S \rightarrow S$ . This function is bijective, meaning that for every  $s \in S$  there exists a unique  $s' \in S$  such that  $\sigma(s) = s'$ .

---

<sup>1</sup>Suppose such  $\omega$  exists, then  $\omega^N = 1$  and  $\omega^j \neq 1$  for  $0 \leq j < N$ .

**Example.** A permutation is a rearrangement of the set, which is in our case:

$$\mathcal{I} = \{(i, j) : \text{such that } 0 \leq i < N, \text{ and } 0 \leq j < 3\}$$

Naturally, the matrix  $V$  induces a permutation  $\sigma$  of this set where  $\sigma((i, j))$  equals to the pair of indices of the next occurrence of the value at position  $(i, j)$ . So, for our example:

$$V = \begin{array}{|c|c|c|} \hline \mathbf{L} & \mathbf{R} & \mathbf{O} \\ \hline 0 & \times & \times \\ \hline 1 & \times & \times \\ \hline 2 & 0 & 3 \\ \hline 1 & 3 & \times \\ \hline \end{array}$$

We have the following permutation:

$$\sigma((0,0)) = (2,1), \sigma((0,1)) = (0,3), \sigma((0,2)) = (0,2)$$

$$\sigma((0,3)) = (0,1), \sigma((2,1)) = (0,0), \sigma((3,1)) = (2,2)$$

For demonstration purposes, we marked in **green** the index of the first and second occurrence of the value 0. For proper  $\sigma$  definition (as it has to be bijective), the application of  $\sigma$  to the last occurrence outputs the first one.

**Permutation Check.** This is probably the most tedious part of PlonK. We split the following derivation into two parts:

- **Set Equality using Polynomials.** We will show how to check whether two sets of field elements are equal using polynomials.
- **Permutation Check using Polynomials.** We will show how to check whether a given function is a permutation using polynomials in several forms.

**Set equality.** Having defined permutation, we can now reduce the second condition of **Proposition 0.4** of valid execution trace matrix into the following check:

$$\forall (i, j) \in \mathcal{I} : T_{i,j} = T_{\sigma(i,j)}$$

You may have noticed how this can be reformulated as equality of two sets  $A$  and  $B$ :

$$\begin{aligned} A &:= \{((i, j), T_{i,j}) : (i, j) \in \mathcal{I}\} \\ B &:= \{(\sigma((i, j)), T_{i,j}) : (i, j) \in \mathcal{I}\} \end{aligned}$$

We can reduce this check down to polynomial equations! Here is how: suppose for simplicity we have two sets with two elements  $A = \{a_0, a_1\}$  and  $B = \{b_0, b_1\}$ . Introduce two sets of polynomials  $A' = \{a_0 + X, a_1 + X\}$  and  $B' = \{b_0 + X, b_1 + X\}$ .

When do we have the set equality  $A' = B'$ ? Well,  $(a_0 + X)(a_1 + X) = (b_0 + X)(b_1 + X)$  works fine. This is true because of linear polynomial unique factorization property, working as prime factors. Now, we can utilize Schwartz-Zippel lemma to replace the latter formula with  $(a_0 + \gamma)(a_1 + \gamma) = (b_0 + \gamma)(b_1 + \gamma)$  for some random  $\gamma \xleftarrow{R} \mathbb{F}$  with overwhelming probability, being at least  $1 - 2/|\mathbb{F}|$ . If we wish to generalize this for arbitrary sets  $A = \{a_0, \dots, a_{k-1}\}$  and  $B = \{b_0, \dots, b_{k-1}\}$ , apply the following equivalent check:

$$\prod_{i=0}^{k-1} (a_i + \gamma) = \prod_{i=0}^{k-1} (b_i + \gamma)$$

Let  $\Omega$  be a domain of the form  $\{1, \omega, \dots, \omega^{k-1}\}$  for some  $k$ -th root of unity  $\omega$ . Let  $f$  and  $g$  be polynomials that we interpolate at  $\Omega$  as follows:

$$f(\omega^j) = a_j + \gamma, \quad g(\omega^j) = b_j + \gamma, \quad j \in [k]$$

Then,  $\prod_{i=0}^{k-1} (a_i + \gamma) = \prod_{i=0}^{k-1} (b_i + \gamma)$  holds if and only if there is a polynomial  $Z \in \mathbb{F}[X]$  such that for all  $h \in \Omega$  we have  $Z(\omega^0) = 1$  and  $Z(h)f(h) = g(h)Z(\omega h)$ .

Now that we can encode equality of sets of field elements, let's expand this to sets of tuples of field elements. Let  $A = \{(a_0, a_1), (a_2, a_3)\}$  and  $B = \{(b_0, b_1), (b_2, b_3)\}$ . Then, similarly, if

$$A' = \{a_0 + a_1Y + X, a_2 + a_3Y + X\}, \quad B' = \{b_0 + b_1Y + X, b_2 + b_3Y + X\},$$

then  $A = B$  if and only if  $A' = B'$ . As before, we can leverage Schwartz-Zippel lemma to reduce this down into sampling two random  $\beta$  and  $\gamma \xleftarrow{R} \mathbb{F}$  and checking equality of:

$$(a_0 + \beta a_1 + \gamma)(a_2 + \beta a_3 + \gamma) = (b_0 + \beta b_1 + \gamma)(b_2 + \beta b_3 + \gamma)$$

**Permutation Check.** Now, to go back to the second condition of [Proposition 0.4](#) which we are trying to formulate in the polynomial domain, it becomes clear that if we somehow encoded inner indices tuple  $(i, j)$  into a one field element, we could use the above fact. Recall that  $i \in [N]$  and  $j \in \{0, 1, 2\}$ . Thus, take the  $3N$ -th primitive root of unity  $\eta$  and define the bijective map  $((i, j), v) \mapsto (\eta^{3i+j}, T_{i,j})$ . Thus, consider the modified sets:

$$\begin{aligned} A &= \{(\eta^{3i+j}, T_{i,j}) : (i, j) \in \mathcal{I}\} \\ B &= \{(\eta^{3k+\ell}, T_{i,j}) : (i, j) \in \mathcal{I}, \sigma((i, j)) = (k, \ell)\} \end{aligned}$$

Sample two random field elements  $\beta$  and  $\gamma \xleftarrow{R} \mathbb{R}$ . Let  $\mathcal{D} = \{1, \eta, \eta^2, \dots, \eta^{3N-1}\}$ . Then, interpolate two polynomials  $f$  and  $g$  over the defined set  $\mathcal{D}$  as follows:

$$\begin{aligned} f(\eta^{3i+j}) &= T_{i,j} + \eta^{3i+j}\beta + \gamma, \quad (i, j) \in \mathcal{I} \\ g(\eta^{3k+\ell}) &= T_{i,j} + \eta^{3k+\ell}\beta + \gamma, \quad (i, j) \in \mathcal{I}, \quad \sigma((i, j)) = (k, \ell) \end{aligned}$$

Similarly to our previous discussion, there should be a polynomial  $Z \in \mathbb{F}[X]$  such that  $\forall d \in \mathcal{D}$ , we have  $Z(\eta^0) = 1$  and  $Z(d)f(d) = g(d)Z(\eta d)$ . This would imply the set equality  $A = B$  with overwhelming probability according to Schwartz-Zippel lemma.

**Shorter Form.** Now, using the  $3N$ -th root of unity is a bit of overkill, so let us try compressing it down to  $\Omega = \{\omega^j\}_{0 \leq j < N}$  where  $\omega$  is the  $N$ th root of unity. We will define three polynomials  $S_{\sigma,1}, S_{\sigma,2}, S_{\sigma,3} \in \mathbb{F}[X]$ , which are interpolated as follows:

$$\begin{aligned} S_{\sigma,1}(\omega^i) &= \eta^{3k+\ell}, \quad (i, 0) \in \mathcal{I}, \quad \sigma((i, 0)) = (k, \ell) \\ S_{\sigma,2}(\omega^i) &= \eta^{3k+\ell}, \quad (i, 1) \in \mathcal{I}, \quad \sigma((i, 1)) = (k, \ell) \\ S_{\sigma,3}(\omega^i) &= \eta^{3k+\ell}, \quad (i, 2) \in \mathcal{I}, \quad \sigma((i, 2)) = (k, \ell) \end{aligned}$$

Let  $k_1$  and  $k_2$  be two field elements such that  $\omega^i \neq \omega^j k_1 \neq \omega^\ell k_2$  for all possible triplets  $i, j, \ell$ . Recall that  $\beta$  and  $\gamma$  are random field elements. Let  $f$  and  $g$  be the polynomials that interpolate, respectively, the following values at  $\Omega$ :

$$\begin{aligned} f(\omega^i) &= (T_{i,0} + \omega^i \beta + \gamma) (T_{i,1} + \omega^i k_1 \beta + \gamma) (T_{i,2} + \omega^i k_2 \beta + \gamma), \quad i \in [N] \\ g(\omega^i) &= (T_{i,0} + S_{\sigma,1}(\omega^i) \beta + \gamma) (T_{i,0} + S_{\sigma,2}(\omega^i) \beta + \gamma) (T_{i,0} + S_{\sigma,3}(\omega^i) \beta + \gamma), \quad i \in [N] \end{aligned}$$

That being said, there is a polynomial  $Z \in \mathbb{F}[X]$  such that  $\forall d \in \mathcal{D}$  we have  $Z(\omega^0) = 1$  and  $Z(d)f(d) = g(d)Z(\omega d)$ , implying  $A = B$  with overwhelming probability. That being said, we now can encode our program using 8 polynomials mentioned at the very beginning:

$$q_L, q_R, q_M, q_O, q_C, S_{\sigma,1}, S_{\sigma,2}, S_{\sigma,3}$$

These are typically called **common preprocessed input**.

### 0.1.6 Summary

Having a program for relation  $\mathcal{R}$ , we saw how it can be represented as a sequence of gates with left, right operands and output. The circuit may be encoded using two matrices  $Q$  — for capturing gates, and  $V$  — for encoding value carries (*wirings*). Upon execution, we get trace execution matrix  $T$  and  $\Pi$  for public inputs.

**Definition 0.7.** Let  $T \in \mathbb{F}^{N \times 3}$  be a trace matrix with columns  $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^N$  and let  $\Pi \in \mathbb{F}^N$  be a public input vector. They correspond to a valid execution instance with public input given by  $\Pi$  if and only if:

1.  $\forall i \in [N] : A_i(Q_L)_i + B_i(Q_R)_i + A_i B_i(Q_M)_i + C_i(Q_O)_i + (Q_C)_i + \Pi_i = 0$
2.  $\forall (i, j), \forall (k, \ell) : V_{i,j} = V_{k,\ell} \implies T_{i,j} = T_{k,\ell}$
3.  $\forall i > n : \Pi_i = 0$ , where  $n$  is the number of public inputs.

Then, we encode these conditions in terms of polynomials.

**Definition 0.8.** Let  $z_\Omega = X^N - 1$  be a vanishing polynomial. Let  $T \in \mathbb{F}^{N \times 3}$  be a trace matrix with columns  $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^N$  and  $\Pi \in \mathbb{F}^N$  be a vector of public signals. They correspond to a valid execution instance with public input given by  $\Pi$  if and only if:

1.  $\exists t_1 \in \mathbb{F}[X] : a q_L + b q_R + a b q_M + c q_O + q_C + \pi = z_\Omega t_1$
2.  $\exists t_2, t_3, z \in \mathbb{F}[X] : z f - g z' = z_\Omega t_2$  and  $(z - 1)L_1 = z_\Omega t_3$ , where  $z'(X) = z(X\omega)$ .

**Remark.** We can reduce every needed check down to one equation, if we introduce randomness. Let  $\alpha$  be a random field element, then:

$$\begin{aligned} z_\Omega t &= a q_L + b q_R + a b q_M + c q_O + q_C + \pi \\ &= \alpha(g z' - f z) \\ &= \alpha^2(z - 1)L_1 \end{aligned}$$

The transition between second and third line is very unobvious and requires a bit of algebraic manipulation. Don't worry if you don't see it immediately.



## 0.2 Plonk Prover and Verifier

In this part, we will observe how the Plonk turns its arithmetization system into the non-interactive proof. We consider how the prover and verifier specifically interact in the protocol.

As in the previous sections, let  $N$  be the size of our program (number of gates). Let  $\omega$  be a primitive  $N$ -th root of unity. Let  $\Omega = \{\omega^j\}_{0 \leq j < N}$ . Define  $Z_\Omega(X) := X^N - 1$  — the vanishing polynomial of  $\Omega$ .

Assume we have conducted arithmetization of this program, obtaining eight polynomials. Now we will demonstrate commitment-scheme agnostic non-interactive (Fiat-Shamir) algorithms for both prover and verifier.

**Remark.** Although the further discussion would involve statements like “the prover picks the random  $\alpha, \beta, \dots$ ”, it is important to note that these values are not actually chosen by the prover, but rather computed deterministically from the transcript using Fiat-Shamir heuristic. In case you are not familiar with Fiat-Shamir heuristic, we recommend you to revisit the ??.

### 0.2.1 Gadgets

Before diving into the protocol, let us introduce the gadgets that will be used.

**Commitment Scheme.** We use the polynomial commitment scheme to commit to the polynomials. Similarly to Groth16, one might safely think of KZG commitment scheme to commit to the polynomials: for  $p(x) \in \mathbb{F}^{(\leq N)}[X]$ , the commitment is  $\text{com}(p) = g^{p(\tau)}$ , which is evaluated using the powers of the generator  $g$ . The opening procedure at point  $\zeta \in \mathbb{F}$  of the commitment is simply finding the encryption  $g^{Q(\tau)}$  of the quotient polynomial  $Q(X) := (p(X) - y)/(X - \zeta)$ , where  $y$  is the value of the polynomial at  $\zeta$ . Finally, verification simply involves applying pairing. Note that other commitment schemes might be used as well.

**Blindings.** In the protocol, to achieve the HZVK property (honest verifier zero-knowledge), the prover must somehow conceal the polynomials he works with. Specifically, consider the polynomial  $b(X)$  that is computed as the interpolated values from the trace table (namely,  $b(\omega^j) = T_{j,1}$ ). However, the prover, by providing the opening  $b(\zeta)$ , should not reveal any information on polynomial  $b(X)$  itself. To achieve this, the prover extends  $b(X) \in \mathbb{F}^{(\leq N)}[X]$  to the higher-degree polynomial  $\tilde{b}(X)$  with the condition that  $\tilde{b}(\omega^j) = b(\omega^j) = T_{j,1}$ . However, for higher degrees, there are multitude of choices for  $\tilde{b}(X)$ , so we need to standardize this choice.

Assume we extend  $b(X)$  to  $\tilde{b}(X) \in \mathbb{F}^{(\leq N')}[X]$  of higher degree  $N' > N$ . For convenience, assume  $N' = N + \delta$  with  $\delta \in \mathbb{N}$ . How can we construct such  $\tilde{b}(X)$ ? We can do this by sampling random values  $\gamma_0, \gamma_1, \dots, \gamma_\delta \xleftarrow{R} \mathbb{F}$  and letting:

$$\tilde{b}(X) = b(X) + z_\Omega(X) \sum_{j=0}^{\delta} \beta_j X^j$$

Why does this work? Let us substitute any  $\omega^i \in \Omega$ :

$$\tilde{b}(\omega^i) = b(\omega^i) + z_\Omega(\omega^i) \sum_{j=0}^{\delta} \beta_j \omega^{ij} = b(\omega^i).$$

Here, we use the fact that  $z_\Omega(\omega^i) = 0$  for all  $i \in [N]$  since  $z_\Omega$  is, by definition, the vanishing polynomial of  $\Omega \ni \omega^i$ .

## 0.2.2 Proving

First, the prover computes execution trace table  $T \in \mathbb{F}^{N \times 3}$ . Then, Let us finally proceed with the protocol.

**Round 1.** Add to the transcript commitments of 8 arithmetizational polynomials:

$$\text{com}(S_{\sigma,1}), \text{com}(S_{\sigma,2}), \text{com}(S_{\sigma,3}), \text{com}(q_L), \text{com}(q_R), \text{com}(q_M), \text{com}(q_O), \text{com}(q_C)$$

Interpolate polynomials  $a', b', c'$  over corresponding columns of  $T$  matrix at the domain  $\Omega$ . Sample random  $b_1, b_2, b_3, b_4, b_5, b_6 \xleftarrow{R} \mathbb{F}$ . Let the blinded polynomials be:

$$\begin{aligned} a &:= (b_1X + b_2)Z_{\Omega}(X) + a'(X) \\ b &:= (b_3X + b_4)Z_{\Omega}(X) + b'(X) \\ c &:= (b_5X + b_6)Z_{\Omega}(X) + c'(X) \end{aligned}$$

Add to the transcript commitments of computed above polynomials:

$$\text{com}(a), \text{com}(b), \text{com}(c)$$

**Round 2.** Sample random scalars  $\beta, \gamma \xleftarrow{R} \mathbb{F}$  from the transcript. Let  $z_0 = 1$  and define recursively for  $k = 0, \dots, N$ :

$$z_{k+1} = z_k \times \frac{(a_k + \beta\omega^k + \gamma)(b_k + \beta\omega^k k_1 + \gamma)(c_k + \beta\omega^k k_2 + \gamma)}{(a_k + \beta S_{\sigma,1}(\omega^k) + \gamma)(b_k + \beta S_{\sigma,2}(\omega^k) + \gamma)(c_k + \beta S_{\sigma,3}(\omega^k) + \gamma)}$$

Interpolate polynomial  $z'$  over evaluations  $(z_0, \dots, z_{N-1})$  at the domain  $\Omega$ .

Sample random  $b_7, b_8, b_9 \xleftarrow{R} \mathbb{F}$ . Let  $z = (b_7X^2 + b_8X + b_9)Z_H + z'$ . Add to the transcript  $\text{com}(z)$ .

**Round 3** Sample  $\alpha \xleftarrow{R} \mathbb{F}$  from the transcript. Let  $\pi(X)$  be the interpolation polynomial of the input matrix  $\Pi$  at the domain  $\Omega$ . Let

$$\begin{aligned} p_1 &= aq_L + bq_R + abq_M + cq_O + q_C + \pi \\ p_2 &= (a + \beta X + \gamma)(b + \beta k_1 X + \gamma)(c + \beta k_2 X + \gamma)z - \\ &\quad - (a + \beta S_{\sigma,1} + \gamma)(b + \beta S_{\sigma,2} + \gamma)(c + \beta S_{\sigma,3} + \gamma)z(\omega X) \\ p_3 &= (z - 1)L_1 \end{aligned}$$

Define the composite polynomial  $p = p_1 + \alpha p_2 + \alpha^2 p_3$ . Compute  $t$  such that  $p = tZ_{\Omega}$ . Write  $t = t_{lo} + X^{N+2}t_{mid} + X^{2(N+2)}t_{hi}$ , with  $t_{lo}, t_{mid}$ , and  $t_{hi} \in \mathbb{F}^{\leq(N+1)}[X]$  polynomials of degree at most  $N + 1$ . Sample random  $b_{10}, b_{11} \xleftarrow{R} \mathbb{F}$  and define:

$$\begin{aligned} t_{lo} &= t'_{lo} + b_{10}X^{N+2} \\ t_{mid} &= t'_{mid} - b_{10} + b_{11}X^{N+2} \\ t_{hi} &= t'_{hi} - b_{11} \end{aligned}$$

Add to the transcript commitments:  $\text{com}(t_{lo}), \text{com}(t_{mid}), \text{com}(t_{hi})$ .

**Round 4** Sample random  $\zeta \xleftarrow{R} \mathbb{F}$  from the transcript. Compute:

$$\bar{a} = a(\zeta), \bar{b} = b(\zeta), \bar{c} = c(\zeta), \bar{S}_{\sigma,1} = S_{\sigma,1}(\zeta), \bar{S}_{\sigma,2} = S_{\sigma,2}(\zeta), \bar{z}_{\omega} = z(\zeta\omega)$$

and add them to the transcript.

**Round 5** Sample random  $v \xleftarrow{R} \mathbb{F}$  from the transcript. Let:

$$\begin{aligned}\hat{p}_{nc1} &= \bar{a}q_L + \bar{b}q_R + \bar{a}\bar{b}q_M + \bar{c}q_o + q_C \\ \hat{p}_{nc2} &= (\bar{a} + \beta\zeta_1 + \gamma)(\bar{b} + \beta k_1\zeta_1 + \gamma)(\bar{c} + \beta k_2\zeta_1 + \gamma)z - \\ &\quad - (\bar{a} + \beta\bar{S}_{\sigma_1} + \gamma)(\bar{b} + \beta\bar{S}_{\sigma_2} + \gamma)(\bar{c} + \beta\bar{S}_{\sigma_3} + \gamma)z(\omega\zeta_1) \\ \hat{p}_{nc3} &= L_1(\zeta_1)z\end{aligned}$$

Define:

$$\begin{aligned}p_{nc} &= p_{nc1} + \alpha p_{nc2} + \alpha^2 p_{nc3} \\ t_{partial} &= t_{lo} + \zeta^{N+2}t_{mid} + \zeta^{2(N+2)}t_{hi}\end{aligned}$$

The subscript  $nc$  stands for "nonconstant," as it is the part of the linearization of  $p$  with nonconstant factors. The subscript  $partial$  indicates that it is a partial evaluation of  $t$  at  $\zeta$ . Partial means that only some power of  $X$  is replaced by the powers of  $\zeta$ . So in particular  $t_{partial}(\zeta) = t(\zeta)$ . Let  $\pi_{batch}$  be the opening proof at  $\zeta$  of the polynomial  $f_{batch}$  defined as:

$$t_{partial} + vp_{nc} + v^2a + v^3b + v^4c + v^5S_{o1} + v^6S_{o2}$$

Let  $\pi_{single}$  be the opening proof at  $\zeta\omega$  of the polynomial  $z$ .

Compute  $\bar{p}_{nc} := p_{nc}(\zeta)$  and  $\bar{t} = t(\zeta)$ .

**Proof**

$$\begin{aligned}com(a), com(b), com(c), com(z), com(t_{lo}), com(t_{mid}), com(t_{hi}), \\ \bar{a}, \bar{b}, \bar{c}, \bar{S}_{o1}, \bar{S}_{o2}, \bar{z}_w, T_{batch}, T_{single}, \bar{p}_{nc}, \bar{t}\end{aligned}$$

### 0.2.3 Verification

**Transcript Initialization** Same as prover, verified adds the following to the transcript:

$$com(S_{o1}), com(S_{o2}), com(S_{o3}), com(q_L), com(q_R), com(q_M), com(q_o), com(q_C)$$

**Extraction of values and commitments** Firstly, the verifier needs to compute all the challenges. For that, he follows these steps:

- Add  $com(a)$ ,  $com(b)$ ,  $com(c)$  to the transcript.
- Sample two challenges  $\beta, \gamma$ .
- Add  $com(z)$  to the transcript.
- Sample a challenge  $\alpha$ .
- Add  $com(t_{lo})$ ,  $com(t_{mid})$ ,  $com(t_{hi})$  to the transcript.
- Sample a challenge  $\zeta$ .
- Add  $\bar{a}, \bar{b}, \bar{c}, \bar{S}_{o1}, \bar{S}_{o2}, \bar{z}_w$  to the transcript.
- Sample a challenge  $v$ .

**Compute  $pi(\zeta)$**  Also, he needs to compute a few values of all these data. First, he computes the  $\Pi$  matrix with the public inputs and outputs. He needs to compute  $pi(\zeta)$ , where  $pi$  is the interpolation of  $\Pi$  at the domain  $H$ . But he doesn't need to compute  $pi$ . He can instead compute  $pi(\zeta)$  as:

$$\sum_{i=0}^n L_i(\zeta) P_{I_i}$$

where  $n$  is the number of public inputs and  $L_i$  is the Lagrange basis at the domain  $H$ .

**Compute claimed values  $p(\zeta)$  and  $t(\zeta)$**  Compute:

$$\bar{p}_c = p_1(\zeta) + \alpha z_w (\bar{c} + \gamma) (\bar{a} + \beta \bar{S}_{\sigma_1} + \gamma) (\bar{b} + \beta \bar{S}_{\sigma_2} + \gamma) - \alpha^2 L_1(\zeta)$$

This is the constant part of the linearization of  $p$ . So, adding it to what the prover claims to be  $\bar{p}_{nc}$ , he obtains  $p(\zeta) = \bar{p}_c + \bar{p}_{nc}$ .

**Compute  $com(t_{partial})$  and  $com(p_{nc})$**  He computes these of the commitments in the proof as follows:

$$com(t_{partial}) = com(t_{lo}) + \zeta^{N+2} com(t_{mid}) + \zeta^{2(N+2)} com(t_{hi})$$

For the second one, compute those:

$$\begin{aligned} \hat{p}_{nc1} &= \bar{a} * com(q_L) + \bar{b} * com(q_R) + (\bar{a}\bar{b}) * com(q_M) + \bar{c} * com(q_o) + com(q_C) \\ \hat{p}_{nc2} &= (\bar{a} + \beta \zeta_1 + \gamma)(\bar{b} + \beta k_1 \zeta_1 + \gamma)(\bar{c} + \beta k_2 \zeta_1 + \gamma) * com(z) - \\ &\quad - (\bar{a} + \beta \bar{S}_{\sigma_1} + \gamma)(\bar{b} + \beta \bar{S}_{\sigma_2} + \gamma)(\bar{c} + \beta \bar{S}_{\sigma_3} + \gamma) * com(z)(\omega \zeta_1) \\ \hat{p}_{nc3} &= L_1(\zeta_1) * com(z) \end{aligned}$$

Then:

$$com(p_{nc}) = com(p_{nc1}) + com(p_{nc2}) + com(p_{nc3})$$

.

**Compute claimed value  $f_{batch}(\zeta)$  and  $com(f_{batch})$**

$$\begin{aligned} f_{batch}(\zeta) &= \bar{t} + v \bar{p}_{nc} + v^2 \bar{a} + v^3 \bar{b} + v^4 \bar{c} + v^5 \bar{S}_{o1} + v^6 \bar{S}_{o2} \\ com(f_{batch}) &= com(t_{partial}) + v * com(p_{nc}) + v^2 * com(a) + \\ &\quad + v^3 * com(b) + v^4 * com(c) + v^5 * com(S_{o1}) + v^6 * com(S_{o2}) \end{aligned}$$

**Proof check** Now the verifier has all the necessary values to proceed with the checks.

- Check that  $p(\zeta)$  equals  $(\zeta^N - 1)t(\zeta)$ .
- Verify the opening of  $f_{batch}$  at  $\zeta$ . That is, check that

$$\text{Verify}([f_{batch}], \pi_{batch}, \zeta, f_{batch}(\zeta)) \text{ outputs Accept.}$$

- Verify the opening of  $z$  at  $\zeta_w$ . That is, check the validity of the proof  $\pi_{single}$  using the commitment  $com(z)$  and the value  $\bar{z}_w$ .

$$\text{That is, check that } \text{Verify}(com(z), \pi_{single}, \zeta_w, \bar{z}_w) \text{ outputs Accept.}$$

If all checks pass, output Accept. Otherwise, output Reject.