$\mathcal{U}\mathrm{ni}\mathcal{P}\mathrm{lon}\mathcal{K}$: $\mathcal{P}\mathrm{lon}\mathcal{K}$ with Universal Verifier

Shumo Chu, Brandon H. Gomes, Francisco Hernández Iglesias,
Todd Norton, Duncan Tebbs *
https://nebra.one

Abstract

We propose $\mathcal{U}\mathrm{ni}\mathcal{P}\mathrm{lon}\mathcal{K}$, a modification of the $\mathcal{P}\mathrm{lon}\mathcal{K}$ protocol that uniformizes the Verifier's work for families of circuits. Specifically, a single fixed-cost "Universal Verifier" can check proofs for circuits of different: sizes, public input lengths, selector polynomials, copy constraints, and even different custom gate sets. $\mathcal{U}\mathrm{ni}\mathcal{P}\mathrm{lon}\mathcal{K}$ therefore extends the universality of $\mathcal{P}\mathrm{lon}\mathcal{K}$ beyond the SRS; it enables a single "Universal Verifier Circuit" capable of verifying proofs from different $\mathcal{P}\mathrm{lon}\mathcal{K}$ circuits.

The Universal Verifier's marginal cost over the ordinary Plonk verifier is small: for circuits using only the vanilla Plonk gate, the Universal Verifier performs a number of additional field multiplications proportional to the logarithm of the maximum supported circuit size; it incurs no additional elliptic curve operations. For circuits using custom gates, the Universal Verifier incurs additional elliptic curve arithmetic only when verifying proofs from circuits that do not use all supported gate types. For circuits that use all supported gates, the Universal Verifier's additional cost consists only of field multiplications proportional to the logarithm of the maximum supported circuit size, the number of custom gate types, and the number of witness variables used by these gates. In both settings (vanilla-only and custom gates) the marginal cost to the prover is a fixed-base MSM of size ℓ, the length of the public input vector.

Contents

*ordered alphabetically, { shumo, bhgomes, francisco, todd, dtebbs } @nebra.one

1 Introduction

In this paper, we propose $\mathcal{U}\mathsf{ni}\mathcal{P}\mathsf{lonK}$, which extends the universality of the family of $\mathcal{P}\mathsf{lonK}$ SNARK schemes by building a "Universal Verifier Circuit" capable of verifying proofs from different $\mathcal{P}\mathsf{lonK}$ circuits, i.e. circuits of different sizes, public input lengths, and different sets of custom gates used. An important feature of our scheme is that it can be used in combination with recent efficient recursive proof schemes such as Halo Infinite [6] or Nova [20], to build efficient *non-uniform* proof aggregation schemes: aggregating proofs from different circuits into a single proof.

SNARKs [5], short for succinct non-interactive arguments of knowledge, have a long history in cryptography research and industry [15, 2, 31, 13, 22, 32, 3, 28, 34, 33, 10]. At a high level, SNARKs are constructed by compiling an Interactive Oracle Proof (IOP) [4] to a SNARK using a suitable cryptographic commitment scheme. Among all Polynomial-IOP-based SNARKs, the Plonk system [13] is most widely used in industry due to its short proof size and low concrete proof verification cost. In Turbplonk[11], it was extended to support custom gates, and in [24, 12] to support lookup gates (UltraPlonk being one implmentation of the combination of these). The Plonk system has also been used with different cryptographic commitment schemes, with notable examples including KZG [17], Inner Product Argument [7] (implemented in Halo2 [27]), and FRI (implemented in Plonky2 [26]).

1.1 Universal Verifier

$\mathcal{U}\mathsf{ni}\mathcal{P}\mathsf{lonK}$ starts from the simple observation that although two Plonk circuits C_1, C_2 may have unequal sizes $n_1 \neq n_2$, their proofs and verifying keys have the same size. Examining the Plonk verifier protocol, we see that if their public input lengths are equal, the C_1 and C_2 verifiers perform nearly identical work.

This observation has practical significance in the context of proof recursion, where the *Verifier* of C is replaced by a *Verifier Circuit* V_C. Broadly speaking, ZK circuits are "rigid" in the sense that emulating control flow is costly, and a circuit performs the work required for *all execution paths* regardless of its inputs. Therefore, in general, the Plonk verifier circuits V_{C_1} and V_{C_2} will not be equal; they do not perform identical work. $\mathcal{U}\mathsf{ni}\mathcal{P}\mathsf{lonK}$ modifies the $\mathcal{P}\mathsf{lonK}$ protocol in such a way that $V_{C_1} = V_{C_2}$ under weak assumptions on C_1 and C_2. Importantly, no material change is required to the prover algorithm (we require only that constants such as the srs match those used by $\mathcal{U}\mathsf{ni}\mathcal{P}\mathsf{lonK}$, and that some small calculation is performed with cost proportional to the number of public inputs).

Specifically, we demonstrate that the Plonk verifier's work can be made uniform for all circuits smaller than some fixed size using custom gates from a predefined set. Necessarily, when implemented as a circuit, the "Universal Verifier" always performs the worst-case number of operations - those that would be required to verify proofs from a circuit in this family. The challenge is to ensure that any *extra* operations performed by the Universal Verifier do not affect the verification result.

The cost of the Universal Verifier is at least that of the most expensive verifier for any circuit in the supported family. We therefore have a trade-off between the flexibility offered by supporting a large family of circuits and the need to pay the worst-case verifier cost for any circuit in the family. However, as we discuss, this overhead is extremely small in comparison to the underlying circuit-independent verification cost and is justified by its utility in recursion-based proof aggregation.

Due to the rigidity of ZK circuits, practical approaches to proof aggregation assume some form of homogeneity (namely, all proofs are for the same circuit or for one of a predefined set of circuits). A

universal verifier for the class of all circuits up to some size represents a significant improvement in flexibility.

1.2 Using $\mathcal{U}\text{ni}\mathcal{P}\text{lonk}$ in Proof Aggregation

Proof aggregation replaces multiple zero-knowledge proofs $\pi_1, \pi_2, \cdots, \pi_k$ with a single aggregated proof π_{agg} which implies the validity of all statements otherwise demonstrated by the individual proofs π_i. Proof aggregation has already been used in zkEVMs [29, 25] to aggregate multiple EVM execution proofs to obtain a better amortized verification cost.

The basic approach to proof aggregation is recursive proofs. One generates an "outer" proof π_{agg} that attests to successful verification of some batch $\pi_1, \pi_2, \cdots, \pi_k$ of "inner" proofs. This approach works well but has two problems: firstly, the cost of the prover of the outer proof is high due to the amount of non-native arithmetic (in particular the pairing check when the inner proof scheme is Groth16 or Plonk). Secondly, this technique can only aggregate a fixed number of the proofs belonging to the same circuit, which limits its usefulness.

To solve the first problem, recent works such as aPlonk [1], SnarkPack [14], and Halo Infinite [6] partially move some of the expensive non-native arithmetic, e.g. pairing, out of the outer circuit. They replace individual pairing checks for each proof with a single pairing check for a random linear combination of the proofs. The verifier only needs to check the accumulated pairing, which they can do out of circuit. This results in a more efficient aggregator. These are examples of *atomic accumulation*[9]. However, this form of aggregation applies only to instances from the same circuit.

Folding (or NIFS, non-interactive folding scheme) based Incremental Verifiable Computation (IVC) schemes [30] such as Nova [20], SuperNova [18], HyperNova [19], Sangria [23], Protostar [8] further improve the aggregator's efficiency and, in some cases, allow non-uniform aggregation. Folding-based IVC moves even more computation out of the recursive circuit: not only the pairing check, but the witness checking as well. Moreover, IVC schemes allow streaming-based aggregation, incrementally aggregating one proof at a time with no restriction on how many proofs can be aggregated. The Nova, Sangria, and Hypernova schemes are limited to uniform IVC: each incremental step checks a proof of a single fixed circuit. The Supernova and Protostar schemes enable non-uniform IVC (NIVC): each incremental step checks a proof of one circuit from a predefined family $\{C_1, \ldots C_I\}$.

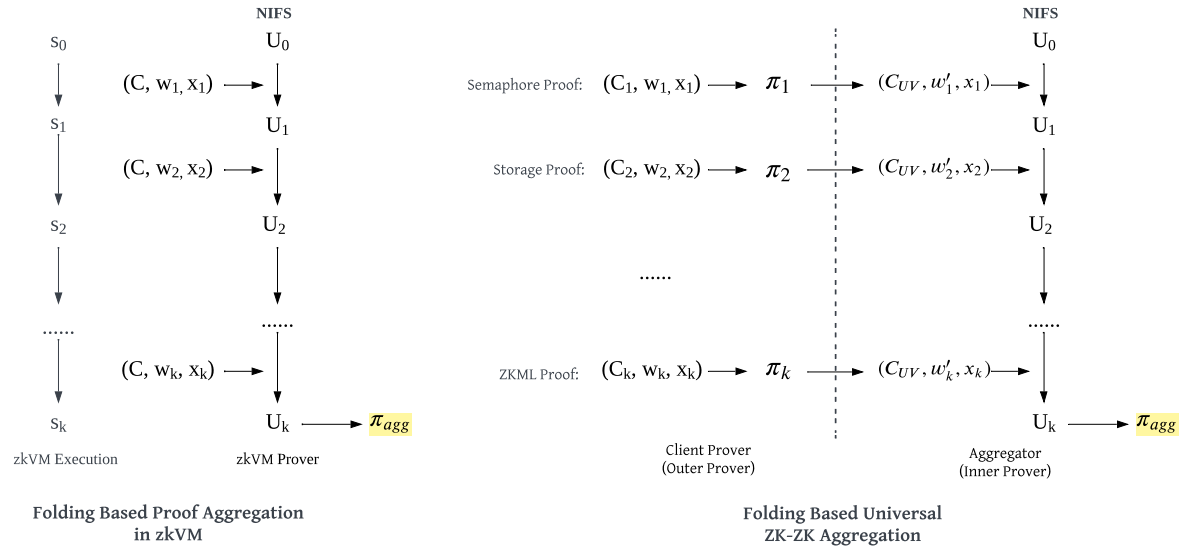

Figure 1: Comparing Universal ZK-ZK Aggreagtion using UniPlonk + NIFS (non-interactive folding scheme) with Proof Aggregation in zkVM using NIFS

UniPlonk could be combined with either atomic accumulation-based aggregation or folding-based aggregation to create *Universal ZK-ZK Aggregation* scheme with:

- *Prover Separation*: The inner prover and the outer prover (aggregator) are not the same party.
- *Witness Privacy*: The outer prover only has access to the inner proofs, not the witnesses. This is very useful in privacy-preserving ZK primitives such as Semaphore.

- *Universality*: The inner prover can support *all* circuits up to a certain size.

Combining UniPlonk with NIFS can improve the efficiency of the outer prover (aggregator) significantly. Fig. 1 demonstrates how a Universal ZK-ZK Aggregation scheme can be constructed using UniPlonk and NIFS. The left-hand side shows a typical NIFS application scenario, generating an aggregated proof for multiple steps of zkVM execution. The right-hand side shows the Universal ZK-ZK Aggregation constructed from UniPlonk and NIFS. In this case, we do not have the luxury of directly folding inner proof witnesses, due to the inner/outer prover separation. However, the outer prover uses the universal verifier circuit (C_{UV} in the figure) to generate a witness for the verification of each inner proof, which can then be folded. Thereby this achieves both universality and efficiency for the ZK-ZK Aggregation scheme. Compared with ZK-ZK aggregation based on atomic accumulation, using NIFS to directly fold the C_{UV} witness significantly reduces the outer prover workload. The universality of UniPlonk's verifier allows this ZK-ZK aggregation scheme to support a wide range of circuits, limited only by circuit size.

2 Preliminaries

2.1 Notation

We use the following conventions (terms such as "configuration" and "global gate set" are defined later):

- n denotes the number of "rows" of a circuit, *i.e.* the length of the witness vector, padded up to a next power of 2.
- $k = \log_2(n)$
- w_i denotes a witness variable for some gate polynomial. When referring to the vanilla Plonk gate we may use the notation a, b, c as in [13].
- m denotes the total number of witness variables used by some circuits.
- q_i denotes a constant variable for some gate polynomial. When referring to the vanilla Plonk gate we may use the notation q_M, q_L, q_R, q_O, q_C as in [13].
- r denotes the total number of constant variables used by some circuits.
- $\mathscr{G}_C$ denotes the set of custom gates used by a circuit (including the vanilla Plonk gate). $\mathscr{G}$ denotes a "global gate set" of all gates used in a particular UniPlonk configuration.
- l denotes the length of a gate set $\mathscr{G}_C$. (The number of public inputs to a circuit shall be denoted with the script ℓ.)
- $b^{(g)}$, $b^{(w)}$, and $b^{(q)}$ are bit vectors referring to the gates, witness variables, and constant variables.

For symbols that refer to integer values (n, k, m, r, l) we use uppercase letters to denote an upper bound on the corresponding quantity:

- N denotes the maximum size of a circuit in a given UniPlonk configuration, K its logarithm.
- M denotes the maximum number of witness variables used by a circuit in a given UniPlonk configuration.
- R denotes the maximum number of constant variables used by a circuit in a given UniPlonk configuration.
- L denotes the length of the global gate set $\mathscr{G}$.

We make use of a function $\mathsf{select} : \{0,1\} \times \mathsf{T} \times \mathsf{T} \to \mathsf{T}$ which can dynamically select between alternative instances of a type T based on the value of a bit. In all cases considered in this work, T is a field or group element (namely a member of $\mathbb{F}$, $\mathbb{G}_1$, $\mathbb{G}_2$) whereby select can be implemented as $\mathsf{select}(b, A, B) = b * A + (1 - b) * B$. We also make make use of *multiplexers*, which allow one of N (where N may be greater than 2) possible values to be selected based on an input x, where x may take values $0, 1, \ldots, N-1$. Multiplexers may be implemented algebraically, similarly to select, or using lookup tables in the case that the values to be selected from are known when the circuit is defined.

2.2 $\mathcal{P}\mathfrak{lonK}$ Recap

For "vanilla" Plonk as described in [13], we adopt the original notation and conventions as used by the authors. We recall some definitions here.

Fix an elliptic curve pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and let $\mathbb{F}$ denote the scalar field of $\mathbb{G}_1$. Let $C = (n, \ell, (q_{L_i}, q_{R_i}, q_{M_i}, q_{O_i}, q_{C_i})_{i=1}^n, \sigma)$ denote a Plonk circuit with $n = 2^k$ rows, ℓ public input values, constant values $q_{L_i}, q_{R_i}, \ldots \in \mathbb{F}$, and copy constraints encoded by a permutation $\sigma : [3n] \to [3n]$. An instance of C is defined by these data and a particular set of public input values $\mathsf{PI} = (\mathsf{PI}_i)_{i=1}^\ell \in \mathbb{F}^\ell$. A witness to an instance of C will be denoted $w = (\{a_i\}_{i=1}^n, \{b_i\}_{i=1}^n, \{c_i\}_{i=1}^n) \in \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n$.

The rows of C are indexed by a multiplicative subgroup H generated by a primitive n^{th} root of unity $\omega \in \mathbb{F}$. Constants $k_1, k_2 \in \mathbb{F}$ are fixed such that the cosets H, k_1H, k_2H are disjoint. The union $H \cup k_1H \cup k_2H$ is used to index the witness values: a_i is indexed by ω^i, b_i is indexed by $k_1\omega^i$, c_i is indexed by $k_2\omega^i$. The polynomial $Z_H(X) = X^n - 1$ vanishes at all points of H.

Vectors $(f_i)_{i=1}^n$ of length n are mapped to polynomials of degree $n-1$ as follows: let $L_i(X)$ denote the degree $n-1$ Lagrange interpolation polynomial which satisfies

$$L_i(\omega^j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

Then the vector $(f_i)_{i=1}^n$ corresponds to the interpolation polynomial

$$f(X) := \sum_{i=1}^n f_i L_i(X)$$

KZG polynomial commitments are defined relative to a fixed structured reference string

$$\mathsf{srs} = \left([1]_1, [x]_1, [x^2]_1, \ldots, [x^{n+5}]_1, [1]_2, [x]_2\right)$$

where $x \in \mathbb{F}$. This "trapdoor" element x is assumed to be fixed but unknown; it is also frequently denoted by τ elsewhere in the literature. Here $[1]_1, [1]_2$ denote fixed generators of the groups $\mathbb{G}_1, \mathbb{G}_2$, respectively. The commitment to a polynomial $f(X)$ is denoted $[f]_1$.

2.2.1 $\mathcal{P}\mathfrak{lonK}$ Setup

The Plonk Setup function returns a set of proving and verifying keys for a circuit C, given a fixed KZG structured reference string:

$$(pk^{\mathcal{P}}, vk^{\mathcal{P}}) \leftarrow \mathcal{P}\mathfrak{lonK}.\mathsf{Setup}(C, \mathsf{srs})$$

A proving key consists of the data

$$\begin{aligned} pk^{\mathcal{P}} = (&q_M(X), q_L(X), q_R(X), q_O(X), q_C(X), \\ &S_{\sigma 1}(X), S_{\sigma 2}(X), S_{\sigma 3}(X)) \end{aligned}$$

where $S_{\sigma 1}, S_{\sigma 2}, S_{\sigma 3}$ are as defined in [13]. A verifying key consists of the data

$$vk^{\mathcal{P}} = ([q_M]_1, [q_L]_1, [q_R]_1, [q_O]_1, [q_C]_1, [S_{\sigma 1}]_1, [S_{\sigma 2}]_1, [S_{\sigma 3}]_1, n)$$

2.2.2 $\mathcal{P}\mathfrak{lonK}$ Prove

We denote a Plonk proof by $\pi^{\mathcal{P}}$. It consists of the following elements:

$$\pi^{\mathcal{P}} = \begin{pmatrix} [a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mid}]_1, [t_{hi}]_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \\ \bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_\omega \end{pmatrix} \tag{1}$$

where the first line consists of $\mathbb{G}_1$ elements and the second line of $\mathbb{F}$ elements.
Plonk proofs are produced by the Prove function, which takes as inputs a proving key $pk^{\mathcal{P}}$ generated by Setup and a witness w:

$$\pi^{\mathcal{P}} \leftarrow \mathcal{P}\mathfrak{lonK}.\mathsf{Prove}(pk^{\mathcal{P}}, w)$$

This function is defined in [13]. We recall some relevant quantities computed by the Plonk Prover:

Blinded Witness Polynomials

The Prover blinds the witness polynomials by randomly sampling $b_1, \dots b_6 \in \mathbb{F}$ and defining

$$\begin{aligned} a(X) &= (b_1 X + b_2) Z_H(X) + \sum_{i=1}^{n} a_i L_i(X) \\ b(X) &= (b_3 X + b_4) Z_H(X) + \sum_{i=1}^{n} b_i L_i(X) \\ c(X) &= (b_5 X + b_6) Z_H(X) + \sum_{i=1}^{n} c_i L_i(X) \end{aligned} \tag{2}$$

Note that these have degree $n + 1$.

Permutation Polynomial

The copy constraints are enforced by proving the properties of the polynomial

$$\begin{aligned} z(X) =& (b_7 X^2 + b_8 X + b_9) Z_H(X) + L_1(X) \\ &+ \sum_{i=1}^{n-1} \left(L_{i+1}(X) \prod_{j=1}^{i} \frac{(a_j + \beta\omega^j + \gamma)(b_j + \beta k_1 \omega^j + \gamma)(c_j + \beta k_2 \omega^j + \gamma)}{(a_j + \beta S_{\sigma 1}(\omega^j) + \gamma)(b_j + \beta S_{\sigma 2}(\omega^j) + \gamma)(c_j + \beta S_{\sigma 3}(\omega^j) + \gamma)} \right) \end{aligned} \tag{3}$$

where b_7, b_8, b_9 are blinding factors sampled by the prover. Note that $\deg z = n + 2$.

Quotient Polynomial

The quotient polynomial $t(X)$ is defined as the polynomial which satisfies

$$\begin{aligned} t(X) Z_H(X) =& a(X) b(X) q_M(X) + a(X) q_L(X) + b(X) q_R(X) + c(X) q_O(X) + \mathsf{PI}(X) + q_C(X) \\ &+ \alpha \left[(a(X) + \beta X + \gamma)(b(X) + \beta k_1 X + \gamma)(c(X) + \beta k_2 X + \gamma) \right] z(X) \\ &- \alpha \left[(a(X) + \beta S_{\sigma 1}(X) + \gamma)(b(X) + \beta S_{\sigma 2}(X) + \gamma)(c(X) + \beta S_{\sigma 3}(X) + \gamma) \right] z(X\omega) \\ &+ \alpha^2 \left(z(X) - 1 \right) L_1(X) \end{aligned} \tag{4}$$

where α is a challenge scalar.
The degree of $t(X)$ is $3n + 5$ and it is represented in degree $n - 1$ chunks as

$$t(X) = t_{lo}(X) + X^n t_{mid}(X) + X^{2n} t_{hi}(X)$$

where $\deg t_{lo} = \deg t_{mid} = n$, $\deg t_{hi} = n + 5$.

Linearization Polynomial

Given a challenge point $\mathfrak{z}$ and evaluations $\bar{a} = a(\mathfrak{z}), \bar{b} = b(\mathfrak{z}), \bar{c} = c(\mathfrak{z}), \bar{s}_{\sigma 1} = S_{\sigma 1}(\mathfrak{z}), \bar{s}_{\sigma 2} = S_{\sigma 2}(\mathfrak{z}), \bar{z}_\omega = z(\mathfrak{z}\omega)$, the linearization polynomial is defined as

$$\begin{aligned} r(X) =& \bar{a}\bar{b} \cdot q_M(X) + \bar{a} \cdot q_L(X) + \bar{b} \cdot q_R(X) + \bar{c} \cdot q_O(X) + \mathsf{PI}(\mathfrak{z}) + q_C(X) \\ &+ \alpha [(\bar{a} + \beta\mathfrak{z} + \gamma)(\bar{b} + \beta k_1 \mathfrak{z} + \gamma)(\bar{c} + \beta k_2 \mathfrak{z} + \gamma) \cdot z(X) \\ &- (\bar{a} + \beta \bar{s}_{\sigma 1} + \gamma)(\bar{b} + \beta \bar{s}_{\sigma 2} + \gamma)(\bar{c} + \beta S_{\sigma 3}(X) + \gamma) \bar{z}_\omega] \\ &+ \alpha^2 \left[(z(X) - 1) L_1(\mathfrak{z}) \right] \\ &- Z_H(\mathfrak{z}) \left(t_{lo}(X) + \mathfrak{z}^n t_{mid}(X) + \mathfrak{z}^{2n} t_{hi}(X) \right) \end{aligned} \tag{5}$$

Showing $r(\mathfrak{z}) = 0$ implies that $t(X)$ satisfies Eq. (4) (except with negligible probability).

Opening Proof Polynomials

The claims $r(\mathfrak{z}) = 0, \bar{a} = a(\mathfrak{z}), \bar{b} = b(\mathfrak{z}), \bar{c} = c(\mathfrak{z}), \bar{s}_{\sigma 1} = S_{\sigma 1}(\mathfrak{z}), \bar{s}_{\sigma 2} = S_{\sigma 2}(\mathfrak{z})$ are batched into a single KZG proof whose opening proof polynomial is (given challenge scalar v)

$$W_{\mathfrak{z}}(X) = \frac{1}{X - \mathfrak{z}} \begin{pmatrix} r(X) \\ + v(a(X) - \bar{a}) \\ + v^2(b(X) - \bar{b}) \\ + v^3(c(X) - \bar{c}) \\ + v^4(S_{\sigma 1}(X) - \bar{s}_{\sigma 1}) \\ + v^5(S_{\sigma 2}(X) - \bar{s}_{\sigma 2}) \end{pmatrix} \tag{6}$$

The claim $\bar{z}_\omega = z(\mathfrak{z}\omega)$ occurs at a different point and so requires a separate opening proof polynomial

$$W_{\mathfrak{z}\omega}(X) = \frac{z(X) - \bar{z}_\omega}{X - \mathfrak{z}} \tag{7}$$

2.2.3 $\mathcal{P}\mathfrak{lon}\mathcal{K}$ Verify

Given a proof $\pi^{\mathcal{P}}$ for a circuit C with public inputs PI, the Plonk Verifier decides whether the proof is valid by evaluating a function $\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(\mathsf{PI}, \pi^{\mathcal{P}})$ defined in [13]. At a high-level, the verifier computes challenge points (Section 2.4), computes a commitment $[r]_1$ to the linearization polynomial $r(X)$, and uses $[W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1$ to verify that $r(\mathfrak{z}) = 0$ and all evaluations supplied in the proof are correct. We recall here some important quantities computed by the verifier:

Linearization Polynomial Constant Term

The verifier computes

$$r_0 = \mathsf{PI}(\mathfrak{z}) - \alpha^2 L_1(\mathfrak{z}) - \alpha(\bar{a} + \beta\bar{s}_{\sigma 1} + \gamma)(\bar{b} + \beta\bar{s}_{\sigma 2} + \gamma)(\bar{c} + \gamma)\bar{z}_\omega \tag{8}$$

from the public input values and the evaluations supplied in $\pi^{\mathcal{P}}$.

Group-encoded Batch Evaluation

The previous term is grouped together with the other claimed evaluations in

$$[E]_1 = \begin{pmatrix} -r_0 + v\bar{a} + v^2\bar{b} + v^3\bar{c} \\ + v^4\bar{s}_{\sigma 1} + v^5\bar{s}_{\sigma 2} + u\bar{z}_\omega \end{pmatrix} [1]_1 \tag{9}$$

Batched Polynomial Commitment

The non-constant part of $r(X)$ and other commitments supplied in $\pi^{\mathcal{P}}$ are grouped together in

$$[F]_1 = [D]_1 + v \cdot [a]_1 + v^2 \cdot [b]_1 + v^3 \cdot [c]_1 + v^4 \cdot [s_{\sigma 1}]_1 + v^5 \cdot [s_{\sigma 2}]_1 \tag{10}$$

where

$$\begin{aligned} [D]_1 = {} & \bar{a}\bar{b} \cdot [q_M]_1 + \bar{a} \cdot [q_L]_1 + \bar{b} \cdot [q_R]_1 + \bar{c} \cdot [q_O]_1 + [q_C]_1 \\ & + \left(\alpha(\bar{a} + \beta\mathfrak{z} + \gamma)(\bar{b} + \beta k_1\mathfrak{z} + \gamma)(\bar{c} + \beta k_2\mathfrak{z} + \gamma) + \alpha^2 L_1(\mathfrak{z}) + u\right)[z]_1 \\ & - \alpha(\bar{a} + \beta\bar{s}_{\sigma 1} + \gamma)(\bar{b} + \beta\bar{s}_{\sigma 2} + \gamma)\beta\bar{z}_\omega [s_{\sigma 3}]_1 \\ & - Z_H(\zeta)\left([t_{lo}]_1 + \mathfrak{z}^n[t_{mid}]_1 + \mathfrak{z}^{2n}[t_{hi}]_1\right) \end{aligned} \tag{11}$$

Final Pairing Check

$\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}$ returns the boolean value

$$e([W_{\mathfrak{z}}]_1 + u \cdot [W_{\mathfrak{z}\omega}]_1, [x]_2) \stackrel{?}{=} e(\mathfrak{z} \cdot [W_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega \cdot [W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2) \tag{12}$$

2.3 $\mathcal{P}\mathfrak{lon}\mathcal{K}$ with Custom Gates

Custom gates were introduced in [11] to allow the proving system to handle arbitrary polynomial constraints on witness values. In this section, we recall the basic ideas and define some notation for custom gates.

2.3.1 Constraint Equations

A custom gate is specified by a polynomial $G(w_1, \ldots w_m, q_1, \ldots, q_r)$, defined on up to m witness variables, and up to r constant variables. The Plonk protocol enforces the constraint

$$G(w_1, \ldots w_m, q_1, \ldots, q_r) = 0$$

For multiple gates, $\{G_i\}_{i=1}^{l}$, each G_i has an associated *selector polynomial* $S_i(X)$ which takes the value 1 at ω^j for each row j in which the gate is enabled, and zero otherwise. Gates are considered to

share the same set $w_1, \dots w_m$ of witness columns and $q_1, \dots q_r$ of constant columns. A random challenge α is used to ensure that when multiple gates are activated on the same row, their constraint polynomials are individually satisfied. The full arithmetic constraint polynomial is given by:

$$\sum_{i=1}^{l} \alpha^{i-1} S_i(X) G_i(w_1(X), \dots, w_m(X), q_1(X), \dots, q_r(X)) \tag{13}$$

where $w_i(X)$ are blinded witness polynomials, as in Eq. (2). The custom gate constraints are satisfied by the witness in each row if and only if this polynomial is divisible by $Z_H(X)$.

The degree of the polynomial in Eq. (13) can be computed from $\deg S_i = n - 1$, $\deg w_i = n + 1$, $\deg q_i = n - 1$ and the degrees of the polynomials G_i. An exact expression requires cumbersome notation, but if we approximate the degrees of S_i, w_i, q_i by n then the degree of Eq. (13) is approximately

$$\max_i(\deg G_i + 1) \cdot n$$

where $\deg G_i$ denotes the degree of G_i as a multivariate polynomial. This contributes a term of degree approximately $\max_i(\deg G_i) \cdot n$ to the quotient polynomial, Eq. (17). Unless some G_i has high degree, the copy constraints usually contribute a higher degree term to the quotient polynomial.

2.3.2 Example

The vanilla Plonk constraint corresponds to a gate with the polynomial

$$G_{\text{vanilla}}(w_1, w_2, w_3, q_M, q_L, q_R, q_O, q_C) = q_M w_1 w_2 + q_L w_1 + q_R w_2 + q_O w_3 + q_C$$

A custom gate to enforce that w_1 and w_2 are the affine coordinates of an elliptic curve point is

$$G_{\text{curve}}(w_1, w_2, q_a, q_b) = w_2^2 - w_1^3 - q_a w_1 - q_b$$

The Plonk proving system with these two gates G_{vanilla} and G_{curve} requires both polynomials to share a common set of variables, so we would redefine G_{curve} as

$$G_{\text{curve}}(w_1, w_2, w_3, q_M, q_L, q_R, q_O, q_C) = w_2^2 - w_1^3 - q_L w_1 - q_R$$

The choice to use q_L, q_R in place of q_a, q_b was arbitrary; any two constant variables could be used to specify the constants of the elliptic curve equation.

Public Input Gate A useful convention for incorporating public inputs into circuits with custom gates is to take G_1 to be a "public input gate" with the constraint polynomial $G_1(w_1, \dots w_m, q_1, \dots q_r) = -w_1$. We use this gate to incorporate public inputs as follows: the definition of the quotient polynomial Eq. (17) includes a term $\alpha^2 \mathsf{PI}(X)$, which is offset by $\alpha^2 S_1(X) G_1(w_1(X), \dots)$ if and only if $w_1(\omega^i) = \mathsf{PI}_i$ and $S_1(\omega^i) = 1$ for $i = 1, \dots \ell$. In practice, G_1 may not need to be a separate gate, as it can be obtained from the vanilla Plonk gate by setting $q_L = -1$ and $q_R = q_M = q_O = q_C = 0$.

2.3.3 Copy Constraints

Witness variables are subject to copy constraints. The vanilla Plonk argument is modified to apply to m witness variables as follows. We fix $k_1, \dots k_m \in \mathbb{F}$ such that the cosets $k_i H$ are mutually disjoint; we take $k_1 = 1$. The witness variable w_i has its values indexed by the points of the coset $k_i H$. The copy constraints are encoded by a permutation σ of the set $\cup_{i=1}^m k_i H$. The generalization of the permutation polynomial in Eq. (3) to the case of m witness columns is

$$\begin{aligned} z(X) =& (b_1 + b_2 X + b_3 X^2) Z_H(X) + L_1(X) \\ &+ \sum_{i=1}^{n-1} \left(L_{i+1}(X) \prod_{j=1}^{i} \frac{\prod_{p=1}^{m} (w_{p,j} + \beta k_p \omega^j + \gamma)}{\prod_{p=1}^{m} (w_{p,j} + \beta S_{\sigma p}(\omega^j) + \gamma)} \right) \end{aligned} \tag{14}$$

Despite its apparent complexity, this is merely Eq. (3) with m witness columns instead of 3. We assume for simplicity that all m witness variables are subject to copy constraints, though this assumption may be relaxed to decrease the number of permutation polynomials, and the complexity of Z_H.

The copy constraints are satisfied if the following polynomials are divisible by $Z_H(X)$:

$$\left(\prod_{i=1}^{m}(w_i(X)+\beta k_i X+\gamma)\right)z(X)-\left(\prod_{i=1}^{m}(w_i(X)+\beta \mathsf{s}_{\sigma i}(X)+\gamma)\right)z(\omega X) \tag{15}$$

and

$$L_1(X)(z(X)-1) \tag{16}$$

The degree of Eq. (15) in X is $m\cdot(n+1)+(n+2)$. Thus the quotient polynomial (Eq. (17)) will have degree at least $m\cdot(n+1)+2$.

Quotient Polynomial

Divisibility of the polynomials Eqs. (13), (15) and (16) by Z_H is proved by forming a random linear combination of them and computing its quotient. For a given challenge scalar $\alpha\in\mathbb{F}$ we define the quotient polynomial $t(X)$ to satisfy

$$\begin{aligned} t(X)Z_H(X) = &\left(\prod_{i=1}^{m}(w_i(X)+\beta k_i X+\gamma)\right)z(X)\\ &-\left(\prod_{i=1}^{m}(w_i(X)+\beta \mathsf{s}_{\sigma i}(X)+\gamma)\right)z(\omega X)\\ &+\alpha L_1(X)\left(z(x)-1\right)\\ &+\alpha^2\mathsf{PI}(X)\\ &+\sum_{i=1}^{l}\alpha^{i+1}S_i(X)G_i(w_1(X),\ldots,w_m(X),q_1(X),\ldots,q_r(X)) \end{aligned} \tag{17}$$

The degree of $t(X)$ is determined by the term coming from Eq. (15) unless some custom gate polynomial G_i has high degree (custom gates used "in the wild" tend to have low degree constraint polynomials). Under this assumption, the degree of $t(X)$ is $m\cdot(n+1)+2$.

This degree is generally too large to commit to $t(X)$ as a single polynomial, so one decomposes t as

$$t(X)=t_1(X)+X^n t_2(X)+\ldots+X^{n(d-1)}t_d(X)$$

with $\deg t_i = n-1$ for $i<d$ and $\deg t_d = n+m+2$. We call d the *degree factor* of the circuit C. In order to compute $[t_d]_1$ the srs must have length at least $n+m+2$.

Linearization Polynomial

Given a challenge evaluation point $\mathfrak{z}$, the linearization polynomial for C with custom gates $G_1,\ldots G_l$ using witness variables $w_1,\ldots w_m$ and constant variables $q_1,\ldots q_r$ is

$$\begin{aligned} r(X) = &\left(\prod_{i=1}^{m}(\bar{w}_i+\beta k_i\mathfrak{z}+\gamma)\right)z(X)\\ &-\left(\prod_{i=2}^{m}(\bar{w}_i+\beta\bar{\mathsf{s}}_{\sigma i}+\gamma)\right)\bar{z}_\omega\cdot((\bar{w}_1+\gamma)+\beta\mathsf{s}_{\sigma 1}(X))\\ &+\alpha L_1(\mathfrak{z})\cdot(z(X)-1)\\ &+\alpha^2\mathsf{PI}(\mathfrak{z})\\ &+\sum_{i=1}^{l}\alpha^{i+1}S_i(X)G_i(\bar{w}_1,\ldots,\bar{w}_m,\bar{q}_1,\ldots,\bar{q}_r)\\ &-Z_H(\mathfrak{z})(t_1(X)+\mathfrak{z}^n t_2(X)+\ldots+\mathfrak{z}^{n(d-1)}t_d(X)) \end{aligned} \tag{18}$$

The verifier computes the commitment to $r(X)$ from a proof and verification key, as

$$\begin{aligned}
[r]_1 = &\left(\prod_{i=1}^{m}(\bar{w}_i + \beta k_i \mathfrak{z} + \gamma)\right)[z]_1 \\
&- \left(\prod_{i=2}^{m}(\bar{w}_i + \beta \bar{\mathsf{s}}_{\sigma i} + \gamma)\right)\bar{z}_\omega \cdot ((\bar{w}_1 + \gamma)\cdot[1]_1 + \beta \cdot [\mathsf{s}_{\sigma 1}]_1) \\
&+ \alpha L_1(\mathfrak{z}) \cdot ([z]_1 - [1]_1) \\
&+ \alpha^2 \mathsf{PI}(\mathfrak{z})[1]_1 \\
&+ \sum_{i=1}^{l} \alpha^{i+1} G_i(\bar{w}_1, \ldots, \bar{w}_m, \bar{q}_1, \ldots, \bar{q}_r)\cdot[S_i] \\
&- Z_H(\mathfrak{z})([t_1]_1 + \mathfrak{z}^n[t_2]_1 + \cdots + \mathfrak{z}^{(d-1)n}[t_d]_1)
\end{aligned} \tag{19}$$

The linearization polynomial could be defined in several ways. For example, the commitment to $\mathsf{s}_{\sigma m}(X)$ could be used in place of $[\mathsf{s}_{\sigma 1}]_1$. In this work, we use the expression above for convenience in the universal verifier (since m is a circuit-dependent quantity and $\mathsf{s}_{\sigma 1}(X)$ is guaranteed to exist for all circuits), however, we note that alternative forms are available, and indeed more optimal forms are likely to exist for any given circuit.

Opening Proof Polynomial

The opening proof polynomial $W_{\mathfrak{z}}(X)$ is modified to

$$W_{\mathfrak{z}}(X) = \frac{1}{X - \mathfrak{z}}\begin{pmatrix} r(X) \\ +\sum_{i=1}^{m} v^i(w_i(X) - \bar{w}_i) \\ +v^m \sum_{i=1}^{r} v^i(q_i(X) - \bar{q}_i) \\ +v^{m+r-1}\sum_{i=2}^{m} v^i(\mathsf{s}_{\sigma i}(X) - \bar{\mathsf{s}}_{\sigma i}) \end{pmatrix} \tag{20}$$

The expression in Eq. (7) for $W_{\mathfrak{z}\omega}(X)$ is unchanged.

2.3.4 Setup for Custom Gates

For a circuit C with custom gates $G_1, \ldots G_l$ using witness variables $w_1, \ldots w_m$ and constant variables $q_1, \ldots q_r$ which computes the prover and verifier keys $pk^{\mathcal{P}}, vk^{\mathcal{P}}$ from the circuit description and srs. These are

$$pk^{\mathcal{P}} = \Big(S_1(X), \ldots S_l(X), q_1(X), \ldots q_r(X), \mathsf{s}_{\sigma 1}(X), \ldots \mathsf{s}_{\sigma m}(X)\Big)$$

$$vk^{\mathcal{P}} = \Big([S_1]_1, \ldots, [S_l]_1, [q_1]_1, \ldots, [q_r]_1, [\mathsf{s}_{\sigma 1}]_1, \ldots, [\mathsf{s}_{\sigma m}]_1\Big)$$

2.3.5 Prove for Custom Gates

For a circuit C with custom gates $G_1, \ldots G_l$ using witness variables $w_1, \ldots w_m$ and constant variables $q_1, \ldots q_r$ a proof consists of

$$\begin{aligned}
\pi^{\mathcal{P}} = \Big([w_1]_1, \ldots, [w_m]_1, [z]_1, [t_1]_1, \ldots, [t_d]_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \\
\bar{w}_1, \ldots, \bar{w}_m, \bar{q}_1, \ldots, \bar{q}_r, \bar{\mathsf{s}}_{\sigma 2}, \ldots \bar{\mathsf{s}}_{\sigma m}, \bar{z}_\omega\Big)
\end{aligned} \tag{21}$$

We note that, unlike vanilla Plonk proofs, Plonk proofs for circuits with custom gates must include evaluations $\bar{q}_i$ of the constant polynomials at the challenge point. This is because, in general, a gate constraint may be non-linear in its constant variables. The verifier will need the evaluations $\bar{q}_i$ to compute the commitment $[r]_1$, Eq. (19). The Prove algorithm follows similar steps to vanilla Plonk:

1. Compute blinded witness polynomials $w_i(X)$ and their commitments $[w_i]_1$
2. Compute permutation challenges β, γ
3. Compute permutation polynomial $z(X)$ and its commitment $[z]_1$
4. Compute quotient challenge α
5. Compute quotient polynomial $t(X)$ and commitments $[t_1]_1, \ldots [t_d]_1$
6. Compute evaluation challenge $\mathfrak{z}$
7. Compute opening evaluations $\bar{w}_1, \ldots \bar{w}_m, \bar{q}_1, \ldots \bar{q}_r, \bar{\mathsf{s}}_{\sigma 2}, \ldots \bar{\mathsf{s}}_{\sigma m}, \bar{z}_\omega$
8. Compute opening challenge v
9. Compute linearization polynomial $r(X)$
10. Compute opening proof polynomials $W_{\mathfrak{z}}(X), W_{\mathfrak{z}\omega}(X)$ and their commitments $[W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1$
11. Return $\pi^{\mathcal{P}}$

2.3.6 Verify for Custom Gates

It is useful to break the verifier's computation of $[r]_1$ into a few intermediate quantities:

$$[r]_1 = r_0 \cdot [1]_1 + r_1[z]_1 + r_2[\mathsf{s}_{\sigma 1}]_1 + [r_3]_1 - [r_4]_1$$

where

$$\begin{aligned}
a &:= \bar{z}_\omega \prod_{i=2}^{m} (\bar{w}_i + \beta \bar{\mathsf{s}}_{\sigma i} + \gamma) \\
r_0 &= -a(\bar{w}_1 + \gamma) - \alpha L_1(\mathfrak{z}) + \alpha^2 \mathsf{PI}(\mathfrak{z}) \\
r_1 &= \prod_{i=1}^{m} (\bar{w}_i + \beta k_i \mathfrak{z} + \gamma) + \alpha L_1(\mathfrak{z}) \\
r_2 &= a\beta \\
[r_3]_1 &= \sum_{i=1}^{l} \alpha^{i+1} G_i(\bar{w}_1, \ldots \bar{w}_m, \bar{q}_1, \ldots \bar{q}_r) \cdot [S_i]_1 \\
[r_4]_1 &= Z_H(\mathfrak{z}) \left([t_1]_1 + \mathfrak{z}^n [t_2]_1 + \ldots + \mathfrak{z}^{n(d-1)} [t_d]_1 \right)
\end{aligned}$$

In terms of these intermediate quantities, the custom gates verifier's group-encoded batch evaluation is

$$[E]_1 := \begin{pmatrix} -r_0 + u\bar{z}_\omega + \\ v\bar{w}_1 + \ldots + v^m \bar{w}_m + \\ v^{m+1}\bar{q}_1 + \ldots + v^{m+r}\bar{q}_r + \\ v^{m+r+1}\bar{\mathsf{s}}_{\sigma 2} + \ldots + v^{2m+r-1}\bar{\mathsf{s}}_{\sigma m} \end{pmatrix} \cdot [1]_1 \tag{22}$$

and the batched polynomial commitment is

$$\begin{aligned}
[F]_1 :=& (r_1 + u) \cdot [z]_1 + r_2[\mathsf{s}_{\sigma 1}]_1 + [r_3]_1 - [r_4]_1 \\
& v[w_1]_1 + \ldots + v^m [w_m]_1 + \\
& v^{m+1}[q_1] + \ldots + v^{m+r}[q_r]_1 + \\
& v^{m+r+1}[\mathsf{s}_{\sigma 2}]_1 + \ldots + v^{2m+r-1}[\mathsf{s}_{\sigma m}]_1
\end{aligned} \tag{23}$$

In terms of these redefined $[E]_1, [F]_1$, the verifier's final pairing check is identical to Eq. (12).

2.4 Transcript and Challenge Points

As part of the Fiat-Shamir transform used to make Plonk a non-interactive protocol, the Plonk Prove function must compute challenge points, namely $\beta, \gamma, \alpha, \mathfrak{z}, v, u$ by applying a hash function to the transcript at several points. Here we specify the details of how transcript is processed and used.

Our model for the transcript is a permutation sponge $\mathscr{S}$ with state $\mathcal{S}$. The sponge $\mathscr{S}$ has an Absorb operation, which accepts (encodings of) $\mathbb{G}$ and $\mathbb{F}$ elements and returns the new sponge state $\mathcal{S}$, and a Squeeze operation which returns an $\mathbb{F}$ element and the new sponge state $\mathcal{S}$. We fix an initial sponge state $\mathcal{S}_0$.

The prover and verifier use the transcript to generate their challenges by absorbing messages sent by the prover and subsequently squeezing challenge scalars. Because in UniPlonk we only provide the verifier with a *commitment* $[\mathsf{PI}]_1$ to public inputs, we also seed the transcript with the commitment $[\mathsf{PI}]_1$ as opposed to the original public input vector PI.

The prover and verifier then generate challenges as follows:

1. Initiate transcript in default state $\mathcal{S}_0$.
2. Absorb all elements of circuit's Plonk verifying key $vk^{\mathcal{P}}$
3. Absorb public input commitment $[\mathsf{PI}]_1$
4. Absorb witness polynomial commitments
   - For vanilla Plonk: $[a]_1, [b]_1, [c]_1$
   - For Plonk with custom gates: $[w_1]_1, \ldots [w_m]_1$
5. Squeeze β, γ challenges
6. Absorb permutation polynomial commitment $[z]_1$
7. Squeeze α challenge
8. Absorb quotient polynomial commitments
   - For vanilla Plonk: $[t_{lo}]_1, [t_{mid}]_1, [t_{hi}]_1$
   - For Plonk with custom gates: $[t_1]_1, \ldots [t_d]_1$
9. Squeeze $\mathfrak{z}$ evaluation challenge
10. Absorb evaluation claims
    - For vanilla Plonk: $\bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_{\omega}$
    - For Plonk with custom gates: $\bar{w}_1, \ldots, \bar{w}_m, \bar{q}_1, \ldots, \bar{q}_r, \bar{\mathsf{s}}_{\sigma 2}, \ldots \bar{\mathsf{s}}_{\sigma m}, \bar{z}_{\omega}$
11. Squeeze v challenge
12. Absorb opening proof polynomial commitments $[W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1$
13. Squeeze u challenge

3 Technical Overview

The main innovation in UniPlonk is the Universal Verifier (UV), which verifies proofs for a class $\mathcal{C}$ of circuits using a fixed set of operations. The UV relies on a small transformation being applied to Plonk proofs but can verify these transformed proofs in constant-time with minimal overhead compared to the original Plonk verifier. By virtue of this, the UV can itself be implemented as a circuit, whereby recursion and proof aggregation techniques can be leveraged to achieve non-homogeneous proof aggregation.

For a given circuit C, the UniPlonk Setup process produces a verification key vk which is augmented with metadata about C that facilitates constant-time verification. The proving key for UniPlonk is the unchanged proving key $pk^{\mathcal{P}}$ from Plonk and can be used with a standard Plonk prover (where the prover follows the conventions we give here for seeding the transcript, and shares the srs and other constants with the UV). Given a Plonk proof $\pi^{\mathcal{P}}$ for a circuit C, with public inputs PI, a *uniformization* step accepts $\pi^{\mathcal{P}}$, PI, and augmented vk and outputs a proof π of fixed size, compatible with the universal

verifier. The UV then takes as inputs the augmented verification key vk, the transformed proof π, and a commitment $[\mathsf{PI}]_1$ to the public inputs.

A primary design objective is for the UniPlonk prover to match the Plonk prover [13] as far as possible, so that UniPlonk can be easily used as a backend for existing circuit programming DSLs such as Circom, PIL, and Noir. We also strive to keep the UV cost as close as possible to the original Plonk verifier to avoid increasing proof recursion overhead.

3.1 Vanilla $\mathcal{U}\mathrm{ni}\mathcal{P}\mathrm{lon}\mathcal{K}$

We let $\mathcal{C}$ be the family of Plonk circuits defined over a common field $\mathbb{F}$ having at most $N = 2^K$ rows and using some common set of public parameters (see Definition 1). Our goal is to construct a UV, a single algorithm that uses a fixed sequence of operations to verify the proof of any circuit in the family $\mathcal{C}$. This UV may then be used to define a single recursive verifier circuit that applies to the entire family $\mathcal{C}$.

3.1.1 Public Input Sizes

The first problem we encounter is handling different public input sizes. The Plonk verifier computes $\mathsf{PI}(\mathfrak{z}) = \sum_{i=1}^{\ell} \mathsf{PI}_i L_i(\mathfrak{z})$. The number of operations required to perform this is proportional to ℓ. Furthermore, the degree of $L_i(X)$ depends on the size n of the circuit in question. This makes the computation of $\mathsf{PI}(\mathfrak{z})$ difficult to perform in a universal verifier circuit without imposing a limit on ℓ.

We instead require that the caller of Verify compute the commitment $[\mathsf{PI}]_1$ to the public input polynomial $\mathsf{PI}(X) = \sum_{i=1}^{\ell} \mathsf{PI}_i L_i(X)$ and pass this to the verifier in place of the inputs themselves. This commitment has a size independent of the number of public inputs and removes the need for circuit-specific computation in the verifier. We compute the commitment $[\mathsf{PI}]_1$ and transform the proof $\pi^{\mathcal{P}}$ in an intermediate step, $\mathcal{U}\mathrm{ni}\mathcal{P}\mathrm{lon}\mathcal{K}.\mathsf{Uniformize}$ (Section 4.3). In Lemma 1 we show that the UniPlonk verifier, working with $[\mathsf{PI}]_1$ and the transformed proof, will agree with the ordinary Plonk verifier.

When we encode the UniPlonk verifier as a Universal Verifier Circuit (UVC) for recursive ZKP, it is important to note that the circuit does *not* check the correctness of the public input commitment. That is, the "outer proof" of the UVC demonstrates the correctness of the "inner proof" under the assumption that the public input commitment is correct. It is the responsibility of the UVC's caller to provide the correct public input commitment.

3.1.2 Circuit Sizes

With public inputs replaced by constant-size commitments, the remaining barriers to a fixed-cost universal verifier are related to the size (number of rows) of a circuit. Consider circuits $C_1, C_2 \in \mathcal{C}$ of sizes $n_1 = 2^{k_1}$, $n_2 = 2^{k_2}$ with $k_1 < k_2$. We walk through the steps of the Plonk verify protocol as presented in [13] (where V_1 and V_2 are the respective verifiers for C_1 and C_2), examining how each step depends on the circuit size. An explanation of how the UV performs the step in constant time is also given.

Steps 1 - 4: Validate the points, sample challenge scalars

Identical for V_1 and V_2.

Step 5: Compute zero polynomial evaluation $Z_H(\mathfrak{z}) = \mathfrak{z}^n - 1$

Given that $n_1 < n_2$, the V_2 verifier must perform additional field arithmetic to compute $\mathfrak{z}^{n_2}$ compared to that required to compute $\mathfrak{z}^{n_1}$. The UV addresses this by computing $\mathfrak{z}^n$ for all[1] $n = 2, 4, \ldots 2^K$, (by squaring a total of K times), and uses a multiplexer to select the required value. The multiplexer can be implemented as the scalar product $\langle b^{(n)}, (\mathfrak{z}^2, \ldots \mathfrak{z}^{2^K}) \rangle$ where $b^{(n)}$ is a boolean vector of length K with $b_k^{(n)} = 1$ iff $n = 2^k$.

Note that computing all these powers of $\mathfrak{z}$ costs $K - k_i$ more field multiplications than would have been required for V_i. The UV incurs this additional cost, as well as the cost of the multiplexer and the bit decomposition.

Step 6: Compute Lagrange polynomial evaluation $L_1(\mathfrak{z}) = \frac{\omega(\mathfrak{z}^n - 1)}{n(\mathfrak{z} - \omega)}$

Armed with $\mathfrak{z}^n$ from the previous step, and the input value n, the computation of $L_1(\mathfrak{z})$ is circuit-independent, but relies on ω being the primitive n-th root of unity used by the prover. To compute this, the UV computes 2-powers of ω (the global primitive N-th root of unity from $\mathscr{U}$) and again uses a

[1] In practice, one would start n at some minimum value higher than 2.

multiplexer to select the required value. (As an optimization, all powers of ω can be precomputed and the correct one selected using the above bit vector $b^{(n)}$.)

Step 7: Compute the public input polynomial $\mathsf{PI}(\mathfrak{z}) = \sum_{i=0}^{\ell-1} w_i L_i(\mathfrak{z})$

As explained above, the UV accepts a commitment $[\mathsf{PI}]_1$ and therefore does not need to perform this step.

Step 8: Compute constant term r_0

The computation of r_0 (Eq. (8)) uses only quantities known from above (proof elements, challenge scalars, and $L_1(\mathfrak{z})$). The computation is therefore identical for V_1 and V_2. The UV omits the term $\mathsf{PI}(\mathfrak{z})$ from r_0 and computes:

$$r_0 = -\alpha^2 L_1(\mathfrak{z}) - \alpha(\bar{a} + \beta\bar{s}_{\sigma 1} + \gamma)(\bar{b} + \beta\bar{s}_{\sigma 2} + \gamma)(\bar{c} + \gamma)\bar{z}_\omega \tag{24}$$

The omission of the $\mathsf{PI}(\mathfrak{z})$ term from this step is compensated for in the next step; the original proof must also be transformed to account for this change. The details and justification are in Lemma 1.

Step 9: Compute first part of batched polynomial commitment $[D]_1$

The computation of $[D]_1$ (Eq. (11)) uses values computed above as well as $\mathfrak{z}^{2n}$. Since $\mathfrak{z}^n$ was already computed above, V_1 and V_2 both simply square it to compute $\mathfrak{z}^{2n}$. Therefore their work is identical for this step.

The UV will include the public input commitment in this term, instead of computing $[D]_1$ as

$$\begin{aligned}
[D]_1 = {} & \bar{a}\bar{b} \cdot [q_M]_1 + \bar{a} \cdot [q_L]_1 + \bar{b} \cdot [q_R]_1 + \bar{c} \cdot [q_O]_1 + [q_C]_1 + [\mathsf{PI}]_1 \\
& + \left(\alpha(\bar{a} + \beta\mathfrak{z} + \gamma)(\bar{b} + \beta k_1 \mathfrak{z} + \gamma)(\bar{c} + \beta k_2 \mathfrak{z} + \gamma) + \alpha^2 L_1(\mathfrak{z}) + u\right)[z]_1 \\
& - \alpha(\bar{a} + \beta\bar{s}_{\sigma 1} + \gamma)(\bar{b} + \beta\bar{s}_{\sigma 2} + \gamma)\beta\bar{z}_\omega [s_{\sigma 3}]_1 \\
& - Z_H(\zeta)\left([t_{lo}]_1 + \mathfrak{z}^n [t_{mid}]_1 + \mathfrak{z}^{2n}[t_{hi}]_1\right)
\end{aligned} \tag{25}$$

Steps 10 - 12:

Identical for V_1 and V_2. The UV computes these as described in [13].

The above modifications result in a constant-work UV that can check proofs from any circuit of size $n \leq N$, given the correct verifying key for the circuit. The original Plonk prover protocol is unchanged. The UV's handling of public inputs requires a preprocessing phase in which we compute the commitment $[\mathsf{PI}]_1$ and modify the original Plonk proof $\pi^{\mathcal{P}}$ by a term $[\Delta W_{\mathfrak{z}}]_1$ which is also computed from the public inputs (see Lemma 1).

3.1.3 Universal Verifier Cost

The price we pay for this flexibility is that the UV's work is at least as large as that of the ordinary Plonk verifier for a circuit of size N. The key cost metric we are concerned with here is the marginal cost of the UniPlonk Verifier compared to the cost of the ordinary Plonk Verifier for a circuit of size N. The additional cost comes from using multiplexers to select values $\mathfrak{z}^n$ and ω in Steps 5 and 6. The cost of adding this multiplexer to a UV circuit depends on the value of N but is small compared to the elliptic curve arithmetic that dominates the circuit's cost. So for circuits of size N the marginal cost of using the UV is the cost of two multiplexers.

For circuits with $n = 2^k < 2^K = N$ the marginal cost of this UV is higher. In addition to the two multiplexers, the UV must compute extra powers of $\mathfrak{z}$ in Step 5. The number of additional powers compared to the ordinary Plonk verifier is $K - k$. So, in general, the marginal cost of using the Uniplonk verifier for a circuit of size n is two multiplexers, one bit-decomposition, and $K - k$ field multiplications. We emphasize that no additional elliptic curve arithmetic is required beyond that performed by the ordinary Plonk verifier.

Thus for circuits that use only the vanilla Plonk gate, the marginal cost of using a universal verifier is negligible compared to other fixed costs for a Plonk verifier (such as EC arithmetic and pairing). For UniPlonk with custom gates, we will see a more complex cost profile (Section 6) that does involve additional EC arithmetic.

3.2 Custom Gates

The above ideas can also be applied to Plonk with custom gates as described in Section 2.3. In order to treat this setting, in addition to $N = 2^K$ we define a "global gate set" $\mathscr{G} = \{G_1, \dots G_L\}$, and let $\mathcal{C}$ be the family of all circuits of size at most N using gates belonging to $\mathscr{G}$. Note that the gates used by C can be *any* subset of $\mathscr{G}$.

UniPlonk aims to introduce no additional overhead for a prover who uses only a subset of the gates in $\mathscr{G}$. Whereas the UV's work will be proportional to $|\mathscr{G}|$, the prover's work will scale only with the number (and complexity) of custom gates actually used in the circuit.

Custom gates present several obstacles to achieving a constant-work verifier algorithm. Firstly, since a circuit of $\mathcal{C}$ *may* uses all the gates of $\mathscr{G}$, the UV must evaluate the constraint polynomials of all gates of $\mathscr{G}$, and discard those for gates not used by the circuit. This is achieved by including a pre-processing step that augments the verification key with a bit vector describing which of the gates in $\mathscr{G}$ are used by the circuit. This Setup step also precomputes certain rearrangements of the standard $vk^{\mathcal{P}}$ to aid in the efficiency of the verifier.

The constraint polynomials of a custom gate may use an arbitrary number of witness variables and constant variables, which presents another challenge. A circuit that uses only a subset of the gates in $\mathscr{G}$ may require fewer columns than a circuit that uses all gates. This could be addressed by requiring the prover of C to keep track of unused columns, however in view of our goal to avoid introducing complexity to the prover, we use the setup phase of the protocol to again augment vk with bit vectors encoding which variables were used in C.

We also define a *uniformization* step for proofs, to convert the original proof $\pi^{\mathcal{P}}$ (with shape specific to C) to a *padded* proof π that is compatible with the UV.

The following gives an outline of these steps. Further discussion and mathematical justification are given in Section 5, and the full rolled-out protocol is given in *Appendix A*.

3.2.1 Missing witness/constant variables

Suppose circuit C using a subset of the gates of $\mathscr{G}$ requires only variables $w_1, \dots w_m$, $q_1, \dots q_r$. For clarity in the following discussion, we assume that a gate requiring m witness columns uses the first m of $w_1, \dots w_M$, and likewise for constant columns. This will be generalized in later sections.

Then a Plonk proof $\pi^{\mathcal{P}}$ of C will include commitments $[w_1]_1, \dots [w_m]_1, [q_1]_1, \dots [q_r]_1$ and corresponding openings $\bar{w}_i, \bar{q}_i$. However, the UV must evaluate constraint polynomials with M witness and R constant variables. To this end, the uniformization phase will "pad" the proof by treating $w_{m+1}, \dots w_M, q_{r+1}, \dots q_R$ as zero polynomials. This alone is fine for the evaluation of gate constraints, since any constraint equations that depend non-trivially on the missing variables $w_{m+1}, \dots w_M, q_{r+1}, \dots q_R$ will be evaluated and then discarded (see below).

However, the permutation argument would not succeed without further modification. For example, the ordinary Plonk verifier for C computes terms like

$$r_1 = \prod_{i=1}^{m} (\bar{w}_i + \beta k_{i\mathfrak{z}} + \gamma) + \alpha L_1(\mathfrak{z})$$

which explicitly depend on the number m of witness variables used by C.

UniPlonk's UV will instead compute this term with products from 1 to M, rather than 1 to m. To ensure that the extra terms do not affect the value of the product, the preprocessing phase computes a bit vector $(b_1^{(w)}, \dots b_M^{(w)})$ with $b_i^{(w)} = 1$ iff circuit C used witness variable w_i. The uniformization step will fill in unused witness entries with a default value, and this bit vector is used by the verifier to discard these terms that were not present when the prover performed the corresponding computation. The Setup step also fills in missing values $\bar{\mathsf{s}}_{\sigma i}$ to ensure the well-formedness of the verification key.

3.2.2 Missing constraint polynomials

To compute the constraint polynomials for all gates, the UV must evaluate a term of the form

$$\sum_{i=1}^{L} a_i \bar{S}_{i_j} G_i(\bar{w}_1, \dots \bar{w}_M, \bar{q}_1, \dots \bar{q}_R)$$

where $\bar{w}_j$ is the evaluation of the witness polynomial at a challenge point. The result of this evaluation must match what an ordinary circuit-specific Plonk verifier would compute. This verifier, working only

with gates G_i for $i \in I \subset \{1, \dots L\}$, computes the term as

$$\sum_{j=1}^{|I|} \alpha^j \bar{S}_{i_j} G_{i_j}(\bar{w}_1, \dots \bar{w}_M, \bar{q}_1, \dots \bar{q}_R)$$

By appropriately assigning the coefficients a_i, we can ensure that the UV computes this term correctly. To do so, the preprocessing phase computes another bit vector $(b_1^{(g)}, \dots b_L^{(g)})$ with $b_i^{(g)} = 1$ iff gate G_i is used by circuit C. We demonstrate in Appendix A how to compute the correct coefficients a_i from the bit vector in constant time. A necessary assumption to make this work is that UniPlonk prover and verifier agree on an ordering of $\mathscr{G}$.

4 $\mathcal{U}\mathrm{ni}\mathcal{P}\mathfrak{lonK}$ construction

We first define the UniPlonk protocol for circuits that use only the "vanilla" Plonk gate. UniPlonk unifies the prover and verifier protocol for all circuits belonging to a certain class $\mathcal{C}$ of all circuits compatible with a given configuration.

Definition 1 (Configuration)**.** *A tuple $\mathscr{U} = (pp, \mathsf{srs}, N, \omega, k_1, k_2)$, where*

- *pp are protocol parameters specifying pairing curves $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, with scalar field $\mathbb{F}$.*
- *srs is the structured reference string for KZG commitments.*
- *$N = 2^K$ is an upper bound on the number of rows in a circuit.*
- *ω is a primitive N-th root of unity in $\mathbb{F}$ which generates a multiplicative subgroup H.*
- *$k_1, k_2 \in \mathbb{F}$ values chosen for use by the permutation argument, as described in Section 2.2, such that H and cosets $k_1 H$ and $k_2 H$ are pairwise disjoint.*

Definition 2 ($\mathscr{U}$-compatible circuit)**.** *We say that a circuit C is compatible with a UniPlonk configuration $\mathscr{U}$ if:*

- *C requires a domain of size $n = 2^k$ with $k \leq K$.*
- *C uses the parameters pp, srs, $\omega_n := \omega^{N/n}$, and the constants k_1 and k_2 specified in $\mathscr{U}$.*

4.1 Setup

Let C be $\mathscr{U}$-compatible for some configuration $\mathscr{U}$. The UniPlonk setup function simply performs the Plonk setup

$$\mathcal{U}\mathrm{ni}\mathcal{P}\mathfrak{lonK}.\mathsf{Setup}(C, \mathsf{srs}) = (pk^{\mathcal{P}}, vk^{\mathcal{P}})$$

as defined in Section 2.2.1. In particular, the verification key

$$vk^{\mathcal{P}} = ([q_M]_1, [q_L]_1, [q_R]_1, [q_O]_1, [q_C]_1, [S_{\sigma 1}]_1, [S_{\sigma 2}]_1, [S_{\sigma 3}]_1, n)$$

contains the domain size n for C. This will be used by the universal verifier.

4.2 Prove

The original Plonk prover algorithm is used:

$$\mathcal{U}\mathrm{ni}\mathcal{P}\mathfrak{lonK}.\mathsf{Prove}(pk, \mathsf{PI}, w) = \mathcal{P}\mathfrak{lonK}.\mathsf{Prove}(pk, \mathsf{PI}, w)$$

The only modification is the transcript conventions of Section 2.4.

4.3 Uniformize

This step computes the commitment to public inputs $[\mathsf{PI}]_1$ and transforms a Plonk proof to account for differences between $\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}$ and $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}$ described below. Let $\pi^{\mathcal{P}}$ be the proof output by $\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Prove}(pk, \mathsf{PI}, w)$ for a $\mathscr{U}$-compatible circuit C of size $n = 2^k$.

Then

$$\pi = \mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Uniformize}(n, \pi^{\mathcal{P}}, \mathsf{PI})$$

is computed as follows:

1. Compute $\mathsf{PI}(X) = \sum_{i=1}^{\ell} \mathsf{PI}_i L_i(X)$, where L_i is the degree $n-1$ Lagrange interpolation polynomial defined by $L_i(\omega_n^j) = \delta_i^j$.
2. Compute $[\mathsf{PI}]_1$
3. Compute the verifier challenge point $\mathfrak{z}$.
4. Compute $\Delta W_{\mathfrak{z}}(X) = \frac{\mathsf{PI}(X) - \mathsf{PI}(\mathfrak{z})}{X - \mathfrak{z}}$
5. Compute $[\Delta W_{\mathfrak{z}}(X)]_1$
6. Let π be $\pi^{\mathcal{P}}$ with the term $[W_{\mathfrak{z}}]_1$ is replaced by $[W_{\mathfrak{z}}]_1 + [\Delta W_{\mathfrak{z}}(X)]_1$
7. Return π.

4.4 Verify

$\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk, [\mathsf{PI}]_1, \pi)$ performs the following steps.

1. Parse
$$\begin{pmatrix} [a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mid}]_1, [t_{hi}]_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \\ \bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_{\omega} \end{pmatrix} \leftarrow \pi$$
and
$$([q_M]_1, [q_L]_1, [q_R]_1, [q_O]_1, [q_C]_1, [S_{\sigma 1}]_1, [S_{\sigma 2}]_1, [S_{\sigma 3}]_1, n) \leftarrow vk$$
2. Check that proof consists of valid $\mathbb{G}_1$ and $\mathbb{F}$ points.
3. Compute the bit representation $b^{(n)}$ of n.
4. Compute the challenge points β, γ, α, $\mathfrak{z}$, v, u according to Section 2.4.
5. Compute all powers of $\mathfrak{z}$: $\vec{z} = (\mathfrak{z}, \mathfrak{z}^2, \ldots \mathfrak{z}^{2^K})$ and the scalar product $\mathfrak{z}^n = \langle b^{(n)}, \vec{z} \rangle$. Compute $Z_{H_k}(\mathfrak{z}) = \mathfrak{z}^n - 1$.
6. Compute $\omega_k = \langle \vec{\omega}, b^{(n)} \rangle$ where $\vec{\omega} = (\omega^{N/2}, \omega^{N/4}, \ldots \omega)$. (Note that the vector $\vec{\omega}$ can be pre-computed.)
7. Compute $L_1(\mathfrak{z}) = \frac{\omega_k Z_{H_k}(\mathfrak{z})}{n(\mathfrak{z} - \omega_k)}$.
8. Compute r_0 according to Eq. (24).
9. Compute $[D]_1$ according to Eq. (25). The value $\mathfrak{z}^{2n}$ is computed by squaring $\mathfrak{z}$ as computed in Step 5.
10. Compute $[F]_1$ according to Eq. (10).
11. Compute $[E]_1$ according to Eq. (9).
12. Return the result of the final pairing check Eq. (12).

Note that Steps 5, 6 and 7 are constant work because there is a fixed K for UniPlonk.

4.5 Verifier Circuit

All steps of $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}$ involve the same number of arithmetic operations for all circuits compatible with a given configuration $\mathscr{U}$. This makes the UniPlonk verifier ideally suited for implementation as a Universal Verifier Circuit (UVC), using recursion to wrap one or more "inner" proofs of $\mathscr{U}$-compatible in an "outer" proof of the UVC.

Let UVC denote a description of $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}$ as a circuit (Plonk, R1CS, or any other arithmetization). The witness to UVC is a UniPlonk proof π as computed by $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Uniformize}$. The public inputs to UVC are $[\mathsf{PI}]_1$ and vk for some $\mathscr{U}$-compatible C. An outer proof π_{UVC} is generated according to some ZK proving schemes. Verifying π_{UVC} against the public inputs $([\mathsf{PI}]_1, vk)$ implies the existence of some proof π such that $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk, [PI]_1, \pi) = 1$. If $[PI]_1$ was computed from public inputs PI, this verification implies that there exists some $\pi^{\mathcal{P}}$ such that $\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk^{\mathcal{P}}, \mathsf{PI}, \pi^{\mathcal{P}}) = 1$ (by the equivalence of UniPlonk and Plonk, Lemma 1).

We emphasize that the UVC takes as input the commitment $[\mathsf{PI}]_1$. Therefore the statement being proven by the UVC is "there exists a valid proof π for a circuit C with verifying key vk and public inputs whose commitment is $[\mathsf{PI}]_1$." Any application that uses the outer proof as a proxy for the validity of the inner proof must therefore check that $[\mathsf{PI}]_1$ is indeed the correct commitment to the expected public inputs PI.

In practice, it is expensive for a circuit to have a large public input set. Rather than passing in the entire verifying key as a public input to the circuit, one might instead include the verifying key as part of the private witness and use only a commitment to it in the circuit's public inputs. Then the UVC must include a check that the verifying key contained in the witness indeed corresponds to the commitment in the public inputs.

4.6 $\mathcal{P}\mathfrak{lon}\mathcal{K}$ Equivalence

We begin with a lemma that justifies UniPlonk's handling of public input values, then conclude with a corollary that UniPlonk is equivalent to Plonk in the sense that UniPlonk's verifier will always agree with the Plonk verifier when given a correctly transformed proof and commitment to public inputs.

Lemma 1. *The Plonk verifier for a circuit C accepts a proof $\pi^{\mathcal{P}}$ with public inputs PI if and only if the UniPlonk verifier accepts the proof $\pi = f_{\mathsf{PI}}(\pi^{\mathcal{P}})$ with the public input commitment $[\mathsf{PI}]_1$. Here f_{PI} is the function that replaces the term $[W_{\mathfrak{z}}]_1$ of $\pi^{\mathcal{P}}$ by $[W_{\mathfrak{z}}]_1 + \left[\frac{1}{X-\mathfrak{z}}\left(\mathsf{PI}(X) - \mathsf{PI}(\mathfrak{z})\right)\right]_1$.*

Therefore $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}$ is equivalent to $\mathcal{P}\mathfrak{lon}\mathcal{K}$ in the sense that for any $\mathscr{U}$-compatible circuit C

$$\begin{pmatrix} \pi \leftarrow \mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Uniformize}(vk, \pi^{\mathcal{P}}, \mathsf{PI}) \\ [\mathsf{PI}]_1 \leftarrow KZG.\mathsf{Commit}(\mathit{srs}, \mathsf{PI}) \\ 1 = \mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk, \pi, [\mathsf{PI}]_1) \end{pmatrix} \iff \left(1 = \mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk^{\mathcal{P}}, \pi^{\mathcal{P}}, \mathsf{PI})\right) \tag{26}$$

Proof. (We denote the ordinary Plonk verifier's quantities with the superscript $\mathcal{P}$.)

The proof $\pi^{\mathcal{P}}$ has the form

$$\pi^{\mathcal{P}} = \left([a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mid}]_1, [t_{hi}]_1, [W^{\mathcal{P}}_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_{\omega}\right)$$

We will demonstrate that the pairing check performed by the Plonk verifier on $\pi^{\mathcal{P}}$ agrees with the pairing check performed by the UniPlonk verifier on

$$\pi = f_{\mathsf{PI}}(\pi^{\mathcal{P}}) = \left([a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mid}]_1, [t_{hi}]_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_{\omega}\right)$$

where $[W_{\mathfrak{z}}]_1 = [W^{\mathcal{P}}_{\mathfrak{z}}]_1 + \left[\frac{1}{X-\mathfrak{z}}\left(\mathsf{PI}(X) - \mathsf{PI}(\mathfrak{z})\right)\right]_1$

Referring to Eqs. (8) and (24), we see that the two verifiers will compute the constant term r_0 differently. The difference in their computations is

$$\Delta r_0 := r_0^{\mathcal{P}} - r_0 = \mathsf{PI}(\mathfrak{z})$$

implying that the difference in their computation of $[E]_1$, Eq. (9) is

$$\Delta[E]_1 := [E^{\mathcal{P}}]_1 - [E]_1 = -\mathsf{PI}(\mathfrak{z}) \cdot [1]_1$$

Referring to Eqs. (11) and (25), we see that the two verifiers will also compute the term $[D]_1$ differently:

$$\Delta[D]_1 := [D^{\mathcal{P}}]_1 - [D]_1 = -[\mathsf{PI}]_1$$

Their computation of the $[F]_1$ term, Eq. (10), will differ by the same amount: $\Delta[F]_1 = -[\mathsf{PI}]_1$. The quantity $[F]_1 - [E]_1$ used in the final pairing check therefore differs by

$$\Delta\left([F]_1 - [E]_1\right) = \mathsf{PI}(\mathfrak{z}) \cdot [1]_1 - [\mathsf{PI}]_1$$

Denoting the final pairing check, Eq. (12), as

$$e(LHS, [x]_2) \stackrel{?}{=} e(RHS, [1]_2)$$

we have

$$\begin{aligned}
\Delta LHS :=& LHS^{\mathcal{P}} - LHS \\
=& [W_{\mathfrak{z}}^{\mathcal{P}}]_1 - [W_{\mathfrak{z}}]_1 \\
=& \left[\frac{1}{X - \mathfrak{z}}\left(\mathsf{PI}(\mathfrak{z}) - \mathsf{PI}(X)\right)\right]_1 \\
\Delta RHS :=& RHS^{\mathcal{P}} - RHS \\
=& \mathfrak{z} \cdot \left([W_{\mathfrak{z}}^{\mathcal{P}}]_1 - [W_{\mathfrak{z}}]_1\right) + \Delta\left([F]_1 - [E]_1\right) \\
=& \mathfrak{z} \cdot \left[\frac{1}{X - \mathfrak{z}}\left(\mathsf{PI}(\mathfrak{z}) - \mathsf{PI}(X)\right)\right]_1 + \left(\mathsf{PI}(\mathfrak{z}) \cdot [1]_1 - [\mathsf{PI}]_1\right)
\end{aligned}$$

Observe that $\left(\mathsf{PI}(\mathfrak{z}) \cdot [1]_1 - [\mathsf{PI}]_1\right)$ is the KZG commitment to the polynomial $\mathsf{PI}(\mathfrak{z}) - \mathsf{PI}(X)$ and that $\left[\frac{1}{X-\mathfrak{z}}\left(\mathsf{PI}(\mathfrak{z}) - \mathsf{PI}(X)\right)\right]_1$ is a KZG proof that this polynomial vanishes at $\mathfrak{z}$. Therefore

$$e(\Delta LHS, [x]_2) = e(\Delta RHS, [1]_2)$$

By linearity of the pairing, this equality implies that the two verifiers will both either accept or reject the final pairing check. □

5 Custom Gates

In this section, we outline how UniPlonk can be extended to handle custom gates. For clarity, we focus on the simple setting of Section 2.3, and later describe how the protocol can be modified to support common extensions such as reduced selectors and references to witness elements in other rows. We expect that the ideas here can be readily applied to other settings with minimal modification.

5.1 $\mathcal{U}\mathrm{ni}\mathcal{P}\mathrm{lon}\mathcal{K}$ circuit definition with custom gates

In order to construct a universal verifier over custom gates, we define a *global gate set* of supported gates, along with other parameters that make up a UniPlonk configuration. Circuits using a subset of the global gate set (along with some basic settings similar to those defined in Definition 1) are compatible with the UniPlonk configuration, and their proofs can be verified by a fixed-cost universal verifier. In Section 6 we analyze the cost of the universal verifier in terms of the parameters in the configuration.

Definition 3 ($\mathcal{U}\mathrm{ni}\mathcal{P}\mathrm{lon}\mathcal{K}$ configuration for custom gates)**.** *A tuple $\mathscr{U} = (pp, \mathsf{srs}, N, \omega, \mathscr{G}, \{k_i\}_{i=1}^{M})$, where*

- *pp are protocol parameters specifying pairing curves $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, with scalar field $\mathbb{F}$.*
- *srs is the structured reference string for KZG.*
- *$N = 2^K$ is an upper bound on the number of rows in a circuit.*
- *ω is a primitive N-th root of unity in $\mathbb{F}$ which generates a multiplicative subgroup H.*
- *$\mathscr{G}$ is the "global gate set" $\{G_i^*\}_{i=1}^{L}$ of L polynomials over M witness and constant columns.*

- $\{k_i\}_{i=1}^{M}$ *values chosen for use by the permutation argument, as described in Section 2.3, such that* $k_1 = 1$ *and cosets* $k_i H$ *are distinct.*

The following configuration variables are implicitly defined by the elements of $\mathscr{U}$ *and are denoted as follows:*

- *the number* L *of gates in* $\mathscr{G}$*.*
- *witness columns* $w_1^*, \ldots, w_M^*$ *and constant columns* $q_1^*, \ldots, q_R^*$*.*
- M *the number of witness columns used by gates in* $\mathscr{G}$*.*
- R *the number of constant columns used by gates in* $\mathscr{G}$*.*
- D *the maximum number of polynomials* $t_1, \ldots t_D$ *required to represent the quotient polynomial in chunks of degree* $N - 1$*.*

By convention $G_1^*(w_1^*, \ldots) = -w_1^*$*, and is referred to as the* public input gate*.*

We now define the conditions under which a circuit may be used with a configuration $\mathscr{U}$.

Definition 4 ($\mathscr{U}$-compatible circuit)**.** *We say that a circuit* C *is compatible with a* $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}$ *configuration* $\mathscr{U}$ *if:*

- C *requires a domain of size* $n < N$*,*
- C *uses gates* $G_1, \ldots, G_l \subset \{G_i^*\}_{i=1}^{L}$*, where the* $\{G_i\}_{i=1}^{l}$ *is* ordered *by* $\{G_i^*\}_{i=1}^{L}$*. (Namely, for any* $i, i' \in [l]$ *with* $i < i'$*, where* $j, j' \in [L]$ *are the corresponding indices such that* $G_i = G_j^*$ *and* $G_{i'} = G_{j'}^*$*, we have* $j < j'$*.)*
- $G_1 = G_1^*$ *(that is,* C *includes the public input gate).*

Let $w_1, \ldots, w_m$ *be the subset of witness columns used by gates in* C*, and* $q_1, \ldots, q_r$ *be the constant columns, where* $m \leq M$ *and* $r \leq R$*. These are assumed to be ordered by* $\{w_i^*\}_{i=1}^{M}$ *and* $\{q_i^*\}_{i=1}^{R}$ *respectively.*

Given a configuration $\mathscr{U}$ and $\mathscr{U}$-compatible circuit C, we can assign injective maps from the indices for gates, witness columns, and constant columns in C, to indices in the global configuration $\mathscr{U}$. We denote these maps $\eta_g : [l] \to [L]$, $\eta_w : [m] \to [M]$ and $\eta_q : [r] \to [R]$ respectively, and note that they are uniquely defined for a specific $\mathscr{U}$ and C pair. (For example, where C uses gates $G_1 = G_1^*$ and $G_2 = G_4^*$, we define $\eta_g : [2] \to [L]$ such that $\eta_g(1) = 1$ and $\eta_g(2) = 4$.)

Remark 1. *We write* $G_i(w_1, \ldots, w_m, q_1, \ldots, q_r)$*, whereas strictly speaking* $G_i = G_j^*$ *for some* j*, which is a polynomial in* M *variables (not* m*). Since* C *is defined such that columns* $w_1, \ldots, w_m$ *and* $q_1, \ldots, q_r$ *contain all* used *by the gates in* C*, we can define* $G_i(w_1, \ldots, w_m, q_1, \ldots, q_r)$ *to mean the evaluation of* G_j^* *using* $w_1, \ldots, w_m$ *as arguments* $w_{\eta_w(1)}^*, \ldots w_{\eta_w(m)}^*$*, and* $q_1, \ldots, q_r$ *as arguments* $q_{\eta_q(1)}^*, \ldots q_{\eta_q(r)}^*$*, with* 0 *passed for all other (unused) arguments.*

5.2 Proof Uniformization with custom gates

We first note that for a Plonk proof $\pi^{\mathcal{P}}$ of the form given in Eq. (21), a transformation can be defined (analogous to that of Section 4.3) such that the modified proof can be verified using a commitment $[PI]_1$ to the public input polynomial, rather than the (circuit-specific) public inputs themselves.

The primary source of complexity for the verifier is that it must be able to compute the commitment to the linearization polynomial and batched evaluation commitment that matches that used by the circuit-specific prover. Therefore the universal verifier must be supplied with a proof of fixed size (i.e. independent of C), and the uniformization step and verifier require metadata about C (stored in the UniPlonk verification key), in order to reformat π and produce a fixed-size proof π' for the universal verifier.

Let C be a $\mathscr{U}$-compatible circuit, and $\pi^{\mathcal{P}}$ be a Plonk proof for C, with public inputs $(w_i)_{i=1}^{\ell}$. The proof $\pi^{\mathcal{P}}$ has the form:

$$\pi^{\mathcal{P}} = \Big([w_1]_1, \ldots, [w_m]_1, [z]_1, [t_1]_1, \ldots, [t_d]_1, [W_{\mathfrak{z}}^{\mathcal{P}}]_1, [W_{\mathfrak{z}\omega}]_1, \\ \bar{w}_1, \ldots, \bar{w}_m, \bar{q}_r, \bar{\mathsf{s}}_{\sigma 2}, \ldots \bar{\mathsf{s}}_{\sigma m}, \bar{z}_\omega\Big)$$

Analogous to the transformation applied in Section 4.3, the transformation required to allow the verifier to accept a commitment to the public input polynomial is as follows:

$$[W_{\mathfrak{z}}]_1 = [W_{\mathfrak{z}}^{\mathcal{P}}]_1 + \alpha^2[\Delta W_{\mathfrak{z}}]_1 \tag{27}$$

where α and $\mathfrak{z}$ are the challenges computed from $[w_1]_1, \ldots, [w_m]_1$, $[z]_1$ and $[t_1]_1, \ldots, [t_d]_1$, and $\Delta W_{\mathfrak{z}}$ is the polynomial $\Delta W_{\mathfrak{z}}(X) = (\mathsf{PI}(X) - \mathsf{PI}(\mathfrak{z}))/(X - \mathfrak{z})$. The commitment $[r]_1$ computed by the universal verifier takes the form:

$$\begin{aligned}
[r]_1 = & \left(\prod_{i=1}^{m}(\bar{w}_i + \beta k_i \mathfrak{z} + \gamma)\right)[z]_1 \\
& - \left(\prod_{i=2}^{m}(\bar{w}_i + \beta \bar{\mathsf{s}}_{\sigma i} + \gamma)\right)\bar{z}_\omega \cdot ((\bar{w}_1 + \gamma)\cdot[1]_1 + \beta \cdot [\mathsf{s}_{\sigma 1}]_1) \\
& + \alpha L_1(\mathfrak{z}) \cdot ([z]_1 - [1]_1) \\
& + \alpha^2 [PI]_1 \\
& + \sum_{k=1}^{l} \alpha^{k+1} G_k(\bar{w}_1, \ldots, \bar{w}_M, \bar{q}_1, \ldots, \bar{q}_Q) \cdot [S_k] \\
& - Z_H(\mathfrak{z})([t_1]_1 + \mathfrak{z}^n[t_2]_1 + \cdots + \mathfrak{z}^{(d-1)n}[t_d]_1)
\end{aligned} \tag{28}$$

(where $PI(\mathfrak{z})$ in Eq. (19) has been replaced by the commitment $[\mathsf{PI}]_1$). Further, the universal verifier does not add the term $\mathsf{PI}(\mathfrak{z})$ (evaluation of the public input polynomial at the challenge point) when performing the calculation equivalent to Eq. (8).

The universal verifier must compute Eq. (28) in terms of the global gate set $\{G_k^*\}$, and must ensure that terms such as k_i, $\bar{\mathsf{s}}_{\sigma i}$ and α^{k+1} appear alongside the correct $\hat{w^*}_i$ and G_k^* elements. To facilitate this, the verification key is augmented with bit vectors describing which gates, and thereby which witness and constant columns, are used by the circuit.

During Setup, commitments to selector polynomials, constant columns, and permutation polynomials are also rearranged in the augmented verification key, so that they appear at the appropriate places according to the configuration $\mathscr{U}$ (where "gaps" are filled with default values). Let the augmented verification key for a circuit C be given by:

$$\begin{gathered}
vk = ([S_1']_1, \ldots, [S_L']_1, [q_1']_1, \ldots, [q_R']_1, \\
[\mathsf{s}_{\sigma 1}']_1, \ldots, [\mathsf{s}_{\sigma M}']_1, k_1', \ldots, k_M', \\
(b_1^{(g)}, \ldots, b_L^{(g)}), (b_1^{(w)}, \ldots, b_M^{(w)}), (b_1^{(q)}, \ldots, b_R^{(q)}), (b_1^{(t)}, \ldots, b_D^{(t)}), n)
\end{gathered}$$

Similarly, Plonk proofs π^P for C (whose size and shape depend on C) must be transformed into a proof π of fixed-size and well-defined format, before they can be processed by the universal verifier. This (along with the transformation described above) is performed by the $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Uniformize}$ algorithm, which uses the augmented verification key to arrange the proof elements appropriately, and again fill in any elements required to transform it into a uniform shape. This process is described in detail in Appendix A.

Let the output of the proof uniformization step be denoted:

$$\begin{gathered}
\pi = ([w_1']_1, \ldots, [w_M']_1, [z]_1, [t_1']_1, \ldots, [t_D']_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \\
\bar{w}_1', \ldots, \bar{w}_M', \bar{q}_1', \ldots, \bar{q}_R', \bar{\mathsf{s}}_{\sigma 2}', \ldots \bar{\mathsf{s}}_{\sigma M}', \bar{z}_\omega)
\end{gathered}$$

where elements $[w_i']_1$, $[t_i']_1$, $\bar{w}_i'$, $\bar{q}_i'$, $\bar{\mathsf{s}}_{\sigma i}'$ represent the sequences $[w_i]_1$, $[t_i]_1$, $\bar{w}_i$, $\bar{q}_i$, $\bar{\mathsf{s}}_{\sigma i}$ from $\pi^{\mathcal{P}}$, but with elements rearranged and gaps filled with default values.

By arranging elements in vk and π appropriately, most terms in Eq. (28) can be computed in a

straightforward way:

$$\left(\prod_{i=1}^{M} \mathsf{select}\left(b_i^{(w)}, (\bar{w}_i' + \beta k_{i\mathfrak{z}}' + \gamma), 1\right)\right)[z]_1$$
$$-\left(\prod_{i=2}^{M} \mathsf{select}\left(b_i^{(w)}, \bar{w}_i' + \beta \bar{\mathsf{s}}_{\sigma i}' + \gamma, 1\right)\right)\bar{z}_\omega \cdot ((\bar{w}_1' + \gamma) \cdot [1]_1 + \beta \cdot [\mathsf{s}_{\sigma 1}']_1)$$
$$+\alpha L_1(\mathfrak{z}) \cdot ([z]_1 - [1]_1)$$
$$+\alpha^2 [PI]_1$$
$$-Z_H(\mathfrak{z})\left(\sum_{i=1}^{D} \mathsf{select}(b_i^{(t)}, \mathfrak{z}^{n(i-1)}, 0)[t_i']_1\right)$$

We note also that the extra elements in vk and π, used to "pad" to the correct format, cannot be used to compromise the soundness of the underlying scheme since they are discarded using the select function.

It remains for the universal verifier to compute the commitment to the gate constraint polynomial Appendix A.4:

$$\sum_{k=1}^{l} \alpha^{k+1} G_k(\bar{w}_1, \ldots, \bar{w}_M, \bar{q}_1, \ldots, \bar{q}_Q) \cdot [S_k]$$

in terms of the elements of vk and π. This involves slightly more complexity in order to match up α^i terms with the correct gate equations but is achieved with a relatively simple algorithm using the variable α' to maintain a *current* α^i term, where bits $(b_1^{(g)}, \ldots, b_L^{(g)})$ is used to sum only the equation for gates that are used in C and to update $\alpha' \leftarrow \alpha' \times \alpha$ only for each used gate:

$[r_3]_1 \leftarrow [0]_1$;
$\alpha' \leftarrow \alpha$;
for $i \leftarrow 1$ **to** L **do**
  $\alpha' \leftarrow \alpha' \times \mathsf{select}\left(b_i^{(g)}, \alpha, 1\right)$;
  $[r_3]_1 \leftarrow [r_3]_1 + \mathsf{select}\left(b_i^{(g)}, \alpha' G_i^*(\bar{w}_1', \ldots, \bar{w}_M', \bar{q}_1', \ldots \bar{q}_R') \cdot [S_i']_1, [0]_1\right)$;
end
return $[r_3]_1$

All details are given in Appendix A.

5.3 Equivalence of Plonk and UniPlonk proofs

We show the analog of Lemma 1 for UniPlonk with custom gates.

For a given circuit C, let (pk, vk) represent the proving and verification keys from $\mathcal{P}\mathfrak{lon}\mathcal{K}$.Setup, and let vk' be the augmented verification key from $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}$.Setup. Given a proof π for C with public inputs $(w_i)_{i=1}^l$, where PI represents the public input polynomials for $(w_i)_{i=1}^l$, we show that:

$$\begin{pmatrix} \pi' \leftarrow \mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Uniformize}(vk', \pi, \mathsf{PI}) \\ [\mathsf{PI}]_1 \leftarrow \mathrm{KZG.Commit}(\mathsf{srs}, \mathsf{PI}) \\ 1 = \mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk', \pi', [\mathsf{PI}]_1) \end{pmatrix} \iff (1 = \mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk, \pi, \mathsf{PI}))$$

The standard $\mathcal{P}\mathfrak{lon}\mathcal{K}$ verification accepts iff

$$e([W_{\mathfrak{z}}]_1 + u[W_{\mathfrak{z}\omega}]_1, [x]_2) = e(\mathfrak{z}[W_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega[W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2)$$

for $[F]_1$ and $[E]_1$ computed as described in Section 2.3.6. Let $[F']_1$ and $[E']_1$ be the equivalent terms computed by $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}$.Verify, and observe from Eqs. (19) and (28) that:

$$[F']_1 = [F]_1 + \alpha^2[\mathsf{PI}]_1$$
$$[E']_1 = [E]_1 + \alpha^2 \mathsf{PI}(\mathfrak{z})[1]_1$$

Using these equations along with Eq. (27), and noting that $[\Delta W_{\mathfrak{z}}]_1$ satisfies:

$$e([\Delta W_{\mathfrak{z}}]_1, [x]_2) = e(\mathfrak{z}[\Delta W_{\mathfrak{z}}]_1 + [\mathsf{PI}]_1 - PI(\mathfrak{z})[1]_1, [1]_2) \tag{29}$$

the pairing check for $\mathcal{UniPlonK}$.Verify has the form:

$$e\left(\left[W'_{\mathfrak{z}}\right]_1 + u[W_{\mathfrak{z}\omega}]_1, [x]_2\right) \stackrel{?}{=} e\left(\mathfrak{z}[W'_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega[W_{\mathfrak{z}\omega}]_1 + [F']_1 - [E']_1, [1]_2\right)$$

from which the equivalence of the pairing checks then follows:

$$\begin{aligned}
&e\left(\left[W'_{\mathfrak{z}}\right]_1 + u[W_{\mathfrak{z}\omega}]_1, [x]_2\right) \\
&= e\left([W_{\mathfrak{z}}]_1 + \alpha^2[h_{\mathfrak{z}}]_1 + u[W_{\mathfrak{z}\omega}]_1, [x]_2\right) \\
&= e\left([W_{\mathfrak{z}}]_1 + u[W_{\mathfrak{z}\omega}]_1, [x]_2\right) \cdot e(\alpha^2[h_{\mathfrak{z}}]_1, [x]_2) \\
&= e\left(\mathfrak{z}[W_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega[W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2\right) \cdot e\left(\alpha^2\mathfrak{z}[h_{\mathfrak{z}}]_1 + [\mathsf{PI}]_1 - PI(\mathfrak{z})[1]_1, [x]_2\right) \\
&= e\left(\mathfrak{z}\left([W_{\mathfrak{z}}]_1 + \alpha^2[h_{\mathfrak{z}}]_1\right) + u\mathfrak{z}\omega[W_{\mathfrak{z}\omega}]_1 + [F]_1 + \alpha^2[\mathsf{PI}]_1 - ([E]_1 + \alpha^2\mathsf{PI}(\mathfrak{z})[1]_1), [1]_2\right) \\
&= e\left(\mathfrak{z}[W'_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega[W_{\mathfrak{z}\omega}]_1 + [F']_1 - [E']_1, [1]_2\right)
\end{aligned}$$

5.4 Transcript handling by prover and verifier

The Prover and Verifier must share a common view of the transcript in order to generate the same challenge points. In Section 2.4, both seed the transcript with the expression $vk \parallel [\mathsf{PI}]_1$, however in the context of custom gates, the UniPlonk verifier's vk contains additional data compared to verifying key $vk^{\mathcal{P}}$ used by the prover. Moreover, π (as produced by the Uniformize step and passed to Prove) will, in general, contain more elements than $\pi^{\mathcal{P}}$ (as produced by the Prove algorithm), and will contain "padding" elements from Uniformize over which the prover does not have control.

One possible approach to this is the use of a cryptographic sponge construction in the transcript generation, whereby the select function is used to dynamically filter out elements that were introduced by the Uniformize step using the bit vector embedded in vk. We assume a sponge as in Section 2.4. Consider the example of commitments to witness columns being added to the transcript. The verifier may perform the following constant-time operations to update the sponge state in a way that matches the prover:

for $i \leftarrow 1$ **to** M **do**
  $\mathcal{S} \leftarrow \mathsf{select}\left(b_i^{(w)}, \mathscr{S}.\mathsf{Absorb}(\mathcal{S}, [w_i]_1), \mathcal{S}\right)$;
end

5.5 Optimizing selector polynomials

In the setting of Section 2.3 the selector polynomials S_i are assumed to be binary selectors, meaning that there is one S_i per gate G_i where at each ω^j, S_i takes the value 1 if the gate is enabled on row j, and 0 otherwise.

In an alternative approach, described in [16] and used by [26, 27], gates can be grouped, and non-binary selector polynomials used. In this way, only one selector polynomial per gate group is required, rather than one per gate. Let I_j be the set of indices of gates in the j-th group for some circuit. For G_i with $i \in I_j$, the gate expression $S_i(X)G_i(w_1, \ldots, w_m, q_1, \ldots, q_r)$ becomes $f_i(X)G_i(w_1, \ldots, w_m, q_1, \ldots, q_r)$, where

$$f_i(X) = S_j(X) \prod_{k \in I_j, k \neq i} (k - S_j(X)) \tag{30}$$

In this way, f_i evaluates to some non-zero value when $S_j = i$, and zero otherwise. For rows in which S_j evaluates to 0, all gates in group j are deactivated.

In computing the commitment to the linearization polynomial, the verifier must evaluate each $f_i(\mathfrak{z})$ or $[f_i(\mathfrak{z})]_1$. Where the degree of f_i in S_j is greater than 1 (for any non-trivial group), the proof π must be modified to contain $S_j(\mathfrak{z})$ and a proof of this evaluation.

In general, a circuit may define circuit groups from 1 up to L gates and therefore may define up to L groups. In order to evaluate the filter polynomials in constant time, the vk' for the circuit is augmented with L bit vector, each of size L, where each bit vector specifies the gates used in each group.

The verifier must evaluate $f_i(\mathfrak{z})$. In UniPlonk, the UV must perform a constant-work computation. The product in Eq. (30) has as many terms as the size of the group to which gate G_i belongs. This grouping depends on which gates are present in circuit C, and so the UV must use a worst-case assumption

on the size of the product. The UV, therefore, supposes that the product contains up to $|\mathscr{G}|$ terms and computes

$$f_i(\mathfrak{z}) = \prod_{j=1}^{|\mathscr{G}|} s_j \tag{31}$$

where

$$s_j = \begin{cases} \eta(j) - S(\mathfrak{z}), & \text{if gate } G_j \text{ has index } \eta(j) \text{ in circuit } C \\ S(\mathfrak{z}), & \text{if } j = i \\ 1, & \text{otherwise} \end{cases}$$

Eq. (31) always computes a product of $|\mathscr{G}|$ terms and agrees with Eq. (30) because the added terms in this product are equal to 1.

In order to use the UV as an in-circuit verifier, the values s_j must be computed in-circuit using a constant work algorithm. This can be done by computing bit vector $b^{(i)}$ during the setup phase, where $b^{(i)}$ encodes which gates belong to the same group as gate G_i. That is, $b^{(i)}_j = 1$ iff gate G_j belongs to the same group as gate G_i. These bit vectors are considered to be part of the augmented verifying key.

5.6 Referencing cells in other rows

In some implementations (notably [27]), custom gates can refer to witness values in other rows, frequently those in neighboring rows. UniPlonk can be extended as described here to support such gates.

As an example, let $(w_{1,i})_{i=1}^n$ and $(w_{2,i})_{i=1}^n$ represent 2 witness columns, and consider a simple gate G defined by:

$$G(w_{1,i}, w_{2,i}, w_{1,i+1}) = w_{1,i} \times w_{2,i} - w_{1,i+1}$$

For any row i where G is activates, $G(w_{1,i}, w_{2,i}, w_{1,i+1}) = 0$ enforces the condition: $w_{1,i+1} = w_{1,i} \times w_{2,i}$.

In the arithmetic constraint polynomial, where $(w_{1,i})_{i=1}^n$ and $(w_{2,i})_{i=1}^n$ are given as polynomials, this is expressed as

$$G(w_1(X), w_2(X), w_1(\omega X))$$

since $w_1(\omega X) = w_1(\omega^{i+1})$ at each value ω^i in the domain.

By extending the proof π to include the evaluation $\bar{w}_{1,\omega} = w_1(\omega\mathfrak{z})$ and requiring the prover to include this evaluation in the batched group evaluation witness $[W_{\mathfrak{z}\omega}]_1$, the verifier can confirm this evaluation and use it as input to G when reconstructing the commitment to the linearization polynomial.

Note that the batch opening proof already contains evaluations at $\mathfrak{z}\omega$, so references to elements in the next row can be handled relatively easily by modifying the elements $[W_{\mathfrak{z}\omega}]_1$. References to other rows may require more elements to be passed in the proof.

6 Cost Analysis

We compare the cost of UniPlonk's Universal Verifier (UV) to the cost of a circuit-specific verifier. This cost analysis is particularly interesting in the context of proof recursion since it determines how much larger a Universal Verifier Circuit (UVC) would be than a circuit-specific verifier circuit. This difference represents the cost of universality. In this section, we simplify the analysis by assuming that only binary selector polynomials are used (as opposed to Section 5.5) and gates do not refer to adjacent rows (as opposed to Section 5.6).

The main factors that determine the UV's cost for a given configuration $\mathscr{U}$ are

- Size of global gate set: $L = |\mathscr{U}.\mathscr{G}|$

  The computation of $[r_3]_1$ in Step 8 consists of L additions and scalar multiplications in $\mathbb{G}_1$, as well as L conditional select statements.

- Number of field operations to evaluate constraint polynomials: $N_{\mathbb{F}}$

  Step 8 also requires some number of field operations to evaluate the constraint polynomials, which depends on the specific form of each gate's polynomial. We denote by $n_{\mathbb{F}}(G_i)$ the number of field multiplications to evaluate the constraint(s) imposed by gate G_i. We have then $N_{\mathbb{F}} = \sum_{G_i \in \mathscr{G}} n_{\mathbb{F}}(G_i)$ field multiplications as well. (We omit counting field additions for the sake of brevity.)

- Maximum number M of witness variables required by any gate $G \in \mathscr{U}.\mathscr{G}$

  The computations of a and r_1 in Step 7 require M and $M+1$ field multiplications, respectively. Each requires M conditional select statements.

  The computation of $[F]_1$ in Step 9 requires $2M-1$ additions and scalar multiplications in $\mathbb{G}_1$, $2M-1$ field multiplications, and $4M-2$ conditional select statements.

  The computation of $[E]_1$ in Step 22 requires $2M-1$ field multiplications and additions, as well as $4M-2$ conditional select statements.

- Maximum number R of constant variables required by any gate $G \in \mathscr{U}.\mathscr{G}$

  The computation of $[F]_1$ in Step 9 requires R additions and scalar multiplications in $\mathbb{G}_1$ and field multiplications, as well as R conditional select statements.

  The computation of $[E]_1$ in Step 22 requires R field multiplications and additions, as well as R conditional select statements.

- Degree factor D of the quotient polynomial

  The computation of r_4 in Step 7 requires $D-1$ additions and scalar multiplications in $\mathbb{G}_1$, $D-1$ field multiplications, and $D-1$ conditional select statements.

- Maximum number of constraints: $\mathscr{U}.N = 2^K$

  As in the cost analysis of vanilla UniPlonk (Section 3.1.3) the UV requires K field multiplications and two multiplexers to compute the correct values of $\mathfrak{z}^n$ and $\omega^{N/n}$.

Letting $\mathbb{G}$ denote the number of constraints required to represent scalar multiplication plus addition in $\mathbb{G}_1$, $\mathbb{F}$ denote the number of constraints for field multiplication, and $\mathbb{S}$ the number of constraints for a conditional select statement, the cost of the UV is

$$\begin{aligned}\text{cost}_{\text{UV}}(L, M, R, D, K, N_{\mathbb{F}}) = \text{constant} &+ (L + 2M + R + D)\mathbb{G} \\ &+ (6M + R + D + K + N_{\mathbb{F}})\mathbb{F} \\ &+ (L + 10M + 2R + D - 5)\mathbb{S} \\ &+ 2 \text{ multiplex}\end{aligned} \tag{32}$$

where the constant term represents costs that are independent of the configuration and would also be present for a circuit-specific verifier (challenge hashes, pairing, and miscellaneous arithmetic). This term is irrelevant for the sake of computing the marginal cost of UniPlonk.

The circuit-specific verifier's cost can is similar to the expression in Eq. (32) with $(L, M, R, D, K, N_{\mathbb{F}})$ equal to the corresponding values used by the circuit: $(l, m, r, d, k, n_{\mathbb{F}})$. The difference is that the circuit-specific verifier would not use any conditional select statements or multiplexing.

Therefore the marginal cost of using the UniPlonk verifier with configuration $\mathscr{U}$ for compatible circuit C is

$$\begin{aligned}\text{marginal cost}_{\text{UV}}(C, \mathscr{U}) = &(\Delta L + 2\Delta M + \Delta R + \Delta D)\mathbb{G} \\ &+ (6\Delta M + \Delta R + \Delta D + \Delta k + \Delta n_{\mathbb{F}})\mathbb{F} \\ &+ (L + 10M + 2R + D - 5)\mathbb{S} \\ &+ 2 \text{ multiplex}\end{aligned}$$

where L, M, R, D, K are set by $\mathscr{U}$ and ΔL denotes the difference between L and the number of custom gates used by C, ΔM is the difference between M and the number of witness polynomials used by C, etc.

Observe that for a circuit C which uses all gates of the global gate set $\mathscr{U}.\mathscr{G}$ we have $\Delta L = \Delta M = \Delta R = \Delta D = \Delta n_{\mathbb{F}} = 0$. Therefore these circuits pay a marginal cost of only

$$\Delta k\mathbb{F} + (L + 10M + 2R + D - 5)\mathbb{S} + 2\text{multiplex}$$

The cost $\mathbb{S}$ of a binary selector is only a single constraint in common arithmetizations, since $\mathsf{select}(b, A, B) = b * A + (1 - b) * B$. The number of constraints to express the multiplexer which selects from among K values is also low, since this may be expressed as a dot product of vectors of length K. This marginal cost is therefore small compared to the overall verifier cost for circuits that make use of all custom gates supported by $\mathscr{U}$.

For "simpler" circuits, meaning those which do not use all gates of $\mathscr{U}.\mathscr{G}$, the marginal cost of using UniPlonk becomes more significant. Firstly, the amount of extra elliptic curve scalar multiplication is proportional to ΔL. Moreover, if this simple circuit can be expressed using $m < M$ witness variables and $r < R$ constant variables, then we again incur an extra $2\Delta M + \Delta R$ elliptic curve scalar multiplications. Similarly, if the simpler circuit has a lower degree factor than D. Additional field multiplications are also incurred proportional to $\Delta M, \Delta R, \Delta D, \Delta n_{\mathbb{F}}$.

Therefore the UVC will be of greatest practical use in a context where the circuits of interest use a similar set of custom gates. Thinking of the UVC as a tool for proof aggregation, a critical design choice will be the global gate set $\mathscr{G}$ to use for the configuration. Having many gates in $\mathscr{G}$ increases the diversity of circuits which can be verified by the same UVC, but increases the UVC's size linearly with the size of $\mathscr{G}$ and the number of variables used by these gates.

7 Knowledge Soundness

UniPlonk inherits its knowledge soundness from the Plonk protocol. Informally, this is because a UniPlonk proof contains within it a Plonk proof and the UniPlonk verifier accepts a proof if and only if the Plonk verifier does. Formally, we can transform a knowledge extractor for the Plonk protocol into a knowledge extractor for UniPlonk with identical knowledge soundness errors.

Based on the definition of Knowledge Soundness given in [13], we define the knowledge soundness game for UniPlonk as: For a fixed $\mathscr{U}$-compatible circuit C representing a relation $\mathcal{R} \subset \mathbb{F}^{\ell} \times \mathbb{F}^{n-\ell}$

1. An algebraic adversary $\mathcal{A}$ chooses input $\mathsf{PI} = (w_i)_{i=1}^{\ell}$ and produces a UniPlonk proof π of C. $\mathcal{A}$ returns π and PI.
2. The knowledge extractor E with access to all of $\mathcal{A}$'s messages during the protocol outputs a witness w.
3. $\mathcal{A}$ wins if
   (a) $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk, \pi, [\mathsf{PI}]_1) = \mathtt{true}$, where vk is the augmented verifying key for C and $[\mathsf{PI}]_1$ is the commitment to PI.
   (b) $(\mathsf{PI}, w) \notin \mathcal{R}$

A protocol has *knowledge soundness error* of ϵ if there exists an extractor E such that for any algebraic adversary, the probability of winning the above game is at most ϵ.

Lemma 2. *Let $\mathcal{R}$ be a relation represented by Plonk circuit C. Let $\mathscr{U}$ be a configuration such that C is $\mathscr{U}$-compatible. Then the UniPlonk protocol for configuration $\mathscr{U}$ applied to C has knowledge soundness with error equal to that of the Plonk protocol.*

Proof. We construct an extractor E for the UniPlonk protocol for the relation $\mathcal{R}$ in the obvious way from the extractor $E^{\mathcal{P}}$ for Plonk with the same relation. E operates as follows:

- Given a UniPlonk proof π and public inputs PI from an adversary $\mathcal{A}$, use the inverse of the mapping in Lemma 1 to compute a Plonk proof $\pi^{\mathcal{P}}$.
- Playing the role of the adversary in the Plonk protocol knowledge soundness game send $\pi^{\mathcal{P}}$ and PI to $E^{\mathcal{P}}$ (where $E^{\mathcal{P}}$ has access to all messages generated by E and thereby $\mathcal{A}$). Note that, by the form of the mapping, E is an algebraic adversary, and let w be the witness returned by $E^{\mathcal{P}}$.
- Return w.

Let ε be the knowledge soundness error of Plonk for $\mathcal{R}$. The probability that $\mathcal{A}$ wins the above game against E, namely that

$$\Big(\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk, \pi, [PI]_1) = \mathtt{true}\Big) \wedge \Big((\mathsf{PI}, w) \notin \mathcal{R}\Big)$$

By Lemma 1, this holds iff

$$\Big(\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(\mathsf{PI}, \pi^{\mathcal{P}}) = \mathtt{true}\Big) \wedge \Big((\mathsf{PI}, w) \notin \mathcal{R}\Big)$$

which, by assumption, has probability ε.

The above shows that the soundness error for $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}$ is *at most* ε. The equality follows by repeating this argument with Plonk and UniPlonk reversed to show that the knowledge soundness error of Plonk is bounded above by that of UniPlonk (because any extractor for UniPlonk can be used to construct an extractor for Plonk). Thus the two protocols have equal knowledge soundness error. □

8 Future Works

We plan to add the support of lookup arguments to UniPlonk, notably, this could combined with effort of supporting lookup argument in NIFS [8, 35, 21] to add lookup argument support to Universal ZK-ZK Aggregation.

Acknowledgements

We would like to specially thank Srinath Setty for the detailed discussion on folding schemes and proof aggregation, as well as David Wong and Zhenfei Zhang for their feedback on an early version of the text.

References

[1] Miguel Ambrona, Marc Beunardeau, Anne-Laure Schmitt, and Raphaël R. Toledo. aplonk : Aggregated plonk from multi-polynomial commitment schemes. Cryptology ePrint Archive, Paper 2022/1352, 2022. https://eprint.iacr.org/2022/1352.

[2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104. ACM, 2017.

[3] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *EUROCRYPT (1)*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019.

[4] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *TCC (B2)*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016.

[5] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349. ACM, 2012.

[6] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *CRYPTO (1)*, volume 12825 of *Lecture Notes in Computer Science*, pages 649–680. Springer, 2021.

[7] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. *IACR Cryptol. ePrint Arch.*, page 499, 2020.

[8] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. Cryptology ePrint Archive, Paper 2023/620, 2023. https://eprint.iacr.org/2023/620.

[9] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. Cryptology ePrint Archive, Paper 2020/1618, 2020. https://eprint.iacr.org/2020/1618.

[10] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT (2)*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2023.

[11] Ariel Gabizon and Zachary J. Williamson. The turbo-plonk program syntax for specifying snark programs. ZKProof Workshop 3, Preprint, 2019. https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf.

[12] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Paper 2020/315, 2020. https://eprint.iacr.org/2020/315.

[13] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. https://eprint.iacr.org/2019/953.

[14] Nicolas Gailly, Mary Maller, and Anca Nitulescu. Snarkpack: Practical SNARK aggregation. In *Financial Cryptography*, volume 13411 of *Lecture Notes in Computer Science*, pages 203–229. Springer, 2022.

[15] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[16] Daira Hopwood. Selector combining, 2022. https://hackmd.io/@daira/SkjDVkLCd.

[17] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[18] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Paper 2022/1758, 2022. https://eprint.iacr.org/2022/1758.

[19] Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. Cryptology ePrint Archive, Paper 2023/573, 2023. https://eprint.iacr.org/2023/573.

[20] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *CRYPTO (4)*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388. Springer, 2022.

[21] levs57. Folding endgame. https://zkresear.ch/t/folding-endgame/106, 2023.

[22] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *CCS*, pages 2111–2128. ACM, 2019.

[23] Nicolas Mohnblatt. Sangria: A folding scheme for plonk. https://github.com/geometryresearch/technical_notes/blob/main/sangria_folding_plonk.pdf, 2023.

[24] Luke Pearson, Joshua Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. Plonkup: Reconciling plonk with plookup. Cryptology ePrint Archive, Paper 2022/086, 2022. https://eprint.iacr.org/2022/086.

[25] Polygon. Polygon zkevm. https://github.com/0xpolygonhermez, 2023.

[26] Mir Protocol. Plonky2. https://github.com/mir-protocol/plonky2, 2023.

[27] ZCash Protocol. Halo2. https://github.com/zcash/halo2, 2023.

[28] Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *CRYPTO (3)*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020.

[29] Scroll Tech. Scroll tech. https://github.com/scroll-tech, 2023.

[30] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[31] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society, 2018.

[32] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO (3)*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764. Springer, 2019.

[33] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *CRYPTO (4)*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328. Springer, 2022.

[34] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *IEEE Symposium on Security and Privacy*, pages 859–876. IEEE, 2020.

[35] Yan X Zhang and Aard Vark. Origami – a folding scheme for halo2 lookups. https://hackmd.io/@aardvark/rkHqa3NZ2#Comparison-with-Sangria, 2023.

A Full Protocol using Custom Gates

We describe the full protocol for custom gates, introduced in Section 5, noting that it can be extended as described in Sections 5.5 and 5.6 to support optimized selector polynomials and gates that reference other rows of the witness.

A.1 $\mathcal{U}\mathrm{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Setup}(\mathscr{U}, C)$

For the circuit C using gates $G_1, \ldots, G_l \subset \mathscr{G}$, define bit vectors:

- *Gate* bits $(b_1^{(g)}, \ldots, b_L^{(g)})$, where $b_i^{(g)} = 1$ if and only if the gate G_i^* is used by C.
- *Witness column* bits $(b_1^{(w)}, \ldots, b_M^{(w)})$ where $b_i^{(w)} = 1$ iff w_i^* is used by one of the gates in C.
- *Constant column* bits $(b_1^{(q)}, \ldots, b_R^{(q)})$ where $b_i^{(q)} = 1$ iff q_i^* is used by one of the gates in C.

Let $(pk^{\mathcal{P}}, vk^{\mathcal{P}})$ be the proving and verifier keys generated for C in the standard way, using pp, srs, ω, $k_1, \ldots, k_m$ etc from $\mathscr{U}$ to obtain

$$pk^{\mathcal{P}} = (S_1, \ldots, S_l, q_1, \ldots, q_r, \mathsf{s}_{\sigma 1}, \ldots, \mathsf{s}_{\sigma m})$$

and

$$vk^{\mathcal{P}} = ([S_1]_1, \ldots, [S_l]_1, [q_1]_1, \ldots, [q_r]_1, [\mathsf{s}_{\sigma 1}]_1, \ldots, [\mathsf{s}_{\sigma m}]_1)$$

To facilitate the universal verifier, we append the following data to $vk^{\mathcal{P}}$ to create and augmented vk:

- Bit fields $(b_i^{(g)})_{i=1}^L$, $(b_i^{(w)})_{i=1}^M$ and $(b_i^{(q)})_{i=1}^R$.
- "Padded" constant column commitments $([q'_1]_1, \ldots, [q'_R]_1)$ where, for each $i = 1, \ldots, r$ with $j = \eta_q(i)$, we set $[q'_j]_1 = [q_i]_1$, and set $[q'_j]_1 = [0]_1$ for all other j:

  $([q'_1]_1, \ldots, [q'_R]_1) \in \mathbb{G}_1^M \leftarrow ([0]_1, \ldots, [0]_1)$;
  for $i \leftarrow 1$ **to** r **do**
  $\quad j \leftarrow \eta_q(i)$;
  $\quad [q'_j]_1 \leftarrow [q_i]_1$;
  end

- "Padded" permutation polynomial commitments $([\mathsf{s}'_{\sigma 1}]_1, \ldots, [\mathsf{s}'_{\sigma M}])$ where, for each $i = 1, \ldots, m$ with $j = \eta_w(i)$, we set $[\mathsf{s}'_{\sigma j}]_1 \leftarrow [\mathsf{s}_{\sigma i}]_1$ and $[\mathsf{s}'_{\sigma j}]_1 \leftarrow [0]_1$ for all other j:

  $([\mathsf{s}'_1]_1, \ldots, [\mathsf{s}'_M]_1) \in \mathbb{G}_1^M \leftarrow ([0]_1, \ldots, [0]_1)$;
  for $i \leftarrow 1$ **to** m **do**
  $\quad j \leftarrow \eta_w(i)$;
  $\quad [\mathsf{s}'_j]_1 \leftarrow [\mathsf{s}_i]_1$;
  end

- "Padded" selector polynomial commitment set $([S'_1], \ldots, [S'_L])$ where, for each $i = 1, \ldots, l$ with $j = \eta_g(i)$, we set $[S'_j]_1 \leftarrow [S_i]_1$ and $[S'_j]_1 \leftarrow [0]_1$ for all other j:

  $([S'_1]_1, \ldots, [S'_L]_1) \in \mathbb{G}_1^L \leftarrow ([0]_1, \ldots, [0]_1)$;
  for $i \leftarrow 1$ **to** l **do**
  $\quad j \leftarrow \eta_g(i)$;
  $\quad [S'_j]_1 \leftarrow [S_i]_1$;
  end

The augmented verification key vk then has the following form, with size and format dependent only on $\mathscr{U}$, and independent of the specific circuit C:

$$\begin{gathered} vk = ([S'_1]_1, \ldots, [S'_L]_1, [q'_1]_1, \ldots, [q'_R]_1, \\ [\mathsf{s}'_{\sigma 1}]_1, \ldots, [\mathsf{s}'_{\sigma M}]_1, k'_1, \ldots, k'_M, \\ (b_1^{(g)}, \ldots, b_L^{(g)}), (b_1^{(w)}, \ldots, b_M^{(w)}), (b_1^{(q)}, \ldots, b_R^{(q)}), (b_1^{(t)}, \ldots, b_D^{(t)}), n) \end{gathered}$$

and the final output of Setup is the pair $(pk = pk^{\mathcal{P}}, vk)$.

Note that the mappings η_g, η_w and η_q, from indices in C to indices in $\mathscr{U}$ can be derived from vk in a deterministic way.

A.2 $\mathcal{U}\mathrm{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Prove}((w_i)_{i=1}^n)$

The Plonk prover for the C can be used, as described in Section 2.3, where srs and all constants (as well as column and gate orderings) are compatible with $\mathscr{U}$.

A.3 $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Uniformize}(vk, \pi^{\mathcal{P}}, \mathsf{PI})$

1. Parse $\pi^{\mathcal{P}}$ as

$$([w_1]_1, \ldots, [w_m]_1, [z]_1, [t_1]_1, \ldots, [t_d]_1, [W_{\mathfrak{z}}^{\mathcal{P}}]_1, [W_{\mathfrak{z}\omega}]_1, \bar{w}_1, \ldots, \bar{w}_m, \bar{q}_1, \ldots, \bar{q}_r, \bar{\mathsf{s}}_{\sigma 2}, \ldots \bar{\mathsf{s}}_{\sigma m}, \bar{z}_\omega)$$

2. Derive the mappings $\eta_g : [l] \to [L]$, $\eta_w : [m] \to [M]$ and $\eta_q : [r] \to [R]$ from the bit vectors in vk'.

3. Create padded witness commitments and evaluations:
   $([w'_1]_1, \ldots, [w'_M]_1) \in \mathbb{G}_1^M \leftarrow ([0]_1, \ldots, [0]_1)$;
   $(\bar{w}'_1, \ldots, \bar{w}'_M) \in \mathbb{F}^M \leftarrow (0, \ldots, 0) \in \mathbb{F}^M$;
   for $i \leftarrow 1$ **to** m **do**
   $j \leftarrow \eta_w(i)$;
   $[w'_j]_1 \leftarrow [w_i]_1$;
   $\bar{w}'_j \leftarrow \bar{w}_i$;
   end

4. Create padded permutation polynomial evaluations:
   $(\bar{\mathsf{s}}'_{\sigma 2}, \ldots, \bar{\mathsf{s}}'_{\sigma M}) \in \mathbb{F}^{M-1} \leftarrow (0, \ldots, 0)$;
   for $i \leftarrow 2$ **to** m **do**
   $j \leftarrow \eta_w(i)$;
   $\bar{\mathsf{s}}'_{\sigma j} \leftarrow \bar{\mathsf{s}}_{\sigma i}$;
   end

5. Create padded constant polynomial evaluations:
   $(\bar{q}'_1, \ldots, \bar{q}'_R) \in \mathbb{F}^R \leftarrow (0, \ldots, 0) \in \mathbb{F}^R$;
   for $i \leftarrow 1$ **to** r **do**
   $j \leftarrow \eta_q(i)$;
   $\bar{q}'_j \leftarrow \bar{q}_i$;
   end

6. Pad $[t_1]_1, \ldots, [t_d]_1$ with entries $[0]_1$ with entries until it is of the length D required by $\mathscr{U}$. Let $\{t'_i\}_{i=1}^D$ be the resulting vector.

7. Compute the polynomial
$$\Delta W_{\mathfrak{z}}(X) = (\mathsf{PI}(X) - \mathsf{PI}(\mathfrak{z}))/(X - \mathfrak{z})$$
   where $\mathsf{PI}(X) = \sum_{i=1}^{\ell} w_i L_i(X)$ and its commitment $[\Delta W_{\mathfrak{z}}]_1$. The new opening proof commitment is then computed as
$$[W_{\mathfrak{z}}]_1 = [W_{\mathfrak{z}}^{\mathcal{P}}]_1 + \alpha^2 [\Delta W_{\mathfrak{z}}]_1$$
   (See Section 5.2 for a full explanation).

8. Return:
$$\pi = ([w'_1]_1, \ldots, [w'_M]_1, [z]_1, [t'_1]_1, \ldots, [t'_D]_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \bar{w}'_1, \ldots, \bar{w}'_M, \bar{q}'_1, \ldots, \bar{q}'_R, \bar{\mathsf{s}}'_{\sigma 2}, \ldots \bar{\mathsf{s}}'_{\sigma M}, \bar{z}_\omega)$$

A.4 $\mathcal{U}\mathfrak{ni}\mathcal{P}\mathfrak{lon}\mathcal{K}.\mathsf{Verify}(vk, \pi, [\mathsf{PI}]_1)$

Note that $[PI]_1$ is computed by the caller using the public inputs $(w_i)_{i=1}^{\ell}$.

The verifier parses the *uniformized* proof:

$$\pi = ([w'_1]_1, \ldots, [w'_M]_1, [z]_1, [t'_1]_1, \ldots, [t'_D]_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \bar{w}'_1, \ldots, \bar{w}'_M, \bar{q}'_1, \ldots, \bar{q}'_R, \bar{\mathsf{s}}'_{\sigma 2}, \ldots \bar{\mathsf{s}}'_{\sigma M}, \bar{z}_\omega)$$

and verifier key:

$$vk = ([S'_1]_1, \ldots, [S'_L]_1, [q'_1]_1, \ldots, [q'_R]_1, [\mathsf{s}'_{\sigma 1}]_1, \ldots, [\mathsf{s}'_{\sigma M}]_1, k'_1, \ldots, k'_M, (b_1^{(g)}, \ldots, b_L^{(g)}), (b_1^{(w)}, \ldots, b_M^{(w)}), (b_1^{(q)}, \ldots, b_R^{(q)}), (b_1^{(t)}, \ldots, b_D^{(t)}), n = (b_1^{(n)}, \ldots, b_K^{(n)}),)$$

1. Verify the well-formedness of $[t'_1]_1, \ldots, [t'_d]_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1$. Validate all $[w'_i]_1$ and $[q'_i]_1$ values. For i with $b_i^{(w)} = 0$, check that $[w'_i]_1$, $\bar{w}'_i$ and $\bar{\mathsf{s}}'_{\sigma i}$ are equal to the expected default values from Appendix A.3. Similarly, for all $[t'_i]_1$ with $b_i^{(t)} = 0$.

2. Validate $(\bar{w}'_1, \ldots, \bar{w}'_M, \bar{q}'_1, \ldots, \bar{q}'_Q, \bar{\mathsf{s}}'_{\sigma 2}, \ldots \bar{\mathsf{s}}'_{\sigma M}) \in \mathbb{F}^{2M+Q}$.

3. Check that $[\mathsf{PI}]_1$ is a $\mathbb{G}_1$ element. Let $\mathsf{transcript}_0 = vk^{\mathcal{P}} \parallel [PI]_1$, ensuring that only elements of $\pi^{\mathcal{P}}$ are added to the transcript as described in Section 5.4.

4. Compute challenges $\beta, \gamma, \alpha, \mathfrak{z}, v, u \in \mathbb{F}$.

5. Compute $\mathfrak{z}^n$ as $\langle (b_1^{(n)}, \ldots, b_K^{(n)}), (\mathfrak{z}, \mathfrak{z}^2, \ldots \mathfrak{z}^{2^K}) \rangle$, and $Z_H(\mathfrak{z}) = \mathfrak{z}^n - 1$.

6. Compute $\omega_k = \langle (b_1^{(n)}, \ldots, b_K^{(n)}), (1, \omega^{2^K}, \omega^{2^{K-1}}, \ldots, \omega^2, \omega) \rangle$, and the evaluation of the first Lagrange polynomial $L_1(\mathfrak{z}) = \frac{\omega_k(\mathfrak{z}^n - 1)}{n(\mathfrak{z} - \omega_k)}$.

7. Split the calculation of $[r]_1$ (see Eq. (18), (19)) as follows:

$$[r]_1 = r_0 \cdot [1]_1 + r_1[z]_1 + r_2[\mathsf{s}_{\sigma 1}]_1 + [r_3]_1 - [r_4]_1$$

   and compute the terms:

$$\begin{aligned}
a &:= \bar{z}_\omega \prod_{i=2}^{M} \mathsf{select}\left(b_i^{(w)}, (\bar{w}'_i + \beta \bar{\mathsf{s}}_{\sigma i} + \gamma), 1\right) \\
r_0 &:= -a(\bar{w}'_1 + \gamma) - \alpha L_1(\mathfrak{z}) \\
r_1 &:= \prod_{i=1}^{M} \mathsf{select}\left(b_i^{(w)}, (\bar{w}'_i + \beta k'_i \mathfrak{z} + \gamma), 1\right) + \alpha L_1(\mathfrak{z}) \\
r_2 &:= a\beta
\end{aligned}$$

8. Compute $[r_3]_1$:

   $[r_3]_1 \leftarrow [0]_1$;
   $\alpha' \leftarrow \alpha$;
   for $i \leftarrow 1$ **to** L **do**
   $\quad \alpha' \leftarrow \alpha' \times \mathsf{select}\left(b_i^{(g)}, \alpha, 1\right)$;
   $\quad [r_3]_1 \leftarrow [r_3]_1 + \mathsf{select}\left(b_i^{(g)}, \alpha' G_i^*(\bar{w}'_1, \ldots, \bar{w}'_M, \bar{q}'_1, \ldots \bar{q}'_R) \cdot [S'_i]_1, [0]_1\right)$;
   end
   return $[r_3]_1$

   and $[r_4]_1$:

   $[r_4]_1 \leftarrow [t'_1]_1$;
   $\mathfrak{z}' \leftarrow 1$;
   for $i \leftarrow 2$ **to** D **do**
   $\quad \mathfrak{z}' \leftarrow \mathfrak{z}' \times \mathfrak{z}^n$;
   $\quad [r_4]_1 \leftarrow [r_4]_1 + \mathsf{select}\left(b_i^{(t)}, \mathfrak{z}' \cdot [t'_i]_1, [0]_1\right)$;
   end
   $[r_4]_1 \leftarrow Z_H(\mathfrak{z}) \cdot [t'_1]_1$;
   return $[r_4]_1$

9. Compute the full batched polynomial commitment, Eq. (23):

$$\begin{aligned}
[F]_1 :=& (r_1 + u) \cdot [z]_1 + r_2[\mathsf{s}_{\sigma 1}]_1 + [r_3]_1 - [r_4]_1 + \alpha^2[\mathsf{PI}]_1 \\
& v[w_1]_1 + \ldots + v^m[w_m]_1 + \\
& v^{m+1}[q_1] + \ldots + v^{m+r}[q_r]_1 + \\
& v^{m+r+1}[\mathsf{s}_{\sigma 2}]_1 + \ldots + v^{2m+r-1}[\mathsf{s}_{\sigma m}]_1
\end{aligned}$$

as follows:

$[F]_1 \leftarrow (r_1 + u) \cdot [z]_1 + r_2[\mathsf{s}_{\sigma 1}]_1 + [r_3]_1 - [r_4]_1$;
$v' \leftarrow 1$;
for $i \leftarrow 1$ **to** M **do**
  $v' \leftarrow v' \times \mathsf{select}\left(b_i^{(w)}, v, 1\right)$;
  $[F]_1 \leftarrow [F]_1 + \mathsf{select}\left(b_i^{(w)}, v' \cdot [w_i']_1, [0]_1\right)$
end
for $i \leftarrow 1$ **to** R **do**
  $v' \leftarrow v' \times \mathsf{select}\left(b_i^{(q)}, v, 1\right)$;
  $[F]_1 \leftarrow [F]_1 + \mathsf{select}\left(b_i^{(q)}, v' \cdot [q_i']_1, [0]_1\right)$
end
for $i \leftarrow 2$ **to** M **do**
  $v' \leftarrow v' \times \mathsf{select}\left(b_i^{(w)}, v, 1\right)$;
  $[F]_1 \leftarrow [F]_1 + \mathsf{select}\left(b_i^{(w)}, v' \cdot [\mathsf{s}_{\sigma i}']_1, [0]_1\right)$
end

10. Compute group-encoded batch evaluation, Eq. (22):

$$[E]_1 := \begin{pmatrix} -r_0 + u\bar{z}_\omega + \\ v\bar{w}_1 + \ldots + v^m \bar{w}_m + \\ v^{m+1}\bar{q}_1 + \ldots + v^{m+r}\bar{q}_r + \\ v^{m+r+1}\bar{\mathsf{s}}_{\sigma 2} + \ldots + v^{2m+r-1}\bar{\mathsf{s}}_{\sigma m} \end{pmatrix} \cdot [1]_1$$

as follows:

$E \leftarrow u\bar{z}_\omega - r_0$;
$v' \leftarrow 1$;
for $i \leftarrow 1$ **to** M **do**
  $v' \leftarrow v' \times \mathsf{select}\left(b_i^{(w)}, v, 1\right)$;
  $E \leftarrow E + \mathsf{select}\left(b_i^{(w)}, v'\bar{w}_i', 0\right)$
end
for $i \leftarrow 1$ **to** R **do**
  $v' \leftarrow v' \times \mathsf{select}\left(b_i^{(q)}, v, 1\right)$;
  $E \leftarrow E + \mathsf{select}\left(b_i^{(q)}, v'\bar{q}_i', 0\right)$
end
for $i \leftarrow 2$ **to** M **do**
  $v' \leftarrow v' \times \mathsf{select}\left(b_i^{(w)}, v, 1\right)$;
  $E \leftarrow E + \mathsf{select}\left(b_i^{(w)}, v'\bar{\mathsf{s}}_{\sigma i}', 0\right)$
end

11. Batch validate all evaluations:

$$e([W_{\mathfrak{z}}]_1 + u \cdot [W_{\mathfrak{z}\omega}]_1, [x]_2) \stackrel{?}{=} e(\mathfrak{z} \cdot [W_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega \cdot [W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2)$$