Origami: Fold a Plonk for Ethereum's VDF

Zhenfei Zhang[1,2]

[1] Ethereum Foundation
[2] Scroll Tech

Abstract. We present Origami verifiable delay function, build from the MinRoot hash and our dedicated plonk proof system that utilizes a tailored custom gate and a folding scheme. MinRoot VDF is the leading candidate for Ethereum adoption. For N iterations of MinRoot hash function, the overall cost of Origami is $N+o(N)$ group operations; improving the previous best known result of $6N$ from a Nova based solution. The proof size is $128k + 224$ bytes if we fold the proofs for k times; and may be further reduce to around 960 bytes, regardless of k, via a standard recursive prover.

1 Introduction

Verifiable delay function (VDF) [BBBF18,KMT22] is a function with a guarantee to be slow to compute. Such a function is very useful for eliminating bias in RANDAO [ran], the source of randomness for the consensus protocol behind the Ethereum beacon chain. In Ethereum's consensus protocol [bea], for a given epoch, each validator contributes its share of randomness via a commit-then-reveal mechanism. Without a VDF, the last validator has the option of either open its randomness share or not, and therefore influences the final output of RANDAO. To mitigate this issue, one may pass the RANDAO output to a VDF, during whose evaluation time, all validators must have already decided whether to reveal their committed value or not, before the VDF output is generated. As such, the unfair advantage of the last validator is removed.

In practice, a VDF needs to be both slow to compute and efficient to verify. Such a VDF can be constructed via the following paradigm:

- Build a long enough hash chain from a parallelization-resistant hash function;
- Generate a succinct (not necessarily zero knowledge) proof, proving the output is the correct evaluation of the hash chain and the given the input.

By design, the hash chain is guaranteed to be slow to compute; in the meantime, a verifier is convinced of the correctness of VDF via the succinct proof.

A crucial point here is that the proof does not take too long to generate. An ideal scenario, as we shall see in the MinRoot VDF design momentarily, is the proving speed is faster than the hash chain itself, and can be pipelined with the hash chain.

The MinRoot VDF [KMT22] is expected to be deployed on Ethereum, contributing to the security of billions on-chain assets. This VDF consists of a chain

of `MinRoot` hashes, and a Nova [KST22] zero knowledge proof (ZKP). Nova is a folding scheme: one can generate multiple instances of a same circuit and fold them into a single one. When applied to the hash chain, one can break up the whole chain into k sub-chains, and start to prove any sub-chain as soon as this sub-chain is been evaluated; and continuously fold the fresh generated instances into the existing instance. Nova's proof size is somewhat large. In [Set], Spartan [Set20] is used to generated a recursive proof: a new proof π_2 asserting that the original proof, π_1, is correct. Since π_2 can be significantly smaller than π_1, this effectively trades the proof size with prover's work.

A reference implementation of the `MinRoot` VDF is available in [Set]. For N iterations of `MinRoot` hash, the proof generation time is dominated by $6N$ group multiplications. This means the proof itself actually requires more computation than the hash chain itself: each hash iteration requires a field exponentiation for the hash, and 6 group multiplications for the proof. Usually a group multiplication is on the order of a few thousands of field multiplications. The reason that the VDF paradigm still works is that the proof is generated along side the hash chain almost simultaneously thanks to the folding feature; and the proof generation itself can be paralleled.

Relaxed rank one constraint system (R1CS) is a popular circuit representations for ZKP statements. Nova uses a relaxed R1CS to support its folding mechanism. The other popular solution in the literature is Plonkish arithmetization [GWC19,BGH19], which allows for more flexible custom gates. It has been shown in [XCZ+22,CBBZ22] that with tailored custom gates, Plonkish arithmetization can be up to 30 times more expressive than R1CS for some of the applications. That is, for a same application, it requires less constraints for Plonk to present the statement than that does for R1CS. The down side is that for a same number of constraints, Plonk family of prove systems usually require more work to generate a proof.

1.1 Our result

We present `Origami` verifiable delay function, build from the `MinRoot` hash and our customized plonk prover.

As alluded earlier, we investigate the direction of representing `MinRoot` hash circuit in Plonkish arithmetization rather than R1CS. To fully harvest the power of Plonkish arithmetization, we design a tailored custom gate for `MinRoot` hash that is capable of proving a `MinRoot` iteration with a single constraint; in comparison, if we were to use the vanilla plonk gate [GWC19] as in Equation 1, or the aforementioned R1CS presentation, we will need 3 constraints per iteration. Our custom gate has only one witness per constraint and does not incur any cost for selectors. It also does not require any copy constraints or permutation constraints. We will explain these notions in the next section. Naively applying our method, in conjunction with popular polynomial commitment scheme, results into a prover that requires $6N$ group multiplications (matching Nova's performance) and a constant number of FFTs of dimension $4N$. This is already

an order of magnitude better than a solution from vanilla plonk; but is not significantly different from Nova based solution.

Next, we show that our custom gate is ideal for folding. Folding allows for aggregating instances. It is commonly used for R1CS systems such as Nova and SuperNova [KS22]. For Plonkish arithmetization, the only known result is [Moh23]. As pointed out by [Moh23], this solution does not scale well with custom gates of degrees beyond 2. Our custom gate's degree is parameterized by a constant α, which is the "root" of the `MinRoot` hash. Concretely $\alpha = 5$ for the curve that we are using. We design a folding scheme for our custom gate. For an N iteration of `MinRoot` hash, the folding scheme reduces the prover's work from $6N$ to $N + o(N)$ group multiplications. Overall we improve the previous best known result by approximately 6 times.

The proof size for `Origami` is $128k + 224$ bytes if we fold the proof for k times. Unlike Nova, in our scheme, the size of the proof increase linearly with the number of folds, and for each fold, the verifier will need to do some extra work of polynomial evaluations. The prover size can be further reduced to about 960 bytes using a recursive circuit such as [Esp,XCZ+22].

Getting back to the VDF paradigm, our `Origami` scheme has no effect on the security or the running time of the `MinRoot` VDF. Both of the aspects are determined by the hash chain rather than the succinct proof. The proof generation time, by design, should not exceed the hash chain time. From this perspective, the saving of `Origami` , compared with Nova based solution, is 5/6 of the computation; that means, generally speaking, to generate the proof for a same hash chain, our solution requires 1/6 of hardware. Considering that, for each block, i.e., every 10 seconds, all validators (and potentially all users, in the final picture of Ethereum) will need to run the VDF, reducing the computation cost for the ZKP component of the VDF, at the very least, has a substantial financial impact for the whole community.

2 Background

2.1 Notation

We make use of the following notations:

- α: 5, the root of the `MinRoot` function;
- β_t and β_e: random linear combination challenges;
- γ: the challenge in the IOP proof;
- ω: the root of unity;
- k: number of instances in the folding system;
- n: number of `MinRoot` iterations in each instances;
- $N := nk$: the total number of `MinRoot` iterations;
- r_i: witness re-randomizer for folding;
- e: error term in a relaxed plonk gate;
- Δ: additive expanding factor of the error term.

2.2 `MinRoot` hash

The $\texttt{MinRoot}_{N,p}(x_0, y_0)$ function, parameterized by an iteration parameter N and a prime field $\mathbb{F}_p$, takes two field elements (x_0, y_0) as inputs and returns two field elements (x_N, y_N). We omit the parameters N and p when there is no ambiguity. It is an iterative function defined by the following equation

$$\texttt{MinRoot} : (x_{i+1}, y_{i+1}) \leftarrow \left((x_i + y_i)^{1/\alpha}, x_i + i\right)$$

where α is the smallest integer that is co-prime with $p - 1$.

In practice, to incur a satisfied delay for RANDAO, we admit N of size up to 2^{40}. Table 1 lists the α parameter for popular choices of curves.

Curves	α	
BN254	5	The Ethereum curve
BabyJubjub	7	Embeded curve on BN254
Grumpkin	5	Embeded curve on BN254
Pasta	5	Cyclic curves
BLS12-381	5	
Jubjub	5	Embeded curve on BLS12-381

Table 1. α parameter for popular curves

2.3 Plonk arithmetization

The vanilla Plonk gate, as describe in [GWC19], is defined as

$$q_L w_0 + q_R w_1 + q_O w_2 + q_M w_0 w_1 + q_C = 0 \tag{1}$$

where w_0, w_1 and w_2 are the witnesses to either the original input or the intermediate values generated during computation; q_L, q_R, q_O, q_M and q_C are the so-called selectors, i.e., they are used to 1) enable basic logic or arithmetic gates such as add and mul gates, and 2) identify which witnesses are selected for the gate.

Since Equations 1 is a super set of addition and multiplication gates, any statement can be expressed as multiple rows of Equations 1. The variables in the equations effectively forms an 8-columns tall matrix where each row is a constraint. From there we need to further constrain that the witnesses in certain row are carried over to those of certain other rows. This is called copy constraint, and is useful to "glue" all the rows together.

We slices through this matrix as a collection of 8 column vectors, i.e., defining

$$\begin{aligned} \boldsymbol{w}_i &:= \langle w_{i,1}, \ldots, w_{i,N} \rangle && \forall i \in \{0, 1, 2\} \\ \boldsymbol{q}_i &:= \langle q_{i,1}, \ldots, q_{i,N} \rangle && \forall i \in \{L, R, M, C, O\} \end{aligned}$$

and then view those vectors as the evaluations of some polynomials (denoted by $w_i[X], q_i[X]$) in their FFT domain, i.e.,

$$w_i[X] := \texttt{IFFT}(\boldsymbol{w}_i) \qquad \forall i \in \{0, 1, 2\}$$
$$q_i[X] := \texttt{IFFT}(\boldsymbol{q}_i) \qquad \forall i \in \{L, R, M, C, O\}$$

Then we can define a polynomial $h[X]$ as

$$h[X] := q_L[X]w_0[X] + q_R[X]w_1[X] + q_O[X]w_2[X] + q_M[X]w_0[X]w_1[X] + q_C[X] \tag{2}$$

Notice that $h[X]$ is 0 at ω^i for $i \in [0, N)$ where ω is the root of unity, if the all constraints are satisfied, as $h[\omega^i]$ in fact is the result of evaluating Equation 1 with inputs from the i-th row of the matrix. This is equivalent to say that $t[X] := h[X]/(X^N - 1)$ is a low degree polynomial of degree $2n$.

The rest of the prove system, handwavingly speaking, is an interactive oracle proof, that proves $t[X]$ is a low degree polynomial. This proof, as well as the knowledge of $q_i[X]$, convinces the verifier the claimed relationship among all witnesses in $w_i[X]$.

2.4 Polynomial commitment scheme

The IOP protocol in the previous section is not succinct, in that the prover needs to send the whole circuit description and the input instances (i.e., polynomials $w_i[X]$ and $q_i[X]$) to the verifier. To achieve succinctness, one may use a polynomial commitment scheme: instead of sending the whole polynomials, the prover may send their commitments to the verifier, along with opening proofs of the polynomials at a random point, γ, chosen by the verifier. After verifying the commitment opening proofs, the verifier then checks the low degree polynomial $t[X]$, evaluated at γ indeed equals $h[X]/(X^N - 1)$ at γ. Via the Schwartz-Zippel lemma, the verifier is convinced $h[X]$ is $t[X](X^N - 1)$ for a low degree polynomial $t[X]$, and hence, $h[X]$ is zero at $\omega^0, \ldots \omega^{N-1}$.

In the literature, there exists a few popular PCS schemes, offering a wide range of functionalities. To name a few, inner-product argument (IPA) [BCC$^+$16,BBB$^+$18], KZG commitment [KZG10] and FRI [BBHR18]. In most part of this paper, we will try to use them as a blackbox with the following syntax.

- `PCS::commit`$(w[X]) \mapsto c$: input a polynomial, generate its commitment;
- `PCS::batch_open`$((w_1[X], \ldots, w_k[X]), \gamma) \mapsto \pi$: generate a proof of opens for $w_1[\gamma], \ldots, w_k[\gamma]$;
- `PCS::batch_verify`$((w_1[\gamma], \ldots, w_k[\gamma]), (c_1, \ldots, c_k), \pi) \mapsto$ `bool`: generate a proof of opens for $w_1[\gamma], \ldots, w_k[\gamma]$.

All functions may optionally take some (trusted) public parameters as an input. Those parameters are implicit and omitted in the syntax for simplicity.

Our desired properties for a PCS are that it takes linear time to commit and open, and allows for batch openings for multiple polynomials and multiple points. Such properties are supported by both IPA and KZG variants; we will concertize the performance of our system with IPA over Baby Jubjub and KZG over BN254 curve.

2.5 Folding scheme

A folding scheme, as defined in [KST22], is a protocol that reduces the task of checking two instances of a relation into the task of checking a single instance of a same relation. The paradigm to build a folding scheme from an existing relation is to relax the relation with an error term, also known as the slack factor, and use this error term to witness the cross terms of two input relations that is generated during the folding. A known limitation here is that the error term grows with the degree of the custom gate.

For a security point of view, a folding scheme must be knowledge sound. That is, given an output witness of the folding schemes, and a number of transcripts, one is able to extract the input witnesses to this folding scheme. This guarantees the output witness of the folding scheme share a same knowledge soundness as the input witnesses.

2.6 Recursive proofs

A recursive proof, denoted by π_2, is a proof asserting that another proof, π_1, is correct. This can be done by expressing π_1's verification algorithm as a circuit in the proof systems of π_2; treating π_1 as an witness to this recursive circuit, and finally generate π_2 asserting the recursive circuit is satisfied given π_1 as input.

A recursive proof is very powerful and enables many applications:

- it can convert an ethereum virtual machine (EVM) incompatible proof to a compatible proof; example applications include token bridges across different blockchains;
- it can be used to aggregate many proofs into a single proof; example applications are all sorts of zk-rollups and zkevm projects;
- it can be used to hide the inputs and the circuit structure to π_1 and result into a private computation; ZEXE is a typical example here.

In our scheme, such a recursive proof is optional. Looking ahead, our `Origami` VDF is linear in the number of foldings, i.e., k, in terms of both the proof size and the verifier computation. With a recursive proof we can remove this k factor and achieve a constant size proof.

3 Customize a Plonk for `MinRoot`

3.1 MinRoot Relation

The `MinRoot` hash function as shown in the original paper [KMT22] is defined as follows:

$$\texttt{MinRoot} : (x_{i+1}, y_{i+1}) \leftarrow \left((x_i + y_i)^{1/\alpha}, x_i + i \right)$$

where α is the minimum integer that is co-prime with p, the order of the field.

That is, given a starting point (x_0, y_0) we have

$$x_1 = (x_0 + y_0)^{1/\alpha}$$

and

$$x_{i+2} = (x_i + x_{i+1} + i)^{1/\alpha}$$

which is

$$x_i + x_{i+1} - x_{i+2}^{\alpha} + i = 0$$

Concretely, the precise statement we are proving is the following:

Definition 1 (MinRoot Relation). *Given a sequence of field elements* $X := \{x_i\}_{i=0}^{N-1}$ *over* $\mathbb{F}_p$. X *satisfies a MinRoot relation if*

$$\forall i \in [0, N-2), \quad x_i + x_{i+1} - x_{i+2}^{\alpha} + i = 0$$

where α *is the smallest integer that is co-prime with* $p - 1$.

For the rest of the paper we will use $\alpha = 5$ as an example. In principal, our result is also applicable to $\alpha = 7$. This will not change the asymptotic complexity for the prover nor the proof size; but will slightly increase the concrete proving time.

We also deal with a more general version of `MinRoot` relation where the inputs are polynomials, and the indexer not necessarily starts from 0. We say polynomials $w_1[X], \ldots, w_k[X]$ and $q_1[X], \ldots, q_k[X]$ satisfy the `MinRoot` relation, if

- the sequence $\{w_1[\omega^0], \ldots w_1[\omega^{n-1}], \ldots, w_k[\omega^0], \ldots, w_k[\omega^{n-1}]\}$ satisfies the above `MinRoot` relation, with a starting index $q_1[\omega^0]$;
- the sequence $\{q_1[\omega^0], \ldots q_1[\omega^{n-1}], \ldots, q_k[\omega^0], \ldots, q_k[\omega^{n-1}]\}$ is incremental, with a different of 1.

3.2 Plonkish arithmetization for MinRoot Relation

We build the following custom gate:

$$w[X] + w[\omega X] - w[\omega^2 X]^5 + q[X] = h[X] \tag{3}$$

where $q[X]$ is implicitly defined as $q[X] := \texttt{IFFT}(\langle 0, 1, 2, ..., N-1 \rangle)$, ω is the root of unity. Recall that, for a given $X = \omega^i$, $w[X]$ points to the witness at the i-th row of the matrix. Accordingly, $w[\omega X] = w[\omega^{i+1}]$ and $w[\omega^2 X] = w[\omega^{i+2}]$ point to the witnesses at the next two rows of the matrix. The constraints are satisfied if $h[\omega^i]$ are all 0s.

Compared with Equation 1, one may notice that our constraint system enjoys the following properties:

- there is only one witness polynomial $w[X]$;
- there is also one constant selector polynomial $q[X]$; it is hard coded to the circuit, so in principle the prover does not need to send it to the verifier;
- the custom gate is of degree 5.

In addition, we don't need any copy constraints or permutation checks in this system; the witnesses are glued together though the rotations: for a given witness w_i for the i-th row, $w[\omega^j X]$ accesses the witness of the $i+j$-th row.

A degree 5 custom gate implies that our $t[X]$ is of degree $4N$. It is common to take N as a power of 2. That is, we can compute $t[X]$ with a $4N$ coset FFT domain without any waste, since the degree of the polynomial matches that of the domain size.

To summarize, a low degree check IOP for the polynomial

$$t[X] = (w[X] + w[\omega X] - w[\omega^2 X]^5 + q[X])/Z[X]$$

is sufficient to convince the verifier that the witnesses satisfy the `MinRoot` relation.

In Algorithms 1 and 2 we show how to build a plonk IOP for instances satisfying such a relationship. For the ease of presentation, this construction omits the polynomial commitment scheme part; and lets the prover explicitly send all necessary polynomials to the verifier.

Algorithm 1 Plonk IOP prover for `MinRoot`

Require: An instance $\langle x_0, \ldots, x_{N-1} \rangle$ satisfying a `MinRoot` relation
Ensure: A proof $\pi := \{w[X], t[X]\}$ asserting this relation
1: Compute $w[X] = \texttt{IFFT}(\langle x_0, x_1, ..., x_{N-1} \rangle)$
2: Compute $q[X] = \texttt{IFFT}(\langle 0, 1, 2, ..., N-1 \rangle)$
3: Set $Z[X] := X^N - 1$
4: Compute $t[X] = (w[X] + w[\omega X] - w[\omega^2 X]^5 + q[X])/Z[X]$
5: **return** $\pi := \{w[X], t[X]\}$

Algorithm 2 Plonk IOP verifier for `MinRoot`

Require: A proof $\pi := \{w[X], t[X]\}$ asserting a `MinRoot` relation
Ensure: a boolean whether the relationship is satisfied
1: **if** $t[X]$'s degree $\geq 4N$ **then**
2: **return** false
3: **end if**
4: Sample a random challenge $\gamma \in \mathbb{F}_q$;
5: Compute $w[\gamma], w[\omega\gamma], w[\omega^2\gamma]$;
6: Compute $t[\gamma]$
7: Compute $q[X] = \texttt{IFFT}(\langle 0, 1, 2, ..., N-1 \rangle)$ and $q[\gamma]$
8: Set $Z[X] := X^N - 1$ and compute $Z[\gamma] := \gamma^N - 1$
9: **return** $t[\gamma]Z[\gamma] \stackrel{?}{=} w[\gamma] + w[\omega\gamma] - w[\omega^2\gamma]^5 + q[\gamma]$

Soundness Since $t[\gamma]Z[\gamma] = w[\gamma]+w[\omega\gamma]-w[\omega^2\gamma]^5+q[\gamma]$ for some random γ, from the Schwartz-Zippel lemma we know $t[X]Z[X] = w[X]+w[\omega X]-w[\omega^2 X]^5+q[X]$.

That is, the right hand side of the equation is divisible by $x^N - 1$, in other worlds, $w[\omega^i] + w[\omega^{i+1}] - w[\omega^{i+2}]^5 + q[\omega^i] = 0$ for $i \in [0, N-2)$. That concludes the proof.

Cost analysis. To convert Algorithm 1 and 2 into a succinct proof scheme via a polynomial commitment scheme is straightforward. We will defer it to the final construction in section 4.2.

Nonetheless, we give an estimation of the proving cost. Concretely, for N iterations of `MinRoot` hash, through our custom gate we build N constraints. The witness polynomial $w[X]$ is of degree N. The polynomial of $t[X]$ is of degree $4N$. If one were to implement the above proving system for the `MinRoot` hash with either IPA or KZG commitment scheme, the prover will need to commit to $w[X]$ and $t[X]$, which incurs $5N$ group multiplications. The prover also needs to open both $w[X]$ and $t[X]$. Assuming batch opening, the opening proof can be computed with N group multiplications. In addition, to compute $t[X]$ we will need to perform coset FFTs over a $4N$-dimension domain.

The overall cost for such a scheme is $6N$ group multiplications; one FFT over a size N domain; and few FFTs over a size $4N$ coset domain.

4 Fold a plonk for `MinRoot`

Now we present a folding scheme that is dedicated to our `MinRoot` relation.

4.1 Intuition

We give intuitions of how to fold our Plonk gates in this section. The security proof is deferred to Section 4.3.

Fold two instances For now it is sufficient to consider a custom gate

$$w[X] + w[\omega X] - w[\omega^2 X]^5 + e[X] = h[X] \tag{4}$$

where $w[X]$ is the witness polynomial as before; $e[X]$ is an error polynomial, a.k.a. the slack factor; $h[X]$ is the polynomial that is divisible by $x^n - 1$. Looking ahead, $e[X]$ receives contributions from both the selector $q[X]$ and the cross terms generated during folding.

Assume we have two witness polynomials $w_a[X]$ and $w_b[X]$, satisfying

$$\begin{aligned} w_a[X] + w_a[\omega X] - w_a[\omega^2 X]^5 + e_a[X] &= h_a[X] \\ w_b[X] + w_b[\omega X] - w_b[\omega^2 X]^5 + e_b[X] &= h_b[X] \end{aligned}$$

with $h_a[\omega^i] = h_b[\omega^i] = 0$ for $i \in [0, n-2)$, our goal is to generate a new witness $w_c[X] := w_a[X] + w_b[X]$, such that

$$w_c[X] + w_c[\omega X] - w_c[\omega^2 X]^5 + e_c[X] = h_c[X] \tag{5}$$

with $h_c[\omega^i] = 0$ for $i \in [0, n-2)$. Notice that

$$\begin{aligned}
&w_c[X] + w_c[\omega X] - w_c[\omega^2 X]^5 \\
&\quad = w_c[X] + w_c[\omega X] \\
&\quad - (w_a[\omega^2 X]^5 + 5w_a[\omega^2 X]^4 w_b[\omega^2 X] + 10w_a[\omega^2 X]^3 w_b[\omega^2 X]^2 \\
&\quad + 10w_a[\omega^2 X]^2 w_b[\omega^2 X]^3 + 5w_a[\omega^2 X] w_b[\omega^2 X]^4 + w_b[\omega^2 X]^5) \\
&\quad = -e_a[X] - e_b[X] - (5w_a[\omega^2 X]^4 w_b[\omega^2 X] + 10w_a[\omega^2 X]^3 w_b[\omega^2 X]^2 \\
&\quad + 10w_a[\omega^2 X]^2 w_b[\omega^2 X]^3 + 5w_a[\omega^2 X] w_b[\omega^2 X]^4)
\end{aligned}$$

For the ease of presentation, denote

$$\begin{aligned}
\Delta[X] := &\, 5w_a[\omega^2 X]^4 w_b[\omega^2 X] + 10w_a[\omega^2 X]^3 w_b[\omega^2 X]^2 \\
&+ 10w_a[\omega^2 X]^2 w_b[\omega^2 X]^3 + 5w_a[\omega^2 X] w_b[\omega^2 X]^4
\end{aligned} \tag{6}$$

as the expanding factor of the error polynomial. Note that $\Delta[X]$ is only a symbolic pointer to a set of monomials as defined in Equation 6 or 11; if it is helpful, readers should replace each $\Delta_j[X]$ with Equation 11 and obtain the explicit formula for $e_i[X]$.

Finally, setting

$$e_c[X] := e_a[X] + e_b[X] + \Delta[X] \tag{7}$$

we obtain Equation 5 as desired.

Fold 2 instances with re-randomization Now we consider the case where one of the witnesses is randomized by a field element r during aggregation, i.e., $w_c[X] = r \cdot w_a[X] + w_b[X]$. This field element is a challenge from the verifier, after both $w_a[X]$ and $w_b[X]$ are committed. This step is crucial for the security, otherwise a malicious prover may use a unsatisfying $w_a[X]$, and use $w_b[X]$ to offset the unsatisfied part in $w_a[X]$.

With re-randomization, we have

$$\begin{aligned}
&w_c[X] + w_c[\omega X] - w_c[\omega^2 X]^5 \\
&\quad = w_c[X] + w_c[\omega X] \\
&\quad - (r^5 \cdot w_a[\omega^2 X]^5 + 5r^4 \cdot w_a[\omega^2 X]^4 w_b[\omega^2 X] + 10r^3 \cdot w_a[\omega^2 X]^3 w_b[\omega^2 X]^2 \\
&\quad + 10r^2 \cdot w_a[\omega^2 X]^2 w_b[\omega^2 X]^3 + 5r \cdot w_a[\omega^2 X] w_b[\omega^2 X]^4 + w_b[\omega^2 X]^5) \\
&\quad = (r - r^5) \cdot w_a[\omega^2 X]^5 - r \cdot e_a[X] - e_b[X] \\
&\quad - (5r^4 \cdot w_a[\omega^2 X]^4 w_b[\omega^2 X] + 10r^3 \cdot w_a[\omega^2 X]^3 w_b[\omega^2 X]^2 \\
&\quad + 10r^2 \cdot w_a[\omega^2 X]^2 w_b[\omega^2 X]^3 + 5r \cdot w_a[\omega^2 X] w_b[\omega^2 X]^4)
\end{aligned}$$

That is, if we set

$$e_c[X] := r \cdot e_a[X] + e_b[X] + \Delta[X] \tag{8}$$

where

$$\begin{aligned}\Delta[X] := 5r^4 \cdot w_a[\omega^2 X]^4 w_b[\omega^2 X] + 10r^3 \cdot w_a[\omega^2 X]^3 w_b[\omega^2 X]^2 \\ + 10r^2 \cdot w_a[\omega^2 X]^2 w_b[\omega^2 X]^3 + 5r \cdot w_a[\omega^2 X] w_b[\omega^2 X]^4 \\ + (r^5 - r) \cdot w_a[\omega^2 X]^5 \end{aligned} \tag{9}$$

then, we have obtained a new pair of witness and error polynomials $w_c[X]$ and $e_c[X]$ satisfying Equation 4.

As a sanity check, if r is 1, then Equations 6 and 7 are identical to Equations 8 and 9 .

To check that the new custom gate is satisfied, the verifier checks that Equations 5 and 8 both hold.

Succinctness It remains to be shown that both $w_c[X]$ and $e_c[X]$ are compact: they are of constant size and do not incur additional communication cost for the prover. This is obvious for $w_c[X]$ which is a random linear combination of $w_a[X]$ and $w_b[X]$, for which the verifier has already received. Then, for $e_c[X]$ during each folding, the number of monomials in the error term is increased; and a degree $5n$ polynomial is accumulated to a random linear combination of $e_a[X]$ and $e_b[X]$. Therefore, $e_c[X]$'s degree is bounded by $5n$, and given $w_a[X]$ and $w_b[X]$ the verifier can locally compute $e_c[X]$ ($e_a[X]$ and $e_b[X]$ are $q_a[X]$ and $q_b[X]$ by definition which are shared with verifier already). So if we continuously fold many times, the verifier can compute all the $e_c[X]$ during each folding.

Fold many instances Our strategy of folding many instances is straightforward from the previous section. Let $w_1[X], \ldots, w_k[X]$ be k witness polynomials that we want to fold. We fold the first two `MinRoot` instances to generate a new instance, and then continuously fold an existing instance with the previously folded instance. During each iteration, we randomize the previous witness, i.e.,

$$\forall j \in [2, k], \quad w'_j[X] = r_j \cdot w'_{j-1}[X] + w_j[X].$$

where r_j-s are random field elements sampled from the transcript. We set

$$e'_j[X] := r_j \cdot e'_{j-1}[X] + e_{j-1}[X] + \Delta_j[X] \tag{10}$$

The prover shares $\{w_j[x], e_j[X]\}_{j=1}^k$, as well as a random linear combination of $\{e'_j[X]\}_{j=1}^k$ with the verifier. The verifier can locally compute all $\{w'_j[x], e'_j[X]\}_{j=1}^k$, and check if the local random linear combination of $\{e'_j[X]\}_{j=1}^k$ matches the one received from the prover.

We remark that an alternative approach to the continuous folding scheme above is to build a binary tree by treating the instances as the leaves. This also requires $k-1$ folding, and does not offer a significant difference from the presented scheme.

4.2 Concrete Scheme

We are in the setting of proving the `MinRoot` relation for a degree $N := kn$ witness polynomial $w[X]$, and its corresponding selector polynomial $q[X]$. We parse them to a list of k witness polynomials $w_1[X], \ldots, w_k[X]$, and corresponding selector polynomials $q_1[X], \ldots, q_k[X]$, each of degree n. $q_1[X], \ldots, q_k[X]$ are shared between prover and verifier at no cost. We want to generate a succinct proof π asserting the `MinRoot` relation for $w[X]$ and $q[X]$.

One minor caveat is how to argue $w_{j+1}[X]$ is a continuation $w_j[X]$ in a `MinRoot` sequence. A naive solution is for the prover to send the first and the last two elements of each $w_j[X]$ to the verifier.

We plug in a PCS to make the system succinct. As mentioned earlier, we use PCS that supports batch opening as a blackbox. In practice one may use KZG or IPA based solutions.

Cost analysis The prover will need to commit to all $w_j[X]$ for $j \in [1, k]$, i.e., $c_{w_j} = \texttt{PCS::commit}(w_j[X])$. This incurs a cost of $N := nk$ group multiplications. The prover will also commit to the final $t[X]$ and $e[X]$ respectively [4], i.e., $c_t = \texttt{PCS::commit}(t[X])$ and $c_e = \texttt{PCS::commit}(e[X])$, which requires $4n + 5n = 9n$ group multiplications. The prover then samples a challenge γ, and opens $w_j[X]$, $t[X]$ and $e[X]$ at γ; $w_j[X]$ at $\omega\gamma$; and $w_j[X]$ at $\omega^2\gamma$. The three batch openings require another $3n$ group multiplications.

The final proof consists of $\{w_j[\gamma], w_j[\omega\gamma], w_j[\omega^2\gamma]\}_{j=1}^{k}, t[\gamma], e[\gamma]$ and the openings. The computation is dominated by $(k + 12)n$ group multiplications and k coset FFTs of dimension $4n$. The proof is of size $128k + 224$ bytes. consists of the following elements. The corresponding size is given assuming KZG commitment over BN254 curve.

- c_{w_j}: the commitments to $w_j[X]$, each of size 32 bytes.
- c_t and c_e: the commitment to $t[X]$ and $e[X]$, each of size 32 bytes.
- $w_j[\gamma], w_j[\omega\gamma], w_j[\omega^2\gamma]$: evaluations of $w_j[X]$ at three positions. For each j this requires 96 bytes.
- $t[\gamma]$ and $e[\gamma]$: evaluations of $t[X]$ and $e[X]$, each of size 32 bytes.
- π_1, π_2, π_3: three batch opening proofs at γ, $\omega\gamma$ and $\omega^2\gamma$, total of size 32 bytes or 96 bytes, depending on the batching mechanism. Here we use 96 bytes.

4.3 Security proof

The proof takes the following steps.

1. The soundness of the scheme in Algorithm 3 and 4 is a direct application of PCS arguments with the Schwartz-Zippel lemma and Fiat-Shamir over the IOP scheme, labeled by Hybrid 1, in Algorithm 5 and 6. If the original

[3] Depending on the actual scheme, π_1, π_2 and π_3 may be further combined.

[4] In practice, one usually decomposes it into 4 and 5 polynomials, respectively, each of dimension n; and commits them individually.

Algorithm 3 Folding Plonk for `MinRoot`: Prover

Require: $w_1[X], \ldots, w_k[X]$ and $q_1[X], \ldots, q_k[X]$ satisfying a `MinRoot` relation

Ensure: $\pi = \{\{c_{w_j}, w_j[\gamma], w_j[\omega\gamma], w_j[\omega^2\gamma]\}_{i=1}^k, c_t, t[\gamma], c_e, e[\gamma], \pi_1, \pi_2, \pi_3\}$ asserting this relation[3]

for $j \leftarrow 1$ to k **do**
  Compute $c_{w_j} = \texttt{PCS::commit}(w_j[X])$ and append c_{w_j} to transcript.
end for
Set $Z[X] = X^n - 1$
Set $e_1[X] = q_1[X]$
Compute $t_1[X] = (w_1[X] + w_1[\omega X] - w_1[\omega^2 X]^5 + e_1[X])/Z[X]$
for $j \leftarrow 2$ to k **do**
  Sample a challenge r_j from transcript
  Compute $w'_j[X] = r_j \cdot w'_{j-1}[X] + w_j[X]$
  Compute $\Delta_j[X]$ as in Equation 11

$$\begin{aligned}\Delta_j[X] = 5r_j^4 \cdot w'_{j-1}[\omega^2 X]^4 w_j[\omega^2 X] + \ 10r_j^3 \cdot w'_{j-1}[\omega^2 X]^3 w_j[\omega^2 X]^2 \\ + 10r_j^2 \cdot w'_{j-1}[\omega^2 X]^2 w_j[\omega^2 X]^3 + 5r_j \cdot w'_{j-1}[\omega^2 X] w_j[\omega^2 X]^4 \\ + (r^5 - r) \cdot w_j[\omega^2 X]^5 \end{aligned} \tag{11}$$

  Set $e_j[X] = r_j \cdot e_{j-1}[X] + q_j[X] + \Delta_j[X]$
  Set $t_j[X] = (w'_j[X] + w'_j[\omega X] - w'_j[\omega^2 X]^5 + e_j[X])/Z[X]$
end for
Sample challenges β_t and β_e from transcript
Compute $t[X] = \sum_{j=1}^k \beta_t^{j-1} \cdot t_j[X]$
Compute $c_t = \texttt{PCS::commit}(t[X])$ and append c_t to transcript
Compute $e[X] = \sum_{j=1}^k \beta_e^{j-1} \cdot e_j[X]$
Compute $e_t = \texttt{PCS::commit}(e[X])$ and append e_t to transcript
Sample a challenge γ from transcript
Generate a proof $\pi_1 = \texttt{PCS::batch\_open}((w_1[X], \ldots w_k[X], t[X], e[X]), \gamma)$
Generate a proof $\pi_2 = \texttt{PCS::batch\_open}((w_1[X], \ldots w_k[X]), \omega\gamma)$
Generate a proof $\pi_3 = \texttt{PCS::batch\_open}((w_1[X], \ldots w_k[X]), \omega^2\gamma)$
return $\pi = \{\{c_{w_j}, w_j[\gamma], w_j[\omega\gamma], w_j[\omega^2\gamma]\}_{j=1}^k, c_t, t[\gamma], c_e, e[\gamma], \pi_1, \pi_2, \pi_3\}$

Algorithm 4 Folding Plonk for `MinRoot`: Verifier

Require: $\pi = \{\{c_{w_j}, w_j[\gamma], w_j[\omega\gamma], w_j[\omega^2\gamma]\}_{i=1}^{k}, c_t, t[\gamma], c_e, e[\gamma], \pi_1, \pi_2, \pi_3\}$
Ensure: A boolean whether the `MinRoot` relation is satisfied for the witness committed in $\{c_{w_j}\}_{i=1}^{k}$

Append $\{c_{w_j}\}_{i=1}^{k}$ to transcript.
for $j \leftarrow 2$ to k **do**
  Sample r_j from transcript.
end for
Sample a challenge β_s and β_e from transcript.
Append c_t and c_e to the transcript.
Sample a challenge γ from transcript.
{Check the correctness of gate evaluations}
Set $Z[X] = X^n - 1$, and compute $Z[\gamma]$
Set $e_1[\gamma] = q_1[\gamma]$
Compute $t_1[\gamma] = (w_1[\gamma] + w_1[\omega\gamma] - w_1[\omega^2\gamma]^5 + e_1[\gamma])/Z[\gamma]$
for $j \leftarrow 2$ to k **do**
  Compute $w'_j[\gamma] = r_j \cdot w'_{j-1}[\gamma] + w_j[\gamma]$
  Compute $\Delta_j[\gamma]$ as in Equation 11
  Set $e_j[\gamma] := r_j \cdot e_{j-1}[\gamma] + q_j[\gamma] + \Delta_j[\gamma]$
  Compute $t_j[\gamma] = (w'_j[\gamma] + w'_j[\omega\gamma] - w'_j[\omega^2\gamma]^5 + e_j[\gamma])/Z[\gamma]$
end for
{Check the correctness of RLC for $t[X]$ and $e[X]$}
if $t[\gamma] \neq \sum_{j=1}^{k} \beta_t^{j-1} \cdot t_j[\gamma]$ **then**
  return false
end if
if $e[\gamma] \neq \sum_{j=1}^{k} \beta_e^{j-1} \cdot e_j[\gamma]$ **then**
  return false
end if
{Verify the polynomial commitments}
if !`PCS::batch_verify`$((c_{w_1}, \ldots, c_{w_k}, c_t, c_e), (w_1[\gamma], \ldots w_k[\gamma], t[\gamma], e[\gamma]), \pi_1)$ **then**
  return false
end if
if !`PCS::batch_verify`$((c_{w_1}, \ldots, c_{w_k}), (w_1[\omega\gamma], \ldots w_k[\omega\gamma]), \pi_2)$ **then**
  return false
end if
if !`PCS::batch_verify`$((c_{w_1}, \ldots, c_{w_k}), (w_1[\omega^2\gamma], \ldots w_k[\omega^2\gamma]), \pi_3)$ **then**
  return false
end if
return true.

scheme is sound, then the IOP scheme must also be sound, where, instead of letting the prover to send the polynomial commitments and their opening, we let the prover to send the exact polynomials.
2. Then, we show that Hybrid 1 in Algorithm 5 and 6 is equivalent to another scheme, Hybrid 2, in Algorithm 7 and Algorithm 8; the only difference is that $\{t_j[X], e_j[X]\}_{j=1}^k$ are checked individually rather than via a random linear combination. The proof follows.

Observe that, given the set $\{w_j[X], q_j[X]\}_{j=1}^k$, the verifier can re-compute all the elements in $\{w'_j[X], e_j[X], t_j[X]\}_{j=1}^k$:

- $w'_1[X]$ is $w_1[X]$ and $e_1[X]$ is $q_1[X]$ by definition;
- $\{w'_j[X]\}_{j=2}^k$ can be obtained recursively since $w'_j[X]$ is a random linear combination of $w_j[X]$ and $w'_{j-1}[X]$;
- with $\{w_j[X], w'_j[X]\}_{j=1}^k$ the verifier knows all $\Delta_j[X]$ via Equation 11;
- $\{e_j[X]\}_{j=2}^k$ can also be obtained recursively via $e_j[X] = r_j \cdot e_{j-1}[\gamma] + q_j[\gamma] + \Delta_j[\gamma]$;
- Finally $\{t_j[X]\}_{j=1}^k$ are obtained via $t_j[X] = (w'_j[X] + w'_j[\omega X] - w'_j[\omega^2 X]^5 + e_j[X])/Z[X]$.

In Algorithm 6, the verifier receives $e[X]$ and $t[X]$. It then checks at a random point γ such that $e[\gamma] = \sum_{j=1}^k \beta_e^{j-1} e_j[\gamma]$ and $t[\gamma] = \sum_{j=1}^k \beta_t^{j-1} t_j[\gamma]$, which convinces it that $e[X] = \sum_{j=1}^k \beta_e^{j-1} e_j[X]$ and $t[X] = \sum_{j=1}^k \beta_t^{j-1} t_j[X]$ are well-formed: they are derived from the witness and selector polynomials as claimed. That is, if the scheme is sound when the verifier receives the random linear combination $e[X]$ and $t[X]$, it must also be sound when the verifier receives the exact $\{e_j[X], t_j[X]\}_{j=1}^k$.

3. Next, Hybrid 2 is $k-1$ iterations of Hybrid 3, described in Algorithm 9 and 10. One caveat here is that when k is large, we will need to invoke the forking lemma (used in Step 4) for k times, and effectively introducing a security loss of k bits. So in theory one should not fold the scheme for too many times. How to achieve a better reduction for polynomial number of recursions is still an open problem.
4. To prove Hybrid 3 is sound, we need to show that the folding scheme is knowledge sound; and the IOP scheme in Algorithm 1 and 2 is sound. The latter has already been demonstrated in the previous section.

The reminder of this section focuses on proving that folding scheme is knowledge sound.

We show the knowledge soundness of Hybrid 3 in Algorithms 9 and 10 via an extractor, who, upon a valid output witness polynomial $w_c[X]$, can extract two witnesses $w_a[X]$ and $w_b[X]$ both satisfy the `MinRoot` relation.

The extractor executes a valid folding scheme and forks at the stage when the randomness r is sampled by the random oracle. With rewinding, the extractor gets two valid, distinct transcripts, denoted by $(w_c[X], t[X], e_c[X], r)$ and

Algorithm 5 Hybrid 1: Prover

Require: $w_1[X], \ldots, w_k[X]$ and $q_1[X], \ldots, q_k[X]$ satisfying a `MinRoot` relation
Ensure: A proof $\pi := \{w_1[X], \ldots, w_k[X], t[X], e[X]\}$ asserting this relation
Set $Z[X] = X^n - 1$
Set $e_1[X] = q_1[X]$ and $w'_1[X] = w_1[X]$
Compute $t_1[X] = (w'_1[X] + w'_1[\omega X] - w'_1[\omega^2 X]^5 + e_1[X])/Z[X]$
for $j \leftarrow 2$ to k **do**
    Sample a challenge r_j
    Compute $w'_j[X] = r_j \cdot w'_{j-1}[X] + \cdot w_j[X]$
    Compute $\Delta_j[X]$ as in Equation 11
    Set $e_j[X] = r_j \cdot e_{j-1}[X] + q_j[X] + \Delta_j[X]$
    Set $t_j[X] = (w'_j[X] + w'_j[\omega X] - w'_j[\omega^2 X]^5 + e_j[X])/Z[X]$
end for
Sample challenges β_t and β_e
Compute $t[X] = \sum_{j=1}^{k} \beta_t^{j-1} \cdot t_j[X]$
Compute $e[X] = \sum_{j=1}^{k} \beta_e^{j-1} \cdot e_j[X]$

Algorithm 6 Hybrid 1: Verifier

Require: $w_1[X], \ldots, w_k[X]$ and $q_1[X], \ldots, q_k[X]$ satisfying a `MinRoot` relation
Require: A proof $\pi := \{w_1[X], \ldots, w_k[X], t[X]\}$ asserting `MinRoot` relation
Ensure: a Boolean whether the assertion is correct
if $t[X]$'s degree $\geq 4n$ **then**
    return false
end if
Sample a random challenge γ
Set $Z[X] = X^n - 1$ and compute $Z[\gamma]$
Compute $t[\gamma]$
Set $e_1[X] = q_1[X]$ and compute $e_1[\gamma]$
Compute $t_1[\gamma] = (w_1[\gamma] + w_1[\omega\gamma] - w_1[\omega^2\gamma]^5 + e_1[\gamma])/Z[\gamma]$
for $j \leftarrow 2$ to k **do**
    Sample a challenge r_j
    Compute $w'_j[X] = r_j \cdot w_{j-1}[X] + w_j[X]$
    Compute $\Delta_j[X]$ as in Equation 11
    Set $e_j[X] = r_j \cdot e_{j-1}[X] + q_j[X] + \Delta_j[X]$ and compute $e_j[\gamma]$
    Compute $t_j[\gamma] = (w'_j[\gamma] + w'_j[\omega\gamma] - w'_j[\omega^2\gamma]^5 + e_j[\gamma])/Z[\gamma]$
end for
Sample challenges β_t and β_e
return $t[\gamma] \stackrel{?}{=} \sum_{j=1}^{k} \beta_t^{j-1} \cdot t_j[\gamma]$ and $e[\gamma] \stackrel{?}{=} \sum_{j=1}^{k} \beta_e^{j-1} \cdot e_j[\gamma]$

Algorithm 7 Hybrid 2: Prover

Require: $w_1[X], \ldots, w_k[X]$ and $q_1[X], \ldots, q_k[X]$ satisfying a `MinRoot` relation
Ensure: A proof $\pi := \{\{w_j[X], t_j[X], e_j[X]\}_{i=1}^{k}\}$ asserting this relation
1: Set $Z[X] = X^n - 1$
2: Set $e_1[X] = q_1[X]$ and $w'_1[X] = w_1[X]$
3: Compute $t_1[X] = (w'_1[X] + w'_1[\omega X] - w'_1[\omega^2 X]^5 + e_1[X])/Z[X]$
4: **for** $j \leftarrow 2$ to k **do**
5: Sample a challenge r_j
6: Compute $w'_j[X] = r_j \cdot w'_{j-1}[X] + \cdot w_j[X]$
7: Compute $\Delta_j[X]$ as in Equation 11
8: Set $e_j[X] = r_j \cdot e_{j-1}[X] + q_j[X] + \Delta_j[X]$
9: Set $t_j[X] = (w'_j[X] + w'_j[\omega X] - w'_j[\omega^2 X]^5 + e_j[X])/Z[X]$
10: **end for**

Algorithm 8 Hybrid 2: Verifier

Require: $w_1[X], \ldots, w_k[X]$ and $q_1[X], \ldots, q_k[X]$ satisfying a `MinRoot` relation
Require: A proof $\pi := \{\{w_j[X], t_j[X], e_j[X]\}_{j=1}^{k}\}$ asserting `MinRoot` relation
Ensure: a Boolean whether the assertion is correct
1: **if** $t_j[X]$'s degree $\geq 4n$ for $j \in [0, k)$ **then**
2: **return** false
3: **end if**
4: Sample a random challenge γ
5: Set $Z[X] = X^n - 1$ and compute $Z[\gamma]$
6: Set $e_1[X] = q_1[X]$ and compute $e_1[\gamma]$
7: **if** $t_1[\gamma] \neq (w_1[\gamma] + w_1[\omega\gamma] - w_1[\omega^2\gamma]^5 + e_1[\gamma])/Z[\gamma]$ **then**
8: **return** false
9: **end if**
10: **for** $j \leftarrow 2$ to k **do**
11: Sample a challenge r_j
12: Compute $w'_j[X] = r_j \cdot w_{j-1}[X] + w_j[X]$
13: Compute $\Delta_j[X]$ as in Equation 11
14: **if** $e_j[\gamma] \neq r_j \cdot e_{j-1}[\gamma] + q_j[\gamma] + \Delta_j[\gamma]$ **then**
15: **return** false
16: **end if**
17: **if** $t_j[\gamma] \neq (w'_j[\gamma] + w'_j[\omega\gamma] - w'_j[\omega^2\gamma]^5 + e_j[\gamma])/Z[\gamma]$ **then**
18: **return** false
19: **end if**
20: **end for**

Algorithm 9 Hybrid 3: Prover

Require: $w_a[X], w_b[X]$, $q_a[X]$ and $q_b[X]$ satisfying a `MinRoot` relation
Ensure: A instance $\{w_c[X], e_c[X]\}$ satisfying a `MinRoot` relation
Ensure: A proof $\pi := \{w_a[X], w_b[X], t[X], e_c[X]\}$ asserting this relation
1: Set $Z[X] = X^n - 1$
2: Sample a challenge r
3: Compute $w_c[X] = r \cdot w_a[X] + \cdot w_b[X]$
4: Compute $\Delta[X]$ as in Equation 11
5: Set $e_c[X] = r \cdot q_a[X] + q_b[X] + \Delta[X]$
6: Set $t[X] = (w_c[X] + w_c[\omega X] - w_c[\omega^2 X]^5 + e_c[X])/Z[X]$

Algorithm 10 Hybrid 3: Verifier

Require: A instance $\{w_c[X], e_c[X]\}$ satisfying a `MinRoot` relation
Require: A proof $\pi := \{w_a[X], w_b[X], t[X], e_c[X]\}$ asserting this relation
Ensure: a Boolean whether the assertion is correct
1: **if** $t[X]$'s degree $\geq 4n$ **then**
2: **return** false
3: **end if**
4: Sample a random challenge γ
5: Set $Z[X] = X^n - 1$ and compute $Z[\gamma]$
6: Sample a challenge r
7: Compute $w_c[X] = r \cdot w_a[X] + w_b[X]$
8: Compute $\Delta[X]$ as in Equation 11
9: **if** $e_c[\gamma] \neq r \cdot q_a[\gamma] + q_b[\gamma] + \Delta[\gamma]$ **then**
10: **return** false
11: **end if**
12: **if** $t[\gamma] \neq (w_c[\gamma] + w_c[\omega\gamma] - w_c[\omega^2\gamma]^5 + e_c[\gamma])/Z[\gamma]$ **then**
13: **return** false
14: **end if**

$(w'_c[X], t'[X], e'_c[X], r')$. Recall that a proof is valid if Equations 5 and 8 both hold. From Equations 5, we obtain

$$\begin{aligned} w_c[X] + w_c[\omega X] - w_c[\omega^2 X]^5 + e_c[X] &= t[X](X^n - 1) =: h_c[X] \\ w'_c[X] + w'_c[\omega X] - w'_c[\omega^2 X]^5 + e'_c[X] &= t'[X](X^n - 1) =: h'_c[X] \end{aligned} \tag{12}$$

where $h_c[\omega^i] = h'_c[\omega^i] = 0$ for $0 \leq i < n - 2$.

Since both transcripts are obtained from a same set of inputs, then there must exist some unknown $\bar{w}_a[X]$ and $\bar{w}_a[X]$ such that

$$\begin{aligned} w_c[X] &:= r \cdot \bar{w}_a[X] + \bar{w}_b[X] \\ w'_c[X] &:= r' \cdot \bar{w}_a[X] + \bar{w}_b[X] \end{aligned} \tag{13}$$

In the meantime, due to Equation 8

$$\begin{aligned} e_c[X] := {} & 5r^4 \cdot \bar{w}_a[\omega^2 X]^4 \bar{w}_b[\omega^2 X] + 10r^3 \cdot \bar{w}_a[\omega^2 X]^3 \bar{w}_b[\omega^2 X]^2 \\ & + 10r^2 \cdot \bar{w}_a[\omega^2 X]^2 \bar{w}_b[\omega^2 X]^3 + 5r \cdot \bar{w}_a[\omega^2 X] \bar{w}_b[\omega^2 X]^4 \\ & + (r^5 - r) \cdot \bar{w}_a[\omega^2 X]^5 \\ & + r \cdot \bar{e}_a[X] + \bar{e}_b[X] \\ e'_c[X] := {} & 5r'^4 \cdot \bar{w}_a[\omega^2 X]^4 \bar{w}_b[\omega^2 X] + 10r'^3 \cdot \bar{w}_a[\omega^2 X]^3 \bar{w}_b[\omega^2 X]^2 \\ & + 10r'^2 \cdot \bar{w}_a[\omega^2 X]^2 \bar{w}_b[\omega^2 X]^3 + 5r' \cdot \bar{w}_a[\omega^2 X] \bar{w}_b[\omega^2 X]^4 \\ & + (r'^5 - r') \cdot \bar{w}_a[\omega^2 X]^5 \\ & + r' \cdot \bar{e}_a[X] + \bar{e}_b[X] \end{aligned} \tag{14}$$

for some unknown $\bar{e}_a[X]$ and $\bar{e}_b[X]$. The check in Step 9 of Algorithm 10 ensures that the same $\bar{w}_a[X]$ and $\bar{w}_b[X]$ are used in building $w_c[X]$ and $e_c[X]$ (as well as $w'_c[X]$ and $e'_c[X]$). Plug in Equations 13 and 14 into Equation 12 we obtain

$$\begin{aligned} r \cdot (\bar{w}_a[X] + \bar{w}_a[\omega X] - \bar{w}_a[\omega^2 X]^5 + \bar{e}_a[X]) \\ + \bar{w}_b[X] + \bar{w}_b[\omega X] - \bar{w}_b[\omega^2 X]^5 + \bar{e}_b[X] = h_c[X] \\ r' \cdot (\bar{w}_a[X] + \bar{w}_a[\omega X] - \bar{w}_a[\omega^2 X]^5 + \bar{e}_a[X]) \\ + \bar{w}_b[X] + \bar{w}_b[\omega X] - \bar{w}_b[\omega^2 X]^5 + \bar{e}_b[X] = h'_c[X] \end{aligned}$$

Since r and r' are distinct outputs from the random oracle, we conclude that $\bar{h}_a[X]$ and $\bar{h}_b[X]$, defined as

$$\begin{aligned} \bar{h}_a[X] &:= \bar{w}_a[X] + \bar{w}_a[\omega X] - \bar{w}_a[\omega^2 X]^5 + \bar{e}_a[X] \\ \bar{h}_b[X] &:= \bar{w}_b[X] + \bar{w}_b[\omega X] - \bar{w}_b[\omega^2 X]^5 + \bar{e}_b[X] \end{aligned}$$

have the property that $\bar{h}_a[\omega^i] = \bar{h}_b[\omega^i] = 0$ for $0 \leq i < n - 2$.

It remains to be shown how we can actually extract $\bar{w}_a[X]$, $\bar{w}_a[X]$, $\bar{e}_a[X]$ and $\bar{e}_b[X]$. First, from Equation 13 we already have

$$\bar{w}_a[X] = \frac{w'_c[X] - w_c[X]}{r' - r}$$
$$\bar{w}_b[X] = \frac{r \cdot w'_c[X] - r' \cdot w_c[X]}{r' - r}$$

Then, define

$$\begin{aligned}
\bar{\Delta}[X] := {} & 5r^4 \cdot \bar{w}_a[\omega^2 X]^4 \bar{w}_b[\omega^2 X] + 10r^3 \cdot \bar{w}_a[\omega^2 X]^3 \bar{w}_b[\omega^2 X]^2 \\
& + 10r^2 \cdot \bar{w}_a[\omega^2 X]^2 \bar{w}_b[\omega^2 X]^3 + 5r \cdot \bar{w}_a[\omega^2 X] \bar{w}_b[\omega^2 X]^4 \\
& + (r^5 - r) \cdot \bar{w}_a[\omega^2 X]^5 \\
\bar{\Delta}'[X] := {} & 5r'^4 \cdot \bar{w}_a[\omega^2 X]^4 \bar{w}_b[\omega^2 X] + 10r'^3 \cdot \bar{w}_a[\omega^2 X]^3 \bar{w}_b[\omega^2 X]^2 \\
& + 10r'^2 \cdot \bar{w}_a[\omega^2 X]^2 \bar{w}_b[\omega^2 X]^3 + 5r' \cdot \bar{w}_a[\omega^2 X] \bar{w}_b[\omega^2 X]^4 \\
& + (r'^5 - r') \cdot \bar{w}_a[\omega^2 X]^5
\end{aligned}$$

From Equation 14 we have

$$e_c[X] = \bar{\Delta}[X] + r \cdot \bar{e}_a[X] + \bar{e}_b[X]$$
$$e'_c[X] = \bar{\Delta}'[X] + r' \cdot \bar{e}_a[X] + \bar{e}_b[X]$$

which allows us to extract $\bar{e}_a[X]$ and $\bar{e}_b[X]$ as

$$\bar{e}_a[X] = \frac{e_c[X] - e'_c[X] + \bar{\Delta}'[X] - \bar{\Delta}[X]}{r - r'}$$
$$\bar{e}_b[X] = \frac{r' \cdot e_c[X] - r \cdot e'_c[X] - r \cdot \bar{\Delta}'[X] - r' \cdot \bar{\Delta}[X]}{r' - r}$$

5 Deployment with recursive proofs

Compared to `MinRoot` VDF, our `Origami` VDF provides faster proving time, at a cost of larger proof size. A typical optimization here is to generated a recursive proof to prove the validity of `Origami` proof. For Ethereum use cases, it may be desirable to have the recursive proof over BN254 curve. In this setting, we may generate the `Origami` VDF proof over the so-called Baby Jubjub curve, a twisted Edwards curve whose base field matches the scalar field of BN254 curve. Two caveats here. First, Baby Jubjub curve's α parameter is 7 which makes the prover a bit less efficient; second, Jubjub curve does not support pairings, which limits our choice of PCS. A natural choice is IPA commitment scheme.

Another option is to use Jubjub (with $\alpha = 5$) and BLS12-381 curves. The hash chain proof can be generated slightly faster than the previous choice; but the recursive proof will be larger and slower due to the increased field size.

Assuming Baby Jubjub and BN254 curves, we estimate the number of constraints for `Origami` verification circuit. The dominate part is the IPA verification algorithm which requires $O(n)$ group operations. [CBBZ22] has shown that the amortized number of circuit constraints per group operation for the Pippenger algorithm is around 300 for a customized plonk system [Esp]. That is, if we assume $n = 1024$ as in `MinRoot` reference code [Set], a recursive circuit will require $O(2^{19})$ constraints which can be generated with around 5 seconds with a 16 core CPU. The final recursive proof of [Esp] is of size 960 Bytes.

Acknowledgement

We would like to thank Dmitry Khovratovich, Tianyi Liu, Nicolas Mohnblatt, and Srinath Setty for helpful discussions on early drafts of this paper.

References

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. pages 315–334, 2018.

BBBF18. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. pages 14:1–14:17, 2018.

BCC+16. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. pages 327–357, 2016.

bea. Ethereum proof-of-stake consensus specifications. `https://github.com/ethereum/consensus-specs`.

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021, 2019. `https://eprint.iacr.org/2019/1021`.

CBBZ22. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. Cryptology ePrint Archive, Paper 2022/1355, 2022. `https://eprint.iacr.org/2022/1355`.

Esp. EspressoSystems.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

KMT22. Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. Minroot: Candidate sequential function for ethereum VDF. *IACR Cryptol. ePrint Arch.*, page 1626, 2022.

KS22. Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *IACR Cryptol. ePrint Arch.*, page 1758, 2022.

KST22. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388. Springer, 2022.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. pages 177–194, 2010.

Moh23. Nicolas Mohnblatt. Sangria: a folding scheme for plonk, 2023. `https://geometry.xyz/notebook/sangria-a-folding-scheme-for-plonk`.

ran. Randao: A dao working as rng of ethereum. `https://github.com/randao/randao`.

Set. Srinath Setty. Nova: Recursive snarks without trusted setup. `https://github.com/microsoft/Nova`.

Set20. Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020.

XCZ+22. Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. VERI-ZEXE: Decentralized private computation with universal setup. Cryptology ePrint Archive, Report 2022/802, 2022. `https://eprint.iacr.org/2022/802`.