Reverie: an end-to-end accumulation scheme from Cyclefold

Lev Soukhanov [*]

December 2023

Abstract

Recent advances in SNARK recursion and incrementally-verifiable computation are vast, but most of the efforts seem to be focused on a particular design goal - proving the result of a large computation known completely in advance.

There are other possible applications, requiring different design tradeoffs. Particularly interesting direction is a case with a swarm of collaborating provers, communicating over a peer-to-peer network - which requires to also optimize the amount of data exchanged between the participants of the swarm.

One notable such application is Ethereum's consensus, which requires to aggregate millions of signatures of individual validators.

In this technical note, we propose an informal notion of an **end-to-end** IVC scheme, which means that the amount of data that the prover needs exchange with the previous prover to continue the computation is small.

We explore the existing design space from this point of view, and suggest an approach to constructing such a scheme by combining the PlonK proof system [8] with the recent Cyclefold construction [11].

1 Purpose and organization of the text

As an opening, author wants to state the purpose of this technical note. This text is mainly written in order to compile already existing approaches into a single tool tailored for a particular design goal.

It seems that most ideas which are brought together in this text are already present in a collective consciousness of zk researchers - either scattered between different articles, buried in github issues, or as folklore. In

[*]Privacy Scaling Exporations, Ethereum Foundation. email:0xdeadfae@gmail.com

this paper, we will only gather these ideas to realise efficient end-to-end IVC. Author's own fundamental contribution is limited by a relatively efficient design of a Cyclefold circuit; and spelling explicitly on how to exactly combine plonk with Cyclefold.

Our note is organized as follows. First, we will propose an (informal) notion of an **end-to-end** IVC scheme, and analyze how various popular IVC schemes - both recursion and folding based fit into this framework. Then, we suggest a method of combining PlonK and Cyclefold to achieve an efficient end-to-end IVC. In a final part of the paper, we suggest a concrete design of a Cyclefold circuit.

Author wants to thank Yar Rebenko and Amir Ismailov for useful discussions and ongoing development of protostar-works library.

2 End-to-end accumulation context

2.1 Incrementally Verifiable Computation

Let us first recall the notion of an IVC scheme [15] (we use common modification of the original notion assuming potential non-determinism). For a potentially non-deterministic step function F, consider the sequence of iterations $(x_0, x_1, x_2, ...)$ such that $F(x_k) = x_{k+1}$.

Definition 1. *An IVC scheme is a new step function P_F, operating on pairs of the form (x_i, π_i), where π_i-s must be computationally sound proofs of knowledge for $F^i(x_0) = x_i$ - which means there exists the verifier program $V(i, x_0, x_i, \pi_i)$, complete and computationally sound in a normal sense; i.e. always accepting the iterative output of P_F, and negligible probability to come up with a passing proof for incorrect output x_i.*

Original definition is formulated for Turing machines, but here we will strongly insist on both F being represented as a circuit (with potentially small i/o and relatively large internal space), and non-determinism.

The standard approach to building IVC is accumulation schemes (which is a more relaxed notion introduced in [6], covering the spectrum from direct recursive schemes to folding schemes).

2.2 End-to-end accumulation schemes

Definition 2. *(Informal, taken from [5])*

Accumulation scheme *is a triple of algorithms P, V, D, called* ***prover***, ***verifier***, *and* ***decider*** *respectively.*

Prover takes as an input sequence of words $q_0, q_1, q_2, \ldots$ of the same language Φ and recursively outputs a sequence of ***accumulators****:*

$$acc_{i+1} = P(acc_i, q_i)$$

Verifier is a step check function, i.e. it has the form

$$V(acc_i, acc_{i+1}, q_i)$$

Decider is a final check function, i.e. it accepts the final accumulator

$$D(acc_n)$$

and this triple satisfies the trivial completeness guarantee:

$$(D(acc_i) = 1) \wedge (\Phi(q_i) = 1) \Rightarrow D(acc_{i+1}) = 1$$

and the soundness guarantee that for any efficiently generated accumulator acc_i, input q_i, new accumulator acc_{i+1} and proof π_i, with 1 - negl probability

$$(D(acc_i) = 1) \wedge (V(acc_{i+1}, acc_i, q_i) = 1) \Rightarrow D(acc_{i+1}) = 1$$

Any direct recursion scheme is evidently also an accumulation scheme, with accumulators being recursive proofs, q-s being the proofs of the separate statements, and decider coinciding with the recursive verifier.

Another example of accumulation scheme is Halo2 [3] - while each individual Halo2 proof requires linear aggregation time, their aggregation can be checked in logarithmic time.

However, this approach has its limitations - notably, sometimes the statement aggregation can be done if prover receives some additional advice string (that verifier does not need). This is captured by a notion of the **split accumulation scheme**, introduced in [5], and it differs only in a sense that q_i-s and, importantly, acc$_i$-s now have "instance" parts, accessible to everybody, and "witness" parts, accessible only to prover *and decider.*

Split accumulation schemes capture all known IVC constructions, in particular, folding. Without going into much details here, we say that in folding q-s are now not proofs, but witnesses to our original NP-statement that we are trying to aggregate + commitments to them, and accumulators are "relaxed" versions of these.

Sometimes, accumulation schemes with slow deciders are appended with additional "final step" proof, which is applied to acc$_n$ and can be verified quickly. Normal recursion schemes are also not exempt from it, because for

STARKs there is a tradeoff between recursion speed and the size of the final proof.

As evident, the idea of having large witness part of acc_i goes against our end goal - as this requires large amount of communication.

Definition 3. *(Informal)*

*In an **end-to-end accumulation scheme**, the total size of acc_i (both instance and witness) must be small. We do not require the scheme to be atomic, only that the total exchange of accumulator data is limited.*

Note 1. *We could call these schemes "succinct" or "laconic", by analogy with NARKs with corresponding properties. However, we want to stress that we are less concerned with asymptotic analysis, and more concerned with concrete efficiency - for example STARKs total size can be constant for a given security parameter, but that does not mean that exchanging megabyte-sized proofs over P2P network is acceptable. So what we are actually trying to say is that for a reasonable security parameter, like* 128 *bit, the size of acc_i should not be larger than* 10-20 *Kb - i.e. it should be concretely, practically small.*

2.3 Possible applications

We envision that any application that requires coordination of multiple proving parties over P2P network will be bandwidth-constrained, and thus requires end-to-end IVC (or, more generally, end-to-end PCD).

One such application is Ethereum's consensus, which needs to aggregate millions of individual validator votes in a relatively short timespan.

Author also envisions distributed networks in which nodes will continuously create proofs of validity of their activity, which seems like a good fit for Mutual Credit Networks and decentralized CDNs.

3 Overview of existing IVCs

3.1 Compression step

As we will soon see, most of the schemes realising IVC require some sort of a final compression SNARK to get to both the small size (let's say no more than few kilobytes), and short verification time.

We call it a "hard step", because it typically takes much more effort to compute the compression SNARK, than to do a single step of IVC.

Moreover, it seems there is a spectrum of tradeoffs - on one side are direct recursion schemes over succinct SNARKs, which do not feature any additional compression, but are relatively slow. On the other hand, IVC can be made relatively fast (achieved both by STARKs [1] and modern folding schemes [13][12][4]), but do require a relatively hard additional step.

3.2 Currently existing approaches

Currently existing approaches fall into 3 broad categories - direct recursion over SNARKs, recursion over STARKs, and other folding and accumulation schemes.

3.2.1 SNARK recursion.

The most successful instantiation of direct recursion over SNARKs is Coda protocol [2], which was used in the first version of a Mina blockchain. The main drawback is the necessity to use a cycle of curves with *both of them having pairings.* Of known constructions, this leaves us with MNT curves, which have an embedding degree 2 and so require large field size to prevent Weil descent attack.

This makes this IVC relatively inefficient compared to other constructions, though it is, of course, end-to-end.

There are also recursion schemes directly emulating non-native arithmetic of the verifier. They are not efficient.

3.2.2 STARK recursion.

STARKs are particular kind of SNARKs, using Merkle trees and Reed-Solomon proofs of proximity for polynomial commitments. They have logarithmic proof size and verification time, and, because they do not require elliptic curve arithmetic, can validate themselves over a native field (if instantiated with algebraic hash such as commonly used Poseidon hash [9]).

However, the proofs are concretely large. Moreover, there exists a tradeoff between the size of a proof and the proving time, so STARK recursion is typically done over proofs of few megabytes in size, and then features not one, but two compression steps - first into a smaller STARK proof (still roughly 100Kb in size in practical instantiations targeting 80-100 bits of security), and second into a constant-sized proof using pairing, such as Groth16 [10] or PlonK [8].

Therefore, they are hardly feasible for P2P applications we are considering, and do not count as end-to-end.

3.2.3 Accumulation and folding schemes.

Most notable accumulation scheme, is, arguably, Halo2 [3] and its versions. Recently, folding schemes such as Nova [13], Hypernova [12] and Protostar [4] came into spotlight as providing recursion potentially competetive with STARKs.

Halo2, technically, **is end-to-end**, but it is limited by a relatively large decider, and non-friendliness to pairings (as it must be instantiated over a cycle of curves with high 2-adicity). In combination, these things severely limit the size of the step-circuit that Halo2 can conveniently support, as well as interoperability with individual pairing-based proofs.

While it likely has somewhat larger overhead than the proposed scheme, Reverie, it should both be a good baseline, and is a nice choice if a (universal) trusted setup is undesirable.

Folding schemes share a common theme - they propagate a lot of accumulator witness data into the next step, but only use commitments to this data in an actual recursive verification circuit. This is, evidently, very good for recursion speed, but very bad for end-to-endness.

Nova (even enhanced with Cyclefold [11]), in particular, has a recursive overhead with witness size $\sim 10k$, which is 320 Kb of data - not including the actual step circuit, which has at least comparable or significantly larger size.

Therefore, folding schemes are not end-to-end.

4 Reverie - an end-to-end recursion using PlonK and Cyclefold

4.1 Dealing with pairings

In an attempt of emulating PlonK verifier inside of a PlonK circuit over the same field, one needs to do a significant amount of non-native arithmetic. Importantly, one needs to process some non-native elliptic curve scalar multiplications, and a single pairing of a form

$$e(A, H_0) = e(B, H_1)$$

where H_0 and H_1 are fixed elements of CRS, living in a pairing group $G2$. Notably, multiple such statements can be combined into a single one by using a random linear combination (originally, each polynomial opening reduces to a single such statement, but then they are combined to a single one).

As pairing is an extremely heavy operation, we need to deal with it. The following construction seems to be folklore (which means that the author does not know the exact reference):

Definition 4. ***Pairing-enhanced PlonK (PEP) circuit*** *is a normal PlonK circuit, with first few public inputs encoding a pair of elliptic curve points A, B. Verifier accepts the proof if and only if in addition to normal PlonK checks, $e(A, H_0) = e(B, H_1)$.*

PEP circuit should be thought of as a normal PlonK setting, in which the prover is allowed to use a single pairing operation for "free" - i.e., require the verifier to check it directly. We will say that such pairing was "deferred".

Lemma 1. *PEP verifier still requires a single pairing.*

Proof. Essentially trivial - normal PlonK checks require multiple pairings which are combined together using a random linear combinations. Additional deferred pairing $e(A, H_0) = e(B, H_1)$ is just batched together with them. □

Lemma 2. *Direct recursion scheme over PEP circuits does not require in-circuit pairings.*

Proof. From previous lemma, PEP verifier only does a single pairing. Therefore, PEP prover can defer it. □

Similarly to normal PlonK proofs, multiple PEP proofs can be batched and still require a single pairing.

Therefore, we only need to deal with non-native scalar multiplications. In order to do it, we need to introduce folding schemes and Cyclefold.

4.2 Protostar / protogalaxy folding scheme

4.2.1 Folding

Here, we present the folding scheme that we will use in our explicit construction. Normal Cyclefold is formulated for Nova paradigm, but Protostar arithmetization seems like a better fit.

Specifically, we use a scheme which is a middleground between Protostar [4] and Protogalaxy [7], and describe it in a simplified, 1-round version.

Let us consider a (single-round) arithmetic circuit of degree d over a large field $\mathbb{F}$ - which means a witness space W, public input space P, and a system of m constraints $f_i(w, p) = 0$ of degree at most d. Let $k = \lceil \log_2(m) \rceil$.

Definition 5. *Consider the following 2-round algebraic protocol (called the **protostar transform**) of an original circuit:*

1. Prover sends $w \in W$, $p \in P$.

2. Verifier picks a random value α, computes and outputs a string of challenges $a_0, a_1, ..., a_{k-1}$, with $a_i = \alpha^{2^i}$. (here, verifier can actually just send α, but it is important that computation of this string is treated as public computation happening outside of the circuit).

3. Verifier checks that

$$\sum_{i=j_0+2j_1+...+a_{k-1}2^{k-1}} a_0^{j_0} a_1^{j_1} ... a_{k-1}^{j_{k-1}} f_i(w,p) = 0$$

Lemma 3. *This 2-round algebraic protocol is statistically sound with soundness error* $\frac{m}{|\mathbb{F}|}$

Proof. Evident from the fact that the sum actually reduces to $\sum\limits_j \alpha^j f_j(w,p)$ and Schwartz-Zippel lemma. □

Note 2. *In what follows, let us denote*

$$\tilde{f}(w,p,a) = \sum_{i=j_0+2j_1+...+a_{k-1}2^{k-1}} a_0^{j_0} a_1^{j_1} ... a_{k-1}^{j_{k-1}} f_i(w,p)$$

Now, we define committed instance of the protocol, and a relaxed committed instance. The folding scheme operates on relaxed instances, so we will have an additional relaxation verifier.

Note 3. *This is a somewhat non-standard point of view. Typically, folding schemes are considered to reduce two potentially relaxed instances to a single relaxed instance. We, however, separate relaxation into a step which "reduces" non-relaxed instance to a relaxed one, and find this point of view particularly illuminating.*

Note 4. *We also avoid requiring homogeneity in any of our definitions. Our relaxed instance only incurs error-term, but not a relaxation factor.*

Let us denote by Hash a hash function used in Fiat-Shamir heuristic, and let us denote $C(w)$ a homomorphically linear collision-resistant commitment scheme from W to an elliptic curve.

Definition 6. ***Committed instance*** *is a pair* (p, c)*, with* c *being a point of elliptic curve.* ***Satisfying witness*** *to this committed instance is a pair* (p, w)*, which satisfies* $f_i(w, p) = 0$*, and* $C(w) = c$*.*

Definition 7. ***Committed relaxed instance*** *is a quadruple* (p, c, a, e) *with* c *being a point of elliptic curve,* $a = (a_0, ..., a_{k-1})$ *- challenge vector and* e *- a single field value called* ***error****.*

Satisfying witness *to this instance is a quadruple* (p, w, a, e) *such that* $C(w) = c$*, and the following "relaxed" equation holds:*

$$\tilde{f}(w, p, a) = e$$

Definition 8. ***Relaxer protocol*** *takes as an input a committed instance, and outputs a committed relaxed instance: given a pair* (p, c)*, sample random* α*, set* $a_i = \alpha^{2^i}$ *and* $e = 0$*, output* $R(p, c) = (p, c, a, e)$*.*

Lemma 4. *It is knowledge-extractable in a following standard sense - for any prover program that, for some distribution of instances, correctly runs the protocol and then outputs a witness to relaxed instance* $R(p, c)$ *there exists an extractor - program running on its code and memory state that outputs either witnesses to the original instances, or a commitment break.*

Proof. Clearly, we can set witness to the outputted relaxed witness w. If it also satisfies the original system of equations, we are done. Assume the contrary - i.e. that not all equations $f_i(w, p)$ are 0, but α is a root of a polynomial with coefficients $f_i(w, p)$.

Replay the relaxer with a different challenge α (randomly chosen one will be a non-root with 1-negl probability). By assumption, adversary manages to output relaxed witness to $R(p, c)$ with non-negligible probability - therefore, it must output a different witness.

Two witnesses to the same commitment is a commitment break. □

Note 5. *In practice, we will use Fiat-Shamir heuristic, which will set a challenge to* $\alpha = \mathrm{Hash}(p, c)$

Therefore, relaxer reduces a normal committed instance to a relaxed one.

Definition 9. ***Folding protocol*** *takes as an input two relaxed instances* (p_0, c_0, a_0, e_0) *and* (p_1, c_1, a_1, e_1) *and reduces them to a single accumulated instance:*

1. *Prover: sends instances, and a univariate polynomial* $e(t)$ *of degree* $d + k$*.*

2. *Verifier: checks that* $e(0) = e_0, e(1) = e_1$*, outputs:*

$$p_{acc} = p_0 + t(p_1 - p_0), c_{acc} = c_0 + t(c_1 - c_0), a_{acc} = a_0 + t(a_1 - a_0), e_{acc} = e(t)$$

Lemma 5. *This protocol is complete and knowledge-extractable (this is known as forking lemma).*

Proof. Completeness is clear - any honest prover just sends the restriction on a line:

$$e(t) = \tilde{f}(w(t), p(t), a(t))$$

where $w(t) = w_0 + t(w_1 - w_0)$, $p(t) = p_0 + t(p_1 - p_0)$, $a(t) = a_0 + t(a_1 - a_0)$.

Extractability holds in a same sense:

By forking the adversary 2 times and passing it different t-s we recover the (alleged) linear polynomial $w(t)$. Either it satisfies $\tilde{f}(a(t), w(t), p(t)) = e(t)$ in $t = 0, t = 1$ (in which case we have succeeded in our extraction), or it doesn't (and then it does not satisfy it in a random point too, by Schwartz-Zippel).

In this case, fork the prover once again, and pass it another random t - with 1-negl probability we will land in a point in which our claimed $w(t)$ is not satisfying. However, adversary will output a different value $w'(t)$, which *is* satisfying.

Considering they are both commitments to $c(t)$, this is a commitment break. $\square$

Definition 10. *Similar to relaxer, we will use Fiat-Shamir heuristic version of the folding protocol. Both of these bundled together will be referred as **NIFS - Non-interactive folding scheme**.*

4.2.2 Design choices

Arguably, the choice of this folding scheme requires some justification (and of course what follows will be largely independent of this explicit choice).

We are planning to use high-degree operations (in order to decrease the witness size) - therefore, it must be something like Protostar (Hypernova would likely also qualify).

We borrow the idea of using k "independent" challenges instead of 2 from Protogalaxy - this is very desirable, because in NIFS it trades off **two** elliptic curve operations for a k additional hashes. k in our case will be around $8 - 10$, which is a great deal, as high-rate hashing with Poseidon is very efficient in-circuit (a very large advantage compared to sequential hashing in Hypernova).

On the other hand, Protogalaxy's rerandomization technique is not required (as we are not doing multifolding), so we end up with a strange middleground solution.

Decision to use affine combination instead of linear is purely an aesthetic one, as it removes the need for the relaxation factor. It does not lead to any significant differences neither in proving, nor verification cost; while decreasing the cognitive load.

4.3 Reverie: dealing with non-native scalar multiplications.

All the content in this section is completely parallel to Cyclefold [11] paper - though in a different context (as our main IVC scheme is a recursion instead of a folding).

Assume that our main PlonK proof system is instantiated over a prime curve C with the base field $\mathbb{F}_q$, and scalar field $\mathbb{F}_r$. Assume also the existence of a dual curve C', with base and scalar field places switched.

Definition 11. *Cyclefold circuit is defined over $\mathbb{F}_q$. It has public i/o s, X, Y, Z, with X, Y, Z themselves being encodings of points of C, and attests to the following requirements:*

1. *X, Y, Z are valid encodings of points of C.*
2. *$sX + Y = Z$.*
3. *s is range-checked to live in some range (we will set them to ~ 128 bit).*

Concrete design of such circuit will be presented in further sections.

Definition 12. ***PEPC (PEP + Cyclefold)*** *proof accumulator instance is the following data: normal PEP proof with additional public input allocated for the hash of relaxed Cyclefold instance.*

PEPC (PEP + Cyclefold) *proof accumulator witness is the instance, plus a satisfying relaxed Cyclefold witness (note that it is a proof witness in split accumulation paradigm - the original circuit witness is still hidden).*

We say that such proof witness verifies if

1. *Normal PEP conditions hold (i.e. it is a valid PlonK proof and deferred pairing claim also holds).*
2. *Provided relaxed Cyclefold witness both commits to the additional exposed public input, and is a valid relaxed witness for the Cyclefold circuit.*

This, as we will see, allows circuit builder to use non-native scalar multiplications.

Theorem 1. ***Reverie:*** *there is an efficient end-to-end split accumulation scheme with total communication $S_P + S_C$ where S_P is the size of our PlonK proof (which depends on the exact layout), and S_C is the total size of a Cyclefold circuit.*

Proof. To accumulate any number of PEPC proofs inside of a PEPC circuit, we shall do the following:

1. Defer pairings as in normal PEP proof.
2. Verify plonk proofs directly, and for each non-native scalar multiplication, spawn a Cyclefold instance with corresponding inputs.
3. Repeatedly fold all present Cyclefold instances (both coming from PEPC proofs that we are accumulating and from all Cyclefold instances for individual non-native escalarmuls).
4. Output the obtained folding relaxed instance.

In order to perform this, it is enough to know all witnesses to the corresponding Cyclefold instances - the deferred ones are known from witness parts of the PEPC proofs, and others are known by construction. □

The rest of the paper is a proposed design of a Cyclefold circuit, and discussion on the expected recursive overhead and communication complexity.

5 Construction

5.1 Complete addition

While developing a library for Protostar, the author has written a circuit for scalar multiplication requiring ~ 500 witness size for a 128-bit scalar. This approach requires offset points, which are extremely annoying in a multi-prover case (as each prover should be given an opportunity to switch an offset point, or this would be a DoS vector).

It turned out that the complete addition formulas from [14] not only solve this problem, but actually are far more efficient from the degree standpoint. This justifies redesigning the circuit, and decreasing the size even further.

Theorem 2. *(Renes, Costello, Batina)*
An an odd-order curve with projective equation $Y^2Z = X^3 + aXZ^2 + bZ^3$ has a complete addition law of bidegree (2,2).

This will be our basic tool in what follows. Notably, affine coordinates for our curve will be chosen as an affine chart $Y = 1$, so 0 is not at infinity and can be represented as a normal affine point.

To proceed further, we need to decide on the maximal degree of our circuit. From author's experience, Protostar prover costs seem to switch from commitment-dominated regime to cross-term dominated regime at the degree $\sim 10 - 12$, though it largely depends on constraints in question and savings provided by making them higher degree.

The additional costs from PEPC circuit doing more hashing also need to be taken into account.

If the obtained witness sizes are still too large (because the application is really constrained in terms of bandwidth), we might either increase the degree further, or try to cut it into pieces and emulate a single 128-bit scalar multiplication by a pair of circuits. Both of these options are undesirable, and it is hard to say which is worse, and how much exactly. In case such a need arises, we leave this as an opportunity for future research.

5.2 Cyclefold circuit

We start by computing a short array of multiples of the point X. As we will be using base-3 decomposition, it will only consist of 3 points:

$$X[\_] = [0, X, 2X]$$

.

Now, for any array of length n we define choice polynomial (basically a short table scan):

$$\mathrm{Choice}(y, \mathrm{Arr}[\_]) = \sum_i L_i(y)\mathrm{Arr}[i]$$

where L_i is a collection of Lagrange polynomials, with $L_i(i) = 1$ vanishing in all points $0..(n-1)$ except i. Clearly, for $0 \leq y < n$ this outputs $\mathrm{Arr}[y]$.

Now, we can define our iteration constraint (which we will apply to accumulated point A in the affine form, expected result A_{next} in the affine form, array of multiples $X[\_]$, ternary digit y, and a projective scale factor λ). We denote by $\overset{\lambda}{=}$ the equality between two projective points up to a scale factor λ.

$$\text{Iter}(A, X[\_], y) : 3A \overset{\lambda}{=} A_{\text{next}} - \text{Choice}(y, X[\_])$$

This constraint attests that A_{next} was obtained from A by tripling it and adding the next ternary digit multiplied by A. The degree of this constraint is 9:

Left hand side has degree 9 - while naively it is represented as $A + 2A$, which for addition law of bidegree (2,2) gives $2 \times 1 + 2 \times 4 = 10$, actually all three coordinates of the result are divisible by y. As the formula receives the affine point with $y = 1$, the result has degree 9.

Right hand side is bidegree (2,2) subtraction of a polynomial of degree 1 and a polynomial of degree 3, so it has degree 8. Scale factor, thus, can be incorporated into right hand side, increasing its degree to 9.

The full Cyclefold circuit, works as follows:

1. Split scalar s into 81 ternary digit. Constrain them to be digits (deg 3 checks), and their linear combination to be s.
2. Compute array $X[\_]$. This only requires a witness of size 2, as $2X$ can be directly constrained to be a multiple of X.
3. Applying Iter repeatedly, constrain execution of triple-and-add method. Each invocation of Iter costs 3 additional witness elements (2 for the new point, and 1 for λ). This requires 3×81 witness elements.
4. Constrain the result sX, Y and Z to satisfy $sX + Y = Z$. This does not require additional witnesses.

Therefore, we obtain a circuit with 326 private witness size, and degree 9. This is the best we have managed to obtain for reasonable degree.

The amount of non-linear constraints is also similar.

5.3 Performance analysis.

326 witness size is our communication overhead (over a normal PlonK proof). For 256-bit field, this amounts to 10 Kb. We believe it is a good practical balance for a lot of scenarios (outperforming even the high-rate STARKs by a decimal order of magnitude and low-rate by 2.5 orders).

Considering prover performance, it is hard to evaluate without knowing the exact layout of the PlonK circuit. Clearly, for aggregation we should prefer very narrow layouts to minimize amount of elliptic curve operations.

We can try to very roughly estimate total prover work for a single non-native 128-bit MSM:

1. Prover needs to range-check inputs to the Cyclefold (let's only count range check on inputs, because output will typically be used in another Cyclefold). Assuming 16-bit lookup each rangecheck requires 16 witness elements (and for Halo2 lookup additional columns will increase total MSM imprint to 48). Considering that our inputs are 2 points and a 128 bit scalar, we get something around $48 \times 4 + 24 = 216$ witness imprint.

2. When folding Cyclefold instances, prover needs to do non-native linear combination. However, inside of a single circuit we can sidestep the necessity of doing modular reduction (by splitting each scalar into 3 limbs of bitsize ≤ 100, we can ensure that multiplying reasonable amount of such limbs by 128-bit scalars and adding them together still does not overflow). So, modular reduction is only done once in the end, and we ignore it in this analysis. To roughly account for native arithmetic on 15 (total) 100-bit limbs, we add 30 to the our (once again, very rough) estimate.

3. Main work, of course, comes from hashing. Let us estimate how many elements do we need to hash in total.

   Non-interactive relaxation: Public inputs fit into 7 elements (3×2 for points, and 1 more for the scalar), the commitment point itself is 2 more, to obtain α. All other data is bound deterministically to α, so no more hashing required. Total tally is 9 parallel hashing work.

   Non-interactive folding: we use in Fiat-Shamir the running hash of the accumulated instance, α, and cross-terms (as our circuit is of degree 9, and amount of protostar challenges is $\lceil \log_2(326) \rceil = 9$ there are 17 cross-terms). This gives 19 more hashing work.

   Notably, Poseidon hash is very efficient in-circuit in cases where hashing is parallel - standard circom implementation of Poseidon(1) has 213 constraints, and Poseidon(10) has 470.

   Sadly, this heavily uses the R1CS structure, and the fact that a big part of partial rounds computations unroll to a single, large linear constraint. In PlonK, linear combinations are not free, but the same principle can be exploited (compare with Neptune strategy https://github.com/lurk-lab/neptune). We are unaware of any current in-circuit implementation using this trick, so we can only guess what the witness imprint will be. Assuming pessimistically that Poseidon requires 700 witness elements per 10 inputs in parallel, we obtain ~ 2100 witness imprint.

4. Finally, we need to perform a native scalar multiplication operation. Once again, it largely depends on tricks used, degree of the circuit and other unknowns (we can not just naively use our Cyclefold circuit as a reference, because PlonK circuit might be of much lower degree). Let's make a pessimistic assumption of roughly 1500 witness size (naive double-and-add without any lookup optimizations).

This seems to indicate that even in the most pessimistic scenario, a single non-native scalar multiplication using this method will require no more than ~ 4000 witness elements, which shows that using it for PlonK verification is going to be entirely feasible on client-side devices.

References

[1] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. https://eprint.iacr.org/2018/046.

[2] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Paper 2020/352, 2020. https://eprint.iacr.org/2020/352.

[3] 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.

[4] 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.

[5] 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.

[6] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Paper 2020/499, 2020. https://eprint.iacr.org/2020/499.

[7] Liam Eagen and Ariel Gabizon. Protogalaxy: Efficient protostar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106, 2023. https://eprint.iacr.org/2023/1106.

[8] 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.

[9] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Paper 2019/458, 2019. https://eprint.iacr.org/2019/458.

[10] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. https://eprint.iacr.org/2016/260.

[11] Abhiram Kothapalli and Srinath Setty. Cyclefold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Paper 2023/1192, 2023. https://eprint.iacr.org/2023/1192.

[12] 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.

[13] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Paper 2021/370, 2021. https://eprint.iacr.org/2021/370.

[14] Joost Renes, Craig Costello, and Lejla Batina. Complete addition formulas for prime order elliptic curves. Cryptology ePrint Archive, Paper 2015/1060, 2015. https://eprint.iacr.org/2015/1060.

[15] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. pages 1–18, 03 2008.