Darlin: Recursive proofs using Marlin

Haböck, Ulrich
ulrich@horizenlabs.io

Garoffolo, Alberto
alberto@horizenlabs.io

Di Benedetto, Daniele
daniele@horizenlabs.io

October 1, 2021

Abstract

This document[1] describes *Darlin*, a succinct zero-knowledge argument of knowledge based on the Marlin SNARK [CHM⁺20] and the 'dlog' polynomial commitment scheme from [BCC⁺16, BBB⁺18]. Darlin addresses recursive proofs by integrating the amortization technique from Halo [BGH19] for the non-succinct parts of the dlog verifier, and we adapt their strategy for bivariate circuit encoding polynomials to aggregate Marlin's inner sumchecks across the nodes the recursive scheme. We estimate the performance impact of inner sumcheck aggregation by about 30% in a tree-like scheme of in-degree 2, and beyond when applied to linear recursion.

Contents

[1]This paper is the full version of the previously published extended abstract 'Darlin: A proof-carrying data scheme based on Marlin'.

1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) are the basis for secure decentralized computations, allowing to verify the correctness of a large number of operations by a single succinct and easy to verify cryptographic proof. Since the advent of SNARKs [Gro10, GGPR13] practical proof systems followed soon after, e.g. Pinocchio [PHGR13], Groth16 [Gro16], and Groth17 [GM17]. Whereas the first SNARKs are intrinsically connected to pairings via non-standard knowledge commitments, proof systems from the second wave, such as Sonic [MBKM19], Aurora [BSCR$^+$19], Marlin [CHM$^+$20], or Plonk [GWC19], are built in a modular way on any polynomial commitment scheme.

To scale over large amounts of data to be processed, recursive arguments or more generally *proof-carrying data (PCD) schemes* [CT10, BCCT13], are inevitable. Recursive arguments verify the existence of a previous such, and their performance is largely dependent on how efficient the verifier itself is translated into an argument. The issue of such a translation step is that typically the assertion to be proven is arithmetized (as a relation or circuit) over a field of a different characteristic than the proof itself, and simulating the arithmetics of a 'foreign' field is costly. The most common approach to tackle the problem is using a 2-cycle of elliptic curves [SS11, CCW18]. Such cycles are pairs of elliptic curves in which the subgroup of one curve is of the same prime order as the base field of the other. Applied to pairing-based SNARKs the cycle approach requires high field sizes. The only known cycles are based on MNT curves of low embedding degree [CCW18], and as such they demand field sizes beyond $1,000$ bit to meet a reasonable level of security [GS19][2]. Second wave SNARKs are not necessarily bound to pairings, hence allow to use cycles of non pairing-friendly curves [BGH19, BLH$^+$], or such in which at least one of the curves is not pairing-friendly [Ol]. Although allowing for smaller field sizes, the use of non pairing-friendly cycles introduces another issue. Due to a lack of better alternatives, such constructions apply (a variant of) the 'dlog' polynomial commitment scheme from [BCC$^+$16] the verifier of which is linear in the size of the circuit to be proven; a serious obstacle for efficient recursion. In their seminal work [BGH19], Bowe et al. showed how to overcome the problem of linear verifier size by a novel approach called *nested amortization*. In nested amortization the proof system aggregates the computational 'hard parts' of the verifier outside the circuit, reducing the verification of all of them to a single expensive check at the recursion end.

Since [BGH19] amortization schemes became an active field of research. Bünz et al. [BCMS20] gave a more modular approach to the [BGH19] concept of amortization (named accumulation scheme therein). However, their approach is less performant than the one in [BGH19], which directly integrates the amortization rounds into the argument system. Boneh et al. [BDFG20] extend the concept of amortization to *private* aggregation schemes for polynomial commitments, which allow to aggregate entire opening proofs along the nodes of a PCD scheme. An even more radical approach for rank one constraint systems (R1CS) is followed by [BCL$^+$20], who aggregate entire R1CS solutions over recursion. Although both approaches lead to a significant speed-up of recursion, they come at the cost of increased proof sizes for the PCD. The private witnesses aggregated across the nodes are as large as the circuit itself. For a Marlin verifier it is at about 1 MiB at minimum, and multiples of that in typical applications [BCL$^+$20].

In this document we describe the *Darlin* proof carrying data scheme, the recursive SNARK for a Latus sidechain of Zendoo [GKO20], a blockchain system which supports cross-chain communication. Latus sidechains are highly customizable blockchains which share the same token as the Zendoo mainchain they are bootstrapped from, and the Darlin scheme is used to provide succinct proofs of correct side chain state transitions. Darlin is based on the Marlin argument system, modified in order to handle the aggregation of both Marlin's inner sumchecks and the 'dlog' hard parts. According to our estimates, we expect the advantage of Darlin over standard Marlin (without inner sumcheck aggregation) to be about 30% when 'merging' two previous proofs, at the cost of only tripling the proof size, cf. Table 1.

[2][BMRS20] uses a cycle of MNT4/MNT6 curves with 753 bit field sizes targeting a security level of 128 bit. However, improvements on the towered number field sieve [GS19, Gui20] enforce to increase the field size up to $1,000$ bits.

Table 1: The impact of inner sumcheck amortization: Comparison of Marlin/dlog [CHM+20] versus Darlin for a PCD node which verifies two previous proofs, both using the Pasta curves [HBG], and using the optimization techniques from [BGH19] to reduce the size of the verifier in circuit. We "segmentize" dlog commitments to speed up the prover. See Section 5.5 for details. As our implementation is not ready yet, the prover times are *estimates* for an Amazon EC2 G4dn instance (4 Intel Xeon@2.5 GHz + 1 NVIDIA T4).

dlog segment size†		2^{19}	2^{18}	2^{17}
constraints	Marlin*	$\approx$ 320 k	$\approx$ 384 k	$\approx$ 520 k
	Darlin	$\approx$ 290 k	$\approx$ 320 k	$\approx$ 390 k
proof size	Marlin*	$\approx$ 4.2 kB	$\approx$ 4.6 kB	$\approx$ 5.3 kB
	Darlin	$\approx$ 15.3 kB	$\approx$ 15.7 kB	$\approx$ 16.8 kB
prover time	Marlin*	$\approx$ 16.5 s	$\approx$ 15.9 s	$\approx$ 15.38 s
	Darlin	$\approx$ 12.3 s (9.6 s**)	$\approx$ 11.7 s (9.1 s**)	$\approx$ 11.4 (8.8 s**)

*Assuming R1CS density $d = 2$, which is large enough in our applications.
**only at the two lowest levels of a proof tree, where aggregation is trivial.

The document is organized as follows. In Section 3 we describe a variant of the univariate sumcheck argument from [BSCR+19, CHM+20], inspired by the grand product argument of Plonk [GWC19]. This variant does not rely on degree bound proofs and allows a more lightweight zero-knowledge randomization. In Section 4 we informally describe our variant of Marlin, which besides using a slightly different matrix arithmetization applies the sumcheck argument from Section 3. In Section 5 we recapitulate the amortization strategy for the dlog hard parts, explain the aggregation of Marlin's inner sumcheck across multiple circuits, and summarize the main recursive argument. All formal definitions and proofs are postponed to the appendix, Section A.

2 Preliminary notes

Whenever appropriate, we formulate our protocols as *algebraic oracle proofs*, with oracles as an information-theoretic model for homomorphic polynomial commitments. An algebraic oracle proof is a multi-round protocol in which the prover responds to verifier challenges with oracles for some *low-degree* polynomials, receives another challenge from the verifier. The prover replies with some other oracles, and so on. The verifier is allowed to query these oracles for the values of any linear combination of their polynomials at any point she chooses. As in algebraic holographic proofs [CHM+20], the verifier may access some of the inputs only via oracle queries, but we do not assume that these oracles stem from a (circuit-specific) setup phase. Algebraic oracle proofs can be viewed as a variant of fully linear interactive oracle protocols [BBCGI19], considering an evaluation query as a linear functional of the polynomial to be queried. However, we shall not dwell on their separate information-theoretic security properties, nor we explicitly provide the compiler which transforms these into their corresponding ones for the resulting interactive argument systems when instantiating the oracles by a polynomial commitment scheme. Instead, the proof of our main recursive argument from Section 5.4 relies on a compiler as used implicitly in the security analysis of the batch evaluation protocol from [BDFG20].

3 A cohomological sumcheck argument

Let F be a finite field, H be a multiplicative subgroup of order n, and assume that $p(X)$ is a polynomial of arbitrary degree. The univariate *sumcheck argument* from [BSCR+19, CHM+20] is an algebraic oracle proof for showing that

$$\sum_{x \in H} p(x) = 0.$$

The sumcheck argument is the key ingredient to Marlin's way of proving a witness polynomial satisfying the rules of a given circuit (see Section 4). It is based on the fact that the above sum is equal to n times the constant term of the polynomial, if $p(X)$ is of *reduced form*, i.e. of a degree strictly less than the domain size $|H| = n$. Hence showing that the reduced form of $p(X)$ has constant term zero, i.e.

$$p(X) = X \cdot g(X) + h(X) \cdot (X^n - 1), \tag{1}$$

for some polynomials $h(X)$ and $g(X)$ whereas $deg(g(X)) < n - 1$, proves the claimed sum. To convince the verifier of (1) the prover provides the oracles for $p(X)$ $g(X)$ and $h(X)$, which we denote by

$$[p(X)], [g(X)], [h(X)],$$

together with a proof that $deg(g(X)) \leq n-1$. In response the verifier samples a random challenge $z \leftarrow\$ F$ on which the oracles are queried for $p(z)$, $g(z)$, $h(z)$. These evaluations are used to validate the identity (1) at $X = z$. In order to obtain (honest verifier) zero-knowledge, the prover samples a random 'mask' polynomial $s(X)$ of degree at least n and proves that

$$\hat{p}(X) = p(X) + s(X) \tag{2}$$

sums up to $\sigma = \sum_{z \in H} s(z)$, which is done by an ordinary sumcheck argument for $\hat{p}(X) - \sigma/n$. See [CHM+20] for the details.

Our sumcheck argument carries over the grand product argument from Plonk [GWC19] to the additive setting. Instead of using the reduced form of the polynomial $p(X)$ in question, the prover shows that the additive *cocycle* with respect to the group action of $\mathbb{Z}$ on H defined by g,

$$f_p(k, X) = \sum_{i=0}^{k-1} p(g^i \cdot X) \tag{3}$$

is a *coboundary*, which is characterized by the following folklore Lemma.

Lemma 1. *Let H be a multiplicative subgroup of a finite field F and let g be a generator of H. For any univariate polynomial $p(X)$ of arbitrary degree we have $\sum_{z\in H} p(z) = 0$ if and only if there exists a polynomial $U(X)$ such that*

$$U(g \cdot X) - U(X) = p(X) \mod (X^n - 1). \tag{4}$$

Proof. Suppose that $\sum_{z\in H} p(z) = 0$. Define $U(X)$ on H by intitializing $U(g^0) = U(1)$ to any arbitrary value, and setting

$$U(g^k) = U(1) + \sum_{i=0}^{k-1} p(g^i)$$

for $k = 1, \dots, n-1$. By definition $U(g^{k+1}) = U(g^k) + p(g^k)$ for all k, $0 \le k \le n-2$. The equation also holds for $k = n-1$, since the full cycle sum $\sum_{i=0}^{n-1} p(g^i) = \sum_{z\in H} p(z)$ vanishes. This shows that $U(g\cdot z) - U(z) = p(z)$ for all z in H, thus any extension $U(X)$ beyond H satisfies the claimed identity $U(g\cdot X) - U(X) = p(X) \bmod (X^n - 1)$. The other direction of the proof is obvious. □

The main advantage of the coboundary approach is that the algebraic oracle proof for equation (4) allows a more lightweight zero-knowledge randomization than that of equation (1): Since no reduced form is needed for $U(X)$, we can simply randomize $U(X)$ by means of the vanishing polynomial of H,

$$\hat{U}(X) = U(X) + (c_0 + c_1 \cdot X) \cdot (X^n - 1), \tag{5}$$

with uniformly random $c_0, c_1 \leftarrow\$ F$, assuming that $\hat{U}(X)$ is not queried beyond the sumcheck protocol. We describe the sumcheck argument as an algebraic oracle proof for polynomials from $R = F[X]/(X^n - 1)$ with the aim to prove that the prover knows an element from R which is subject to the sumcheck $\sum_{x\in H} p(X) = 0$.

Protocol 1 (Coboundary sumcheck)**.** *Let H be a multiplicative subgroup of a finite field F, g be a generator of H having order n. The prover is given $p(X)$ from $R = F[X]/(X^n - 1)$ subject to $\sum_{x\in H} p(x) = 0$, and the verifier is given the oracle of a random representant $\hat{p}(X) = p(X) + r(X)\cdot(X^n - 1)$, where $r(X)$ is sampled uniformly from the set of polynomials of degree strictly less than $b+1$.* [3]

(1) The prover P computes $U(X)$ of $deg(U(X)) < n$ according to the coboundary identity (4). It computes $\hat{U}(X)$ as in (5), with $c_0, c_1 \leftarrow\$ F$, and the quotient polynomial $h(X)$ satisfying

$$\hat{U}(g\cdot X) - \hat{U}(X) = \hat{p}(X) + h(X)\cdot(X^n - 1). \tag{6}$$

P then sends $[\hat{U}(X)]$, $[h(X)]$ to the verifier.

[3]The bound $b \ge 0$ corresponds to the maximum number of allowed queries for $[\hat{p}(X)]$ beyond the sumcheck protocol.

(2) The verifier V samples a random challenge $z \leftarrow\!\$\, F \setminus H$ and queries the oracles $[\hat{U}(X)]$, $[h(X)]$, and $[\hat{p}(X)]$ for their values at z. (The oracle aborts, if $z \in H$.) V uses these values to verify identity (6) *at $X = z$, and accepts if valid. (Otherwise, the verifier rejects.)*

The security analysis of Protocol 1 (applied to a specific $\hat{p}(X)$) is given in the course of the proof of Theorem 2. As a separate algebraic oracle proof it is perfectly complete and computationally knowledge sound, assuming the size of H is negligible compared to the size of the field F. It is succinct and perfectly honest verifier zero-knowledge, assuming that each the oracles $[\hat{p}(X)]$ is queried outside the protocol at most another b times (and $[\hat{U}(X)]$ is not queried at all). The latter is an immediate consequence of the fact that the conditional distribution of

$$(v_1, v_2, v_3, v_4) = \big(\hat{U}(g \cdot z), \hat{U}(z), \hat{p}(z), h(z)\big),$$

conditional to $z \notin H$, is uniform on the relation $\mathcal{R}_z = \{(v_1, v_2, v_3, v_4) \in F^4 : v_1 - v_2 - v_3 = v_4 \cdot (z^n - 1)\}$. If one instantiates the oracle with a computationally binding (Definition 8) and perfectly hiding (Definition 7) polynomial commitment scheme, the opening proof of which is an argument of knowledge (Definition 5), then the protocol is compiled into a succinct honest verifier zero-knowledge argument of knowledge.

4 Coboundary Marlin

This section describes *Coboundary Marlin*, a slight variant of the Marlin SNARK [CHM+20]. We introduce two changes: First, we replace Marlin's sumcheck argument by the coboundary argument from Section 3. Second, we[4] make use of the Lagrange kernel

$$L_n(X, Y) = \frac{1}{n} \cdot \frac{Y \cdot (X^n - 1) - X \cdot (Y^n - 1)}{X - Y} \tag{7}$$

instead of the non-normalized version $R(X, Y) = \frac{X^n - Y^n}{X - Y}$. The Lagrange kernel shares the same key properties as $R(X, Y)$. It can be evaluated *succinctly*, and allows a *practical* sumcheck representation for the bivariate circuit polynomials, as shown below. However, we point out that our favor for the Lagrange kernel is mainly for esthetic reasons. Using it allows us to argue directly with the bivariate circuit polynomials instead of a derivative in both of Marlin's sumcheck arguments as well as our aggregation strategy from Section 5.

[4]We would like to thank A. Querol for pointing out that [CFF+20] also choose the Lagrange kernel. As a consequence, our version of the lincheck is exactly the same as theirs.

4.1 Arithmetization

We assume an arithmetic circuit $\mathcal{C}$ over F being represented by a *rank-one constraint system (R1CS)*, i.e.

$$(A \cdot y) \odot (B \cdot y) = C \cdot y, \tag{8}$$

where we assume that A, B, C are $n \times n$ matrices over F, $\cdot$ is the vector matrix product and $\odot$ denotes the entry-wise (Hadamard) product of vectors. The witness vector $y \in F^n$ is composed of a public part x and a private part w, i.e. $y = (x \| w)$. Notice assuming quadratic matrices is no loss in generality, as the constraint system may always be padded with dummy constraints or variables. Moreover, we presume that $|F| - 1$ is divisible by a high power of two, assuring the existence of sufficiently large multiplicative subgroups of 2-adic order. Subgroups of such smooth order allow for a fast Fourier transform which runs in time $O(n \log(n))$, where n is the order of the subgroup. (In the sequel we call such subgroups *FFT domains*.)

In Marlin the R1CS equations are expressed over the FFT domain $H = \{z \in F : z^n - 1 = 0\}$ using Lagrange encoding. That is, given an arbitrary enumeration $\{z_1, \ldots, z_n\}$ of H a vector $y = (y_k)_{k=1}^n$ is associated with the polynomial

$$y(X) = \sum_k y_k \cdot L(X, z_k). \tag{9}$$

In other words, (y_k) is the vector of coordinates with respect to the Lagrange basis $(L(X, x_k))_k$. Therefore $y \in F^n$ is a solution of (8) if and only if its associated polynomial $y(X) = \sum_k y_k \cdot L(X, z_k)$ satisfies

$$y_A(X) \cdot y_B(X) = \sum_{z \in H} C(X, z) \cdot y(z) \mod (X^n - 1), \tag{10}$$

where

$$y_A(X) = \sum_{z \in H} A(X, z) \cdot y(z) \mod (X^n - 1), \tag{11}$$

$$y_B(X) = \sum_{z \in H} B(X, z) \cdot y(z) \mod (X^n - 1). \tag{12}$$

In these equations, $A(X, Y)$, $B(X, Y)$, $C(X, Y)$ are the bivariate polynomials with the entries of the R1CS matrices A, B, C respectively as Lagrange coordinates,

$$M(X, Y) = \sum_{i,j=1}^{n} M_{i,j} \cdot L(X, z_i) \cdot L(Y, z_j), \tag{13}$$

for $M = A, B, C$. The double sum in (13) is made amenable to a univariate sumcheck argument by indexing its non-zero terms over yet another FFT domain $K = \{w \in F : w^m - 1 = 0\}$, again assuming the existence of a sufficiently large smooth multiplicative subgroup. As in Marlin, we denote by

$$val_M(X), row_M(X), col_M(X) \in F[X]/(X^m - 1) \tag{14}$$

the polynomials of degree $< m$ which index M's non-zero values, their row and column indices (the latter two regarded as points from H, as in (13)), so that

$$M(X,Y) = \sum_{w \in K} val_M(w) \cdot L(X, row_M(w)) \cdot L(Y, col_M(w)).$$

Since $L_n(X,z) = \frac{1}{n} \cdot \frac{z \cdot (X^n - 1)}{X - z}$ whenever z is from H, we have

$$M(X,Y) = \frac{(X^n - 1) \cdot (Y^n - 1)}{n} \cdot \sum_{w \in K} \frac{val_M(w) \cdot row_M(w) \cdot col_M(w)}{(X - row_M(w)) \cdot (Y - col_M(w))} \mod (X^m - 1). \quad (15)$$

This representation, which differs slightly from [CHM+20], is the one we use for the second sumcheck argument, the 'inner sumcheck'. We assume that for $M = A, B, C$, the precomputed polynomials

$$row.col_M(X) = row_M(X) \cdot col_M(X) \mod (X^m - 1), \quad (16)$$
$$val.row.col_M(X) = val_M(X) \cdot row_M(X) \cdot col_M(X) \mod (X^m - 1), \quad (17)$$

regarded of degree $< m$, are also part of the verifier key.

4.2 The protocol

In Marlin, public circuit inputs $x = (x_i)$ define the Lagrange representation of the *input polynomial*

$$x(X) = \sum_i x_i \cdot L_\ell(x_i, X) \in F[X]/(X^\ell - 1)$$

over an properly sized input domain $I \leq H$ of size ℓ, and the full circuit state polynomial $y(X)$ is combined via

$$y(X) = x(X) + (X^\ell - 1) \cdot w(X), \quad (18)$$

using a gauged witness polynomial $w(X) \in F^{<n-\ell}[X]$. The prover provides the oracles for the private witness polynomial $w(X)$, $y_A(X)$, $y_B(X)$ and convinces the verifier of the R1CS identities (10), (11), and (12). These three identities are reduced to a single one by building a random linear combination based on a challenge $\eta \leftarrow\!\$\, F$, i.e.

$$y_\eta(X) = \sum_{z \in H} T_\eta(X,z) \cdot y(z) \mod (X^n - 1), \quad (19)$$

with

$$y_\eta(X) = y_A(X) + \eta \cdot y_B(X) + \eta^2 \cdot y_A(X) \cdot y_B(X),$$

and

$$T_\eta(X,Y) = A(X,Y) + \eta \cdot B(X,Y) + \eta^2 \cdot C(X,Y).$$

The linear identity (19) is reduced to a sumcheck over H by sampling a polynomial $R(X,\alpha)$ using a suitable kernel $R(X,Y)$, $\alpha \leftarrow\!\!\$\, F$, and applying it via scalar product to both sides of the equation. This yields

$$\sum_{z\in H} \langle R(X,\alpha), T_\eta(X,z)\rangle_H \cdot y(z) = \langle R(X,\alpha), y_\eta(X)\rangle_H,$$

hence

$$\sum_{z\in H} \langle R(X,\alpha), T_\eta(X,z)\rangle_H \cdot y(z) - R(z,\alpha)\cdot y_\eta(z) = 0. \tag{20}$$

Choosing the Lagrange kernel $L_n(X,Y)$ for $R(X,Y)$, $\langle L_n(X,\alpha), T_\eta(X,z)\rangle_H = T_\eta(\alpha,z)$, since $T_\eta(X,z)$ is of degree less than n (see Appendix A.10). Hence equation (20) is equal to

$$\sum_{z\in H} T_\eta(\alpha,z)\cdot y(z) - L_n(z,\alpha)\cdot y_\eta(z) = 0. \tag{21}$$

Equation (21) is the central identity to be proven by the protocol.

We describe the protocol as algebraic oracle proof.

Initialization

In the first step the prover computes the polynomials[5] $w(X)$, $y_A(X)$, $y_B(X) \in F[X]/(X^n - 1)$ from their Lagrange representations, and chooses random representants

$$\hat{w}(X), \hat{z}_A(X), \hat{z}_B(X) \in F^{<n+1}[X]$$

according to the sampling rule $\hat{p}(X) \leftarrow\!\!\$\, p(X) + F\cdot(X^n - 1)$ using randomizer polynomials of degree zero. It sends their oracles $[\hat{w}(X)]$, $[\hat{y}_A(X)]$, $[\hat{y}_B(X)]$ to the verifier, who returns the randomnesses $\eta \leftarrow\!\!\$\, F$ and $\alpha \leftarrow\!\!\$\, F \setminus H$ for Equation (21).

Outer sumcheck

To prove equation (21) we apply the coboundary argument from Section 3 to $\hat{p}(X) := T_\eta(\alpha,X)\cdot\hat{y}(X) - L_n(X,\alpha)\cdot\hat{y}_\eta(X)$, where

$$\hat{y}(Y) := x(Y) + (Y^\ell - 1)\cdot\hat{w}(Y),$$
$$\hat{y}_\eta(Y) := \hat{y}_A(Y) + \eta\cdot\hat{y}_B(Y) + \eta^2\cdot\hat{y}_A(Y)\cdot\hat{y}_B(Y).$$

The prover computes the boundary polynomial $U_1(X) \in F[X]/(X^n - 1)$, chooses a random representant

$$\hat{U}_1(X) \in F^{<n+2}[Y]$$

[5]Unless stated otherwise we assume polynomials $p(X)$ from $F[X]/(X^n - 1)$ of reduced form, i.e. of degree $< n$.

of it, and computes $h_1(X) \in F^{<2\cdot n+1}[X]$ for the outer sumcheck identity

$$\begin{aligned} T_\eta(\alpha, X)\cdot \hat{y}(X) - L_n(X,\alpha)\cdot \hat{y}_\eta(X) \\ = \hat{U}_1(gX) - \hat{U}_1(X) + h_1(X)\cdot(X^n - 1), \quad (22) \end{aligned}$$

where g is a generator of H. It then sends $[\hat{U}_1(X)]$, $[h_1(X)]$ together with $[T_\eta(\alpha, X)]$ to the verifier. The verifier samples another random challenge $\beta \leftarrow\$ F \setminus H$ and queries the oracles for $\hat{w}(\beta), \hat{y}_A(\beta), \hat{y}_B(\beta), T_\eta(\alpha,\beta), \hat{U}_1(g\cdot\beta)$, $\hat{U}_1(\beta), h_1(\beta)$, which are used for checking the identity (22) at $Z = \beta$.

Inner sumcheck

To prove that $T_\eta(\alpha,\beta)$ as provided by the oracle in fact stems from the circuit polynomials $M(X,Y)$, $M = A, B, C$ we adapt Marlin's inner sumcheck to our representation (15). Using these we obtain

$$T_\eta(\alpha,\beta) = \sum_{w\in K}\sum_{M=A,B,C} \eta_M \cdot \frac{val.row.col_M(w)}{(\alpha - row_M(w))\cdot(\beta - col_M(w))}, \quad (23)$$

where $(\eta_A, \eta_B, \eta_C) = \frac{(1-\alpha^n)\cdot(1-\beta^n)}{n^2}\cdot(1,\eta,\eta^2)$. We apply the coboundary sumcheck to

$$p(X) = \sum_{M=A,B,C} \eta_M \cdot \frac{val.row.col_M(X)}{(\alpha - row_M(X))\cdot(\beta - col_M(X))},$$

regarded as a reduced element from $F[X]/(X^m - 1)$. The prover computes $U_2(X)$ from $F[X]/(X^m-1)$ satisfying

$$p(X) = \frac{T_\eta(\alpha,\beta)}{m} + U_2(g_K X) - U_2(X) \mod (X^m - 1),$$

and then multiplies both sides with the denominator

$$\begin{aligned} b(X) &= \prod_{M=A,B,C} (\alpha - row_M(X))\cdot(\beta - col_M(X)) \\ &= \prod_{M=A,B,C} (\alpha\beta + \beta\cdot row_M(X) + \alpha\cdot col_M(X) + row.col_M(X)), \end{aligned}$$

where $row.col_M(X)$ are the precomputed products (16) from the prover key. This yields the *inner sumcheck* identity

$$\begin{aligned} \sum_{M=A,B,C} \eta_M \cdot val.row.col_M(X) \\ = b(X)\cdot\left(\frac{T_\eta(\alpha,\beta)}{m} + U_2(g_K X) - U_2(X)\right) + h_2(X)\cdot(X^m - 1), \quad (24) \end{aligned}$$

where g_K is a generator of K and $h_2(X) \in F^{<3\cdot m-3}[X]$. The prover sends the oracles $[U_2(X)]$ and $[h_2(X)]$ to the verifier, who samples a random challenge $\gamma \leftarrow\$ F$, on which the oracles are queried for $row_M(\gamma), col_M(\gamma), row.col_M(\gamma)$, $val.row.col_M(\gamma)$, where $M = A, B, C$, and $U_2(g_K\cdot\gamma), U_2(\gamma), h_2(\gamma)$. These values are used by the verifier to check the identity (24) at $X = \gamma$.

Security

The security analysis of Coboundary Marlin is similar to that of our main recursive argument, Theorem 2. As for Theorem 2, we stress the fact that we use the Halevi-Micali [HM98] notion of proof of knowledge with negligible knowledge error. The proof can be found in the appendix, Section A.5.

Theorem 1. *Instantiating the oracle by a computationally binding and perfectly hiding polynomial commitment scheme (Definition 7 and 8, Coboundary Marlin is a succinct, perfect honest verifier zero-knowledge (Defintion 6) argument of knowledge (Definition 5) for the R1CS relation*

$$\mathcal{R} = \big\{((A,B,C,x),w) : y = (x,w) \text{ satisfies } (A\cdot y)\odot(B\cdot y) = C\cdot y\big\}.$$

Using the Fiat-Shamir transform the interactive argument is transformed into a zk-SNARK with analog security properties in the random oracle model.

4.3 A note on performance

Marlin's outer sumcheck takes place over the FFT domain H, the size of which covers the number of constraints/variables of the constraint system. In practice circuits yield about the same number of variables as constraints, hence it is reasonable to take n the number of constraints as measure for the computational effort of the outer sumcheck, assuming a sufficiently smooth order of F^* to optimally match n. The inner sumcheck runs over the FFT domain K of size $m \approx \max_{M=A,B,C}\|M\|$ ($\|M\|$ is the number of non-zero entries in M), again under the assumption of sufficient smoothness. This domain is by the factor

$$d = \frac{\max_{M=A,B,C}\|M\|}{n}$$

larger, where d is the *R1CS density* of the circuit. The R1CS density is the average number of variables per constraint. In practice, we observed values between $d = 1.5$ and $d = 2$ for the circuits we target. (These circuits implement elliptic curve arithmetics over non-extension fields and the x^5-Poseidon hash [GKR+21] with an internal state of 3 field elements.)

5 Recursion

Our recursive scheme is based on Coboundary Marlin and the [BCMS20] variant of the *dlog polynomial commitment scheme* from [BCC+16]. We take Coboundary Marlin without inner sumcheck as succinct argument, and we aggregate both the non-succinct parts of the opening proof verifier, as well as the correctness checks usually served by the inner sumchecks, which is verifying that the commitment intended for

$$T_\eta(\alpha,Y) = \sum_{M=A,B,C}\eta_M\cdot M(\alpha,Y)$$

Table 2: Computational effort of the (coboundary) zk-Marlin prover, using an elliptic curve based linear polynomial commitment scheme. We only count fast Fourier transforms $\mathsf{FFT}(a)$ in terms of their domain size a, and elliptic curve multi scalar multiplications $\mathsf{MSM}(b)$ in terms of the number of scalars b. (Without opening proof.)

	polynomial arithm.	commit
intial round	$3\ \mathsf{FFT}(n)$	$3\ \mathsf{MSM}(n)$
outer sumcheck	$2\ \mathsf{FFT}(n) + 2\ \mathsf{FFT}(2n)$ $+3\ \mathsf{FFT}(3n)$	$2\ \mathsf{MSM}(n) + 1\ \mathsf{MSM}(2n)$
inner sumcheck	$1\ \mathsf{FFT}(m) + 1\ \mathsf{FFT}(4m)$	$1\ \mathsf{MSM}(m) + 1\ \mathsf{MSM}(3m)$
overall	$\approx (15 + 5\cdot d)\ \mathsf{FFT}(n)$	$\approx (7 + 4\cdot d)\ \mathsf{MSM}(\mathsf{n})$

in fact carries these polynomials. Aggregation of the non-succinct part of the dlog verifier (the *dlog hard parts*) relies on the same principle as introduced by Halo [BGH19]. The way we aggregate the inner sumchecks is a generalization of Halo's strategy for their circuit encoding polynomial $s(X,Y)$, and we extend it across circuits to serve a reasonable number of instances $\mathcal{C}_i = \{A_i, B_i, C_i\}$ simultaneously. As a separate 'stand-alone' protocol, our strategy may be taken as *public aggregation scheme* in the sense of [BDFG20], or an *(atomic) accumulation scheme* according to [BCMS20, BCL$^+$20]. However, for efficiency reasons we choose Halo's 'interleaved' approach instead of the blackbox constructions from [BCMS20, BCL$^+$20, BDFG20], and let the rounds of both the argument system and the aggregation scheme share the same opening proof.

In our recursive argument certain previous proof elements $(acc_i)_{i=1}^{\ell}$ called *accumulators* are 'passed' through inputs of the 'current' circuit and post-processed within the run of the current argument. Formally, $(acc_i)_{i=1}^{\ell}$ satisfy a given predicate ϕ,

$$\phi(acc_i) = 1, \quad i = 1, \ldots, \ell,$$

and are mapped to dedicated inputs of the current circuit. Beyond the rounds for proving satisfiability of the current circuit, the accumulators $(acc_i)_{i=1}^{\ell}$ are aggregated within some extra rounds into a new instance, the 'current' accumulator acc, which is again subject to $\phi(acc) = 1$. Altogether our recursive argument is of the form

$$\left\langle \mathsf{Prove}((acc_i)_{i=1}^{\ell}, (x,w), pk), \mathsf{Vf}((acc_i)_{i=1}^{\ell}, x, vk)\right\rangle,$$

where (x,w) are public and private circuit witnesses, pk and vk are the prover and verifier key for both Marlin and the aggregation scheme, and the new acc is output to both prover and verifier.

5.1 Inner sumcheck aggregation

Here, the accumulator consists of a commitment C and the succinct description of the circuit polynomial $T_\eta(z,Y)$ intended to be represented by C, i.e. the point $z \in F$ and the randomnesses $\vec{\eta} = (\eta_A, \eta_B, \eta_C) \in F^3$,

$$acc_T = (z, \vec{\eta}, C).$$

The corresponding predicate ϕ_T is satisfied if and only if C is the commitment of $T_\eta(z, Y)$ (using commitment randomness zero). The prover reduces the correctness of several accumulator instances to that of a single new one, and the verifier validates the correctness of this reduction while keeping track of the polynomial descriptions (i.e. the point z and the coefficient vector $\vec{\eta}$) by herself. We sketch the strategy assuming a single previous accumulator.

There, a previous instance $(\alpha', \vec{\eta}', C')$ is 'merged' with $(\alpha, \vec{\eta}, C)$ of the current outer sumcheck. In a first step, the prover reduces the 'multi-point', 'multi-polynomial' instance[6] $T_{\vec{\eta}'}(\alpha', Y)$, $T_{\vec{\eta}}(\alpha, Y)$ to a single-point, multi-polynomial instance

$$T_{\vec{\eta}'}(X, \beta), T_{\vec{\eta}}(X, \beta),$$

with random $\beta \leftarrow\$ F$, by providing the commitments to these new polynomials and proving consistency via polynomial testing: If the old polynomials evaluate at the challenge β to the same values as the new polynomials at the old point, respectively, then correctness of the new polynomials overwhelmingly implies that of the old ones. Using the same principle once again, correctness of the single-point multi-polynomial instance is then reduced in batch to a single-point single-polynomial instance

$$\lambda \cdot T_{\vec{\eta}'}(\alpha'', Y) + T_{\vec{\eta}}(\alpha'', Y) = T_{\lambda \cdot \vec{\eta}' + \vec{\eta}}(\alpha'', Y),$$

where $\lambda, \alpha'' \leftarrow\$ F$ are random. Note that the resulting polynomial is again of the form $T_{\vec{\eta}''}(\alpha'', Y)$ with $\eta'' = \lambda \cdot \vec{\eta}' + \vec{\eta}$. For the reduction, the prover shows that the linear combination $\lambda \cdot T_{\vec{\eta}'}(X, \beta) + T_{\vec{\eta}}(X, \beta)$ opens at the new challenge $X = \alpha''$ to the same value as the new polynomial $\lambda \cdot T_{\vec{\eta}'}(\alpha'', Y) + T_{\vec{\eta}}(\alpha'', Y)$ at the old point $Y = \beta$. Again, correctness of the new polynomial overwhelmingly implies correctness of the old ones.

Protocol 2 is regarded as a subprotocol of our complete recursive argument Protocol 4, right after the outer sumcheck. We formulate it as an algebraic oracle protocol, considering commitments as oracles.

Protocol 2 (Inner sumcheck aggregation). *Suppose that $acc'_T = (\alpha', \vec{\eta}', [T'(Y)])$ is a previous accumulator, intended to represent an oracle for $T'(Y) = T_{\vec{\eta}'}(\alpha', Y)$, and $(\alpha, \vec{\eta}, [T(Y)])$ is as provided by the prover in the current outer sumcheck, intended to represent an oracle for $T(Y) = T_{\vec{\eta}}(\alpha, Y)$, with $\vec{\eta} = (1, \eta, \eta^2)$. Aggregation of acc'_T and $(\alpha, \vec{\eta}, [T(Y)])$ is done according to the following steps immediately processed after the outer sumcheck.*

(1) *Given β, the random challenge from the outer sumcheck, the prover sends the oracles for the 'bridging polynomials'*

$$T_{\vec{\eta}}(X, \beta), T_{\vec{\eta}'}(X, \beta) \in F[X]/(X^n - 1),$$

[6]Here 'multi-point' refers to the different points α, α', and 'multi-polynomial' to the different polynomials defined by $\vec{\eta}$, $\vec{\eta}'$.

on which the verifier responds with random $\lambda, \gamma \leftarrow\!\$ F$.

(2) Given λ, γ from the verifier, the prover 'responds' with the oracle for

$$T''(Y) = T_{\vec{\eta}}(\gamma, Y) + \lambda \cdot T_{\vec{\eta}'}(\gamma, Y).$$

The verifier queries $[T_{\vec{\eta}}(X, \beta)]$, $[T_{\vec{\eta}'}(X, \beta)]$ for their corresponding values v_1, v_2 at $X = \alpha$ and α', and checks them against the values of $[T(Y)]$, $[T'(Y)]$ at $Y = \beta$, respectively. It also queries $[T''(Y)]$ at $Y = \beta$ and checks its value against that of the linear combination $[T_{\vec{\eta}}(X, \beta)] + \lambda[T_{\vec{\eta}'}(X, \beta)]$ at $X = \gamma$. If these checks succeed, then the verifier accepts and the new accumulator is

$$acc''_T = (\alpha'', \vec{\eta}'', C'') = (\gamma, \vec{\eta} + \lambda \cdot \vec{\eta}', [T'(Y)]).$$

A formal analysis of Protocol 2 is given in the course of the security proof of the complete recursive argument. As a stand-alone argument having its own opening proof, the protocol defines a *(perfectly) complete* and *sound accumulation scheme* for the predicate ϕ_T in the sense of [BCMS20]: If both acc'_T and (α, η, C) satisfy the predicate ϕ, so does acc''_T. And if $\phi(acc''_T) = 1$, then with overwhelming probability both $\phi(acc'_T)$ and $\phi(\alpha, \eta, C) = 1$.

5.2 Generalization to several circuits

The aggregation strategy from Section 5.1 is easily extended to serve multiple circuits $C_1, \ldots, \mathcal{C}_L$ simultaneously. This 'cross-circuit' generalization is especially useful in 'non-homogeneous' chemes which are composed by a variety of recursive circuits. Lets assume that the R1CS matrices A_i, B_i, C_i of the circuits $\mathcal{C}_i$, $i = 1, \ldots, L$, are padded to the same square dimension so that we may regard their

$$A_i(X, Y), B_i(X, Y), C_i(X, Y),$$

as bivariate polynomials over the same domain $H \times H$. As in the single-circuit setting we leverage the linearity of the commitment scheme and keep track of a single *cross-circuit polynomial*

$$T_H(\alpha, Y) = \sum_{i=1}^{L} T_{i,\vec{\eta}_i}(z, Y) = \sum_{i=1}^{L} \sum_{M = A_i, B_i, C_i} \eta_{M,i} \cdot M(\alpha, Y) \tag{25}$$

by means of the cross-circuit coefficient vector $H = (\vec{\eta}_1, \vec{\eta}_2, \ldots, \vec{\eta}_L)$. The *cross-circuit accumulator* for the collection $\mathcal{C} = \{\mathcal{C}_1, \ldots, C_L\}$ is of the form

$$acc_{\mathcal{C}} = (\alpha, H, C),$$

with $\alpha \in F$, coefficient vector $H = (\vec{\eta}_1, \ldots, \vec{\eta}_L) \in (F^3)^L$, and an element C from the commitment group. The corresponding predicate $\phi_{\mathcal{C}}$ is satisfied if and only if C is in fact the dlog commitment of $T_H(\alpha, Y)$, using blinding randomness zero.

5.3 Accumulating the dlog hard parts

The aggregation strategy for the non-succinct part of the dlog verifier is identical to that in [BCMS20]. The opening proof for the dlog commitment is an inner product argument that uses the folding technique from [BCC⁺16] to gradually reduce the opening claim on the initial full-length polynomial to one of half the size, until ending up with the opening claim of a single coefficient polynomial. The final committer key G_f of the opening proof is a single group element which is the result of a corresponding folding procedure on the full-length committer key of the dlog scheme. It equals the commitment of the succinct *reduction polynomial*

$$h(\vec{\xi}, X) = \prod_{i=0}^{k-1} (1 - \xi_{k-1-i} \cdot X^{2^i}), \tag{26}$$

where $k = \log|H| = n$ is the number of reduction steps and $\vec{\xi} = (\xi_i)_{i=0}^{k-1}$ their challenges. The *dlog accumulator* is of the form

$$acc_{dlog} = (\vec{\xi}, C),$$

where $\vec{\xi} \in F^k$ and C is from the commitment group, and the corresponding accumulator predicate ϕ_{dlog} is satisfied if and only if C is the commitment of $h(\vec{\xi}, X)$, using blinding randomness zero.

As Protocol 2, the aggregation strategy is regarded as a subprotocol of the complete recursive argument Protocol 4, and for efficiency reasons we reuse the challenge γ from the inner sumcheck aggregation. We again restrict to the case of a single previous accumulator.

Protocol 3 (dlog hard parts aggregation)**.** *Suppose that* $acc'_{dlog} = (\vec{\xi}', [h'(X)])$ *is a previous dlog accumulator, with* $[h'(X)]$ *representing an oracle for* $h'(X) = h(\vec{\xi}', X)$*. The following step is part of the complete recursive argument and processed immediately after Protocol 2:*

(1) The verifier queries $[h'(X)]$ *at for its value* v' *at* $X = \gamma$ *from step* (2) *of Protocol 2.*

If $v' = h(\vec{\xi}', \gamma)$ *then the verifier accepts. The new accumulator* $acc''_{dlog} = (\vec{\xi}'', C'')$ *is the one from the dlog opening proof at the end of the complete protocol.*

5.4 The main recursive argument

The complete recursive argument is a composition of Coboundary Marlin's outer sumcheck for the 'current' circuit, choosing 'zero-knowledge bound' $b = 1$, the aggregation rounds from the cross-circuit variant of Protocol 2, and Protocol 3. As in Section 5.2 we assume that the bivariate circuit polynomials $A_i(X,Y)$, $B_i(X,Y)$, $C_i(X,Y)$ are over the same domain $H \times H$, where $|H| = n$.

The query phases of these subprotocols are gathered at the end of the protocol, which is then concluded by the batch evaluation argument from [BDFG20].

We formulate the complete argument with oracles for polynomials replaced by their dlog commitments, while keeping with the same notation $[p(X)]$. For simplicity, we again restrict to the case of a single previous accumulator. The general case is straight-forward.

Protocol 4 (Complete recursive argument)**.** *Given a composed accumulator $acc' = (acc'_{\mathcal{C}}, acc'_{dlog})$, where $acc_{\mathcal{C}} = (\alpha', H', C'_T)$ is a cross-circuit accumulator for the collection $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$ and $acc'_{dlog} = (\vec{\xi}, C')$ is a dlog accumulator. The recursive argument for an instance (x, w) of the 'current' circuit $\mathcal{C}_k$ from $\mathcal{C}$ is composed by the following steps.*

(1) Intitialization for $\mathcal{C}_k$: The prover computes the gauged witness polynomial $w(X)$, $z_A(X)$, and $z_B(X)$ from $F[X]/(X^n - 1)$ and chooses random representants

$$\hat{w}(X), \hat{z}_A(X), \hat{z}_B(X) \in F^{<n+1}[X]$$

as described in Section 4. It sends their dlog commitments $[\hat{w}(X)]$, $[\hat{z}_A(X)]$, and $[\hat{z}_B(X)]$ to the verifier, who responds with $\eta, \alpha \leftarrow\$ F$.

(2) Outer sumcheck for $\mathcal{C}_k$: The prover computes

$$T_{\vec{\eta}}(\alpha, X) = \eta_A \cdot A(\alpha, X) + \eta_B \cdot B(\alpha, X) + \eta_C \cdot C(\alpha, X) \in F[Y]/(X^n - 1)$$

of the current circuit, using $\vec{\eta} = (\eta_A, \eta_B, \eta_C) = (1, \eta, \eta^2)$, and

$$\hat{U}_1(X) \in F^{<n+2}[Y], h_1(X) \in F^{<2\cdot n+1}[X]$$

subject to the outer sumcheck identity (22)*. It sends $[T_{\vec{\eta}}(\alpha, X)]$, $[\hat{U}_1(X)]$, $[h_1(X)]$ to the verifier, who returns another random challenge $\beta \leftarrow\$ F$.*

(3) Inner sumcheck aggregation, Step 1: The prover computes the 'bridging' polynomials for

$$T_{\vec{\eta}}(X, \beta), T_{H'}(X, \beta) \in F[X]/(X^n - 1),$$

and sends $[T_{\vec{\eta}}(X, \beta)], [T_{H'}(X, \beta)]$ to the verifier, who answers with another random $\lambda, \gamma \leftarrow\$ F$.

(4) Inner sumcheck aggregation, Step 2: The prover computes the cross-circuit linear combination

$$T_{H''}(\gamma, Y) = T_{\vec{\eta}}(\gamma, Y) + \lambda \cdot T_{H'}(\gamma, Y) \in F[Y]/(Y^n - 1),$$

and $[T_{H''}(\gamma, Y)]$ to the verifier.

After these steps, both prover and verifier engage in the batch evaluation argument from [BDFG20] for the dlog commitment scheme, applied to the queries as listed below. If the queried values pass the checks of the outer sumcheck, Protocol 2 and Protocol 3, and if $(acc'_{\mathcal{C}}, acc'_{dlog})$ match with the public input x of the circuit, then the verifier accepts. The new accumulator is $acc'' = (acc''_{\mathcal{C}}, acc''_{dlog})$ with[7]

$$acc''_{\mathcal{C}} = (\gamma, H'', C'') = (\gamma, \vec{\eta} \cdot \delta_k + \lambda \cdot H', [T_{H''}(\gamma, Y)]),$$

and $acc''_{dlog} = (\vec{\xi}, G_f)$ from the above batch evaluation proof.

The multi-point queries to be proven by the batch evaluation argument are as follows.

- $[\hat{w}(X)], [\hat{z}_A(X)], [\hat{z}_B(X)], [\hat{U}_1(X)], [h_1(X)]$, $[T_{\vec{\eta}}(\alpha, X)]$ at β, as well as $[\hat{U}_1(X)]$ at $g \cdot \beta$,
- $[T_{\vec{\eta}}(X, \beta)]$ at α, $[T_{H'}(X, \beta)]$ at α', and $[T_{H''}(\gamma, Y)]$, C'_T from $acc'_{\mathcal{C}}$ at β,
- $[T_{\vec{\eta}}(X, \beta)] + \lambda \cdot [T_{H'}(X, \beta)]$ at γ, and C' from acc'_{dlog} at γ.

For the sake of completeness we summarize the batch evaluation argument in Section A.8.

The following theorem states that the main recursive argument, i.e. Protocol 4 extended by the predicate check on $acc''_{\mathcal{C}}$, is a zero-knowledge argument of knowledge. We point out that we use the Halevi-Micali [HM98] notion of proof of knowledge for negligible soundness error, see Definition 4 and Defintion 5.

Theorem 2. *If the dlog commitment scheme is computationally binding (Definition 8) then Protocol 4, extended by the predicate verification on the resulting inner sumcheck accumulator $acc''_{\mathcal{C}}$, is a perfectly honest verifier zero-knowledge (Definition 6) argument of knowledge (Definition 5) for the relation*

$$\begin{aligned}\mathcal{R} = \big\{((\mathcal{C}, acc'_{\mathcal{C}}, acc'_{dlog}, x), w) : (x, w) \in R_{\mathcal{C}_k} \wedge \phi(acc'_{\mathcal{C}}) = 1 \\ \wedge\, \phi_{dlog}(acc'_{dlog}) = 1 \wedge (acc'_{C}, acc'_{dlog}) \textit{ is consistent with } x\big\},\end{aligned}$$

where $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$ is a collection of rank-one constraint systems. Here, $R_{\mathcal{C}_k}$ denotes the R1CS relation given by the circuit $\mathcal{C}_k$, and ϕ and ϕ_{dlog} are as in Section 5.2 and Section 5.3

The proof of Theorem 2 is given in Section A.4. In practice we use the Fiat-Shamir transform to turn Protocol 4 into a non-interactive argument of knowledge which is zero-knowledge against arbitrary polynomial time adversaries.

[7]Here, $\vec{\eta} \cdot \delta_k$ denotes the vector which is $\vec{\eta}$ at the position of the current circuit $\mathcal{C}_k$ in the cross-circuit accumulator, and zero elsewhere.

5.5 A note on performance

Inner sumcheck aggregation is particularly effective when the number of previous accumulators is low, as seen from the operations counts in Table 3. For a single previous accumulator ($\ell = 1$) representing the case of linear recursion, the prover effort for the recursive argument is comparable to that of standard Marlin for a circuit of R1CS density $d = 1$. Having $\ell = 4$ previous accumulators, as in our Darlin PCD scheme, the equivalent density is about $d = 1.5$.

Table 3: Recursion prover with and without inner sumcheck aggregation in terms of FFT operations and multi-scalar multiplications, for in-degree ℓ (without opening proof).

	polynomial arith.	commit
intial round	$3\ \mathsf{FFT}(n)$	$3\ \mathsf{MSM}(n)$
outer sumcheck	$2\ \mathsf{FFT}(n) + 2\ \mathsf{FFT}(2n)$ $+3\ \mathsf{FFT}(3n)$	$2\ \mathsf{MSM}(n) + 1\ \mathsf{MSM}(2n)$
aggregation rounds	$(4+\ell)\ \mathsf{FFT}(n)$	$(2+\ell)\ \mathsf{MSM}(n)$
overall	$\approx (15+\ell)\cdot\ \mathsf{FFT}(n)$	$\approx (9+\ell)\ \mathsf{MSM}(n)$
without aggregation	$\approx (15+5\cdot d)\ \mathsf{FFT}(n)$	$\approx (7+4\cdot d)\ \mathsf{MSM}(n)$

Compared to a standard Marlin prover for circuits with density $d = 2$ the performance improvement is estimated at 27%, as indicated by our estimates from Table 1 in Section 1. The timing estimates from this table are based on a detailed simulation of a Darlin prover (in terms of MSM, FFT, vector and vector-matrix operations), run on an Amazon EC2 G4dn instance (with 4 Intel Xeon@2.5 GHz and 1 NVIDIA T4) currently offered at a rate of 0.526 USD per hour. The number of constraints for verifying two previous proofs stem from detailed paper-and-pencil counts, where our circuit design follows the 'deferred arithmetics' technique from [BGH19], which postpones non-native arithmetic checks to the 'next' circuit in recursion, in which these operations are again native. (We moreover apply their endomorphism-based scalar multiplication which reduces the number of constraints significantly.) We vary over different segment sizes for the dlog commitment scheme (cf. Section A.9 on segmentation of homomorphic polynomial commitment schemes) starting with the smallest possible domain size $|H| = 2^{19}$ to cover the two verifier, and then reducing the committer key to 2^{18} and 2^{17}. Consequently, the prover times decrease at the cost of increasing proof sizes and the number of constraints for the verifier circuit.

6 Future work

We will implement Darlin as the recursive main argument of our upcoming Darlin proof carrying data suite [HB21], using a 2-cycle of ordinary elliptic curves such as the Pasta curves [HBG]. The full suite will cover pure proof merging nodes (for in-degree 1 and 2) as well as special purpose nodes with additional consensus specific logic. Beyond that a separate transformation chain of arguments for converting Darlin proofs into ordinary Marlin proofs will be provided. A formal description, including an in-depth security analysis will be given in [HB21].

7 Acknowledgements

The first author is indebted to Maus and Bowie for their appreciated feedback. Without them, the main recursive argument would miss its most important feature, the whisker feedback loop in the cross-meal aggregation of fish, chicken and beef. One of the first readers is also grateful to Peperita, that helped moving away from pairings in exchange for tasty kibble.

Bibliography

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *in IEEE Symposium on Security and Privacy*, pages 315–334, 2018.

[BBCGI19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *CRYPTO 2019*, volume 11694 of *LNCS*, pages 67–97. Springer, 2019. full version:`https://eprint.iacr.org/2019/188.pdf`.

[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. In M. Fischlin and J.S. Coron, editors, *EUROCRYPT 2016*, volume 9666 of *LNCS*. Springer, 2016. Full version: `https://eprint.iacr.org/2016/263`.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *STOC'13*, 2013.

[BCL+20] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *IACR preprint archive 2020/1618*, 2020. `https://eprint.iacr.org/2020/1618`.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. In *IACR preprint archive 2020/499*, 2020. `https://eprint.iacr.org/2020/499`.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo Infinite: recursive zk-snarks from any additive polynomial commitment scheme. In *IACR preprint archive 2020/1536*, 2020. `https://eprint.iacr.org/2020/1536`.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. In *IACR preprint archive 2019/1021*, 2019. `https://eprint.iacr.org/2019/1021`.

[BLH$^+$] Sean Bowe, Ying Tong Lai, Daira Hopwood, Jack Grigg, and Steven Smith. Halo 2. `https://github.com/zcash/halo2`.

[BMRS20] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. In *IACR preprint archive 2020/352*, 2020. `https://eprint.iacr.org/2020/352`.

[BSCR$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Y. Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019*, volume 11476 of *LNCS*. Springer, 2019.

[CCW18] Alessandro Chiesa, Lyn Chua, and Matthew Weidner. On cycles of pairing-friendly elliptic curves. In *SIAM Journal on Applied Algebra and Geometry*, volume 3(2), 2018.

[CFF$^+$20] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaï s Querol, and Hádrian Rodríguez. Lunar: a toolbox for more efficient and updatable zkSNARKs and commit-and-prove extensions. In *IACR preprint archive 2020/1069*, 2020. `https://eprint.iacr.org/2020/1069`.

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT 2020*, volume 12105 of *LNCS*, 2020.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS'10*. Tsinghua University Press, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Brian Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Nguyen P.Q., editor, *EUROCRYPT 2013*, volume 7881 of *LNCS*. Springer, 2013.

[GKO20] Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov. Zendoo: a zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. In *IACR preprint archive 2020/123*, 2020. `https://eprint.iacr.org/2020/123`.

[GKR$^+$21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roz, and Markus Schofnegger. POSEIDON: A new hash function for zero-knowledge proof systems. In *USENIX Security Symposium 2021*, 2021.

[GM17] Jens Groth and Mary Maller. Snarky Signatures: Minimal signatures of knowledge from simulation-extractable snarks. In Shacham H. Katz J., editor, *CRYPTO 2017*, volume 10402 of *LNCS*. Springer, 2017.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Abe M., editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*. Springer, 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In M. Fischlin and J.S. Coron, editors, *EUROCRYPT 2016*, volume 9666 of *LNCS*. Springer, 2016.

[GS19] Aurore Guillevic and Shashank Singh. On the alpha value of polynomials in the tower number field sieve algorithm. In *https://hal.inria.fr/hal-02263098*, 2019.

[Gui20] Aurore Guillevic. A note on MNT4 and MNT6 curves: Estimation of STNFS cost. (personal communication), 2020.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical non-interactive arguments of knowledge. In *IACR preprint archive 2019/953*, 2019. https://eprint.iacr.org/2019/953.

[HB21] Ulrich Haböck and Daniele Di Benedetto. The Darlin proof carrying data protocol suite. in preparation, 2021.

[HBG] Daira Hopwood, Sean Bowe, and Jack Grigg. The Pasta Curves for Halo 2 and beyond. https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/.

[HM98] Shai Halevi and Silvio Micali. More on proofs of knowledge. In *IACR preprint archive 1998/015*, 1998. Appeared in the THEORY OF CRYPTOGRAPHY LIBRARY and has been included in the ePrint Archive. https://eprint.iacr.org/1998/015.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *ACM SIGSAC Conference on Computer and Communication Security*, pages 2111–2128, 2019.

[Ol] O1-labs. Mina Protocol. https://minaprotocol.com.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *in IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[SS11] Joseph H. Silverman and Katherine E. Stange. Amicable pairs and aliquot cycles for elliptic curves. In *Experimental Mathematics*, volume 20(3), 2011.

[WTS+18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zk-SNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943, 2018.

A Appendix

A.1 Notation

We denote the security parameter by λ, where we throughout consider it in unary representation. A function $f(\lambda)$ is *negligible* if for every polynomial $p(\lambda)$, it holds that $\lim_{\lambda \to \infty} f(\lambda) \cdot p(\lambda) = 0$, or in short $f(\lambda) = o(1/p(\lambda))$.

Probabilistic algorithms are denoted by capital letters A, B, etc., and we write $y \leftarrow \mathsf{A}(x)$ if an algorithm A ouputs a string y given an input string x while using some internal random coins r uniformly sampled from $\{0,1\}^*$. Whenever we need to refer to the used random coins r, we shall explicitly write $y = \mathsf{A}(x;r)$. We say that A is *probabilistic polynomial time (p.p.t.)*, if its run time $T_{x,r}$ on input x and internal random coins r is bounded by some fixed polynomial $p(|x|)$ independent of the random coins, where $|x|$ denotes the length of its input. We say that A is *expected polynomial time* if the expected run time $E(T_{x,r})$, where the expectation is taken over all random coins r, is bounded by some polynomial in the length of the input. The interaction of two interactive probabilistic algorithms A and B is denoted $\langle \mathsf{A}, \mathsf{B} \rangle$, where we explicitly clarify what are the inputs and outputs of both algorithms.

A.2 Interactive arguments

Let $\mathcal{R}$ be a polynomial time decidable binary relation. An interactive argument system for $\mathcal{R}$ consists of three probabilistic polynomial time algorithms

$$(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf}).$$

Given the security parameter λ in unary representation, $\mathsf{Setup}(\lambda)$ outputs a common reference string crs which supports all statement-witness pairs (x,w) up to a certain maximum length $N = N(\lambda)$, which we write in short $(x,w) \in \mathcal{R}_N$. Given $(x,w) \in \mathcal{R}_N$, the algorithms Prove and Vf are used to interactively reason about whether x belongs to the language defined by $\mathcal{R}$ or not. We denote their interaction by $tr \leftarrow \langle \mathsf{Prove}(x,w), \mathsf{Vf}(x) \rangle$ with tr as the transcript of the interaction, and we assume that both algorithms have access to the *crs* without explicitly declaring them as inputs. After at most polynomially many steps the verifier accepts or rejects, and we say that *tr* is accepting or rejecting.

Definition 3 (Perfect completeness)**.** An interactive argument system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ satisfies *perfect completeness* if

$$\Pr\left[\begin{array}{l}\langle \mathsf{Prove}(x,w), \mathsf{Vf}(x)\rangle \\ \text{is accepting}\end{array}\middle|\begin{array}{l} crs \leftarrow \mathsf{Setup}(\lambda), \\ (x,w) \leftarrow \mathcal{A}(\lambda), \text{ with} \\ (x,w) \in \mathcal{R}_N \end{array}\right] = 1.$$

We define knowledge-soundness in the style of [HM98]. However we do not dwell on the structure or the message distribution of the blackbox extractor. The reason for this choice of definition is the modularity of our proof of Theorem 2, which refers to the security result on the batch evaluation argument from [BDFG20].

Definition 4 (Knowledge-soundness)**.** An interactive argument system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ for the relation $\mathcal{R}$ is *knowledge sound* if for every x from $\mathcal{L}_{\mathcal{R}}$ and every adversary $\mathcal{A}$ which makes $\langle \mathcal{A}, \mathsf{Vf}(x)\rangle$ accept with non-negligible probability $\varepsilon(x) > \mathsf{negl}(\lambda)$, there is a strict polynomial time algorithm $\mathcal{E} = \mathcal{E}^{\mathcal{A}}$ with blackbox access to $\mathcal{A}$ which does at most $\mathsf{poly}(|x|, \lambda)/\mathsf{poly}(\varepsilon)$ calls, and overwhelmingly outputs a witness w such that $(x,w) \in \mathcal{R}$.

Definition 5. We say that an interactive argument system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ is an *argument of knowledge*, if it is perfectly complete and knowledge sound as defined above. It is said to be *succinct*, if the size of the transcript is sublinear in the size of $(x,w) \in \mathcal{R}$.

As we do not require any trust assumptions for the setup, our definition of zero-knowledge does not make use of trapdoors.

Definition 6 (Perfect honest verifier zero-knowledge)**.** An interactive argument system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ is *perfect honest verifier zero-knowledge* if there is a p.p.t. algorithm Sim such that for every p.p.t. algorithm $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}(x,w) \in \mathcal{R}_N \\ \wedge \\ A(tr') = 1\end{array}\middle|\begin{array}{l} crs \leftarrow \mathsf{Setup}(\lambda), \\ (x,w) \leftarrow \mathcal{A}(crs), \\ tr' \leftarrow \mathsf{Sim}(crs, x) \end{array}\right]$$
$$= \Pr\left[\begin{array}{c}(x,w) \in \mathcal{R}_N \\ \wedge \\ \mathcal{A}(tr) = 1\end{array}\middle|\begin{array}{l} crs \leftarrow \mathsf{Setup}(\lambda), \\ (x,w) \leftarrow \mathcal{A}(crs), \\ tr \leftarrow \langle \mathsf{Prove}(x,w), \mathsf{Vf}(x)\rangle \end{array}\right].$$

A.3 Forking Lemmas

We use the forking Lemma from [BCC+16] and we obtain strict polynomial time of the sampling algorithm by truncation. Assume that $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$

is an $(2r+1)$-move public-coin argument, by which we mean that in each round the verifier messages are chosen uniformly at random from a sample space S. Given a transcript tr resulting from the interaction of $\mathcal{A}(\ .\ )$ with the verifier $\mathsf{Vf}(crs, x)$, we denote by $tr|_{\leq i}$, with $i = 0, \ldots, r$, the partial transcript consisting of the prover and verifier messages of the first $2i+1$ moves. An $(n_1, \ldots, n_r)$-*tree of accepting transcripts* is a tree of depth r which is rooted in a prover's first message $tr|_{\leq 0}$ and in which each node at level $i \in \{0, \ldots, r-1\}$ represents a partial transcript $tr|_{\leq i-1}$ and has exactly n_i children nodes extending this transcript. The tree has overall $K(\lambda) = \prod_{i=1}^{r} n_i$ leafs standing for complete transcripts in which the verifier eventually accepts. We assume that the size of S grows superpolynomially in λ, so that

$$\Pr[x_1 \neq x_2 | x_1, x_2 \leftarrow\$\, S] > 1 - \mathsf{negl}(\lambda).$$

Lemma 2 ([BCC⁺16]). *Let* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ *be a* $(2r+1)$*-move public-coin interactive proof, and* $\mathcal{A}$ *a p.p.t. adversary which runs in expected time* $t_{\mathcal{A}}$ *and succeeds* $\langle \mathcal{A}(\ .\ ), \mathsf{Vf}(crs, x)\rangle$ *with non-negligible probability* $\varepsilon = \varepsilon(x)$ *on public input* x*. If* $n_1, \ldots, n_r \geq 2$ *are such that* $K(\lambda) = \prod_{i=1}^{r} n_i$ *is polynomially bounded, then there exists a p.p.t. algorithm* $\mathcal{T}$ *that calls the next message function of* $\mathcal{A}$ *at most* $2 \cdot K(\lambda)/\varepsilon$ *times and with non-negligible probability* $\varepsilon/2$ *outputs an* $(n_1, \ldots, n_r)$*-tree of accepting transcripts in which all pairs of sibling-node challenges* x_1, x_2 *are subject to* $x_1 \neq x_2$*.*

For the sake of completeness we shortly sketch the construction of $\mathcal{T}$ as claimed in Lemma 2. The tree finder algorithm $\mathcal{T}'$ from [BCC⁺16] is a rejection sampler which is allowed to fail at every first completion of $\langle \mathcal{A}, \mathsf{Vf}(crs, x)\rangle$ on a given partial transcript $tr|_{\leq i}$. It succeeds with probability

$$\begin{aligned} p_{tr|_{\leq i}} &= \Pr[\mathcal{T}' \text{ completes subtree for } tr|_{\leq i}] \\ &= \Pr[\langle \mathcal{A}(\ .\ ), \mathsf{Vf}(crs, x)\rangle \text{ succeeds on } tr|_{\leq i}]. \end{aligned}$$

and runs in expected polynomial time

$$\mathbb{E}[\# \text{ of } \mathcal{A} \text{ calls given } tr|_{\leq i}] \leq n_i \cdot n_{i+1} \cdot \ldots \cdot n_r.$$

Overall $\mathcal{T}'$ is of expected polynomial time calling $\mathcal{A}$ at most $K(\lambda)$ times on average, and $\mathcal{T}'$ succeeds in producing a complete $(n_1, ..., n_r)$-tree of accepting transcript with probability ε. The probability that such a complete tree of accepting transcripts has collisions (i.e. two sibling challenges coincide) is negligible, see the full version of [BCC⁺16]. Finally limiting the run time of $\mathcal{T}'$ to $2 \cdot K(\lambda)/\varepsilon$ calls of $\mathcal{A}$ (and returning $\perp$ in that case) yields a strict polynomial time algorithm which still succeeds with a probability of at least $\varepsilon/2$.

Lemma 3 ([BDFG20]). *Let* δ *be such that* $0 < \delta \leq \frac{\varepsilon^2}{8K(\lambda)}$*. Then with probability at least* $\varepsilon/4$ *the tree finding algorithm* $\mathcal{T}$ *from Lemma 2 outputs a tree of accepting transcripts with the following property: For every partial transcript* $tr|_{\leq i}$ *of length* i *in the tree, the conditional success probability for* $\mathcal{A}(\ .\ )$ *continuing the partial transcript* $tr|_{\leq i}$ *is at least* δ*.*

Proof. The tree sampler $\mathcal{T}$ from Lemma 2 tests at most $2 \cdot K(\lambda)/\varepsilon$ partial transcripts $tr|_{\leq i}$. Such transcript succeeds with the probability $p_{tr|_{\leq i}}$ as above. Therefore, the probability that one of these $p_{tr|_{\leq i}}$ is smaller than a given δ is at most $2 \cdot K(\lambda)/\varepsilon \cdot \delta$. Choosing the latter of at most $\varepsilon/4$ yields the assertion of the lemma. $\square$

We note that the factors 1/2 in Lemma 2 and 1/8 in Lemma 3 are arbitrary. Any other choice of these factors $> 1 - 1/\mathsf{poly}(\lambda)$ is possible.

A.4 Proof of Theorem 2

Theorem 2 is subject to Protocol 4, extended by the verification of the resulting $acc'' = (\gamma, H'', [T_{H''}(\gamma, Y)])$. We refer to this extended protocol as the full protocol. The batch evaluation proof Eval is regarded as a subprotocol, and we use knowledge soundness and honest verifier zero-knowledge of it (Theorem 11 and Theorem 10) to infer the same properties for the full protocol.

Knowledge soundness

Assume maximum degree for the polynomial commitment scheme is $d = \mathsf{poly}(\lambda)$ with $d \geq 2 \cdot n + b$.The proof is divided into two steps. In the first one, we special soundness of the algebraic oracle proof. The second step uses the transcript sampler from Lemma 2 to construct a strict polynomial time extractor from the strict polynomial time extractor of the batch evaluation argument.

Step 1. Special soundness.

Consider the protocol as an interactive oracle proof where the oracles are guaranteed having a degree of at most $d = \mathsf{poly}(\lambda)$. Besides the arithmetic checks on the evaluation claims, the verifier checks the oracle for $T_{H''}(\gamma, Y)$ by reading it in full length[8] and compare it against the polynomial described by H'' as computed in the protocol. We claim that this 'algebraic oracle proof' is (m_1, m_2, m_3, m_4)-special sound, with

$$(m_1, m_2, m_3, m_4, m_5) = (3, n, 2 \cdot d + 1, 2, d + 1),$$

in the following sense: Given an $(m_1, m_2, m_3, m_4, m_5)$-tree of accepting transcripts with pairwise distinct verifier challenges for $\eta \in F, \alpha, \beta \in F \setminus H, \lambda \in F, \gamma \in F$, respectively, then

- the polynomial in the oracle from acc'_{dlog} is the claimed reduction polynomial $h(\vec{\xi}', X)$,
- the oracles intended for $T_{H'}(\gamma', Y)$ and $T_\eta(\alpha, Y)$ in fact carry the correct polynomials, and

[8]Equivalently, the verifier may query the oracle at $d+1$ different points and reconstructs the polynomial from the values.

- the polynomial $y(X) = x(X) + (X^\ell - 1) \cdot \hat{w}(X) \bmod (X^n - 1)$ with $\hat{w}(X)$ from $[\hat{w}(X)]$ satisfies the R1CS identities (10), (11), (12).

This is true for the following reasons:

- $m_1 = 3$ pairwise distinct $\eta_1, \eta_2, \eta_3 \in F$ are sufficient to derive the R1CS identities (10), (11), (12) from the 'lincheck' identity (19). (The Vandermonde matrix for different choices of η is invertible.)

- $m_2 = n$ pairwise distinct $\alpha_1, \ldots, \alpha_n \in F \setminus H$ allow inverting the reduction of the lincheck identity (19) to the sumcheck identity (1) by means of the Lagrange kernel. (Recall that the sumcheck is obtained from the lincheck by applying $\langle L_n(X, \alpha), . \rangle$. By Lemma 5 the inner products for any n different values of α allow to uniquely reconstruct the lincheck polynomial modulo $(X^n - 1)$.)

- $m_3 = 2 \cdot d + 1$ pairwise distinct $\beta_1, \ldots, \beta_{2d+1} \in F \setminus H$ are sufficient to infer the outer sumcheck identity on the full domain F, as well as the identity for the first step of the inner sumcheck aggregation. (The polynomials are of degree at most d.)

- $m_4 = 2$ distinct $\lambda_1, \lambda_2 \in F$ allow for reconstructing the component polynomials from their linear combination $T_{\vec{\eta}}(X, \beta) + \lambda \cdot T_{H'}(X, \beta)$ in the second step of the inner sumcheck aggregation. (Again, since the Vandermonde matrix is invertible.)

- and $m_5 = d + 1$ pairwise distinct $\gamma_1, \ldots, \gamma_{d+1} \in F$ are sufficient to infer both the correctness of the polynomial behind the linear combination $[T_{\vec{\eta}}(X, \beta)] + \lambda \cdot [T_{H'}(X, \beta)]$, and the polynomial behind the oracle in acc'_{dlog}. (Again, all polynomials are of degree at most d.)

Step 2. Extractor.

Suppose that $\mathcal{A}$ is a probabilistic polynomial time adversary which succeeds the complete verifier with non-negligible probability ε on given inputs ($\mathcal{C}$, $acc'_{\mathcal{C}}$, acc'_{dlog}, x). As $K(\lambda) = m_1 \cdot \ldots \cdot m_5$ is polynomial in λ, Lemma 2 guarantees a strict polynomial time algorithm $\mathcal{T}$ which calls $\mathcal{A}$ at most $2 \cdot K(\lambda)/\varepsilon = \mathsf{poly}(\lambda)$ times and succeeds with a non-negligible probability of $\varepsilon/2$ in sampling an $(m_1, m_2, m_3, m_4, m_5)$-tree of accepting transcripts as needed for Step 1. Each partial transcript $tr|_{i \le 5}$ records the messages until and including the sampling of the last verifier challenge γ, before entering the batch evaluation protocol Eval. By Lemma 3 we may assume that for each of these partial transcripts $tr|_{i \le 5}$, the probability that $\mathcal{A}$ succeeds on it is at least $\delta = \frac{\varepsilon^2}{8 \cdot K(\lambda)} > \mathsf{negl}(\lambda)$. By knowledge-soundness of the batch evaluation argument Eval, there is a strict polynomial time extractor which calls $\mathcal{A}$ at most $\mathsf{poly}(\lambda)/\mathsf{poly}(\delta)$ times on each of the transcripts $tr|_{i \le 5}$ and outputs the witness polynomials from $F^{<d+1}[X]$ (including commitment randomnesses) for

- $[\hat{w}(X)], [\hat{z}_A(X)], [\hat{z}_B(X)], [\hat{U}_1(X)], [h_1(X)]$, $[T_{\vec{\eta}}(\alpha, X)]$,
- $[T_{H'}(\alpha', X)]$ from $acc'_{\mathcal{C}}$, as well as $[T_{\vec{\eta}}(X,\beta)]$, $[T_{H'}(X,\beta)]$,
- $[T_{\vec{\eta}}(X,\beta)] + \lambda \cdot [T_{H'}(X,\beta)]$, and $[h(\vec{\xi}', X)]$ from acc'_{dlog},

of each of the $(\eta, \alpha, \beta, \lambda, \gamma) = (\eta_{i_1}, \alpha_{i_2}, \beta_{i_3}, \lambda_{i_4}, \gamma_{i_5})$ in the transcript tree. These polynomials have values which pass all the verifier checks of the protocol. Assuming the dlog commitment is computationally binding, the witness polynomials and commitment randomnesses for $[\hat{w}(X)]$, $[\hat{z}_A(X)]$, $[\hat{z}_B(X)]$, $[\hat{U}_1(X)]$, $[h_1(X)]$, as well as the witness polynomials for the accumulator commitments $[T_{H'}(X,\beta)]$, $[h(\vec{\xi}', X)]$ overwhelmingly coincide for all the verifier challenges $(\eta_{i_1}, \alpha_{i_2}, \beta_{i_3}, \lambda_{i_4}, \gamma_{i_5})$, and for the same reason the polynomials for $[T_{\vec{\eta}}(\alpha, X)]$ do not depend on $(\beta_{i_3}, \lambda_{i_4}, \gamma_{i_5})$. Hence if we replace the commitments by these polynomials we obtain an $(m_1, m_2, m_3, m_4, m_5)$-tree of accepting transcripts as needed for Step 1 to conclude that $y(X) = x(X) + (X^\ell - 1)\cdot \hat{w}(X) \bmod (X^n - 1)$ satisfies the R1CS identities for $\mathcal{C}_k$, and both acc'_T and acc'_{dlog} are correct. The overall run-time of the extractor is strictly bounded by $K(\lambda)\cdot \mathsf{poly}(\lambda)/\mathsf{poly}(\delta) = \mathsf{poly}(\lambda)/\mathsf{poly}(\varepsilon)$ and succeeds with a non-negligible probability of at least $\varepsilon/4$. By amplification we obtain the claimed extractor for knowledge-soundness.

Zero-knowledge

Perfect honest verifier zero-knowledge of Protocol 4 is an immediate consequence of perfect honest verifier zero-knowledge of the batch evaluation argument (Theorem 11) and the same property for the coboundary outer sumcheck. The latter is obtained from the following auxiliary lemma. The proof of it is straightforward, and we leave it to the reader.

Lemma 4. *Assume that P follows Protocol 4. Then the conditional distribution of* $(v_1, v_2, v_3, v_4, v_5, v_6) = (\hat{w}(\beta), \hat{y}_A(\beta), \hat{y}_B(\beta), \hat{U}(g\cdot\beta), \hat{U}(\beta), h(\beta))$*, conditional to* (η, α, β)*, is uniform on the relation* $\mathcal{R}_{\eta,\alpha,\beta}$ *of all* $(v_1, v_2, v_3, v_4, v_5, v_6)$ *satisfying the outer sumcheck equation*

$$T_\eta(\alpha,\beta)\cdot(x(\beta) + (\beta^\ell - 1)\cdot v_1) - L_n(\beta,\alpha)\cdot(v_2 + \eta\cdot v_3 + \eta^2 \cdot v_2 \cdot v_3) \\ - v_4 + v_5 = v_6 \cdot (\beta^n - 1).$$

Using Lemma 4 the simulator for the outer sumcheck is constructed as follows. Given a consistent previous accumulator $acc' = (acc'_{\mathcal{C}}, acc'_{dlog})$ and any circuit input $x \in F^\ell$, it first samples η and (α, β) uniformly from F and $(F \smallsetminus H)^2$, respectively, and $(v_1, v_2, v_3, v_4, v_5, v_6)$ uniformly from $\mathcal{R}_{\eta,\alpha,\beta}$ (by choosing $v_1, \ldots, v_5$ uniformly from F and v_6 as determined by the outer sumcheck equation), and then crafts arbitrary polynomials $\hat{w}(X)$, $\hat{y}_A(X)$, $\hat{y}_B(X)$, $\hat{U}(X)$ and $h(X)$ of degree less than d which evaluate at $X = \beta$ (and $\hat{U}(X)$ also at $X = g\cdot\beta$) to the corresponding values. All these polynomials are committed using hiding randomnesses, and the aggregation rounds Step (3) and Step (4) of Protocol 4 are performed as in an honest prover-verifier interaction. Since the dlog commitment scheme is perfectly hiding, the resulting conversation is identically distributed as in an ordinary honest prover-verifier interaction for Step (1-4). These conversations are completed by calling the simulator for the batch evaluation argument on the collected commitments. Since the latter partial transcripts are identically distributed to an honest prover-verifier interaction, so are the completed conversations.

A.5 Proof of Theorem 1

The proof of Theorem 1 is almost identical to that of Theorem 2, hence we only point out the differences. We assume that the indexer polynomial $row_M(X)$, $col_M(X)$, $row.col_M(X)$ and $val.row.col_M(X)$, $M = A, B, C$, as defined in Section 4 are verified against their commitments in a precomputation phase of the protocol.

For knowledge-soundness, observe that Coboundary Marlin viewed as an algebraic oracle proof, is (m_1,m_2,m_3,m_4)-special sound with

$$(m_1, m_2, m_3, m_4) = (3, n, 2 \cdot d + 1, 4 \cdot m - 3).$$

Here, $d = \mathsf{poly}(\lambda)$ is the maximum degree of the oracle polynomials, n and m are the sizes of the domains H and K, and m_1, m_2, m_3, m_4 correspond to the verifier challenges $\eta \in F$, $\alpha, \beta, \gamma \in F \setminus H$, respectively. The reasoning for m_1, m_2, m_3 is as befor, and $m_2 = 4 \cdot m - 2$ by the degree of the inner sumcheck identity (24). Based on these modified soundness numbers, the strict polynomial time extractor is constructed as in Section A.4 by using the forking lemma (Lemma 2) and the extractor for the batch evaluation argument.

The proof for perfect honest verifier zero-knowledge can be taken over almost verbatim, replacing completing of the simulated transcript for the outer sumcheck by an honest prover-verifier run of the inner sumcheck instead of the aggregation rounds.

A.6 Polynomial commitment schemes

We regard a polynomial commitment scheme consisting of four probabilistic polynomial time algorithms

$$(\mathsf{Setup}, \mathsf{Com}, \mathsf{Prove}, \mathsf{Vf}).$$

Given the security parameter λ, $(ck, vk) \leftarrow \mathsf{Setup}(\lambda)$ generates a common reference string consisting of a committer key ck and a verifier key vk supporting polynomials over some finite field F having degree of at most $N = N(\lambda)$, where N is polynomial in λ. Given the committer key ck, the commitment of a polynomial $p(X)$ of degree at most N is computed by $C = \mathsf{Com}(p(X); r)$, where r denotes the used random coins. (We again omit ck from the inputs for brevity.)

We regard $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ as a succinct interactive argument system for the relation

$$\mathcal{R} = \left\{((C, x, v), (p(X), r)) \ : \ C = \mathsf{Com}(p(X); r) \wedge p(x) = v\right\},$$

and call the polynomial commitment scheme to satisfy completeness, zero-knowledge and witness-extended emulation if the interactive argument system does. We refer to the interaction $\langle \mathsf{Open}, \mathsf{Vf}\rangle$ as opening proof for the polynomial $p(X)$ at the point $x \in F$.

The security notions *computational binding* and *perfect hiding* are as for general non-interactive commitment schemes $(\mathsf{Setup}, \mathsf{Com})$. For the sake of brevity, we directly cite them applied to polynomial commitment schemes:

Definition 7 (Perfect Hiding)**.** We say that a polynomial commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Vf})$ is *perfectly hiding* if every polynomial adversary $\mathcal{A}$ has no advantage over pure guessing when distinguishing the commitments of two adversarially chosen polynomials,

$$\Pr\left[b^\star = b \;\middle|\; \begin{array}{l} (ck, vk) \leftarrow \mathsf{Setup}(\lambda), \\ \text{for } i = 1, 2 \\ \quad p_i(X) \leftarrow \mathcal{A}(ck, vk),\ \deg(p_i(X)) \le N(\lambda), \\ \quad C_i \leftarrow \mathsf{Com}(p_i(X)) \\ b \leftarrow\!\$\, \{0, 1\},\ b^\star \leftarrow \mathcal{A}(C_b, C_{1-b}) \end{array} \right] = \frac{1}{2}.$$

Definition 8 (Computational Binding)**.** A polynomial commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Vf})$ is *computationally binding* if for every p.p.t. adversary $\mathcal{A}$, the probability to find two different messages $(p_i(X), r_i)$, $i = 1, 2$, having the same commitment is negligible:

$$\Pr\left[\begin{array}{c} \mathsf{Com}(p_1(X); r_1) = \mathsf{Com}(p_2(X); r_2) \\ \wedge \\ (p_1(X), r_1) \neq (p_2(X), r_2) \end{array} \;\middle|\; \begin{array}{l} (ck, vk) \leftarrow \mathsf{Setup}(\lambda), \\ (r_1, p_1(X), r_2, p_2(X)) \leftarrow \mathcal{A}(ck, vk) \end{array} \right] \\ = \mathsf{negl}(\lambda).$$

We further make use the notion of a homomorphic schemes, again directly applied to polynomial commitment schemes:

Definition 9 (Homomorphic commitment)**.** A polynomial commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Vf})$ with commitments in a *commitment group* $(\mathcal{G}, +)$ is *homomorphic* (or, *linear*), if

$$\mathsf{Com}(p_1(X); r_1) + \mathsf{Com}(p_2(X); r_2) = \mathsf{Com}(p_1(X) + p_2(X); r_1 + r_2).$$

A.7 The dlog commitment scheme from [BCMS20]

The dlog polynomial commitment scheme from [BCMS20] is an ordinary Pedersen vector commitment. Given the coefficient vector $\vec{c} = (c_i)_{i=0}^{d-1}$ of a polynomial

$p(X)$ from $F[X]$ of degree at most d, its commitment is the Pedersen linear combination

$$\mathsf{Com}(p(X); r) = r \cdot S + c_0 \cdot G_0 + \ldots + c_{d-1} \cdot G_{d-1},$$

where S and (G_i) is the committer key, and the optional hiding randomness r is uniformly drawn from F. As in [BCC⁺16], [BBB⁺18], or [WTS⁺18], the opening proof is an inner product argument, which uses $k = \log(d)$ rounds to gradually reduce the size polynomial in question by one half until ending up with a single-coefficient instance. In the course of the reduction the committer key is repeatedly folded into a final committer key G_f for the single-coefficient claim, which depends linearly on the initial committer key by

$$G_f = h_0 \cdot G_0 + \ldots + h_{d-1} \cdot G_{d-1},$$

where (h_i) are the coefficients of the *reduction polynomial*

$$h(\vec{\xi}, X) = \prod_{i=0}^{k-1} (1 - \xi_{k-1-i} \cdot X^{2^i}),$$

where $\vec{\xi} = (\xi_i)_{i=0}^{k-1}$ are the random challenges of the reduction steps. See [BCMS20] for a detailed description.

Theorem 10 ([BCC⁺16, BCMS20])**.** *Under the assumption that the dlog commitment scheme is computationally binding, the opening argument is an honest verifier zero-knowledge (Definition 6) argument of knowledge (Definition 5).*

Zero-knowledge is proven in [BCMS20]. To obtain a strict polynomial time extractor as demanded by Definition 5, one proceeds as in [BCC⁺16] to construct an expected polynomial time extractor with average runtime μ inverse proportional to the success probability ε of the adversary. Truncating it to strict polynomial time $2 \cdot \mu / \varepsilon$ zields a running time bounded by $\mathsf{poly}(\lambda)/\mathsf{poly}(\varepsilon)$ calls of $\mathcal{A}$ and still maintains a success probability of $\varepsilon/2$.

A.8 The batch evaluation protocol from [BDFG20]

We give an informal description of the protocol from [BDFG20], Section 5.2, for proving a multi-point opening claim

$$p_i(x_i) = y_i, \quad i = 1, \ldots, m,$$

of given polynomials $p_i(X)$, $i = 1, \ldots, m$, for a linear commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Vf})$. The multi-point evaluation claim is equivalent to the following system of algebraic identities over the set $\Omega = \{x_1, \ldots, x_m\}$,

$$p_i(X) \cdot L_m(x_i, X) - y_i = 0 \mod z(X), \quad i = 1, \ldots, m,$$

where $L_m(x_i, X)$ is the Lagrange polynomial for Ω, and $z(X) = \prod_{x\in\Omega}(X - x)$ is the vanishing polynomial of Ω. A more practical system of identities is achieved if we replace the Lagrange polynomials by their non-normalized variant

$$z_i(X) = \frac{z(X)}{X - x_i} = \prod_{x\in\Omega\setminus\{x_i\}} (X - x).$$

These m identities are reduced to a single identity over Ω in the usual manner: The prover receives a random scalar $\rho \leftarrow\$ F$ from the verifier, and shows the combined identity

$$\sum_{i=1}^{m} \rho^{i-1} \cdot (p_i(X) - y_i) \cdot z_i(X) = 0 \mod z(X)$$

instead. For this, the verifier provides an oracle for the quotient polynomial $q(X)$ in the non-modular identity

$$\sum_{i=1}^{m} \rho^{i-1} \cdot (p_i(X) - y_i) \cdot z_i(X) = q(X) \cdot z(X),$$

which is probed at a fresh random point $x \leftarrow\$ F$. This identity is verified by showing that the linear combination $\sum_{i=1}^{m} \rho^{i-1} \cdot z_i(x) \cdot p_i(X) - z(x) \cdot q(X)$ opens at x to the expected value $v = \sum_{i=1}^{m} \rho^{i-1} \cdot y_i \cdot z_i(x)$.

Theorem 11 ([BDFG20], Theorem 6)**.** *Assume that the commitment scheme is computationally binding. If the opening argument for the polynomial commitment scheme is a perfect honest verifier zero-knowledge argument of knowledge (Definition 5), then the same holds for the batch evaluation protocol.*

A.9 Segmentation of linear commitment schemes

Segmentation of a homomorphic polynomial commitment scheme is a useful technique to improve the computational effort of the prover at the cost of increasing the commitment size[9]. One chooses an undersized committer key $ck = (G_0, \dots, G_{s-1})$, where the *segment size* s is typically magnitudes smaller than the targeted maximum degree d, and extends its domain beyond degree $s - 1$ by decomposing a polynomial $p(X)$ into

$$p(X) = p_0(X) + X^s \cdot p_1(X) + \dots + X^{(k-1)\cdot s} \cdot p_k(X),$$

with each $p_i(X)$, $i = 1, \dots, k$, of degree at most s. (If d is the degree of $p(X)$ then the number k of segment polynomials is equal to $\lceil (d+1)/s \rceil$.) The commitment of $p(X)$ is then defined as the vector of the commitments of its segment polynomials using ck,

$$\mathsf{Com}(p(X)) := (\mathsf{Com}(p_0(X)), \dots, \mathsf{Com}(p_k(X)))\,.$$

[9]We learned this technique from [Ol] but believe that it is commonly known.

Every evaluation claim $p(x) = v$ of the full-size polynomial is translated to the same claim on the linear combination of its segment polynomials,

$$LC_x(p_0(X), \ldots, p_k(X)) = p_0(X) + x^s \cdot p_1(X) + \ldots + x^{(k-1)\cdot s} \cdot p_k(X),$$

which is efficiently proven by leveraging the homomorphic property of the scheme.

A.10 Facts on the Lagrange kernel

Let $H = \{x : x^n - 1 = 0\}$ be an order n subgroup of the multiplicative group of a finite field F. The Lagrange kernel

$$L_n(X, Y) = \frac{1}{n} \cdot \left(1 + \sum_{i=1}^{n-1} X^i \cdot Y^{n-i}\right)$$

is the unique bivariate symmetric polynomial of individual degree at most $n-1$, such that for $y \in H$ the function $L(X, y)$ restricted to H equals the Lagrange function $L_y(X)$, which evaluates to one at $X = y$, and zero otherwise. The kernel has the succinct representation

$$L_n(X, Y) = \frac{1}{n} \cdot \frac{Y \cdot Z_H(X) - X \cdot Z_H(Y)}{X - Y},$$

where $Z_H(X) = X^n - 1$ is the vanishing polynomial of H. Lagrange kernels represent point evaluation, as characterized by the following simple Lemma.

Lemma 5. *Suppose that $H \subset F^*$ is a multiplicative subgroup of order n and $p(X)$ is a polynomial of degree $deg(p(X)) \leq n - 1$. Then for every z in F,*

$$\left\langle L_n(X, z), p(X)\right\rangle_H = \sum_{x \in H} L_n(z, x) \cdot p(x) = p(z).$$

Proof. Since $p(X) = \sum_{y \in H} p(y) \cdot L_n(X, y)$, it suffices to show the claim for $p(X) = L_n(X, y)$, with $y \in H$. By the property of $L_n(X, y)$, we have $\langle L_n(X, z), L_n(X, y)\rangle_H = L_n(y, z)$, which by symmetry is equal to $L_n(X, y)$ at $X = z$. This completes the proof of the Lemma. $\square$

Marlin [CHM+20] uses the generalized derivative

$$u_H(X, Y) = \frac{Z_H(X) - Z_H(Y)}{X - Y}$$

instead of $L_n(X, Y)$ (again as element from $F[X, Y]/\langle X^n - 1, Y^n - 1\rangle$). For the generalized derivative a similar inner product formula holds: For every $z \in F$,

$$\begin{aligned}\left\langle u_H(X, z), p(X)\right\rangle_H &= \left\langle u_H(X, X) \cdot L_H(X, z), p(X)\right\rangle_H \\ &= \left\langle L_n(X, z), u_H(X, X) \cdot p(X)\right\rangle_H = p^*(z),\end{aligned}$$

where $p^*(X) = u_H(X, X) \cdot p(X) \bmod (X^n - 1)$.