Darlin: A proof carrying data scheme based on Marlin

(Extended abstract)

Haböck, Ulrich
ulrich@horizenlabs.io

Garoffolo, Alberto
alberto@horizenlabs.io

Di Benedetto, Daniele
daniele@horizenlabs.io

July 10, 2021

Abstract

In this document we describe the *Darlin* proof carrying data scheme for the distributed computation of block and epoch proofs in a Latus sidechain of Zendoo [GKO20]. Recursion as well as base proofs rest on Marlin [CHM+20] using the Pasta curves [HBG] and the 'dlog' polynomial commitment scheme from [BCC+16, BBB+18]. We apply the amortization technique from Halo [BGH19] to the non-succinct parts of the verifier, and we adapt their strategy for bivariate circuit encoding polynomials to aggregate Marlin's inner sumchecks across the nodes of the proof carrying data scheme. Regarding performance, the advantage of Darlin over a scheme without inner sumcheck aggregation is about 30% in a tree-like scenario as ours, and beyond when applied to linear recursion.

Contents

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 their advent [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 commmitment 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 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 2-cycles 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][1]. 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 yet 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 recursion end.

Since [BGH19] amortization schemes became an active field of research. Bünz et al. [BCMS20] give 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 integrates the amortization rounds directly 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, which for a Marlin verifier 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. We apply tree-like recursion whenever timing is critical (e.g. when computing block proofs) and otherwise sequentially. For our implementation we choose the Pasta cycle [HBG] of non-pairing friendly curves, and use the optimization techniques from [BGH19] to reduce the size of the verifier in circuit. According to our estimates, we expect the advantage of Darlin over standard Marlin (without inner sumcheck aggregation) to be about 43% at the two lowest levels of a proof tree, and 30% beyond, at the cost of only tripling the proof size, cf. Table 1.

[1] [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,

Table 1: The impact of inner sumcheck amortization: Comparison of Marlin [CHM+20] versus Darlin for a PCD node which verifies two proofs from 'below', both using the Pasta curves [HBG] and split verification as described in Section 7. 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*	≈ 320 k	≈ 384 k	≈ 520 k
	Darlin	≈ 290 k	≈ 320 k	≈ 390 k
proof size	Marlin*	≈ 4.2 kB	≈ 4.6 kB	≈ 5.3 kB
	Darlin	≈ 15.3 kB	≈ 15.7 kB	≈ 16.8 kB
prover time	Marlin*	≈ 16.5 s	≈ 15.9 s	≈ 15.38 s
	Darlin	≈ 12.3 s (9.6 s**)	≈ 11.7 s (9.1 s**)	≈ 11.4 (8.8 s**)

*Assuming R1CS density $d = 2$, which is large enough in our applications.
**only at the two lowest levels of the 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. Finally, we conclude with a description of the Darlin proof carrying data scheme in Section 6 and give a short description of our implementation strategy in Section 7.

2 Preliminary Notes

The largest part of this document is kept informal, while a rigorous treatment is postponed to the full version. However, even then some notions need to be settled at the beginning, at least to such an extent so that a subsequent exposure does not lack clarity.

Assume that $\mathcal{R}$ is a polynomial decidable binary relation, consisting of pairs (x, w), where we call x statement and w witness. Informally, an *interactive argument* for $\mathcal{R}$ is a protocol between a computationally bounded prover and a verifier with the aim to prove that for a given statement x there exists a w such that $(x, w) \in \mathcal{R}$. Typically, the argument has a setup procedure which

Gui20] enforce to increase the field size up to $1,000$ bits.

provides a common reference string supporting instances (x, w) up to a certain maximum size. Beside that common reference string, the verifier is given only x while the prover has w, and after a certain number of steps the verifier is convinced of the fact that $x \in \mathcal{R}$ and accepts, otherwise she rejects. The notions of completeness, zero-knowledge and knowledge soundness are given in Appendix A.2. If the transcript of the conversation (and hence the number of steps) is sub-linear in the size of the instance (x, w), then the argument is called *succinct*.

A *polynomial commitment scheme* is a commitment scheme for polynomials over some finite field which additionally comes with an interactive argument for proving the value of a committed polynomial at a queried point. This *evaluation argument* (or, *opening proof*) is a succinct interactive argument for statements of the form (C, x, y), where C is the commitment of a witness polynomial $p(X)$ for which $y = p(x)$. See Appendix A.3 for the formal definitions.

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, on which he 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 is given some oracles as protocol inputs, but we do not assume that these oracles stem from a (circuit specific) public precomputation phase. Our input oracles contain private or public polynomials, typically created at running time before calling the protocol. Formal definitions of algebraic oracle proofs will be given in the full version of the document.

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 of 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 above sum is equal to n times the constant term of the polynomial, if $p(X)$ is of *reduced form*, i.e. of 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.

Plonk [GWC19] uses an alternative approach for proving a 'grand product' $\prod_{z \in H} p(z) = 1$ in their permutation argument. For that, the prover provides the oracle for a polynomial $Z(X)$ which satisfies

$$Z(g \cdot X)/Z(X) = p(X) \mod (X^n - 1), \tag{3}$$

where g is a generator of the multiplicative subgroup H. Given the action of $\mathbb{Z}$ on H defined by the 'shift' transformation $T(x) = g \cdot x$, consider the cumulative products

$$f_p(k, x) = \prod_{i=0}^{k-1} p(T^i(x)),$$

for $x \in H$ and $k \in \mathbb{Z}$. The function f_p is a *cocycle* with values in the multiplicative group F^* of the field, which means that $f_p(m+k, x) = f_p(m, T^k(x)) \cdot f_p(k, x)$ for every $m, k \in \mathbb{Z}$ and $x \in H$. In terms of cocycles, identity (3) states that f_p is a *coboundary*, which means that there exists a polynomial $Z(X)$ such that

$$f_p(k, x) = Z(T^k(x)) \cdot Z(x)^{-1},$$

for every $x \in H$, $k \in \mathbb{Z}$. This polynomial is called the *boundary* of the coboundary f_p. It is a well-known fact that a cocycle is a coboundary if and only if the products over the closed loops of the action on H vanish, that is $\prod_{z \in H} p(z) = 1$, as g is a generator of H.

Our sumcheck argument simply carries over Plonk's grand product argument to the additive setting. Instead of using the reduced form of the polynomial $p(X)$ in question, the prover shows that the additive cocycle

$$f_p(k, X) = \sum_{i=0}^{k-1} p(g^i \cdot X) \tag{4}$$

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*

$$U(g \cdot X) - U(X) = p(X) \mod (X^n - 1). \tag{5}$$

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 $k = 1, \ldots, n-1$. By definition $U(g^{k+1}) = U(g^k) + p(g^k)$ for all k, $0 \le k \le n-2$. But 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, and hence 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 (5) 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,

$$\hat{U}(X) = U(X) + (c_0 + c_1 \cdot X) \cdot (X^n - 1), \tag{6}$$

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 proof 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 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 $b + 1$.* [2]

(1) The prover P computes $U(X)$ of $deg(U(X)) < n$ according to the coboundary identity (5). It computes $\hat{U}(X)$ as in (6), with $c_0, c_1 \leftarrow\$ F$, and the quotient polynomial $h(X)$ satisfying

$$\hat{U}(g \cdot X) - \hat{U}(X) = \hat{p}(X) + h(X) \cdot (X^n - 1). \tag{7}$$

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

(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 (7) at $X = z$, and accepts if valid. (Otherwise, the verifier rejects.)

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

The security analysis of Protocol 1 will be given in the full version of the document. Informally, Protocol 1 is perfectly a complete and computationally knowledge sound algebraic oracle proof, if the size of H is negligible compared to the size of the field F. It is succinct and moreover 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 7) and perfectly hiding (Definition 6) 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 Our Marlin variant

In this section we describe *Coboundary Marlin*, which is a slight variant of the Marlin proving system [CHM+20]. We introduce two changes: First, we replace the Marlin's sumcheck argument by the coboundary argument from Section 3. Second, we[3] make use of the Lagrange kernel

$$L(X, Y) = \frac{1}{|H|} \cdot \frac{Y \cdot Z_H(X) - X \cdot Z_H(Y)}{X - Y} \tag{8}$$

instead of the non-normalized version $R(X, Y) = \frac{Z_H(X) - Z_H(Y)}{X - Y}$. The Lagrange kernel shares the same key properties as the generalized derivative $R(X, Y)$. 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.

4.1 Arithmetization

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

$$(A \cdot y) \odot (B \cdot y) = C \cdot y, \tag{9}$$

[3]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.

where 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. $y = (x \| w)$. Assuming quadratic matrices is no loss in generality, as the constraint system may be always 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 two-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 $H = \{z \in F : z^n - 1 = 0\}$ using Lagrange encoding. That is, given an arbitrary enumeration $\{x_1, \ldots, x_n\}$ of H a vector $y = (y_k)_{k=1}^n$ is associated with the polynomial $y(X) = \sum_k y_k \cdot L(X, z_k)$. 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 (9) 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 having the matrix entries of 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 by indexing its non-zero terms over yet another FFT domain $K = \{w \in F : w^m - 1 = 0\}$, again assuming the existence of such sufficiently large multiplicative subgroup. In practice $M = A, B, C$ are sparse, yielding that K is about 2 to 5 times larger than H, see Section 4.3. As in Marlin, we denote by

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

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 for every z in H, $L(X, z) = \frac{1}{|H|} \cdot \frac{z \cdot Z_H(X) - X \cdot Z_H(z)}{X - z} = \frac{1}{n} \cdot \frac{z \cdot (X^n - 1)}{X - z}$, we have

$$M(X, Y) = \frac{(X^n - 1) \cdot (Y^n - 1)}{n} \sum_{w \in K} \frac{val_M(w) \cdot row_M(w) \cdot col_M(w)}{(X - row_M(w)) \cdot (Y - col_M(w))}. \tag{14}$$

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

$$row.col_M(X) = row_M(X) \cdot col_M(X) \mod (X^m - 1), \tag{15}$$

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

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

4.2 The protocol

In Marlin, the prover provides the oracles for $y(X)$, $y_A(X)$, $y_B(X)$ and convinces the verifier of the 'R1CS identities' (10), (11), (12). For efficiency reasons these three identities are reduced to a single one by building a random linear combination based on a challenge $\eta \leftarrow\$ F$, i.e.

$$y_\eta(X) = \sum_{z \in H} T_\eta(X, z) \cdot y(z) \mod (X^n - 1), \tag{17}$$

with

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

and

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

(We notice that using the powers of η slightly differs from choosing arbitrary random scalars η_A, η_B, η_C as in [CHM+20].) Equation (17) 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

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

and hence

$$\sum_{z \in H} \langle R(X, \alpha), T_\eta(X, z) \rangle_H \cdot y(z) - R(z, \alpha) \cdot y_\eta(z) = 0. \tag{18}$$

We choose the Lagrange kernel $L(X, Y)$ for $R(X, Y)$, instead of a non-normalized variant as in [CHM+20]. Since $T_\eta(X, z)$ is of degree less than n, it holds that $\langle L(X, \alpha), T_\eta(X, z) \rangle_H = T_\eta(\alpha, z)$ (see Appendix A.5), and equation (18) is equal to

$$\sum_{z \in H} T_\eta(\alpha, z) \cdot y(z) - L(z, \alpha) \cdot y_\eta(z) = 0. \tag{19}$$

Equation 19 is the central identity to be proven by the protocol.

We give only an informal description of the main steps of the protocol, and skip the discussion on zero-knowledge. See the full version of the paper for a detailed treatment.

Initialization

In the first step the prover computes the polynomials[4]

$$y(X), y_A(X), y_B(X) \in F[X]/(X^n - 1)$$

from their Lagrange represenations, and sends their oracles $[y(X)]$, $[y_A(X)]$, $[y_B(X)]$ to the verifier, who returns the randomnesses $\eta \leftarrow\$ F$ and $\alpha \leftarrow\$ F \setminus H$ for equation (19).

Outer sumcheck

To prove equation (19) we apply the coboundary argument from Section 3 to $p(X) = T_\eta(\alpha, X) \cdot y(X) - L(X, \alpha) \cdot y_\eta(X)$. For this the prover computes

$$U_1(X) \in F[X]/(X^n - 1),\ h_1(X) \in F^{<n}[X]$$

satisfying

$$T_\eta(\alpha, X) \cdot y(X) - L(X, \alpha) \cdot y_\eta(X) \\ = U_1(gX) - U_1(X) + h_1(X) \cdot (X^n - 1), \quad (20)$$

where g is generator of H, and sends their oracles, together with that of $T_\eta(\alpha, X)$, to the verifier. The verifier samples another random challenge $\beta \leftarrow\$ F \setminus H$ and queries the oracles for $y(\beta), y_A(\beta), y_B(\beta), T_\eta(\alpha, \beta), U_1(g\cdot\beta), U_1(\beta), h_1(\beta)$, which are used for checking the identity (20) at $Z = \beta$.

Inner sumcheck

To prove that the value $T_\eta(\alpha, \beta)$ 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 slightly different representation 14. Using these we obtain

$$T_\eta(\alpha, \beta) = \sum_{w \in K} \sum_{M=A,B,C} \eta_M \cdot \frac{val.row.col_M(w)}{(\alpha - row_M(w)) \cdot (\beta - col_M(w))}, \quad (21)$$

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

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

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

and then multiplies both sides with the denomiator

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

where $row.col_M(X)$ are the precomputed products (15) from the prover key. This yields the *inner sumcheck* identity

$$\begin{aligned} &\sum_{M=A,B,C} \eta_M \cdot val.row.col_M(X) \\ &\qquad = b(X) \cdot \left(\frac{T_\eta(\alpha,\beta)}{m} + U_2(g_K X) - U_2(X) \right) + h_2(X) \cdot (X^m - 1), \quad (22) \end{aligned}$$

where g_K is a generator of K and $deg(h_2(X)) \leq 3 \cdot m - 4$. 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 (22) at $X = \gamma$.

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, and in practice we observed values between 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.)

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	$\approx (15 + 5 \cdot d)$ $\mathsf{FFT}(n)$	$\approx (7 + 4 \cdot d)$ $\mathsf{MSM}(\mathsf{n})$

5 Marlin in recursion

Our proof carrying data scheme described in Section 6 is based on Marlin and the [BCMS20] variant of the *dlog polynomial commitment scheme* from [BCC+16]. We take Coboundary Marlin without inner sumcheck as non-efficient[5] succinct argument, and we aggregate both the non-succinct parts of the opening proof verifier, as well as the correctness checks usually served by inner sumchecks, which is verifying that the commitment intended for

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

in fact stems from that polynomial. 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 the 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]. For efficiency reasons we choose the 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 protocol rounds for proving satisfiability of the current circuit, the $(acc_i)_{i=1}^{\ell}$ are aggregated within some extra rounds into a new instance, the 'current' accumulator

[5] By non-efficient we mean that the verifier effort is linear in the size of the circuit.

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. (We assume that $(acc_i)_{i=1}^{\ell}$ is consistent with the circuit inputs x.)

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 z and $\vec{\eta}$) 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

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

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 its commitment as oracle.

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'*

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

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 full version of the document. 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' recursive schemes such as our Darlin PCD scheme from Section 6. 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

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

$i = 1, \ldots, L$, as bivariate polynomials over the same domain $H \times H$. As in the single-circuit setting we leverage linearity of the commitment scheme, and keep track of a single *cross-circuit polynomial*

$$T_H(\alpha, Y) = \sum_{i=1}^{L} T_{i,\vec{\eta}_i}(z, Y) = \sum_{i=1}^{L} \sum_{M=A_i,B_i,C_i} \eta_{M,i} \cdot M(\alpha, Y) \tag{23}$$

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 ϕ 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 commmitment is an inner product argument which uses the repetitive 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*

$$h(\vec{\xi}, X) = \prod_{i=0}^{k-1} (1 - \xi_{k-1-i} \cdot X^{2^i}), \tag{24}$$

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 a commitment, 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 Putting the pieces together

The complete recursive argument is a composition of Marlin's outer sumcheck for the 'current' circuit, 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 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 previous 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

$$w(X), z_A(X), z_B(X) \in F[X]/(X^n - 1)$$

and sends their dlog commitments $[w(X)]$, $[z_A(X)]$, and $[z_B(X)]$ to the verifier, who responds with $\eta, \alpha \leftarrow\$ F$.

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

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

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

$$U_1(Y) \in F[Y]/(Y^n - 1), h_1(Y) \in F^{<n}[Y],$$

according to the outer sumcheck identity (20). It then sends their commitments $[T_{\vec{\eta}}(\alpha, Y)], [U_1(Y)], [h_1(Y)]$ 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 sends $H'' = \delta_k \cdot \vec{\eta} + H'$ *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 following multi-point queries:

- $[w(X)], [z_A(X)], [z_B(X)], [U_1(X)], [h_1(X)]$, $[T_{\vec{\eta}}(\alpha, X)]$ *at* β, *as well as* $[U_1(X)]$ *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.

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*

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

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

A detailed security analysis of Protocol 1 is given in the full version of the document. We summarize its result in the following theorem.

Theorem 1. *If the dlog commitment scheme is computationally binding (Definition 7) then Protocol 4, extended by the predicate verification on the resulting inner sumcheck accumulator* $acc''_{\mathcal{C}}$, *is an argument of knowledge (Definition 5) for the relation*

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

indexed by the circuit collection $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$. *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*

We will use the Fiat-Shamir transform to turn Protocol 4 into a non-interactive argument of knowledge as used in our Darlin PCD scheme. As the proof for Theorem 1 we postpone a security analysis in the random oracle model to the full version of the paper.

5.5 A note on performance

Inner sumcheck aggregation is particularly effective when the number of previous accumulators is low, as can be seen from the operations counts in Table 3. For $\ell = 1$ previous accumulators 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 $d = 1.5$. Compared to a standard Marlin prover for circuits with density $d = 2$ the perfomance improvement is about 27%, cf. Table 1.

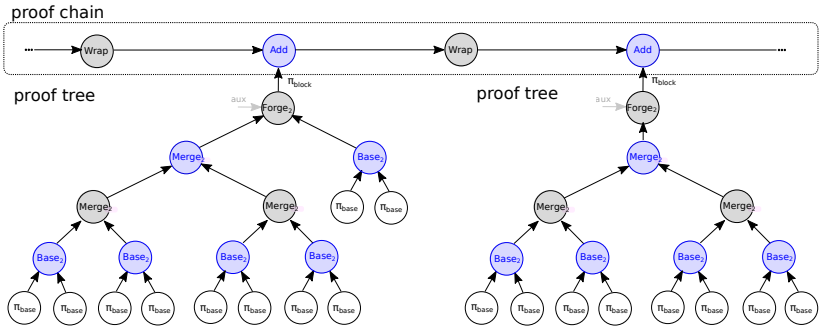
Table 3: Marlin prover with and without inner sumcheck aggregation in terms of FFT operations and multi-scalar multiplications. (Without opening proof.)

	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	$\approx (15 + 5 \cdot d)$ $\mathsf{FFT}(n)$	$\approx (7 + 4 \cdot d)$ $\mathsf{MSM}(n)$

6 The Darlin PCD scheme

Zendoo [GKO20] is a cross chain transfer protocol allowing the exchange of assets between a Bitcoin-like *mainchain* and *sidechains* of different types, without knowing their internal structure (e.g., what consensus protocol is used, what types of transactions are supported). The coin transfers happening from sidechains to mainchain are performed by submitting to mainchain a special kind of transaction named *certificate*, containing the list of sidechain-to-mainchain transfers (so-called 'backward transfers') and a SNARK assuring their correctness (i.e. they followed the sidechain rules), optionally in zero-knowledge. A *Latus sidechain* is a specific Zendoo enabled sidechain structure which leverages recursive SNARKs to prove the correctness of backward transfers by validating the state of the sidechain (e.g., the state can be the collection of unspent transaction outputs). In order to assure practical performance and system decentralization, such succinct proof is computed in a distributed manner by several *SNARK provers*. The SNARK provers must follow the Latus incentive scheme [GKO21] which is specifically designed to enable decentralized proof creation and fair proving costs. In this chapter we give a high-level description of the Darlin proof carrying data (PCD) scheme, which is the recursive scheme that serves such succinct proof of state within a Latus sidechain.

Figure 1: The Darlin PCD scheme and its subgraphs. Normal Marlin base proofs π_{base} are merged by Darlin nodes into block proofs π_{block}, which are incrementally added to the overall state of the sidechain.



An in-depth discussion, including the recursive arithmetic circuits is given in the full protocol specifications [BH21].

The Darlin PCD scheme is based on *Darlin*, i.e. the recursive argument from Section 5.4, using a 2-cycle of ordinary elliptic curves. Such cycle consists of two prime order elliptic curves $E_1 = EC(F_1)$ and $E_2 = EC(F_2)$ over prime fields F_1 and F_2 such that the order of the one group equals the field characteristic of the other. The scheme applies a small collection of recursive circuits $\mathcal{C}(F_1)$ and $\mathcal{C}(F_2)$ over the two fields, used to process Darlin proofs along a (directed and acyclic) communication graph of nodes. The communication graph, as depicted in Figure 1, is divided into three subgraphs:

- The *proof chain* is a sequential chain of proofs designed to keep track of the last sidechain state and its validating proof. The chain consists of a sequence of *Add* and *Wrap* nodes over the curves E_1 and E_2, used to merge the 'previous' state proof and the 'current' block proof in order to create a new proof certifying the validity of the state transitions derived by the block.

- The *proof tree* is a dynamically arranged graph using *Merge* nodes[7]over the two curves E_1 and E_2 in order to end up with a unique block proof that is the result of merging all the base proofs. The finalizing *Forge* node is a special purpose node which, in addition to proof merging, serves some additional (e.g. consensus related) logic.

- The *base layer* is composed by Marlin nodes over the curve E_2 which serve the *base proofs* certifying the state transition for a dedicated fraction of

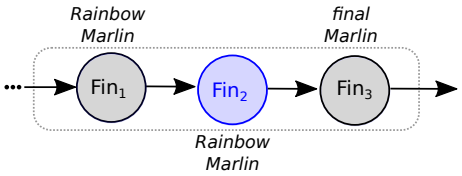
[7]And their variants at the 'bottom' border of the Darlin PCD scheme, the nodes which verify the normal Marlin proofs from the base layer.

block data. The proofs provided by these nodes validate both the common sidechain transaction rules and the custom logic needed by the specific sidechain.

Technically, the base layer forms a PCD scheme on its own, serving ordinary Marlin proofs which are then merged by the Darlin PCD into block proofs. This hybrid design is a trade off between flexibility and performance. It allows to keep the number of circuits supported by the cross-circuit accumulator of the Darlin PCD scheme independent of the number of custom base proofs, and hence do not overstretch the memory usage of the SNARK provers.

After a specified number of blocks (an *epoch*) the sidechain commits its state (plus transfers, beside some extra information collected over the epoch) to the mainchain, together with a succinct proof. (For more details on the features of sidechain certificates, see [GKO20].) To allow a mainchain verifier being agnostic of the circuits aggregated along the Darlin PCD, a separate *exiting chain* of specific purpose nodes gradually transform a Darlin proof with its cross-circuit inner sumcheck accumulators into a normal Marlin proof (plus two deferred dlog accumulators). The exiting chain, as illustrated in Figure 2 consists of three nodes. The first two nodes of the exiting chain prove correctness of the two cross-circuit aggregators (one over E_1, and the other one over E_2), by running an inner sumcheck argument for all the involved circuit matrices. Technically, these nodes run a cross-circuit variant of Marlin, called *Rainbow Marlin*, which we describe in detail in our full protocol specification [BH21]. The third node of the exiting chain is again a normal Marlin node, which verifies the succinct part of the previous Rainbow Marlin proof; the dlog accumulators of the two previous Rainbow Marlin proofs are simply passed via public inputs. The final proof is a normal Marlin proof, together with two dlog accumulators (one over E_1, and one from E_2).

Figure 2: The exiting chain of the Darlin PCD scheme gradually transforms a Darlin proof into a normal Marlin proof.



7 Implementation

We instantiate our Darlin PCD scheme using the Pasta curves [HBG], which form a 2-cycle of prime order elliptic curves over 255 bit prime fields. The Pasta curves come with a variety of practical properties, such as high 2-adicity

of their scalar fields (as needed for FFT), the existence of a non-trivial curve automomorphism (to support endomorphism-based scalar multiplication), as well as the existence of field automorphisms of low algebraic degree (for arithmetic hash functions). We choose x^5-Poseidon [GKR+21] as circuit-friendly hash function, with a suitable choice of rate and capacity. In the design of recursive circuits we moreover apply all the optimizations from Halo [BGH19]:

- We defer non-native arithmetics: We split the Darlin verifier into its native and non-native part, where the non-native part is implemented in the 'next' recursive circuit, in which these operations are again native. This splittling technique is applied to the algebraic checks that come along with the Darlin verifier, as well as for the Poseidon-based Fiat-Shamir sponge.
- We use Halo's endomorphism-based scalar multiplication.

We further use segmentation for the dlog commitment scheme and tune its size to balance prover speed-up versus increased verifier circuits. Our constraint estimate for a *Merge*-2 circuit, which is the typical recursive circuit in our Darlin PCD scheme, stays below 2^{19} constraints while using a segment size of 2^{17}, as can be seen in Table 1. There, the timing estimates are based on a careful 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 0.587 USD per hour. Given these timings we expect a proof tree for a base layer of overall 2^{27} = 128 million constraints to run in less than 105 seconds, assuming 256 parallel SNARK provers and neglecting communication costs. Compared to a proof tree based on Marlin without inner sumcheck aggregation, the Darlin PCD scheme is expected to be faster by a factor of 0.74.

8 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.

[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.

[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`.

[BH21] Daniele Di Benedetto and Ulrich Haböck. The Darlin PCD scheme protocol specification. in preparation, 2021.

[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`.

[GKO21] Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov. Latus incentive scheme: Enabling decentralization in blockchains based on recursive snarks. In *IACR preprint archive 2021/399*, 2021. `https://eprint.iacr.org/2021/399`.

[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 noninteractive arguments of knowledge. In *IACR preprint archive 2019/953*, 2019. `https://eprint.iacr.org/2019/953`.

[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/`.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constand-round secure two-party computation (full version). In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 171–189. Springer, 2001. full version:`https://eprint.iacr.org/2001/107`.

[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.

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 $f(\lambda) = o(1/p(\lambda))$. Two functions $f(\lambda)$ and $g(\lambda)$ are *computationally indistinguishable*, denoted by $f \approx g$, if their difference is negligible in λ.

Probabilistic algorithms are denoted by capital letters A, B, etc., and we write $y \leftarrow \mathsf{A}(x)$ if an algorithm A 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 will 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 2 (Perfect completeness)**.** An interactive argument system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ satisfies *perfect completeness* if

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

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

Definition 3 (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 with access to the common reference string and such that for every p.p.t. algorithm $\mathcal{A}$,

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

Since we intend to use arguments as components of larger cryptographic schemes, we follow [BCC+16], [BGH19] and use the notion of *witness-extended emulation* [Lin01] for defining arguments of knowledge. An interactive argument system has witness-extended emulation if every computationally bounded adversary can be turned into a witness-extended emulator of comparable efficiency, which serves both transcripts tr' indistinguishable from the interactions of $\mathcal{A}$, and a witnesses w for tr' whenever it is accepting.

Definition 4 (Witness-extended Emulation)**.** An interactive argument system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ has *witness-extended emulation* if for every p.p.t. adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there is a p.p.t. algorithm $\mathcal{E} = \mathcal{E}^{\mathcal{A}}$ with black-box access to the next-message functions of $\mathcal{A}$ and such that

$$\Pr\left[\begin{array}{l|l} \mathcal{A}_2(tr') = 1 \wedge & crs \leftarrow \mathsf{Setup}(\lambda), \\ (\text{if } tr' \text{ accepts, then } (x,w) \in \mathcal{R}_N) & x \leftarrow \mathcal{A}(crs)\ , \\ & (w, tr') \leftarrow \mathcal{E}^{\mathcal{A}} \end{array}\right]$$
$$\approx \Pr\left[\mathcal{A}_2(tr) = 1 \;\middle|\; \begin{array}{l} crs \leftarrow \mathsf{Setup}(\lambda), \\ x \leftarrow \mathcal{A}_1(crs), \\ tr \leftarrow \langle \mathcal{A}_1, \mathsf{Vf}(x) \rangle \end{array}\right].$$

We note that any expected polynomial time emulator with non-negligible success probability can be turned into one with strict polynomial running time still having non-negligible success probability, hence our variant of Definition 4 is equivalent to the ones from [BCC+16], [BGH19].

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

A.3 Polynomial commitment schemes

We regard a polynomial commitment scheme consisting of four p.p.t. algorithms

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

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 $(\mathsf{Setup}, \mathsf{Open}, \mathsf{Vf})$ as succinct interactive argument system for the relation

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

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 6 (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 adversarily chosen polynomials,

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

Definition 7 (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 commmitment 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 8 (Homomorphic commitment)**.** A polynomial commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Vf})$ with commitments in a *commitment group* $(\mathcal{G}, +)$ is *homomorphic*, 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.4 Pre-processing arguments with universal SRS

An *indexed relation* $\mathcal{R} = \bigcup_i R_i$ is a collection of binary relations R_i where the index i ranges over some index set $\mathcal{I}$. In our context, $\mathcal{I}$ will be set of all R1CS systems over the family of fields given by a polynomial commitment scheme (see Appendix A.3) and so that the sizes of the R1CS matrices are polynomial in the security parameter. We consider a *universal setup pre-processing argument system*

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

for an polynomial decidable binary indexed relation $\mathcal{R}$ as an interactive argument system in which the setup is divided into phases. Given the security parameter λ, $\mathsf{Setup}(\lambda)$ generates the universal part of the common reference string, which we again denote by crs. This is the universal setup, and done in the so-called 'offline' phase. In a subsequent 'online' phase, the *indexer* Index is used to generate the index specific part of the common reference string, which consists of a prover and a verifier key for a given index i, $(pk, vk) \leftarrow \mathsf{Index}(crs, i)$.

The security definitions completeness, zero-knowledge and witness-extended emulation are as Definition 2, Definition 3 and Definition 4, with the following minor differences: We replace (x, w) by (i, x, w), hence the adversary is allowed to chose the index i, $(i, x, w) \leftarrow \mathcal{A}(crs)$, followed by computing $(pk, vk) \leftarrow \mathsf{Index}(crs, i)$. Although we consider (i, pk, vk) as part of the common reference string, we will explicitly state pk and vk as inputs for the prover and the verifier, respectively, e.g. $tr \leftarrow \langle \mathsf{Prove}(pk, x, w), \mathsf{Vf}(vk, x) \rangle$.

A.5 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

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

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 to zero otherwise. The kernel has the succinct representation

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

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 2. *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,*

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

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