GAPP: Generic Aggregation of Polynomial Protocols

Chaya Ganesh[1], Sikhar Patranabis[2], Shubh Prakash[3], and Nitin Singh[4]

[1,3]Indian Institute of Science
[2,4]IBM Research India

Abstract

We construct a new bivariate polynomial commitment scheme, bPCLB, with succinct verification, $O(m+n)$ sized public parameters, and $O(m+n)$ cryptographic operations to generate evaluation proofs for bivariate polynomials of degree (n, m). bPCLB commits to polynomials using their Lagrange coefficients. This is in contrast to existing bivariate schemes, which either incur $O(mn)$-sized public parameters and $O(mn)$ cost for evaluation proofs, or do not natively support committing to polynomials in the Lagrange basis. We present the idea of a packing gadget that allows packing a non-constant number of univariate polynomials into one bivariate polynomial. We implement the packing gadget with bPCLB to achieve the following results.

We propose a generic framework called GAPP for aggregation of polynomial protocols. This allows proving n instances of a polynomial protocol using a single *aggregate* proof that has $O(\log n)$ size, and can be verified using $O(\log n)$ operations. We construct an information-theoretic protocol for proving the satisfiability of the bivariate polynomial identity, which is then compiled using any bivariate polynomial commitment scheme (PCS) to yield an argument of knowledge for the aggregation relation. GAPP can be applied to several popular SNARKs over bilinear groups that are modeled as polynomial protocols in a black-box way. Using bPCLB yields an efficient instantiation of GAPP for aggregating PLONK proofs where the prover only performs sublinear cryptographic operations in the evaluation proof.

We illustrate the packing property of bPCLB in two applications. We first construct a new lookup argument which supports tables of m-tuples. Second, we show a generalized grand-product protocol over a non-constant (say m) number of vectors. While existing approaches for both these applications incur $O(m)$ proof size and verification complexity, our constructions achieve $O(\log m)$ dependence. We leverage these applications together with our GAPP framework to design an À la carte proof system for non-uniform computations.

1 Introduction

Zero-knowledge (ZK) proofs [GMW86, For87, GMR89] allow a prover to convince a verifier about the truth of a statement without revealing anything beyond this. Consider an NP relation $\mathcal{R}$ that defines the language $\mathcal{L}$ of all statements x for which there exists a witness w so that $\mathcal{R}(x, w) = 1$. In a zero-knowledge proof for $\mathcal{R}$, the goal is for a prover who knows a witness w to convince a verifier that $x \in \mathcal{L}$ without revealing any additional information about w. When considering proofs that are only *computationally sound* (called argument systems) the communication complexity can be smaller than the length of the witness [BCC88], and are called *succinct arguments* [Kil92, Mic94].

Succinct Arguments. Succinct Non-interactive ARguments of Knowledge (SNARKs) enable one to prove the integrity of a computation such that the proof size and the verifier's work to check the proof do not scale with the size of the computation. Zero-knowledge variants (zkSNARKs) additionally guarantee that the proof hides all private inputs involved in the computation. zkSNARKs are a fundamental building block in modern cryptographic systems that crucially need small proofs and efficient verification (e.g., ZCash [BCG+14] and Monero [NMT]). These have been constructed in several works [Gro10, Lip12, BCCT12] [BCI+13, GGPR13, PHGR13, BCG+13, Lip13, BCTV14].

Polynomial Protocols. A design methodology underlying several recent constructions of efficient SNARKs is the following. First, an unconditionally secure idealized protocol is obtained for some NP complete language, which is then compiled into a computationally sound argument via a cryptographic compiler. Polynomial protocols, and related notions of Polynomial Interactive Oracle Proofs (PIOPs) and Algebraic Holographic Proofs (AHPs) offer a mathematically elegant framework for constructing secure idealized protocols. Informally, the prover in this idealized setting is restricted to sending low-degree polynomial oracles to the verifier, who infers the membership of the statement in the NP complete language by checking certain identities on the polynomials provided by the prover. In the compiled cryptographic argument, the polynomial oracles are realized via a polynomial commitment scheme (PCS) that allows the prover to send a short commitment to a polynomial and then open evaluations in a verifiable way. PCSs (and the compiled zkSNARKs) are either in the Structured Reference String (SRS) model or in idealized models (like ROM,GGM,AGM) or both. Polynomial protocols have been used to build popular SNARKs [CHM+20, GWC19], lookup arguments [BCG+18, GW20, EFG22, CFF+24] (which are particularly useful in constructing efficient SNARKs), as well as arguments for several other useful relations such as permutations etc.

Our Work. We propose *generic aggregation of polynomial protocols* where n instances of a polynomial protocol can be proved via a single *aggregate* proof that has $O(\log n)$ size, and can be verified using $O(\log n)$ operations. Our proof aggregation applies in a black-box manner to several popular SNARKs over bilinear groups that are modeled as polynomial protocols. A key technical tool of our framework that is of independent interest is a new bivariate polynomial commitment scheme, bPCLB. As applications of bPCLB, we obtain new *lookup arguments over tuples* and a generalized grand-product protocol. Putting these together with our proof aggregation, we design *efficient proofs of non-uniform computation* with "à la carte" prover cost. We expand on our results below.

1.1 Our Contributions

Underlying our contributions is the idea of a *packing gadget* that allows packing a non-constant number of univariate polynomials into one bivariate polynomial. This naturally leads us to proof aggregation. As we elaborate later, a *bivariate polynomial in Lagrange basis* is the "right" tool for our packing gadget. The packing gadget together with the tool of our bivariate polynomial commitment are the technical tools that yield our GAPP framework, lookup arguments over tuples and a generalized grand-product protocol. These applications are illustrative of the idea, and we believe that the packing gadget together with our bivariate PCS will lead to many applications.

Bivariate PCS in the Lagrange Basis. As a core technical contribution, we present a new bivariate PCS, which we call bPCLB. At a high level, bPCLB is an analogue of the AFG [AFG+16, BMM+21] commitment in the Lagrange basis. In comparison to KZG-based bivariate PCS [ZBK+22, LXZ+24], which incurs $O(mn)$-sized public parameters and $O(mn)$ cryptographic operations to generate evaluation proofs for bivariate polynomials of degree (n, m), the corresponding costs for bPCLB are $O(m + n)$. While this is similar to the overheads incurred by AFG-based bivariate PCS [AFG+16, BMM+21] in isolation, the fact that bPCLB operates directly over Lagrangian components of the bivariate polynomial makes it more suitable for certain applications, such as the aggregation of polynomial protocols, as described next. To achieve succinct verification in bPCLB, we require a new folding technique that we call *Lagrangian folding*. We believe that the bivariate PCS bPCLB and the Lagrangian folding scheme are of independent interest beyond aggregating polynomial protocols. See Section 3 for the technical details.

GAPP. We propose a generic technique for aggregating polynomial protocols, that we call GAPP. Our framework allows proving n instances of a polynomial protocol using a single $O(\log n)$-sized aggregate proof, which can be succinctly verified. At a high level, we reduce the satisfiability of several univariate polynomial identities over a domain to the satisfiability of a *single* bivariate polynomial identity over a related domain, where the bivariate polynomials interpolate *a batch of* univariate polynomials over the domain. We construct an information-theoretic protocol for proving the satisfiability of the bivariate polynomial identity, which is then compiled using any bivariate polynomial commitment scheme (PCS) to yield an argument of knowledge for the aggregation relation. We refer to Section 4 for a formal exposition.

The use of a bivariate polynomial in Lagrange basis to capture a batch of univariate constraints has appeared in Caulk [ZBK+22], Sublonk [CGG+24], and Pianist [LXZ+24]. However, in all of these prior works, the techniques are presented for very specific polynomial protocols, namely, for multi-unity proof aggregation in [ZBK+22, CGG+24], and PLONK PIOP in [LXZ+24]. Our framework generalizes these approaches to *arbitrary* polynomial protocols, and can be viewed as a natural analogue of representing a batch of arithmetic constraints via a polynomial identity over interpolated polynomials, which is a key step in several SNARK constructions. Unlike prior works [ZBK+22, CGG+24, LXZ+24], our framework is described in a manner that is agnostic to the specific bivariate PCS used. This abstraction highlights the role of the bivariate PCS as the key primitive determining the efficiency of aggregated proof generation.

It turns out that instantiating the GAPP framework with existing bivariate PCS (such as those based on AFG [AFG+16, BMM+21], Dory [Lee21], Samaritan [GPS25] and KZG variants [ZBK+22, LXZ+24]) incurs large overheads with respect to the size of the public parameters and proof generation (see Section 1.2 for a more detailed discussion). We present a concrete instantiation of the GAPP framework using bPCLB. Since bPCLB operates directly over Lagrangian components of the bivariate polynomial, it is more suitable for instantiating the GAPP framework than AFG based as well as Dory-based bivariate PCS [Lee21], which work over the monomial representation of polynomials. Unlike the aggregation frameworks from [GMN22, ABST23, YZRM24, LXZ+24] that are tailored to specific protocols (such as Groth16 or PLONK), our scheme is generally applicable to all polynomial protocols, and features a universal setup that can be reused across different polynomial IOPs.

Illustration: PLONK *Proof Aggregation.* As an illustration, we use the instantiation of GAPP described in Section 4.3 together with the PLONK PIOP to obtain a simpler and modular proof aggregation for PLONK with universal setup. In terms of proof generation, our scheme outperforms the popular aggregation scheme aPlonk [ABST23] currently used to specifically aggregate PLONK proofs in validity rollups for the Tezos blockchain. This demonstrates that the generality of our proof aggregation framework does not come at the cost of additional prover overhead. See Section **??** and Appendix 4.4 for more details.

Scheme	Setup	t_P	t_V	$\|\pi\|$
Naïve	$O(t)$	$O(m+t)$	$O(m)$	$O(m)$
[CGG$^+$24]	$O(mt)$	$O(mt)$	$O(1)$	$O(1)$
[DXNT24]	$O(mt)$	$O(mt)$	$O(1)$	$O(1)$
Our Work	$O(m+t)$	$O(m+t)$	$O(\log t)$	$O(\log t)$

Table 1: Comparison of arguments of knowledge for tuple lookups. Here m denotes the tuple size, n denotes the size of the subvector, while k denotes the size of the parent table/vector and $t = \max(n, k)$. We denote prover cost by t_P, verification cost by t_V and argument size by $|\pi|$. We only report cryptographic operations for t_P and t_V.

À la carte Proof System for Non-Uniform Computations. We use our bivariate PCS bPCLB to construct a new lookup protocol for tuples, and together with the argument for the relation $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$, we design an À la carte proof system for non-uniform computations.

Tuple lookup. We present a new lookup protocol for tuples based on our bivariate PCS bPCLB. For integers k, m, n, the vector of m-tuples $\mathbf{A} = (\mathbf{a}_0, \ldots, \mathbf{a}_{n-1})$ is a *subvector* of $\mathbf{T} = (\mathbf{t}_0, \ldots, \mathbf{t}_{k-1})$, if for all $i \in [n]$, there exists $j \in [k]$ such that $\mathbf{a}_i = \mathbf{t}_j$. Typically, we wish to check that $\mathbf{A}$ is a subvector of $\mathbf{T}$ given commitments to $\mathbf{A}$ and $\mathbf{T}$ under a suitable commitment scheme. When $m = 1$, this is a regular lookup argument. For most of the $m = 1$ schemes, the case of $m > 1$ can be obtained by committing vectors of m-tuples and then using a random linear combination over the field vectors to reduce the subvector relation over tuples to the one over field vectors. However, this straightforward approach causes both the commitment size and verification complexity to scale as $O(m)$. Recent works [CGG$^+$24, DXNT24] achieve argument size and verification cost independent of m, but incur a multiplicative overhead ($O(mn)$) in the size of the public parameters and the cryptographic operations required by the prover.

We present an approach for tuple lookup based on bPCLB where the size of public parameters is $O(m+n)$, proof generation requires $O(m+n)$ cryptographic operations, and the argument size and verification cost are logarithmic. We construct protocols for lookup (the vectors are committed) and committed index lookup (both the vectors and the positions are committed). We compare our tuple lookup with [CGG$^+$24] and [DXNT24] in Table 1.

À la carte Prover. À la carte prover cost profile refers to the prover's complexity being proportional only to the size (sum of sizes) of circuit(s) of the operations invoked by a program execution (and independent of the size of circuits corresponding to non-invoked instructions). We use our aggregation techniques in conjunction with the above tuple lookup to achieve an "à la carte" proof system for non-uniform computations captured in PLONK constraints. Towards this, we also use our bivariate PCS bPCLB to construct a generalization of PLONK-style grand-product for permutation checks that is of independent interest. Our scheme makes purely black-box use of cryptographic primitives, and serves as a practical alternative to non-uniform IVC schemes [KS22, BC23, KS24]. Our scheme also achieves faster proof generation than existing approaches that make black-box use of cryptographic primitives, such as [DXNT24, CGG$^+$24]. In both the prior works, non-uniform proof generation involves two key steps (i) prover uses the tuple lookup argument to lookup "sub-circuits" (modeled as tuples) involved in the computation, from a pre-defined table of such sub-circuits and later (ii) proves the correctness of the circuit assembled from the looked up sub-circuits. The first step incurs $\widetilde{O}(mn)$ cryptographic operations, while the second step invokes Plonk/Marlin prover on the $O(mn)$-sized assembled circuit. Following a similar approach, we first use tuple lookup to commit to circuit polynomials for each step (potentially in an input defined manner), and then provide an aggregated proof for all the steps. As noted earlier, our tuple lookup is substantially faster. Further, our proof aggregation is $25-30\%$ more efficient than the monolithic proof for $O(mn)$ sized circuit.

We reiterate that our focus is only to showcase the versatility of the GAPP framework and its efficient instantiation from our bivariate PCS bPCLB. While à la carte prover for non-uniform computation can also be used to obtain proofs of machine execution (zkVM), where current instruction determines the invoked computation, we leave detailed, end-to-end optimized construction of zkVM using our methods and its comparison with existing approaches as future work.

1.2 Technical overview

Our Bivariate PCS. We first present an overview of our new bivariate polynomial commitment scheme bPCLB. Similar to the approach in [BMM+21], we use KZG PCS to create commitments $\{C_i\}$ to the Lagrange coefficients $\{p_i(Y)\}$ of the bivariate polynomial $P(X,Y)$, and then compute $C_P = \sum_{i=0}^{n-1} e(C_i, w_i)$, where $\mathbf{w} = (w_0, \dots, w_{n-1}) \in \mathbb{G}_2^n$ is a commitment key. Our core technical innovation is in achieving a logarithmic size opening for this commitment. To illustrate the challenges thereof, we recall the techniques used in [BMM+21, GMN22].

To open the bivariate polynomial $P(X,Y)$ with commitment C_P (computed using the monomial basis), the existing approaches in [BMM+21, GMN22] generalize the "split and fold" technique used in the inner product protocols in [BCC+16, BBB+18] and in the compressed sigma protocol framework of [AC20]. In more detail, to open the polynomial commitment to a value v at (x,y), the prover first computes the commitment C_p to the univariate polynomial $P(x,Y)$ and sends it to the verifier. Since $P(x,Y) = \sum_{i=0}^{n-1} p_i(Y)x^i$, the homomorphic property of the KZG commitment scheme implies that $C_p = \sum_{i=0}^{n-1} x^i C_i$ is a commitment to $P(x,Y)$. Next, the prover opens the univariate polynomial $P(x,Y)$ to the value v at $Y = y$. The key step in the protocol is for the prover to convince the verifier that the commitment C_p to the univariate polynomial is consistent with the commitment C_P to the bivariate polynomial. In other words, the prover proves knowledge of the vector $C_0, \dots, C_{n-1} \in \mathbb{G}_1^n$ such that:

$$C_P = \sum_{i=0}^{n-1} e(C_i, w_i) \bigwedge C_p = \sum_{i=0}^{n-1} x^i C_i \qquad (1)$$

In the above, C_P can be considered a commitment to $\mathbf{C} = (C_0, \dots, C_{n-1})$ under the commitment key $\mathbf{w} = (w_0, \dots, w_{n-1})$ which is *doubly homomorphic*[1], whereas C_p can be considered a *linear form* given by the key $(1, x, \dots, x^{n-1})$.

Note that the relation in Equation 1 can be proved using the split and fold technique of [BCC+16, BBB+18, AC20]. However, naïvely using the techniques from these works results in a linear time verifier. This is due to the fact that verifier is required to compute the "folded" commitment key and linear form in each round. To address this, the prior works [BMM+21, GMN22] consider commitment keys w_i, $i \in [n]$ with monomial structure, i.e. $w_i = \tau^i \cdot g_2 = \left[\tau^i\right]_2$ for some trapdoor $\tau \leftarrow \mathbb{F}$. The linear form already inherits the monomial structure $(1, x, \dots, x^{n-1})$ from the representation of the bivariate polynomial $P(X,Y)$ in the monomial basis consisting of powers of X. The key observation made about such structured commitment keys is that the folding can be delegated to the prover, and the verifier can efficiently (in $O(\log n)$ time) check the correctness of the final commitment key and the linear form.

[1]The commitment is homomorphic both in commitment key and message

Our GAPP Framework. We use $[a]$ for $a \in \mathbb{N}$ to denote the set $[0, a-1]$. A polynomial protocol involves proving that a polynomial identity holds over some subset $\mathbb{V}$ of $\mathbb{F}$, where the polynomials involved in the identity are committed using some polynomial commitment scheme PC. Typically, the subset $\mathbb{V}$ is taken to be the set of m^{th} roots of unity for some $m \in \mathbb{Z}$. Formally, the goal is to prove a polynomial identity of the form $G(p_0(Y), \ldots, p_{\ell-1}(Y)) = 0$ vanishes over $\mathbb{V}$, for some multivariate polynomial G, given a set of commitments $(C_0, \ldots, C_{\ell-1})$, where each C_j is a commitment to the polynomial $p_j(Y)$ under the polynomial commitment scheme (PCS) PC. Without loss of generality, we assume that, for some $k \in [\ell]$, the commitments $(C_0, \ldots, C_{k-1})$ are honestly generated, while the remaining commitments $(C_k, \ldots, C_{\ell-1})$ are adversarially generated. For example, proof generation in PLONK [GWC19] involves showing that a polynomial $G_{\mathsf{p}}(q_M(Y), q_L(Y), q_R(Y), q_O(Y), q_C(Y), a(Y), b(Y), c(Y))$ vanishes over $\mathbb{V}$, where $G_{\mathsf{p}}(q_M, q_L, q_R, q_O, q_C, a, b, c) = q_M ab + q_L a + q_R b + q_O c + q_C$. Here, commitments to the circuit polynomials $(q_M, q_L, q_R, q_O, q_C)$ are trusted (being outputs of one-time preprocessing), while commitments to the witness polynomials (a, b, c), generated by the prover, may be malicious.

Aggregating Polynomial Protocols. Suppose a prover wishes to prove n homogeneous polynomial identities $G(p_{i,0}(Y), \ldots, p_{i,\ell-1}(Y)) = 0 \bmod Z_{\mathbb{V}}(Y)$ for each $i \in [n]$,given a set of commitments $(C_{i,0}, \ldots, C_{i,\ell-1})$, where $C_{i,j}$ is a commitment to $p_{i,j}(Y)$ under the PCS PC. Note that the naïve approach would be to run n instances of any polynomial protocol compatible with PC. This approach involves proving a statement of size $O(n\ell)$, and results in an argument size of $O(n\ell|\pi_{\mathsf{PC}}|)$ (where $|\pi_{\mathsf{PC}}|$ denotes the opening size of PC).

Aggregation using Bivariate Polynomials. First, to reduce the statement size from $n \times \ell$ to ℓ, we create a single commitment to a *vector of polynomials* $(p_{i,j})_{i \in [0,n-1]}$ for a given $j \in [\ell]$. Let $\mathbb{H} = \{1, \omega, \ldots, \omega^{n-1}\}$ be the subgroup consisting of the n^{th} roots of unity in $\mathbb{F}$, and let $\mu_i^{\mathbb{H}}(X)$ be the corresponding Lagrange polynomial for each $i \in [n]$. We say that the *packed* polynomial corresponding to the vector of univariate polynomials $(p_{i,j})_{i \in [n]}$ is the bivariate polynomial $P_j(X, Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) p_{i,j}(Y)$. For this packing scheme, the satisfiability of a set of n univariate polynomial identities over $\mathbb{V}$ reduces to showing that the bivariate polynomial $Q(X, Y) = G(P_0(X, Y), \ldots, P_{\ell-1}(X, Y))$ vanishes over $\mathbb{H} \times \mathbb{V}$. This is analogous to compressing n arithmetic constraints into an equivalent polynomial constraint over interpolated polynomials in several SNARK constructions. See Lemma 4.1 for a precise exposition.

Now we wish to prove knowledge of bivariate polynomials $(P_0, \ldots, P_{\ell-1})$ corresponding to commitments $(C_0, \ldots, C_{\ell-1})$ s.t. $G(p_{i,0}(Y), \ldots, p_{i,\ell-1}(Y))$ vanishes over $\mathbb{V}$ for each $i \in [n]$, where for each $j \in [\ell]$, $p_{0,j}(Y), \ldots, p_{n-1,j}(Y)$ are the uniquely determined univariate components of $P_j(X, Y)$ with respect to the polynomial basis $(\mu_0^{\mathbb{H}}(X), \ldots, \mu_{n-1}^{\mathbb{H}}(X))$. In a real application, for some $k \in [\ell]$, the commitments $(C_0, \ldots, C_{k-1})$ would be honestly generated (i.e., trusted by the verifier), while the commitments $(C_k, \ldots, C_{\ell-1})$ would serve as aggregated commitments to the prover's witness (potentially adversarially generated).

The GAPP Relation. Let bPC be a bivariate PCS with public parameter pp. We informally define the relation $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ as follows:

Definition 1.1 (GAPP (informal)). *Let* $\mathbf{C} = (C_0, \ldots, C_{\ell-1})$ *be a vector of commitments, and let* $\mathbf{w} = (P_0(X, Y), \ldots, P_{\ell-1}(X, Y))$ *be a vector of bivariate polynomials. We say that* $(\mathbf{C}, \mathbf{w}) \in \mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ *if:*

1. *For each* $j \in [\ell]$, $P_j(X, Y)$ *opens the commitment* C_j.

2. *For each $i \in [n]$, $G(P_0(\omega^i, Y), \ldots, P_{\ell-1}(\omega^i, Y)) = 0 \bmod Z_{\mathbb{V}}(Y)$, where ω is the canonical primitive n^{th} root of unity in $\mathbb{F}$, and where $P_j(\omega^i, Y)$ is the i^{th} univariate polynomial $p_{i,j}(Y)$ "packed" into P_j.*

In Section 4, we formally define a more generalized version of this relation that allows capturing polynomial protocols where the same polynomial appears with different parameterizations. See Definition 4.1 for the details.

Argument of Knowledge for GAPP. We construct an information-theoretic protocol where the prover's messages are restricted to be low-degree (univariate and bivariate) polynomials, and compile this using bPCLB to obtain an argument of knowledge for $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ with *succinct verification* and an *efficient prover*. The information-theoretic protocol requires the prover to show that $Q(X, Y) = G(P_0(X, Y), \ldots, P_{\ell-1}(X, Y))$ vanishes over $\mathbb{H} \times \mathbb{V}$. At a high level, the prover produces low-degree polynomials $Q(X, Y)$ and $H(X, Y)$ such that $Q(X, Y) - Z_{\mathbb{V}}(Y)H(X, Y)$ vanishes over $\mathbb{H}$, to which the verifier has oracle access. The verifier probabilistically checks the identity $Q(X, y) - Z_{\mathbb{V}}(y)H(X, y) = 0 \bmod Z_{\mathbb{H}}(X)$ for $y \leftarrow \mathbb{F}$. Concretely, the prover produces univariate polynomials $q(X) = Q(X, y)$, $h(X) = H(X, y)$, and $u(X)$, and the verifier probabilistically checks $q(X) - Z_{\mathbb{V}}(y)h(X) = u(X)Z_{\mathbb{H}}(X)$ by querying the oracles for $q(X), h(X)$ and $u(X)$ at a random point. Finally, to establish that $q(X) = Q(X, y)$ and $h(X) = H(X, y)$, the verifier samples $x \leftarrow \mathbb{F}$, queries the polynomials Q and H at (x, y), queries the polynomials q and h at x, and checks that $Q(x, y) = q(x)$ and $H(x, y) = h(x)$.

À la carte Proof System for Non-Uniform Computations. We combine our aggregation techniques with our tuple lookup and permutation check from the generalized grand-product described below to design "à la carte" proof system for non-uniform computations captured in PLONK constraints.

Generalized Grand-product. We use our packing gadget idea and the bivariate PCS bPCLB to construct a generalization of PLONK-style grand-product for permutation checks. In the grand product protocol, given a polynomial $f(X)$ that encodes a set of wires $\mathbf{a} = (a_0, \ldots, a_{n-1})$, the protocol proves that $a_{\sigma(i)} = a_i$ for all $i \in [n]$. In our generalization, given a bivariate polynomial $A(X, Y)$ with $\deg_X(A) < n$ and $\deg_Y(m) < m$, encoding a set of wires $\mathbf{a} = (a_0, \ldots, a_{nm-1})$ and a permutation $\sigma : [nm] \to [nm]$, we present a *bivariate* PIOP, that checks $a_{\sigma(i)} = a_i$ for all $i \in [nm]$. Using bPCLB, we obtain a logarithmic-sized proof as opposed to a proof size of $O(n)$ resulting from the straightforward protocol that commits to each of the univariate polynomials.

'A la carte Proof System. The template of our construction is the following. We consider an n-step non-uniform computation F using the PLONK PIOP. We use the tuple lookup argument to prove that correct circuit polynomials are used for each step, i.e, they correspond to selector wires τ_i that pick a function from a family. We then use our aggregation technique to provide an aggregate proof of correctness for all selected circuits. This is done by showing that witness polynomials satisfy the PLONK constraints for F_{τ_i} for all $i \in [n]$. Finally, we need to show that inputs and outputs of different circuits are consistent as captured by some permutation σ (analogous to the wiring/copy constraints in regular PLONK). This is shown using our generalized grand-product.

1.3 Additional Related Work

Proof Aggregation. Popular proof aggregation schemes include SnarkPack [GMN22] for aggregating Groth16 proofs, aPlonk [ABST23] for aggregating PLONK [GWC19] proofs, and aHyperProofs [YZRM24] which is a multivariate counterpart to aPlonk and aggregates HyperPlonk [CBBZ23] proofs. Hyperproofs [SCP+22] provides Merkle-like proofs based on polynomial commitments which can be efficiently aggregated. [GMN22, ABST23, YZRM24] rely on multi-polynomial commitments and inner pairing-product arguments from [BMM+21] to fold pairing checks for n proof verifications into one pairing check over multi-commitments. The PIOP based schemes [ABST23, YZRM24] subsequently verify the polynomial identities inside a *meta* arithmetic circuit, whose correctness is proved using a separate SNARK proof. These existing aggregation frameworks require relation-specific setup for each instance due to their dependence on the arithmetic circuit defining the aggregation relation. These approaches are also tailored to specific protocols, such as Groth16 and PLONK. In contrast, our methods apply more generally and avoid the need for relation-specific setup.

The recent work of [LXZ+24] uses bivariate polynomials in Lagrange basis to capture a batch of univariate constraints, albeit towards a completely different goal of distributing a SNARK prover. Their distributed protocol also requires proof aggregation as a key technique. As noted earlier, their techniques are very specific to PLONK and crucially rely on customized KZG commitments, which are incompatible with existing PLONK circuits based on publicly available universal SRS generated using `powers of tau` ceremony [pot]. Our proof aggregation mechanism applies generally to any polynomial protocol, and avoids the need for such customized KZG commitments. The technique of using bivariate polynomials is used in an adhoc way and for a specific proof system, PLONK. In this paper, we formalize the paradigm into a general framework applicable to any polynomial protocol; and additionally present a new bivariate polynomial commitment scheme that yields an efficient instantiation of this framework.

Lookup Arguments. Early lookup arguments such as Arya [BCG+18] and Plookup [GW20] were constructed as means for "custom gates" in SNARKs. These schemes incur proving cost of $O(n + k)$ cryptographic operations where n and k are the sizes of the table and the subvector, respectively. Recent works [BCG+18, GW20, Hab22, ZBK+22, PK22, ZGK+22, GK22, EFG22, CFF+24] have introduced substantial improvements. The works of [EFG22, CFF+24] based on "cached quotients" incur proving cost of $O(k)$, substantially improving over earlier works when $k \ll n$. Moreover, recent work [DGP+24] also obtains committed index lookups via an efficient reduction to un-indexed lookups. In the context of lookups over m-tuples, all of these works focus on $m = 1$. The naïve approach of constructing m-tuple lookup for $m > 1$ using these works incurs $O(m)$ multiplicative overhead in commitment/argument size and verification complexity.

Recent works [CGG+24, **?**] avoid the $O(m)$ multiplicative overhead by extending CQ [EFG22] and Plookup [GW20] PIOPs with additional algebraic constraints to atomically associate each m-tuple in a table $\mathbf{T}$ consisting of n such tuples with m distinct positions in a flattened table $\widetilde{\mathbf{T}}$ of size mn. An m tuple is now interpreted as a set of contiguous positions (equivalently, a *segment*) in [CGG+24], or as a coset of a subgroup of order n in [DXNT24]. While both the works achieve argument size and verification independent of m, they both require a setup of size $O(mn)$ and proof generation involving $\widetilde{O}(mn)$ cryptographic operations. In contrast, we propose a lookup argument for m-tuples with $O(m + n)$ setup, $O(m + n)$ cryptographic operations for proof generation, and logarithmic argument size and verification cost.

À la carte Prover. Proving correct machine computation involves proving the state transition function determined at each step by the specific instruction type. This typically incurs large prover costs due to the use of a universal circuit to capture any instruction supported by the machine. General purpose SNARKs for such applications require large universal circuits that encompass all possible execution paths, incurring similar proof generation overheads. the prover cost should only depend on the actual execution path (i.e., the clause that is actually executed), such that the cost only grows with the sizes of circuits corresponding to the the operations invoked by the program execution. Such "à la carte" prover cost profile when proving machine executions is enabled by recent works on non-uniform IVC [KS22, BC23, KS24]. However, these works make use of non-black-box use of cryptographic objects such as hash functions, groups etc, which limits their portability. Certain recent works [AST24, STW24] avoid non-uniform verification of the CPU state transition by proposing an approach based on lookup arguments.

Verifying non-uniform relations while only making black-box use of cryptography has been explored in MuxProofs [DXNT24] and SubPlonk [CGG+24]. The authors of [DXNT24, CGG+24] leverage SNARKs with *updatable* setup to verifiably obtain *computation commitments* (similar to relation-specific public parameters) for the active sub-circuit determined by the inputs. The correctness of the committed active sub-circuit is then proved by a specific SNARK, namely Marlin [CHM+20] in [DXNT24] and PLONK in [CGG+24]. Our proposed scheme also makes black-box use of cryptoprimitives, and achieves faster proof generation than [DXNT24, CGG+24].

For disjunctive NP relations, [GGHAK22, GHAKS23] describe stacking protocols for efficiently handling disjunctions by compiling IOPs and Sigma protocol composition. This approach incurs prover cost that depends only on the size of the clause executed and is independent of the total number of clauses. Other recent works such as [YHH+23] have used the MPC-in-the-head paradigm to construct efficient protocols for disjunctions and batched disjunctions.

2 Preliminaries

We present preliminary background material in this section.

Notations. We use $[n]$ to denote the set of integers $\{0, \ldots, n-1\}$ and $\mathbb{F}$ to denote a prime field of order p. We denote by λ a security parameter. We use negl to denote a negligible function: for any integer $c > 0$, there exists $n \in \mathbb{N}$, such that $\forall\ x > n$, $\mathsf{negl}(x) \leq 1/x^c$. We assume a bilinear group generator BG which on input λ outputs parameters for the protocols. Specifically $\mathsf{BG}(1^\lambda)$ outputs $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t)$ where: $\mathbb{F} = \mathbb{F}_p$ is a prime field of super-polynomial size in λ, with $p = \lambda^{\omega(1)}$; $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order p, and e is an efficiently computable non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; Generators g_1, g_2 are uniformly chosen from $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively and $g_t = e(g_1, g_2)$. We write groups $\mathbb{G}_1$ and $\mathbb{G}_2$ additively, and use the shorthand notation $[x]_1$ and $[x]_2$ to denote group elements $x \cdot g_1$ and $x \cdot g_2$ respectively for $x \in \mathbb{F}$. We implicitly assume that all the setup algorithms for the protocols invoke BG to generate descriptions of groups and fields over which the protocol is instantiated. We will als use sets $\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ to specify the type of operations, where additionally, we have $\mathbb{P}$ to denote pairings and $\mathbb{G}_1$ to denote multiexponentiation.

Lagrange Polynomials. At different points, we use groups $\mathbb{H}$, $\mathbb{V}$ and $\mathbb{K}$ generated by primitive n^{th}, m^{th} and k^{th} roots of unity. We use $\{\mu_i^{\mathbb{H}}(X)\}_{i=0}^{n-1}$, $\{\mu_i^{\mathbb{V}}(X)\}_{i=0}^{m-1}$ and $\{\mu_i^{\mathbb{K}}(X)\}_{i=0}^{k-1}$ as the Lagrange polynomials for sets $\mathbb{H}$, $\mathbb{V}$ and $\mathbb{K}$ respecitvely. We use $Z_{\mathbb{H}}(X)$, $Z_{\mathbb{V}}(X)$ and $Z_{\mathbb{K}}(X)$ to denote the vanishing polynomials of the respective sets. We will generally use ω as the n^{th} primitive root of unity and ν as the primitive m^{th} root of unity.

Succinct Arguments of Knowledge. Let $\mathcal{R}$ be a NP-relation and $\mathcal{L}$ be the corresponding NP-language, where $\mathcal{L} = \{x : \exists\ w \text{ such that } (x, w) \in \mathcal{R}\}$. Here, a prover $\mathcal{P}$ aims to convince a verifier $\mathcal{V}$ that $x \in \mathcal{L}$ by proving that it knows a witness w for a public statement x such that $(x, w) \in \mathcal{R}$. An interactive argument of knowledge for a relation $\mathcal{R}$ consists of a PPT algorithm Setup that takes as input the security parameter λ, and outputs the public parameters pp, and a pair of interactive PPT algorithms $\langle \mathcal{P}, \mathcal{V} \rangle$, where $\mathcal{P}$ takes as input (pp, x, w) and $\mathcal{V}$ takes as input (pp, x). An interactive argument of knowledge $\langle \mathcal{P}, \mathcal{V} \rangle$ must satisfy completeness and knowledge soundness.

Definition 2.1 (Completeness)**.** *For all security parameter $\lambda \in \mathbb{N}$ and statement x and witness w such that $(x, w) \in \mathcal{R}$, we have*

$$\Pr\left(b = 1 \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ b \leftarrow \langle \mathcal{P}(w), \mathcal{V} \rangle(\mathsf{pp}, x) \end{array} \right) = 1.$$

Definition 2.2 (Knowledge Soundness)**.** *For any PPT malicious prover $\mathcal{P}^* = (\mathcal{P}_1^*, \mathcal{P}_2*)$, there exists a PPT algorithm $\mathcal{E}$ such that the following probability is negligible:*

$$\Pr\left(\begin{array}{c} b = 1 \wedge \\ (x, w) \notin \mathcal{R} \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, \mathsf{st}) \leftarrow \mathcal{P}_1^*(1^\lambda, \mathsf{pp}) \\ b \leftarrow \langle \mathcal{P}_2^*(\mathsf{st}), \mathcal{V} \rangle(\mathsf{pp}, x) \\ w \leftarrow \mathcal{E}^{\mathcal{P}_2^*}(\mathsf{pp}, x) \end{array} \right).$$

A *succinct* argument of knowledge $\langle \mathcal{P}, \mathcal{V} \rangle$ for a relation $\mathcal{R}$, must satisfy completeness and knowledge soundness and additionally be *succinct*, that is, the communication complexity between prover and verifier, as well as the verification complexity is bounded by $\mathsf{poly}(\lambda, \log |w|)$.

Polynomial Commitment Scheme. A polynomial commitment scheme (PCS) introduced in [KZG10] allows a prover to open evaluations of the committed polynomial succinctly. A PCS over $\mathbb{F}$ is a tuple $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ where:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i \in [n]})$. On input security parameter λ, number of variables n and upper bounds $D_i \in \mathbb{N}$ on the degree of each variable X_i for a n-variate polynomial, Setup generates public parameters pp.

- $(C, \tilde{\mathbf{c}}) \leftarrow \mathsf{Com}(\mathsf{pp}, f(X_1, \cdots, X_n), d_1, \ldots, d_n)$. On input the public parameters pp, and a n-variate polynomial $f(X_1, \cdots, X_n) \in \mathbb{F}[X_1, \ldots, X_n]$ with degree at most $\deg(X_i) = d_i \leq D_i$ for all i, Com outputs a commitment to the polynomial C, and additionally an opening hint $\tilde{\mathbf{c}}$.

- $b \leftarrow \mathsf{Open}(\mathsf{pp}, f(X_1, \cdots, X_n), d_1, \ldots, d_n, C, \tilde{\mathbf{c}})$. On input the public parameters pp, the commitment C and the opening hint $\tilde{\mathbf{c}}$, a polynomial $f(X_1, \cdots, X_n)$ with $d_i \leq D_i$, Open outputs a bit indicating accept or reject.

- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, (d_1, \ldots, d_n), (x_1, \ldots, x_n), v; f(X_1, \cdots, X_n))$. A public coin interactive protocol $\langle P_{\mathsf{eval}}(f(X_1, \cdots, X_n)), V_{\mathsf{eval}} \rangle(\mathsf{pp}, C, (d_1, \ldots, d_n), (x_1, \ldots, x_n), v)$ between a PPT prover and a PPT verifier. The parties have as common input public parameters pp, commitment C, degree d, evaluation point x, and claimed evaluation v. The prover has, in addition, the opening

$f(X_1, \cdots, X_n)$ of C, with $\deg(X_i) \leq d_i$. At the end of the protocol, the verifier outputs 1 indicating accepting the proof that $f(x_1, \ldots, x_n) = v$, or outputs 0 indicating rejecting the proof.

A polynomial commitment scheme must satisfy completeness, binding and extractability.

Definition 2.3 (Completeness)**.** *For all polynomials $f(X_1, \cdots, X_n) \in \mathbb{F}[X_1, \ldots, X_n]$ with degree $\deg(X_i) = d_i \leq D_i$, for all $(x_1, \ldots, x_n) \in \mathbb{F}^n$,*

$$\Pr\left(b = 1 \ : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i\in[n]}) \\ (C, \tilde{\mathbf{c}}) \leftarrow \mathsf{Com}(\mathsf{pp}, f(X_1, \cdots, X_n), d_1, \ldots, d_n) \\ v \leftarrow f(x) \\ b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, (d_1, \ldots, d_n), (x_1, \ldots, x_n), v; f(X_1, \cdots, X_n)) \end{array} \right) = 1.$$

Definition 2.4 (Binding)**.** *A polynomial commitment scheme* PC *is binding if for all PPT $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left(\begin{array}{c} \mathsf{Open}(\mathsf{pp}, f_0, \mathbf{d_0}, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1 \wedge \\ \mathsf{Open}(\mathsf{pp}, f_1, \mathbf{d_1}, C, \tilde{\mathbf{c}}_\mathbf{1}) = 1 \wedge \\ f_0 \neq f_1 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i\in[n]}) \\ (C, f_0, f_1, \tilde{\mathbf{c}}_\mathbf{0}, \tilde{\mathbf{c}}_\mathbf{1}, \mathbf{d_0}, \mathbf{d_1}) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array} \right).$$

Definition 2.5 (Knowledge Soundness)**.** *For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a PPT algorithm $\mathcal{E}$ such that the following probability is negligible in λ:*

$$\Pr\left(\begin{array}{c} b = 1 \wedge \\ \mathcal{R}_{\mathsf{Eval}}(\mathsf{pp}, C, \mathbf{x}, v; \tilde{f}, \tilde{\mathbf{c}}) = 0 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i\in[n]}) \\ (C, \mathbf{d}, \mathbf{x}, v, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ (\tilde{f}, \tilde{\mathbf{c}}) \leftarrow \mathcal{E}^{\mathcal{A}_2}(\mathsf{pp}, C, d) \\ b \leftarrow \langle \mathcal{A}_2(\mathsf{st}), V_{\mathsf{eval}} \rangle (\mathsf{pp}, C, \mathbf{d}, \mathbf{x}, v) \end{array} \right).$$

where the relation $\mathcal{R}_{\mathsf{Eval}}$ is defined as follows:

$$\begin{array}{c} \mathcal{R}_{\mathsf{Eval}} = \{((\mathsf{pp}, C \in \mathbb{G},\ \mathbf{x} \in \mathbb{F}^n,\ v \in \mathbb{F});\ (f(X_1, \cdots, X_n), \tilde{\mathbf{c}})) : \\ (\mathsf{Open}(\mathsf{pp}, f, \mathbf{d}, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1) \wedge v = f(\mathbf{x})\} \end{array}$$

We denote by $\mathsf{Prove}, \mathsf{Verify}$, the non-interactive prover and verifier algorithms obtained by applying FS to the Eval public-coin interactive protocol, giving a non-interactive PCS scheme $(\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, d), C \leftarrow \mathsf{Com}(\mathsf{pp}, f(\mathbf{X})), (v, \pi) \leftarrow \mathsf{Prove}(\mathsf{pp}, f(\mathbf{X}), x),\ b \leftarrow \mathsf{Verify}(\mathsf{pp}, C, v, x, \pi)$.

Definition 2.6 (Knowledge Soundness for Non-Interactive PCS)**.** *For any PPT adversary $\mathcal{A}$, there exists a PPT algorithm $\mathcal{E}$ such that the following probability is negligible in λ:*

$$\Pr\left(\begin{array}{c} b = 1 \wedge \\ \mathcal{R}_{\mathsf{Eval}}(\mathsf{pp}, C, \mathbf{x}, v; \tilde{f}, \tilde{\mathbf{c}}) = 0 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i\in[n]}) \\ (C, \mathbf{d}, \mathbf{x}, v, \pi) \leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pp}) \\ (\tilde{f}, \tilde{\mathbf{c}}) \leftarrow \mathcal{E}^{\mathcal{A}, \mathsf{RO}}(\mathsf{pp}, C, d) \\ b \leftarrow \mathsf{Verify}(\mathsf{pp}, C, \mathbf{d}, \mathbf{x}, v, \pi) \end{array} \right).$$

where the relation $\mathcal{R}_{\mathsf{Eval}}$ is defined as follows:

$$\begin{array}{c} \mathcal{R}_{\mathsf{Eval}} = \{((\mathsf{pp}, C \in \mathbb{G},\ \mathbf{x} \in \mathbb{F}^n,\ v \in \mathbb{F});\ (f(X_1, \cdots, X_n), \tilde{\mathbf{c}})) : \\ (\mathsf{Open}(\mathsf{pp}, f, \mathbf{d}, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1) \wedge v = f(\mathbf{x})\} \end{array}$$

Definition 2.7 (Succinctness)**.** *We require the commitments and the evaluation proofs to be of size independent of the degree of the polynomial, that is the scheme is* proof succinct *if $|C|$ is* $\mathsf{poly}(\lambda)$*, $|\pi|$ is* $\mathsf{poly}(\lambda)$ *where π is the transcript obtained by applying FS to* Eval*. Additionally, the scheme is* verifier succinct *if* Eval *runs in time* $\mathsf{poly}(\lambda) \cdot \mathsf{log}(d)$ *for the verifier.*

Fiat-Shamir. An interactive protocol is *public-coin* if the verifier's messages are uniformly random strings. Public-coin protocols can be transformed into non-interactive arguments in the Random Oracle Model (ROM) by using the Fiat-Shamir (FS) [FS87] heuristic to derive the verifier's messages as the output of a Random Oracle. All protocols in this work are public-coin interactive protocols in the structured reference string (SRS) model where both the parties have access to a SRS, that are then compiled into non-interactive arguments using FS. We denote by $\mathsf{Prove}, \mathsf{Verify}$, the non-interactive prover and verifier algorithms obtained by applying FS to the Eval public-coin interactive protocol, giving a non-interactive PCS scheme ($\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, d), (C, \tilde{\mathbf{c}}) \leftarrow \mathsf{Com}(\mathsf{pp}, f(\mathbf{X})), (v, \pi) \leftarrow \mathsf{Prove}(\mathsf{pp}, f(\mathbf{X}), x)$, $b \leftarrow \mathsf{Verify}(\mathsf{pp}, C, v, x, \pi)$.

KZG PCS. The KZG univariate PCS was introduced in [KZG10]. We denote the KZG scheme by the tuple of PPT algorithms ($\mathsf{KZG.Setup}$, $\mathsf{KZG.Commit}$, $\mathsf{KZG.Prove}$, $\mathsf{KZG.Verify}$) as defined below.

Definition 2.8 (KZG PCS). *Let* $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t)$ *be output of bilinear group generator* $\mathsf{BG}(1^\lambda)$.

- $\mathsf{KZG.Setup}$ *on input* $(1^\lambda, d)$*, where* d *is the degree bound, outputs*

$$\mathsf{srs} = (\{[\tau]_1, \ldots, [\tau^d]_1\}, \{[\tau]_2, \ldots, [\tau^d]_2\}$$

- $\mathsf{KZG.Commit}$ *on input* $(\mathsf{srs}, p(X))$*, where* $p(X) \in \mathbb{F}_{\leq d}[X]$*, outputs* $C = [p(\tau)]_1$

- $\mathsf{KZG.Prove}$ *on input* $(\mathsf{srs}, p(X), \alpha)$*, where* $p(X) \in \mathbb{F}_{\leq d}[X]$ *and* $\alpha \in \mathbb{F}$*, outputs* (v, π) *such that* $v = p(\alpha)$ *and* $\pi = [q(\tau)]_1$*, for* $q(X) = \frac{p(X)-p(\alpha)}{X-\alpha}$

- $\mathsf{KZG.Verify}$ *on input* $(\mathsf{srs}, C, v, \alpha, \pi)$*, outputs* 1 *if the following equation holds, and* 0 *otherwise:* $e(C - v[1]_1 + \alpha\pi, [1]_2) \stackrel{?}{=} e(\pi, [\tau]_2)$

Polynomial Protocols. A modular approach for designing efficient succinct arguments consists of two steps: (i) constructing an information theoretic protocol in an idealized model, (ii) compiling the information-theoretic protocol via a cryptographic compiler to obtain an argument system. Informally, the prover and the verifier interact where the prover provides oracle access to a set of polynomials, and the verifier accepts or rejects by checking certain identities over the polynomials output by the prover and possibly public polynomials known to the verifier. Such a *polynomial protocol* is compiled into a succinct argument of knowledge by realizing the polynomial oracles using a *polynomial commitment scheme*. A polynomial commitment scheme allows a prover to commit to polynomials, and later verifiably open evaluations at chosen points by giving evaluation proofs. This enables the verifier to probabilistically check polynomial identities at random points of $\mathbb{F}$. Many recent constructions of zkSNARKs [BFS20, CHM+20, GWC19] follow this approach where the information theoretic object is a polynomial protocol and the cryptographic compiler is a polynomial commitment scheme.

Algebraic Group Model. We analyze the security of our protocols in the Algebraic Group Model (AGM) introduced in [FKL18]. An adversary $\mathcal{A}$ is called *algebraic* if every group element output by $\mathcal{A}$ is accompanied by a representation of that group element in terms of all the group elements that $\mathcal{A}$ has seen so far (input and output). In the AGM, an adversary $\mathcal{A}$ is restricted to be *algebraic*, which in our SRS-based protocol means a PPT algorithm satisfying the following: Given $\mathsf{srs} = (\mathsf{srs}_1, \mathsf{srs}_2)$, whenever $\mathcal{A}$ outputs an element $A \in \mathbb{G}_i, i \in 1, 2$, it is accompanied by its representation, i.e, $\mathcal{A}$ also outputs a vector $\mathbf{v}$ over $\mathbb{F}$ such that $A = \langle \mathbf{v}, \mathsf{srs}_i \rangle$.

The q-DLOG Assumption. We define the q-DLOG assumption below.

Definition 2.9 (q-DLOG Assumption). *The q-DLOG assumption with respect to $\mathcal{G}$ holds if for all λ and for all PPT $\mathcal{A}$, we have:*

$$\Pr\left[\begin{array}{c}\tau = \tau' \\ \tau' \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp})\end{array} : \begin{array}{c}(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t) \leftarrow \mathsf{BG}(1^\lambda),\ \tau \leftarrow \mathbb{F} \\ \mathsf{pp} := (g_1^\tau, g_1^{\tau^2}, \ldots, g_1^{\tau^q}, g_2^\tau, g_2^{\tau^2}, \ldots, g_2^{\tau^q})\end{array}\right] \leq \mathsf{negl}(\lambda)$$

3 Bivariate PCS in Lagrange Basis: bPCLB

In this section, we present our new bivariate PCS bPCLB. Technically, we rely on Inner Product Arguments (IPA) [BCC+16, BBB+18, BMM+21] and Compressed Sigma Protocols (CSP) [AC20], which we describe with respect to a bilinear group BG below (see Appendix A for a more general exposition).

Inner Product Arguments, Compressed Sigma Protocols. Let CM denote the commitment scheme with key space $\mathbb{G}_2^n$, message space $\mathbb{G}_1^n$ and commitment space as $\mathbb{G}_T$, where commitment to $\mathbf{u} \in \mathbb{G}_1^n$ under the key $\mathbf{v} \in \mathbb{G}_2^n$ is given by $C_u = \mathsf{CM}(\mathbf{v}, \mathbf{u}) = \sum_{i=0}^{n-1} e(u_i, v_i)$. For simplicity, we have considered a *non-hiding* commitment here, though the modification to a hiding commitment is straightforward. Similarly, for $\mathbf{a} \in \mathbb{F}^n$, we call the map $L_a : \mathbb{G}_1^n \to \mathbb{G}_1$ defined by $\mathbf{u} \mapsto \sum_{i=0}^{n-1} a_i u_i$ as the linear form on $\mathbb{G}_1^n$, defined by the vector $\mathbf{a}$. We also conveniently denote the commitment CM and the linear form as inner products $\langle \mathbf{v}\,,\, \mathbf{u} \rangle_\otimes$ and $\langle \mathbf{a}\,,\, \mathbf{u} \rangle$ respectively. The CSPs and IPAs provide elegant arguments of knowledge for proving linear forms over committed vectors. In particular, they provide $O(\log n)$ size argument for the following relation over $(\mathbf{v}, \mathbf{a}, C_P, C_p; \mathbf{C})$ given by $\langle \mathbf{v}\,,\, \mathbf{C} \rangle_\otimes = C_P$ and $\langle \mathbf{a}\,,\, \mathbf{C} \rangle = C_p$.

3.1 The Bivariate PCS bPCLB

The bivariate PCS bPCLB is summarized in Figure 1. We also describe the key steps below.

Setup. The setup for bPCLB for degree bound (d_x, d_y) consists of KZG setup $(\mathsf{uPC.pk}, \mathsf{uPC.ck}) = \left(([\tau^i]_1)_{i=0}^{d_y}, [\tau]_2\right)$ for $\tau \leftarrow \mathbb{F}$, which forms the *inner commitment scheme*. For *outer commitment*, the setup generates $\mathbf{v} = ([\beta^i]_2)_{i=0}^{d_x}$ for $\beta \leftarrow \mathbb{F}$. The setup outputs $\mathsf{pk} = (\mathsf{uPC.pk}, \mathbf{v})$ and $\mathsf{ck} = (\mathsf{uPC.ck}, [\beta]_1)$.

Remark 3.1. *Both [BMM+21] and [GMN22] note that the outer commitment $\mathbf{C} \mapsto \sum_{i=0}^n e(C_i, [\beta^i]_2)$ is not binding, as $[\beta]_1$ in* ck *can be used to construct a collision; in particular, $([\beta]_1, [0]_1)$ and $([0]_1, [1]_1)$ yield the same commitment. To ensure binding, [BMM+21] encodes only even powers of β in the vector $\mathbf{v}$, while [GMN22] also commits the vector using an independently sampled key in $\mathbb{G}_2^n$. However, [ABST23] observed that these mitigation strategies are not required when using* KZG *as the inner commitment scheme with an independently sampled setup trapdoor τ. We follow the same approach and do not constrain the commitment key $\mathbf{v}$ in any way.*

Commitment. To commit to $P \in \mathbb{F}[X, Y]$ with $deg_X(P) < n$ and $\deg_Y(P) < m$, the prover writes P in Lagrange basis as $P(X, Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) p_i(Y)$, and computes inner commitments $(C_i, \tilde{\mathbf{c}}_i) = \mathsf{uPC.Com}(\mathsf{uPC.pk}, m, p_i)$ for $i \in [n]$. Finally, the prover computes the outer commitment $(C_P, \tilde{\mathbf{C}}) = \mathsf{CM}(\mathbf{v}, \mathbf{C})$ with $C_P = \sum_{i=0}^{n-1} e(C_i, v_i)$ as the commitment to P. Since, we are considering non-hiding outer commitment, we may assume that $\tilde{\mathbf{C}} = [0]_1$.

Let $\mathsf{BG} = (\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t)$, be a bilinear group with efficiently computable non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Let uPC denote the KZG univariate PCS with commitments in $\mathbb{G}_1$. bPCLB consists of PPT algorithms ($\mathsf{bPCLB.Setup}, \mathsf{bPCLB.Com}, \mathsf{bPCLB.Eval}, \mathsf{bPCLB.Open}$) defined below.

- $\underline{\mathsf{bPCLB.Setup}(1^\lambda, d_x, d_y)}$ takes degree bounds d_x and d_y in variables X and Y respectively as inputs. It outputs as follows:
  1. $(\mathsf{uPC.pk}, \mathsf{uPC.ck}) = \mathsf{KZG.Setup}(1^\lambda, d_y)$.
  2. $\mathbf{v} = \left(\left[\beta^i\right]_2 \right)_{i \in [d_x]}$ for $\beta \leftarrow \mathbb{F}$.
  3. $\mathbb{H} = \{1, \omega, \ldots, \omega^{d_x - 1}\}$, where $\omega \in \mathbb{F}$ is primitive d_x^{th} root of unity.
  4. $\mathsf{pk} = (\mathsf{uPC.pk}, \mathbf{v})$, $\mathsf{ck} = (\mathsf{uPC.ck}, [\beta]_1)$.
  5. Output $(\mathsf{pk}, \mathsf{ck})$
- $\underline{\mathsf{bPCLB.Com}(\mathsf{pk}, F, (n, m))}$ takes proving key pk and polynomial $F \in \mathbb{F}[X, Y]$ with $\deg_X(F) < n$ and $\deg_Y(F) < m$. The algorithm outputs commitment C_F as follows:
  1. Compute $(C_i, \tilde{c}_i) \leftarrow \mathsf{uPC.Com}(\mathsf{uPC.pk}, F(\omega_n^i, Y), m)$ for $i \in [n]$. Here $\omega_n \in \mathbb{H}$ is a primitive n^{th} root of unity. We assume that $n \,|\, d_x$.
  2. Compute $C_F = \mathsf{CM}_e(\mathbf{v}, \mathbf{C}) = \sum_{i=0}^{n-1} e(C_i, v_i)$. Here CM_e denotes inner product commitment given by bilinear operator $v_i \otimes C_i = e(C_i, v_i)$.
  3. Output $(C_F, \widetilde{\mathbf{c}})$ where $\widetilde{\mathbf{c}} = (\tilde{c}_0, \ldots, \tilde{c}_{n-1})$.
- $\underline{\mathsf{bPCLB.Eval}}$ is an interactive protocol between $\mathcal{P}(\mathsf{pk}, F, \widetilde{\mathbf{c}}, (n, m), (x, y), v)$ and $\mathcal{V}(\mathsf{ck}, C_F, (n, m), (x, y), v)$.
  - **Round 1**: Prover commits to univariate restriction.
    1. $\mathcal{P}$ computes $f(Y) = F(x, Y)$, commitment $C_f = \mathsf{uPC.Com}(\mathsf{uPC.pk}, f)$.
    2. $\mathcal{P}$ computes $\pi \leftarrow \mathsf{uPC.Prove}(\mathsf{uPC.pk}, f, y)$.
    3. $\mathcal{P}$ sends the commitment C_f and opening proof π.
  - **Round 2**: Verifier checks consistency of univariate restriction.
    1. $\mathcal{P}$ and $\mathcal{V}$ run a CSP argument of knowledge π_{csp} to prove knowledge of $\mathbf{w} = (w_0, \ldots, w_{n-1}) \in \mathbb{G}_1^n$ for the relation:
    $$\sum_{i=0}^{n-1} e(w_i, v_i) = C_F \;\wedge\; \sum_{i=0}^{n-1} \mu_i(x) \cdot w_i = C_f \tag{2}$$
    2. $\mathcal{V}$ outputs 1 if the π_{csp} verifier accepts and if
    $$\mathsf{uPC.Verify}(\mathsf{uPC.ck}, C_f, \pi, y, v) = 1$$

Figure 1: The bivariate polynomial commitment scheme bPCLB.

Succinct Evaluation Proof. As in [BMM+21], the evaluation proof of a bivariate polynomial $P(X, Y)$ at the point (x, y) proceeds in two steps: In the first step, the prover commits to the univariate polynomial $p(Y) = P(x, Y)$. Subsequently, the prover uses the univariate PCS to prove that $p(y) = v$, where v is the claimed evaluation. Next, the prover to shows that the commitment C_p to the polynomial $p(Y)$ is consistent with the commitment C_P to $P(X, Y)$ by proving knowledge of the commitments $(C_0, \ldots, C_{n-1}) \in \mathbb{G}_1^n$ to the univariate components of P satisfying:

$$\sum_{i=0}^{n-1} e(C_i, v_i) = C_P \quad \wedge \quad \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(x) \cdot C_i = C_p \tag{3}$$

The above relation can be proved using compressed sigma protocols. We now describe the concrete CSP for Equation 3, which we denote by π_{csp}. In the description below, "boxes" highlight those computations which are delegated by the verifier to obtain succinct verification. Their correctness is succinctly verified in the final round.

Interactive Protocol π_{csp}. Let $P_0 = C_P$, $v_0 = C_p$, $\mathbf{w}_0 = (C_0, \ldots, C_{n-1})$, $\mathsf{ck}_0 = \mathbf{v} = ([1]_2, [\beta]_2, \ldots, [\beta^{n-1}]_2)$, $\mathbf{a}_0 = (\mu_0^{\mathbb{H}}(x), \ldots, \mu_{n-1}^{\mathbb{H}}(x))$. The prover and verifier now interact in ℓ rounds where $\ell = \log(n)$. In round $i \in [\ell]$, the prover ($\mathcal{P}$) and verifier ($\mathcal{V}$) proceed as follows:

- $\mathcal{P}$ computes: Prover splits the vectors $\mathbf{w}_i, \mathsf{ck}_i$ and $\mathbf{a}_i$ of size $n/2^i$ into two vectors of size $n/2^{i+1}$ each by splitting them in the middle.

$$(\mathbf{w}_i^{(L)}, \mathbf{w}_i^{(R)}) \leftarrow \mathrm{split}(\mathbf{w}_i),$$

$$\boxed{(\mathsf{ck}_i^{(L)}, \mathsf{ck}_i^{(R)}) \leftarrow \mathrm{split}(\mathsf{ck}_i),\ (\mathbf{a}_i^{(L)}, \mathbf{a}_i^{(R)}) \leftarrow \mathrm{split}(\mathbf{a}_i)}$$

- $\mathcal{P}$ computes: $A_i = \langle \mathsf{ck}_i^{(L)}, \mathbf{w}_i^{(R)} \rangle_{\otimes}$, $A_i' = \langle \mathsf{ck}_i^{(R)}, \mathbf{w}_i^{(L)} \rangle_{\otimes}$, $u_i = \langle \mathbf{a}_i^{(L)}, \mathbf{w}_i^{(R)} \rangle$, $u_i' = \langle \mathbf{a}_i^{(R)}, \mathbf{w}_i^{(L)} \rangle$. It sends A_i, A_i', u_i and u_i' to $\mathcal{V}$.
- The verifier sends a challenge $c_i \leftarrow \mathbb{F}$.
- The prover computes folded vectors $\mathbf{w}_{i+1}$, ck_{i+1} and $\mathbf{a}_{i+1}$, as follows:

$$\mathbf{w}_{i+1} = \mathbf{w}_i^{(L)} + c_i^{-1} \cdot \mathbf{w}_i^{(R)},$$

$$\boxed{\mathsf{ck}_{i+1} = \mathsf{ck}_i^{(L)} + c_i \cdot \mathsf{ck}_i^{(R)},\ \mathbf{a}_{i+1} = \mathbf{a}_i^{(L)} + c_i \cdot \mathbf{a}_i^{(R)}}$$

- $\mathcal{P}$ and $\mathcal{V}$ compute: $P_{i+1} = c_i \cdot A_i' + P_i + c_i^{-1} \cdot A_i$, $v_{i+1} = c_i u_i' + v_i + c_i^{-1} u_i$.

Final Check. After ℓ iterations as above, the prover sends size 1 vectors $\mathsf{ck}_\ell, \mathbf{a}_\ell$ and $\mathbf{w}_\ell$ to the verifier. The verifier checks $e(\mathbf{w}_\ell, \mathsf{ck}_\ell) = P_\ell$ and $\mathbf{a}_\ell \cdot \mathbf{w}_\ell = v_\ell$. Note that we treat size 1 vectors as scalars here.

Verifier Checks Folding of Inner Product Commitment Key. As noted in prior works [BMM$^+$21, Lee21], the final folded commitment key ck_ℓ is a KZG commitment the polynomial $g(X) = \prod_{i=0}^{\ell-1}(1 + c_i X^{2^{\ell-1-i}})$ in the group $\mathbb{G}_2$ with $\mathbf{v}$ as the commitment key. To check this, the verifier requests the prover for an evaluation proof at a point $r \leftarrow \mathbb{F}$. Subsequently, the verifier checks the evaluation proof using ck_ℓ as the commitment to g, and $g(r)$ as the evaluation which it computes itself in $O(\log n)$ time.

Verifier Checks Folding of Linear Form. The commitment key for the linear form is $\mathbf{a}_0 = (\mu_0^{\mathbb{H}}(x), \ldots, \mu_{n-1}^{\mathbb{H}}(x))$, where the polynomials $\{\mu_i^{\mathbb{H}}\}_{i=0}^{n-1}$ are the Lagrange basis polynomials for the subgroup $\mathbb{H} = \langle \omega_n \rangle$ generated by primitive n^{th} root of unity. To verify folding of such structured keys succinctly, we introduce a new technique which we call *Lagrangian Folding*. Looking ahead, the prover shows that $p(x) = \mathbf{a}_\ell$ for a polynomial $p \in \mathbb{F}[X]$, where the verifier "knows" coefficients of p in Lagrange basis. Applying the Lagrangian folding technique yields an evaluation proof for bPCLB of size $O(\log n)$, with verification dominated by $O(\log n)$ group operations (see Theorem 3.1 for a formal statement and proof).

We now describe the Lagrangian folding technique, where we first consider folding vectors of polynomials.

Definition 3.1. *Let $n = 2^\ell$ for $\ell \geq 1$ and let $\mathbf{A} = (a_0(X), \ldots, a_{n-1}(X))$ be a vector of polynomials in $\mathbb{F}[X]$. For a scalar $c \in \mathbb{F}$, we define $\mathsf{Fold}(\mathbf{A}, c)$ as a vector $\mathbf{A}'$ of polynomials of length $n/2$, where $\mathbf{A}'[\,i\,] = \mathbf{A}[\,i\,] + c \cdot \mathbf{A}[\,n/2 + i\,]$ for $i \in [n/2]$. For $\mathbf{A} = (a(X))$ of size 1, we define $\mathsf{Fold}(\mathbf{A}) = a(X)$.*

Our next definition captures successive folding of keys to obtain the final commitment key.

Definition 3.2. *For a vector of polynomials $\mathbf{A} = (a_0(X), \ldots, a_{n-1}(X)) \in \mathbb{F}[X]^n$ with $n = 2^\ell$, $\ell > 1$, and scalar vector $\mathbf{c} = (c_0, \ldots, c_{\ell-1}) \in \mathbb{F}^\ell$, we define $\mathsf{FullFold}(\mathbf{A}, \mathbf{c})$ to be the polynomial $\mathsf{FullFold}(\mathsf{Fold}(\mathbf{A}, c_0), (c_1, \ldots, c_{\ell-1}))$. For $\ell = 1$, we define $\mathsf{FullFold}(\mathbf{A}, \mathbf{c}) = \mathsf{Fold}(\mathbf{A}, c_0)$ where $\mathbf{c} = (c_0)$.*

It is readily observed that if $(a_0(x), \ldots, a_{n-1}(x))$ is the initial commitment key for some polynomials $a_0(X), \ldots, a_{n-1}(X)$, the final commitment key after folding challenges $c_0, \ldots, c_{\ell-1}$ for $\ell = \log n$ is given by $p(x)$, where $p(X) = \mathsf{FullFold}(\mathbf{A}, (c_0, \ldots, c_{\ell-1}))$ with $\mathbf{A} = (a_0(X), \ldots, a_{n-1}(X))$. The logarithmic verification of prior works is based on the fact that for $\mathbf{A} = (1, X, \ldots, X^{n-1})$, the polynomial $\mathsf{FullFold}(\mathbf{A}, (c_0, \ldots, c_{\ell-1})) = \prod_{i=0}^{\ell-1}(1 + c_i \cdot X^{2^{\ell-1-i}})$ can be evaluated at any point $x \in \mathbb{F}$ in $O(\log n)$ time. Our key idea is to fold the vector $\mathbf{A} = (\mu_0^{\mathbb{H}}(X), \ldots, \mu_{n-1}^{\mathbb{H}}(X))$ in Lagrange basis, where the polynomial $\mu_i^{\mathbb{H}}(X)$ is given by the i^{th} standard unit vector $\mathbf{e}_i \in \mathbb{F}^n$. It is readily seen that folding this vector of polynomials successively using challenges $c_0, \ldots, c_{\ell-1}$ results in the polynomial with Lagrange coefficients as $\mathbf{u} = (1, c_0) \otimes (1, c_1) \otimes \cdots \otimes (1, c_{\ell-1})$. In the next subsection, we show a protocol to prove evaluation of the polynomial whose Lagrange coefficients exhibit a tensor structure such as above. In particular, the protocol allows the verifier to determine the correctness of final folded linear form ck_ℓ.

3.2 Proving Evaluation of Folded Polynomial

Given integers $\ell > 0$ and $n = 2^\ell$, we present a protocol which allows a prover to show that for $\mathbf{c} = (c_0, \ldots, c_{\ell-1}) \in \mathbb{F}^\ell$, vector $\mathbf{u} \in \mathbb{F}^n$ given by $\mathbf{u} = (1, c_0) \otimes (1, c_1) \otimes \cdots \otimes (1, c_{\ell-1})$, $x \in \mathbb{F}$ and $v \in \mathbb{F}$, the unique polynomial $p \in \mathbb{F}[\,X\,]$ of degree at most $n - 1$ such that $p(\omega^i) = \mathbf{u}[\,i\,]$ for $i \in [n]$ satisfies $p(x) = v$ where ω is a primitive n^{th} root of unity in $\mathbb{F}$. In other words, the prover claims that the polynomial with $\mathbf{u}$ as its Lagrange coefficients over the set $\mathbb{H} = \langle \omega \rangle$ evaluates to v at x. Our divide and conquer approach is reminiscent of the Gemini protocol to prove evaluation of a multilinear polynomial committed as a univariate polynomial with the same coefficient vector.

To start the protocol, the prover computes the polynomial $p(X)$ with vector $\mathbf{u}$ as its evaluations over the roots of unity (we shall see that this can be done in $O(n)$ $\mathbb{F}$-operations). The prover commits to $p(X)$ using KZG commitment C. The prover then proves two claims: (i) $p(x) = v$ for the polynomial p committed by C and (ii) $\mathbf{u}$ is the vector of evaluations of p over the domain $\mathbb{H}$. The first claim is easily proved using the KZG proof of evaluation. To prove the second claim, the prover commits to the polynomials $p_i(X)$ for $i = 1, \ldots, \ell - 1$ where $p_i(X)$ is of degree at most $2^i - 1$ and $p_i(X)$ interpolates the subvector $\mathbf{u}_i$ of $\mathbf{u}$ given by $\mathbf{u}_i = (1, c_0) \otimes \cdots \otimes (1, c_{i-1})$. Next, the prover shows the following for $i = 1, \ldots, \ell$

$$p_i(X) = \frac{X^{n_i} + 1}{2} \cdot p_{i-1}(X) - \frac{X^{n_i} - 1}{2} \cdot c_{i-1} \cdot p_{i-1}(\omega_i^{-1} X) \tag{4}$$

where we set $p_0(X) = 1$, $p_\ell(X) = p(X)$, $n_i = 2^{i-1}$, $\omega_i := \omega^{n/2^i}$ is a primitive 2^i-th root of unity. The above decomposition mirrors the fact that $\mathbf{u}_i$ consists of vector $\mathbf{u}_{i-1}$ at even positions and the scaled vector $c_{i-1}\mathbf{u}_{i-1}$ at odd positions, and thus the interpolating polynomial can be written in terms of interpolating polynomial for $\mathbf{u}_{i-1}$, with co-prime polynomials $X^{n_i} + 1$ and $X^{n_i} - 1$ selecting even and odd roots respectively. We state the following lemma.

Lemma 3.1. *If polynomials $p_0, \ldots, p_\ell$ satisfy Equation (4), for all $i = 1, \ldots, \ell$ with $p_0 = 1$, then $p(X) := p_\ell(X)$ is the unique polynomial in $\mathbb{F}^{<n}[\,X\,]$ such that $p(\omega^j) = \mathbf{u}[\,j\,]$ for all $j \in [n]$.*

Proof. We use induction on i to show that $p_i(X)$ interpolates the vector $\mathbf{u}_i$ over the subgroup $\mathbb{H}_i = \langle \omega_i \rangle$ where $\omega_i = \omega^{2^{(\ell-i)}}$. Consider the base case $i = 1$. Then, by Equation (4) we have:

$$\begin{aligned} p_1(X) &= \frac{X^{n_1}+1}{2} \cdot p_0(X) - \frac{X^{n_1}-1}{2} \cdot c_0 \cdot p_0(\omega_1^{-1}X) \\ &= \frac{X+1}{2} - c_0 \cdot \frac{X-1}{2} \end{aligned}$$

Clearly, it can be seen that p_1 evaluates to $\mathbf{u}_1 = (1, c_0)$ over $\mathbb{H}_1 = \{1, -1\}$. Now suppose the claim is true for $i-1$ for some $i \leq \ell$, i.e, $p_{i-1}(\omega_{i-1}^j) = \mathbf{u}_{i-1}[\,j\,]$ where ω_{i-1} is a primitive 2^{i-1}-th root of unity and $\mathbf{u}_{i-1}$ is $(1, c_0) \otimes \cdots (1, c_{i-2})$ as defined earlier. Let ω_i be the primitive 2^i-th root of unity satisfying $\omega_{i-1} = \omega_i^2$. From Equation (4), we have for all $j \in [2^i]$,

$$\begin{aligned} p_i(\omega_i^j) &= \frac{\omega_i^{jn_i}+1}{2} \cdot p_{i-1}(\omega_i^j) - \frac{\omega_i^{jn_i}-1}{2} \cdot c_{i-1} \cdot p_{i-1}(\omega_i^{-1}\omega_i^j) \\ &= \frac{\omega_i^{2^i \cdot j/2}+1}{2} \cdot p_{i-1}(\omega_i^j) - c_{i-1}\frac{\omega_i^{2^i \cdot j/2}-1}{2} \cdot p_{i-1}(\omega_i^{(j-1)}) \end{aligned}$$

From the above, when j is even, the second term is zero and we have $p_i(\omega_i^j) = p_{i-1}(\omega_i^j) = p_{i-1}(\omega_{i-1}^{j/2})$ $= \mathbf{u}_{i-1}[\,j/2\,]$. Similarly when j is odd, we have $p_i(\omega_i^j) = c_{i-1}p_{i-1}(\omega_{i-1}^{(j-1)/2}) = c_{i-1}\mathbf{u}_{i-1}[\,(j-1)/2\,]$. In other words, the vector $\mathbf{u}_i$ of evaluations $\left(p_i(\omega_i^j)\right)_j$ is given by $\mathbf{u}_i = \mathbf{u}_{i-1} \otimes (1, c_{i-1})$. This completes the induction step and proves the claim for all $i \leq \ell$. □

The prover successively computes polynomials $p_1, \ldots, p_\ell$ based on Equation (4) and computes corresponding commitments $C_1, \ldots, C_\ell$. It can be seen that at step i, the prover work is $O(2^i)$ group and field operations, which yields a total of $O(n)$ group and field operations. Next, the verifier checks the identities for $i = 1, \ldots, \ell$ in Equation (4) at a random point $\beta \leftarrow \mathbb{F}$. To this effect, the prover sends claimed evaluations $v_i = p_i(\beta)$ and $v_i' = p_i(\omega_i^{-1}\beta)$ for $i = 1, \ldots, \ell$ and corresponding KZG evaluation proofs. The evaluations at β can be checked via a single KZG proof via standard batching techniques. Thus the prover 2ℓ $\mathbb{F}$-elements and $\ell + 1$ $\mathbb{G}_1$-elements to prove the identities. The total communication cost is therefore $2\ell\ \mathbb{F} + (2\ell + 2)\ \mathbb{G}_1$. The verification incurs $O(\ell)\ \mathbb{G}_1$ operations and one pairing check via standard batching of pairing checks for KZG verification.

3.3 Parameters for bPCLB

In Theorem 3.1, we summarize the parameters attained by the non-interactive scheme $\mathsf{bPCLB} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Prove}, \mathsf{Verify})$ (as per Definition 1) obtained by applying Fiat-Shamir heuristic to $\mathsf{bPCLB.Eval}$ in Figure 1.

Theorem 3.1. *Assuming that the q-DLOG assumption holds for the bilinear group generator* BG, *the scheme* $\mathsf{bPCLB} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Prove}, \mathsf{Verify})$ *obtained by applying Fiat-Shamir heuristic to the interactive procedure* $\mathsf{bPCLB.Eval}$ *in Figure 1 is a polynomial commitment scheme for polynomials in $\mathbb{F}[X, Y]$ in the algebraic group model (AGM) and achieves following efficiency parameters:*

$$\begin{aligned} |\pi^{\mathsf{bPCLB}}| &= 2\log n\,\mathbb{G}_T + 2\log n\,\mathbb{G}_1 + 2\log n\,\mathbb{F} \\ \mathrm{t}_{\mathrm{C}}^{\mathsf{bPCLB}} &= mn\,\mathbb{G}_1 + n\,\mathbb{P} + mn\,\mathbb{F} \\ \mathrm{t}_{\mathrm{P}}^{\mathsf{bPCLB}} &= O(m+n)\,\mathbb{G}_1 + O(n)\,\mathbb{P} + O(n)\,\mathbb{F} \\ \mathrm{t}_{\mathrm{V}}^{\mathsf{bPCLB}} &= 2\log n\,\mathbb{G}_T + 4\log n\,\mathbb{G}_1 + O(1)\,\mathbb{P} + O(\log n)\,\mathbb{F} \end{aligned}$$

In the above, n and m denote the degree bounds on variables X and Y respectively, while $|\pi|$, $\mathrm{t_C}$, $\mathrm{t_P}$, $\mathrm{t_V}$ denote the proof size, commit time, prover time and verifier time respectively. We exclude the time to evaluate F at the evaluation point in the prover time $\mathrm{t_P}$.

Proof. Assume that for all nodes p at height d labeled with the statement C_p ,the extractor outputs vector $(w_0^p(Y), \ldots, w_{h-1}^p(Y))$ such that $\sum_{i=0}^{h-1} e([w_i^p(\tau)]_1, \mathsf{ck}^p[\,i\,]) = C_p$, where we have $h = 2^d$ and ck_p is the folded key ck corresponding to challenges determined by node p. It can be seen that for a node q at height $d+1$, the vector $(w_0^q(Y), \ldots, w_{h'-1}^q(Y))$, with $h' = 2^{d+1}$ such that $\sum_{i=0}^{h'-1} e([w_i^q(\tau)]_1, \mathsf{ck}^q[\,i\,]) = C_q$ can be obtained by considering child nodes of q for three distinct challenges $c_{\ell-1-d}$. This follows from the usual extraction in CSPs, noticing that underlying representation can also be linearly combined. All that we need to do is to argue the case $d = 0$, for which we invoke the AGM assumption as follows. Consider a node in the tree at depth ℓ (and thus height 0), after all CSP challenges have been specified. Let the node correspond to the statement $(C_\ell, \mathbf{w}_\ell, \mathsf{ck}_\ell)$ where C_ℓ is the folded commitment computed by the verifier, $\mathbf{w}_\ell$ and ck_ℓ are the folded witness and commitment key output by $\mathcal{A}$. Since $\mathcal{A}$ is algebraic, it outputs polynomials $w(Y), \bar{g}(X)$ of degree at most d_y and d_x respectively such that $\mathbf{w}_\ell = [w(\tau)]_1$, $\mathsf{ck}_\ell = [\bar{g}(\beta)]_2$. Let z be the subsequent evaluation challenge at depth $\ell+1$, with π as the KZG evaluation proof output by $\mathcal{A}$. Again, $\mathcal{A}$ outputs $q(X)$ such that $\pi = [q(\beta)]_2$. Let $g(X)$ be the polynomial $\prod_{i=0}^{\ell-1}(1 + c_i X^{2^{\ell-1-i}})$. For an accepting transcript, we must have: $e(\mathbf{w}_\ell, \mathsf{ck}_\ell) = C_\ell$, $e([\beta - z]_1, \pi) = e([1]_1, \mathsf{ck}_\ell - [g(z)]_2)$. By q-DLOG assumption, the second pairing equality implies $q(X)(X-z) = \bar{g}(X) - g(z)$, otherwise β can be obtained by factoring the non-zero difference polynomial. Then, with overwhelming probability, $\bar{g}(X) = g(X)$, as z was sampled independent of $\bar{g}$ and g. Now, from first pairing equality we have $e([w(\tau)]_1, [g(\beta)]_2) = C_\ell$. The extractor outputs $w(Y)$, thus proving the base case of induction. □

4 Generic Aggregation of Polynomial Protocols

In this section, we present our GAPP framework for generic aggregation of polynomial protocols. We begin with a general treatment, followed by a concretely efficient instantiation using our bivariate PCS bPCLB from Section 3.

Let $G(X_0, \ldots, X_{\ell-1}) \in \mathbb{F}[X_0, \ldots, X_{\ell-1}]$ be an ℓ-variate polynomial. As outlined in the overview, we consider a scenario where a prover wishes to prove a set of n polynomial identities of the form

$$G(p_{i,0}(Y), \ldots, p_{i,\ell-1}(Y)) = 0 \bmod Z_{\mathbb{V}}(Y) \quad \forall i \in [n], \tag{5}$$

given a set of commitments $(C_{i,0}, \ldots, C_{i,\ell-1})$, where for each $(i,j) \in [n] \times [\ell]$, $C_{i,j}$ is a commitment to the polynomial $p_{i,j}(Y) \in \mathbb{F}[Y]$ under a polynomial commitment scheme PC.

4.1 Aggregation using Bivariate Polynomials

Packed Polynomials. Let bPC be a generic bivariate polynomial commitment scheme. Let $\mathbb{H} = \{1, \omega, \ldots, \omega^{n-1}\}$ be the subgroup consisting of the n^{th} roots of unity in $\mathbb{F}$, and let $\mu_i^{\mathbb{H}}(X)$ be the corresponding Lagrange polynomial for each $i \in [n]$. We say that the *packed* polynomial corresponding to the vector of univariate polynomials $(p_{i,j}(Y))_{i \in [n]}$ is the bivariate polynomial

$$P_j(X, Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) p_{i,j}(Y). \tag{6}$$

It turns out that, for the above packing scheme, the satisfiability of a set of n univariate polynomial identities over $\mathbb{V}$ reduces to a single bivariate polynomial identity over the packed polynomials over the domain $\mathbb{H} \times \mathbb{V}$, given by

$$Q(X,Y) = G(P_0(X,Y), \dots, P_{\ell-1}(X,Y)) \text{ vanishes over } \mathbb{H} \times \mathbb{V}.$$

We capture this formally using the following lemma.

Lemma 4.1 (Packing Lemma)**.** *Let $m, n, \ell \in \mathbb{N}$ be positive integers. Let $\mathbb{V} = \langle \nu \rangle$ and $\mathbb{H} = \langle \omega \rangle$ be the subgroups generated by primitive m^{th} and n^{th} roots of unity in $\mathbb{F}$ respectively. Let $G(X_0, \dots, X_{\ell-1}) \in \mathbb{F}[X_0, \dots, X_{\ell-1}]$ be an ℓ-variate polynomial, and let $p_{i,j}(Y)$ be a univariate polynomial in $\mathbb{F}[Y]$ for each $(i,j) \in [n] \times [\ell]$. Let $P_j(X,Y)$ denote the packed (bivariate) polynomial corresponding to the vector of univariate polynomials $(p_{0,j}(Y), \dots, p_{n-1,j}(Y))$ as in Equation 6. Then the univariate polynomial $G(p_{i,0}(Y), \dots, p_{i,\ell-1}(Y))$ vanishes over $\mathbb{V}$ for all $i \in [n]$ if and only if the bivariate polynomial $Q(X,Y) = G(P_0(X,Y), \dots, P_{\ell-1}(X,Y))$ vanishes over the set $\mathbb{H} \times \mathbb{V}$.*

Proof. We first state some identities. Recall that $\mathbb{H} = \{1, \omega, \dots, \omega^{n-1}\}$ is the subgroup consisting of the n^{th} roots of unity in $\mathbb{F}$. It follows from Equation 6 that for each $i \in [N]$ and each $j \in [\ell]$, we have

$$P_j(\omega^i, Y) = \sum_{i'=0}^{n-1} \mu_{i'}\left(\omega^i\right) p_{i',j}(Y) = p_{i,j}(Y)$$

which follows from the facts that: (i) $\mu_i\left(\omega^i\right) = 1$ for each $i \in [N]$, and (ii) $\mu_{i'}\left(\omega^i\right) = 0$ for each $i, i' \in [N]$ such that $i \neq i'$. Hence, *for all* $i \in [n]$:

$$Q(\omega^i, Y) = G(p_{i,0}(Y), \dots, p_{i,\ell-1}(Y)) \tag{7}$$

We now prove the "if" part of the statement of Lemma 4.1. Suppose that

$$Q(X,Y) = G(P_0(X,Y), \dots, P_{\ell-1}(X,Y))$$

vanishes over $\mathbb{H} \times \mathbb{V}$. This implies that the following must be true for each $i \in [n]$: $Q(\omega^i, Y) = 0 \bmod Z_{\mathbb{V}}(Y)$. By Equation 7, we have that *for all* $i \in [n]$, the following univariate polynomial holds

$$Q(\omega^i, Y) = 0 \bmod Z_{\mathbb{V}}(Y) \implies G(p_{i,0}(Y), \dots, p_{i,\ell-1}(Y)) = 0 \bmod Z_{\mathbb{V}}(Y)$$

as desired. We now prove the "only if" part of the statement of Lemma 4.2. Suppose that *for all* $i \in [n]$: $G(p_{i,0}(Y), \dots, p_{i,\ell-1}(Y)) = 0 \bmod Z_{\mathbb{V}}(Y)$. By equation 7, *for all* $i \in [n]$: $Q(\omega^i, Y) = 0 \bmod Z_{\mathbb{V}}(Y)$. But this precisely implies that $Q(X,Y)$ vanishes over $\mathbb{K}$, as desired. This completes the proof of Lemma 4.1. □

The GAPP Relation. In the GAPP relation defined in the overview, we considered ℓ polynomial commitments for an ℓ-variate form G, intuitively, binding each polynomial to a distinct variable of G. Often we need to associate the *same* commitment with more than one variable in G, specifically when a polynomial appears in the identity with different parameterizations. Looking ahead, in the application of GAPP to aggregate PLONK proofs, the form of polynomial identity is given by Equation (9), in which the polynomial z appears with parameterizations as $z(Y)$ and $z(\nu Y)$ for $\nu \in \mathbb{F}$.

Formally, we consider commitments $(C_0, \dots, C_{r-1})$ to $r \leq \ell$ polynomials $(P_0, \dots, P_{r-1})$, each of which is potentially bound to several variables in G with different parameterizations. In general, we consider s parameterization polynomials $h_i(Y)$ for $i \in [s]$, and the maps $\kappa : [\ell] \to [r]$ and $\theta : [\ell] \to [s]$. For each $i \in [\ell]$, we define $K_i(X,Y) = P_{\kappa(i)}(X, h_{\theta(i)}(Y))$ for $i \in [\ell]$, where K_i specifies the i^{th} input to G.

Let bPC be a bivariate polynomial commitment scheme as before with commitment space $\mathcal{C}$. Given $\mathsf{pp} \leftarrow \mathsf{bPC.Setup}(1^\lambda, (d_x, d_y))$, we define the relation $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ for degree bounds $(n, m) \leq (d_x, d_y)$ as follows:

Definition 4.1 (GAPP Relation)**.** *Let* $\mathbf{C} = (C_0, \ldots, C_{r-1}) \in \mathcal{C}^r$ *be a vector of commitments,* $\mathbf{w}_0 = (P_0(X,Y), \ldots, P_{r-1}(X,Y)) \in (\mathbb{F}[X,Y])^r$ *be a vector of bivariate polynomials, and let* $\mathbf{w}_1 = (\tilde{\mathbf{c}}_0, \ldots, \tilde{\mathbf{c}}_{r-1})$ *be a vector of opening hints. Additionally, let* $\mathbf{h} = (h_0(Y), \ldots, h_{s-1}(Y))$ *be a vector of parameterization polynomials, maps* $\kappa : [\ell] \to [r]$ *and* $\theta : [\ell] \to [s]$ *assigning witness polynomials with parameterization for variables in* G*. We say that* $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ *for* $\mathbf{x} = (\kappa, \theta, \mathbf{h}, \mathbf{C})$ *and* $\mathbf{w} = (\mathbf{w}_0, \mathbf{w}_1)$ *if:*

1. *For each* $j \in [r]$*,* $\mathsf{bPC.Open}(\mathsf{pp}, P_j(X,Y), (d_x, d_y), C_j, \tilde{\mathbf{c}}_j) = 1$.
2. *For each* $i \in [n]$*,* $G\big(K_0(\omega^i, Y), \ldots, K_{\ell-1}(\omega^i, Y)\big) = 0 \bmod Z_{\mathbb{V}}(Y)$*, where* ω *is the canonical primitive* n^{th} *root of unity in* $\mathbb{F}$ *and where for each* $j \in [\ell]$*, we have* $K_j(X,Y) = P_{\kappa(j)}(X, h_{\theta(j)}(Y))$.

4.2 Argument of Knowledge for the GAPP Relation

In this subsection, we present a argument of knowledge for $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ given *any* univariate and bivariate polynomial commitment schemes. This argument of knowledge relies on certain algebraic observations which we state next. Recall that the formal definition of $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ involves proving that the bivariate polynomial $Q(X,Y) = G(P_0(X,Y), \ldots, P_{\ell-1}(X,Y))$ vanishes over $\mathbb{H} \times \mathbb{V}$. We use the following algebraic criterion to show that a bivariate polynomial Q vanishes over the domain $\mathbb{H} \times \mathbb{V}$.

Lemma 4.2. *Let* $m, n, \ell \in \mathbb{N}$ *be positive integers and let* $\mathbb{V}$ *and* $\mathbb{H}$ *as before be the subgroups consisting of the* m^{th} *and* n^{th} *roots of unity in* $\mathbb{F}$*, respectively. A polynomial* $Q \in \mathbb{F}[X,Y]$ *vanishes over* $\mathbb{H} \times \mathbb{V}$ *if and only there exists polynomial* $H \in \mathbb{F}[X,Y]$ *with* $\deg_X(H) < n$ *such that* $Q(X,Y) - Z_{\mathbb{V}}(Y)H(X,Y) = 0 \bmod Z_{\mathbb{H}}(X)$*. Moreover, the polynomial* H *is explicitly described as:*

$$H(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) \frac{Q(\omega^i, Y)}{Z_{\mathbb{V}}(Y)} \tag{8}$$

Proof. First, assume that Q vanishes on $\mathbb{H} \times \mathbb{V}$. Then for $i \in [n]$, $Q(\omega^i, Y)$ vanishes on $\mathbb{V}$ and hence is divisible by $Z_{\mathbb{V}}(Y)$. Thus all the univariate components of H are indeed polynomials. Substituting $X = \omega^k$ in expressions for polynomials Q and H we see that $Q(\omega^k, Y) = Z_{\mathbb{V}}(Y)H(\omega^k, Y)$. Thus, by factor theorem, $(X - \omega^k)$ divides $Q(X,Y) - Z_{\mathbb{V}}(Y)H(X,Y)$ for all $k \in [n]$. Since these factors are relatively prime we have $Z_{\mathbb{H}}(X) = \prod_{k=0}^{n-1}(X - \omega^k)$ divides $Q(X,Y) - Z_{\mathbb{V}}(Y)H(X,Y)$ which proves the claim. The other direction is trivial, as existence of H satisfying $Q(X,Y) - Z_{\mathbb{V}}(Y)H(X,Y) = 0 \bmod Z_{\mathbb{H}}(X)$ implies Q vanishes over $\mathbb{H} \times \mathbb{V}$. □

Argument of Knowledge for $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$. We describe in Figure 2 an argument of knowledge for the relation $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ (see Appendix **??** for additional textual description). We present this as an interactive *public-coin* protocol (i.e., all of the verifier's messages are uniformly random strings). It can be made non-interactive using the standard FS transform. We present a detailed description of the interactive protocol in Figure 2 below.

- **Setup**: Setup generates the following public parameters:

$$\mathsf{pp}_{\mathsf{uPC}} \leftarrow \mathsf{uPC.Setup}(1^\lambda, d_x), \quad \mathsf{pp}_{\mathsf{bPC}} \leftarrow \mathsf{bPC.Setup}(1^\lambda, (d_x, d_y)).$$

- **Common Input**: $\mathbf{C} = (C_0, \ldots, C_{r-1}) \in \mathcal{C}^r$, $\mathbf{h} = (h_0(Y), \ldots, h_{s-1}(Y))$, maps $\kappa : [\ell] \to [r]$ and $\theta : [\ell] \to [s]$
- **Prover's Input**: $(\mathbf{w}_0, \mathbf{w}_1)$ where $\mathbf{w}_0 = (P_0(X,Y), \ldots, P_{r-1}(X,Y)) \in (\mathbb{F}[X,Y])^r$, and $\mathbf{w}_1 = (\tilde{\mathbf{c}}_0, \ldots, \tilde{\mathbf{c}}_{r-1})$, such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ for $\mathbf{x} = (\kappa, \theta, \mathbf{h}, \mathbf{C})$ and $\mathbf{w} = (\mathbf{w}_0, \mathbf{w}_1)$.
- **Additional Notations**: $K_j(X,Y) = P_{\kappa(j)}(X, h_{\theta(j)}(Y))$ for $j \in [\ell]$ and

$$Q(X,Y) = G(K_0(X,Y), \ldots, K_{\ell-1}(X,Y))$$

- **Round 1**: $\mathcal{P}$ commits to the polynomial $H(X,Y)$ computed as in Lemma 4.2.
  1. $\mathcal{P}$ sends commitment $C_H \leftarrow \mathsf{bPC.Com}(\mathsf{pp}_{\mathsf{bPC}}, H)$.
  2. The verifier $\mathcal{V}$ sends $y \leftarrow \mathbb{F}$.
- **Round 2**: $\mathcal{P}$ commits to an auxiliary univariate polynomial $u(X)$.
  1. $\mathcal{P}$ computes $u(X) = (Q(X,y) - Z_{\mathbb{V}}(y)H(X,y))/Z_{\mathbb{H}}(X)$.
  2. $\mathcal{P}$ sends the univariate commitment $C_u \leftarrow \mathsf{uPC.Com}(\mathsf{pp}_{\mathsf{uPC}}, u)$.
  3. $\mathcal{V}$ sends $x \leftarrow \mathbb{F}$.
- **Round 3:** $\mathcal{V}$ checks: $G(K_0(X,y), \ldots, K_{\ell-1}(X,y)) = Z_{\mathbb{V}}(y)h_y(X) + u(X)Z_{\mathbb{H}}(X)$.
  1. $\mathcal{P}$ computes: $(\tilde{h}, \{\tilde{k}_j\}_{\mathrm{j}\in[\ell]}, \tilde{u})$ where $\tilde{h} = H(x,y)$, $\tilde{u} = u(x)$, and $\tilde{k}_j = K_j(x,y)$ for each $j \in [\ell]$.
  2. $\mathcal{P}$ computes
     - $\pi_h \leftarrow \mathsf{bPC.Prove}(\mathsf{pp}_{\mathsf{bPC}}, H(X,Y), (x,y))$.
     - $\pi_u \leftarrow \mathsf{uPC.Prove}(\mathsf{pp}_{\mathsf{uPC}}, u(X), x)$.
     - $\pi_j \leftarrow \mathsf{bPC.Prove}(\mathsf{pp}_{\mathsf{bPC}}, P_{\kappa(j)}(X,Y), (x, h_{\theta(j)}(y)))$ for each $j \in [\ell]$.
  3. $\mathcal{P}$ sends $((\tilde{h}, \pi_h), \{\tilde{k}_j, \pi_j\}_{j\in[\ell]}, (\tilde{u}, \pi_u))$.
  4. $\mathcal{V}$ performs the following verification checks:
     - $b_h = \mathsf{bPC.Verify}(\mathsf{pp}_{\mathsf{bPC}}, C_H, (n,m), (x,y), \tilde{h}, \pi_h)$.
     - $b_u = \mathsf{uPC.Verify}(\mathsf{pp}_{\mathsf{uPC}}, C_u, n, x, \tilde{u}, \pi_u)$.
     - $b_j \leftarrow \mathsf{bPC.Verify}(\mathsf{pp}_{\mathsf{bPC}}, C_{\kappa(j)}, (n,m), (x, h_{\theta(j)}(y)), \tilde{k}_j, \pi_j)$ for each $j \in [\ell]$.
  5. $\mathcal{V}$ also checks that $G(\tilde{k}_j, \ldots, \tilde{k}_{\ell-1}) = Z_{\mathbb{V}}(y)\tilde{h} + \tilde{u}Z_{\mathbb{H}}(x)$.
  6. If all of the above verification checks pass, $\mathcal{V}$ accepts. Otherwise, it rejects.

Figure 2: Argument of knowledge for the extended relation $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$.

Setup and Inputs. The setup phase of the protocol generates the following public parameters for uPC and bPC: $\mathsf{pp}_{\mathsf{uPC}} \leftarrow \mathsf{uPC.Setup}(1^\lambda, d_x)$ and $\mathsf{pp}_{\mathsf{bPC}} \leftarrow \mathsf{bPC.Setup}(1^\lambda, (d_x, d_y))$. The public input (common to both the prover $\mathcal{P}$ and the verifier $\mathcal{V}$) consists of:

- A vector of bivariate commitments $\mathbf{C} = (C_0, \ldots, C_{r-1}) \in \mathcal{C}^r$.
- A vector of parameterization polynomials $\mathbf{h} = (h_0(Y), \ldots, h_{s-1}(Y))$.
- The maps $\kappa : [\ell] \to [r]$ and $\theta : [\ell] \to [s]$.

The (honest) prover $\mathcal{P}$ additionally inputs its witness $(\mathbf{w}_0, \mathbf{w}_1)$ where

$$\mathbf{w}_0 = (P_0(X,Y), \ldots, P_{r-1}(X,Y)) \in (\mathbb{F}[X,Y])^r, \quad \mathbf{w}_1 = (\tilde{\mathbf{c}}_0, \ldots, \tilde{\mathbf{c}}_{r-1})$$

such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ for $\mathbf{x} = (\kappa, \theta, \mathbf{h}, \mathbf{C})$ and $\mathbf{w} = (\mathbf{w}_0, \mathbf{w}_1)$. We define the following auxiliary polynomials: $K_j(X,Y) = P_{\kappa(j)}(X, h_{\theta(j)}(Y))$ for $j \in [\ell]$ and $Q(X,Y) = G(K_0(X,Y), \ldots, K_{\ell-1}(X,Y))$.

The Interactive Protocol. Given the public parameters $(\mathsf{pp}_{\mathsf{uPC}}, \mathsf{pp}_{\mathsf{bPC}})$ and the inputs as described above, $\mathcal{P}$ and $\mathcal{V}$ engage in an interactive protocol that proceeds as follows:

Round-1: In the first round, $\mathcal{P}$ computes the polynomial $H(X,Y)$ according to Lemma 4.2, and sends a commitment C_H to the polynomial $H(X,Y)$ under the bivariate PCS bPC. To check that $Q(X,Y) = Z_{\mathbb{V}}(Y)H(X,Y) \mod Z_{\mathbb{H}}(X)$, $\mathcal{V}$ sends a random challenge $y \leftarrow \mathbb{F}$, and asks $\mathcal{P}$ to prove that $Z_{\mathbb{H}}(X)$ divides the univariate polynomial $Q(X,y) - Z_{\mathbb{V}}(y)H(X,y)$.

Round-2: $\mathcal{P}$ computes $u(X) = (Q(X,y) - Z_{\mathbb{V}}(y)H(X,y))/Z_{\mathbb{H}}(X)$, and sends a commitment C_u to $u(X)$ under the univariate PCS uPC. Note that $\mathcal{P}$ computes $Q(X,y)$ as $G(K_0(X,y), \ldots, K_{\ell-1}(X,y))$ without explicitly computing the bivariate polynomial Q. At this point, $\mathcal{V}$ wishes to check

$$Q(X,y) = G(K_0(X,y), \ldots, K_{\ell-1}(X,y)) = Z_{\mathbb{V}}(y)H(X,y) + u(X)Z_{\mathbb{H}}(X)$$

To check this, $\mathcal{V}$ sends a second random challenge $x \leftarrow \mathbb{F}$, and asks $\mathcal{P}$ to send a set of polynomial evaluations $(\tilde{h}, \tilde{u}, \{\tilde{k}_j\}_{j\in[\ell]})$ where $\tilde{h} = H(x,y)$, $\tilde{u} = u(x)$, and for each $j \in [\ell]$, $\tilde{k}_j = K_j(x,y) = P_{\kappa(j)}(x, h_{\theta(j)}(y))$.

Round-3: $\mathcal{P}$ sends the above polynomial evaluations to $\mathcal{V}$, along with the corresponding evaluation proofs, computed as:

- $\pi_h \leftarrow \mathsf{bPC.Prove}(\mathsf{pp}_{\mathsf{bPC}}, H(X,Y), (x,y))$
- $\pi_u \leftarrow \mathsf{uPC.Prove}(\mathsf{pp}_{\mathsf{uPC}}, u(X), x)$.
- $\pi_j \leftarrow \mathsf{bPC.Prove}(\mathsf{pp}_{\mathsf{bPC}}, P_{\kappa(j)}(X,Y), (x, h_{\theta(j)}(y)))$ for each $j \in [\ell]$.

Final Verification Checks: $\mathcal{V}$ verifies the evaluation proofs sent by $\mathcal{P}$ as:

- $b_h = \mathsf{bPC.Verify}(\mathsf{pp}_{\mathsf{bPC}}, C_H, (n,m), (x,y), \tilde{h}, \pi_h)$.
- $b_u = \mathsf{uPC.Verify}(\mathsf{pp}_{\mathsf{uPC}}, C_u, n, x, \tilde{u}, \pi_u)$.
- $b_j = \mathsf{bPC.Verify}(\mathsf{pp}_{\mathsf{bPC}}, C_{\kappa(j)}, (n,m), (x, h_{\theta(j)}(y)), \tilde{k}_j, \pi_j)$ for each $j \in [\ell]$.

$\mathcal{V}$ also verifies that the following relation holds with respect to the evaluations sent by $\mathcal{P}$: $G(\tilde{k}_0, \ldots, \tilde{k}_{\ell-1}) = Z_{\mathbb{V}}(y)\tilde{h} + \tilde{u}Z_{\mathbb{H}}(x)$. If all of these verification checks pass, $\mathcal{V}$ accepts. Otherwise, it rejects.

Theorem 4.1. *Assuming that* uPC *and* bPC *are polynomial commitment schemes as defined in Section 2, the above protocol is a succinct argument of knowledge for the relation* $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ *in Definition 4.1.*

Proof. We argue both knowledge-soundness and succinctness for the protocol in Figure 2 below.

Knowledge-Soundness. Consider a PPT cheating prover $\mathcal{A}$ that interacts with an honest verifier $\mathcal{V}$ to produce an accepting transcript of the form

$$(\mathbf{C} = (C_0, \ldots, C_{r-1}), (x,y), C_H, C_u, (\tilde{h}, \pi_h), (\tilde{u}, \pi_u), \{(\tilde{p}_j, \pi_j)\}_{j\in[\ell]})$$

where $x, y \leftarrow \mathbb{F}$. Let $\mathcal{E}_{\mathsf{uPC}}$ and $\mathcal{E}_{\mathsf{bPC}}$ be the PPT extractors for uPC and bPC, respectively, as per Definition 2.6. We construct an extractor $\mathcal{E}_{\mathsf{GAPP}}$ with oracle access to $\mathcal{A}$ as follows:

- $\mathcal{E}_{\mathsf{GAPP}}$ uses its oracle access to $\mathcal{A}$ to extract the following:
  - $(P_j(X,Y), \tilde{\mathbf{c}}_j) \leftarrow \mathcal{E}^{\mathcal{A}}_{\mathsf{bPC}}(C_j, (n,m), (x,y), \tilde{p}_j, \pi_j)$ for each $j \in [r]$.
  - $(H(X,Y), \tilde{\mathbf{c}}_H) \leftarrow \mathcal{E}^{\mathcal{A}}_{\mathsf{bPC}}(C_H, (n,m), (x,y), \tilde{h}, \pi_h)$.
  - $(u(X), \tilde{\mathbf{c}}_u) \leftarrow \mathcal{E}^{\mathcal{A}}_{\mathsf{uPC}}(C_u, n, x, \tilde{u}, \pi_u)$.
- $\mathcal{E}_{\mathsf{GAPP}}$ uses the vector of parameterization polynomials $\mathbf{h} = (h_0(Y), \ldots, h_{s-1}(Y))$, and the maps $\kappa : [\ell] \to [r]$ and $\theta : [\ell] \to [s]$ to compute for each $j \in [\ell]$

$$K_j = P_{\kappa(j)}(X, h_{\theta(j)}(Y)).$$

- $\mathcal{E}_{\mathsf{GAPP}}$ outputs $\perp$ if any of the following hold:
  - Any of the above extractions fail.
  - For some $j \in [\ell]$, $\widetilde{p}_j \neq K_j(x,y) = P_{\kappa(j)}(x, h_{\theta(j)}(y))$, or $\widetilde{h} \neq H(x,y)$, or $\widetilde{u} \neq u(x)$.
  - For some $j \in [r]$, $\mathsf{bPC.Open}(\mathsf{pp}, P_j(X,Y), (d_x, d_y), C_j, \tilde{\mathbf{c}}_j) = 0$.
- Otherwise, $\mathcal{E}$ outputs $(\mathbf{w}_0, \mathbf{w}_1)$ where

$$\mathbf{w}_0 = (P_0(X,Y), \ldots, P_{r-1}(X,Y)), \quad \mathbf{w}_1 = (\tilde{\mathbf{c}}_0, \ldots, \tilde{\mathbf{c}}_{r-1})$$

First of all, assuming the knowledge-soundness of uPC and bPC, $\mathcal{E}_{\mathsf{GAPP}}$ outputs $\perp$ with negligible probability for any PPT adversary $\mathcal{A}$. Indeed, if this is not the case for some PPT adversary $\mathcal{A}$, then one can use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}'$ that breaks knowledge-soundness of either uPC or bPC with non-negligible probability.

Now, assuming that $\mathcal{E}_{\mathsf{GAPP}}$ does not output $\perp$, we argue that we must have $(\mathbf{C}, (\mathbf{w}_0, \mathbf{w}_1)) \in \mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$, except with negligible probability. To see this, observe that since the above transcript passes all verification checks by an honest $\mathcal{V}$, we must have $G(\tilde{p}_j, \ldots, \tilde{p}_{\ell-1}) = Z_{\mathbb{V}}(y)\tilde{h} + \tilde{u}Z_{\mathbb{H}}(x)$. This follows immediately from the description of the protocol in Figure 2, since the verifier would reject otherwise. Further, since $\mathcal{E}_{\mathsf{GAPP}}$ did not output $\perp$, we must have

$$\widetilde{p}_j = K_j(x,y) = P_{\kappa(j)}(x, h_{\theta(j)}(y)) \;\; \forall j \in [\ell], \quad \widetilde{h} = H(x,y), \quad \widetilde{u} = u(x).$$

Since $x \leftarrow \mathbb{F}$ and all of the commitments $(\{C_j\}_{j \in [r]}, C_H, C_u)$ were produced by $\mathcal{A}$ before the honest $\mathcal{V}$ sent across the challenge x, by the Schwartz-Zippel lemma, the following must be true (except with probability $(r+2)/|\mathbb{F}| = \mathsf{negl}(\lambda)$):

$$Q(X,y) = G(p_0(X,y), \ldots, p_{\ell-1}(X,y)) = Z_{\mathbb{V}}(y)H(X,y) + u(X)Z_{\mathbb{H}}(X)$$

which is turn implies $Q(X,y) - Z_{\mathbb{V}}(y)H(X,y) = 0 \bmod Z_{\mathbb{H}}(X)$. Finally, since $y \leftarrow F$ and the commitments $(\{C_j\}_{j \in [r]}, C_H)$ were produced by $\mathcal{A}$ before the honest $\mathcal{V}$ sent across the challenge y, by the Schwartz-Zippel lemma, the following must again be true (except with probability $(r+1)/|\mathbb{F}| = \mathsf{negl}(\lambda)$):

$$Q(X,Y) - Z_{\mathbb{V}}(Y)H(X,Y) = 0 \bmod Z_{\mathbb{H}}(X)$$

This in turn implies that, except with negligible probability, we must have $(\mathbf{C}, (\mathbf{w}_0, \mathbf{w}_1)) \in \mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ by Lemma 4.2.

Succinctness. The succinctness of the protocol in Figure 2 follows immediately from the succinctness of uPC and bPC. Concretely, to argue proof-succinctness of the protocol, it suffices to observe that: (i) $|C_j| = \mathsf{poly}(\lambda)$ and $|\pi_j| = \mathsf{poly}(\lambda)$ for each $j \in [r]$, (ii) $|C_H| = \mathsf{poly}(\lambda)$ and $|\pi_h| = \mathsf{poly}(\lambda)$, and (iii) $|C_u| = \mathsf{poly}(\lambda)$ and $|\pi_u| = \mathsf{poly}(\lambda)$. The first two requirements are satisfied assuming the proof-succinctness of bPC, while the third requirement is satisfied assuming the proof-succinctness of uPC. Finally, verifier-succinctness of the protocol follows immediately from the verifier-succinctness of uPC and bPC.

This completes the proof of Theorem 4.1. □

4.3 Efficient Protocol for GAPP using KZG and bPCLB

We finally present a concrete instantiation of the the protocol in Figure 2, with KZG as the univariate PCS and bPCLB as the bivariate PCS, respectively, over the bilinear group BG. The corresponding parameters for this instantiation is described in the theorem below.

Theorem 4.2. *Assuming q-DLOG assumption holds for the bilinear group generator* BG*, there exists an argument of knowledge for the relation* $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ *in the algebraic group model. Moreover, the setup* $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda, d_x, d_y)$ *is universal for all relations* $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ *satisfying* $n \leq d_x$, $m \cdot \deg(G) \leq d_y$. *The protocol satisfies the following efficiency parameters:*

$$\begin{aligned}
|\pi^{\mathsf{GAPP}}| \quad &= (\ell + 2)\,\mathbb{F} + |\pi^{\mathsf{bPCLB}}(n, m)| \\
\mathrm{t}_{\mathrm{P}}^{\mathsf{GAPP}} \quad &= n \log(\deg(G)) \cdot \mathsf{FFT}(m \deg(G))\,\mathbb{F} + nm \deg(G)\,\mathbb{G}_1 + \mathrm{t}_{\mathrm{P}}^{\mathsf{bPCLB}}(n, m) \\
\mathrm{t}_{\mathrm{V}}^{\mathsf{GAPP}} \quad &= \ell\,\mathbb{G}_T + ||G||\,\mathbb{F} + \mathrm{t}_{\mathrm{V}}^{\mathsf{bPCLB}}(n, m)
\end{aligned}$$

In the above $||G||$ *denotes the size of arithmetic circuit computing* G, ℓ *denotes number of variables in* G, *while* π, t_{P} *and* t_{V} *denote proof size, prover complexity and verifier complexity respectively.*

Proof. The proof follows from the proof of Theorem 4.1, the security of our bPCLB commitment scheme proved in Theorem 3.1, and the security of the KZG commitment scheme. □

4.4 Application of GAPP to PLONK Proof Aggregation

Proof aggregation is a natural application of GAPP framework introduced in this paper. We use the instantiation of GAPP described in Section 4.3 with PLONK PIOP to obtain a simpler and modular proof aggregation for PLONK with universal setup. In terms of proof generation, our scheme outperforms the popular aggregation scheme aPlonk [ABST23] currently used in validity rollups for the Tezos blockchain. Our scheme also supports more efficient proof generation than the recently introduced scheme from [LXZ+24] as well as proof aggregation approaches based on incrementally verifiable computation (IVC) [KST22, KS22, BC23, KS24]. Some other advantages of our scheme are:

- Unlike [LXZ+24], we avoid the need for customized KZG commitments, which are incompatible with existing PLONK circuits based on publicly available universal SRS generated using `powers of tau` ceremony [pot].
- Unlike IVC-based aggregation frameworks [KST22, KS22, BC23, KS24] that make non-black-box use of cryptographic primitives (e.g., hash functions, bilinear groups etc.) and are therefore difficult to instantiate in a plug-and-play manner, we use cryptographic primitives in a fully black-box manner.

We benchmark our scheme against the naïve baseline of generating n separate PLONK proofs in Figure 3. We achieve 25-30% faster proof generation than the baseline. In comparison with [ABST23,

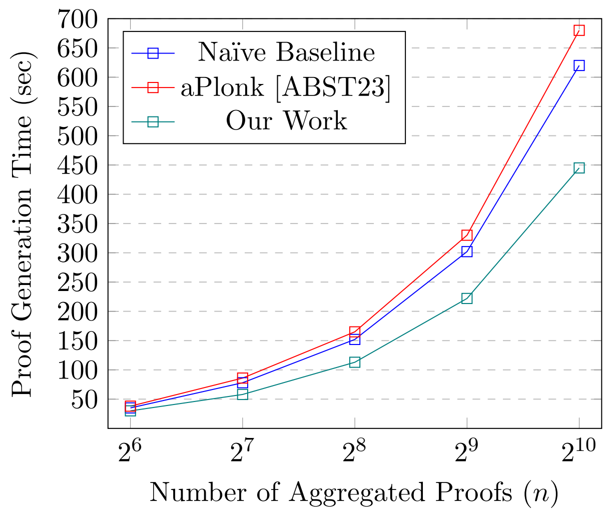

Figure 3: Comparative benchmarks for aggregation of PLONK proofs. We conservatively estimate a 10% overhead for aPlonk over the naïve baseline, based on performance reported in [ABST23]. Our verification time is similar to aPlonk, and proof size scales as $\approx 2.8 \log n$ KB for BLS12-381 curve. Individual circuits are of size 2^{14}. The comparisons were run using six cores on a desktop class machine with 32GB RAM.

LXZ$^+$24, KST22, KS22, BC23, KS24], all of which incur additional prover overheads on top of the baseline, we support significantly more efficient proving, while retaining succinct verification. Our advantage in proof generation over the baseline stems from the fact that we avoid computing evaluation proofs for the "quotient" polynomials for each instance, and instead only compute an evaluation proof for the bivariate polynomial aggregating them. The cryptographic operations incurred by evaluation proof using our bivariate PCS bPCLB are almost the same as that for *one* instance of univariate evaluation proof.

Our Approach. For simplicity, we describe proof aggregation for n identical PLONK circuits. This is a common setting in rollup applications, with each circuit verifying a batch of transactions. We also benchmark our scheme against the naive approach of generating n separate proofs. The circuit for a relation in PLONK PIOP is described by the vector

$$\big((q_M(Y), q_L(Y), q_R(Y), q_O(Y), q_C(Y), S_a(Y), S_b(Y), S_c(Y))\big) \in (\mathbb{F}[Y])^8$$

where the first five polynomials represent the gate constraints, while the latter three represent the "wiring" constraints. The commitments to above polynomials are known to the verifier. As part of the proof, the prover commits to "witness" polynomials $a(Y)$, $b(Y)$ and $c(Y)$ (we ignore public inputs for simplicity, the parts of witness may be opened orthogonal to the aggregation). Following the verifier's challenges after prover commits to the witness, a PLONK proof essentially requires prover to show that the polynomial $q(Y)$ as defined by

$$\begin{aligned}
q(Y) \equiv\ & \big(q_M(Y)a(Y)b(Y) + q_L(Y)a(Y) + q_R(Y)b(Y) + q_O(Y)c(Y) + q_C(Y)\big) \\
& + \alpha\big((a(Y) + \beta Y + \gamma)(b(Y) + \beta k_1 Y + \gamma)(c(Y) + \beta k_2 Y + \gamma)z(Y)\big) \\
& - \alpha\big((a(Y) + \beta S_a(Y) + \gamma)(b(Y) + \beta S_b(Y) + \gamma)(c(Y) + \beta S_c(Y) + \gamma)z(\nu Y)\big) \\
& + \alpha^2 \mu_0^{\mathbb{H}}(Y)(z(Y) - 1) \qquad (9)
\end{aligned}$$

vanishes over the domain $\mathbb{V}$ consisting of m roots of unity $\{1, \nu, \ldots, \nu^{m-1}\}$. Here m denotes the number of gates in the circuit, while α, β and γ denote uniform challenges from the verifier. The polynomials $a(Y), b(Y), c(Y)$ and $z(Y)$ are supplied by the prover. Let $(a_i(Y), b_i(Y), c_i(Y), z_i(Y))_{i \in [n]}$ be the prover's polynomials corresponding to each of the n proofs. Define multivariate polynomial G as below:

$$
\begin{aligned}
&G(X_0, X_1, \ldots, X_{12}, X_{13}) := \\
&\quad X_0 X_8 X_9 + X_1 X_8 + X_2 X_9 + X_3 X_{10} + X_4 \\
&\quad + \alpha\big((X_8 + \beta X_{13} + \gamma) \cdot (X_9 + \beta k_1 X_{13} + \gamma) \cdot (X_{10} + \beta k_2 X_{13} + \gamma) \cdot X_{11}\big) \\
&\quad - \alpha\big((X_8 + \beta X_5 + \gamma) \cdot (X_9 + \beta X_6 + \gamma) \cdot (X_{10} + \beta X_7 + \gamma) \cdot X_{12}\big) \\
&\quad + \alpha^2 \mu_0^{\mathbb{V}} \cdot (X_{12} - 1)
\end{aligned} \tag{10}
$$

The aggregate proof requires the prover to show that for all $i \in [n]$, the polynomial

$$G(q_M, q_L, q_R, q_O, q_C, S_a, S_b, S_c, a_i, b_i, c_i, z_i, \bar{z}_i, Y)$$

vanishes over $\mathbb{V}$ for $\bar{z}_i(Y) = z_i(\nu Y)$. Now, by Lemma 4.1, above is equivalent to proving that the polynomial

$$Q(X, Y) = G(q_M, q_L, q_R, q_O, q_C, S_a, S_b, S_c, A, B, C, Z, \overline{Z}, Y) \tag{11}$$

where $\overline{Z}(X, Y) = Z(X, \nu Y)$ vanishes over the domain $\mathbb{H} \times \mathbb{V}$. The aggregate proof using GAPP framework is described in Figure 4, where we omit details of PLONK argument which can be found in [GWC19]. Broadly, in the first two messages, the prover commits to packed polynomials $A(X, Y)$, $B(X, Y)$, $C(X, Y)$, $Z(X, Y)$, corresponding to univariate polynomials $a_i(Y), b_i(Y), c_i(Y)$, $z_i(Y)$ for $i \in [n]$. We note that for polynomials which are constant across n instances (circuit polynomials, Y, etc), their packed polynomial is identical to the univariate polynomial. The prover and verifier now invoke the protocol in Figure 2 to prove that $(\kappa, \theta, \mathbf{h}, \chi)$ is a valid statement in $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp}, G, n, m}$ where

$$\chi = (\chi_M, \chi_L, \chi_R, \chi_O, \chi_C, \chi_a, \chi_b, \chi_c, \chi_A, \chi_B, \chi_C, \chi_Z, \perp)$$

is a vector of commitments to polynomials, with the first eight corresponding to circuit polynomials, and last four to the prover polynomials. We set the final commitment to polynomial Y as $\perp$, as it can be succinctly evaluated. Next, we set $\mathbf{h} = (Y, \nu Y)$ and set maps κ and θ to ensure that variable X_{11} is bound to $Z(X, Y)$ while variable X_{12} is bound to $Z(X, \nu Y)$. Above protocol is trivially extended to the case, when different circuit polynomials are used in different instances. Let $\mathcal{C} = (\mathcal{C}_i)_{i=0}^{n-1}$ as a family of circuits, each of which is represented using at most m PLONK constraints. We define the language $\mathcal{R}^{\mathsf{agg}}_{\mathsf{pp}, \mathcal{C}}$ and say that $\mathbf{w} = (\mathbf{w}_0, \ldots, \mathbf{w}_{n-1}) \in \mathcal{R}^{\mathsf{agg}}_{\mathsf{pp}, \mathcal{C}}$ if $\mathcal{C}_i(\mathbf{w}_i) = 1$ for all $i \in [n]$.

Lemma 4.3. *Assuming that q-DLOG holds for bilinear group generator* BG *there exists an argument of knowledge for the langugage* $\mathcal{R}^{\mathsf{agg}}_{\mathsf{pp}, \mathcal{C}}$ *in the algerbraic group model. For the case of identical circuits, the protocol in Figure 4 is an argument of knowledge for the sub-language of* $\mathcal{R}^{\mathsf{agg}}_{\mathsf{pp}, \mathcal{C}}$ *consisting of identical circuits, with following efficiency parameters:*

$$
\begin{aligned}
|\pi^{\mathsf{agg}}| &= |\pi^{\mathsf{GAPP}}(n, m)| \\
\mathrm{t}_{\mathrm{P}}^{\mathsf{agg}} &= \mathrm{t}_{\mathrm{P}}^{\mathsf{GAPP}}(n, m) \\
\mathrm{t}_{\mathrm{V}}^{\mathsf{agg}} &= \mathrm{t}_{\mathrm{V}}^{\mathsf{GAPP}}(n, m) + 12\,\mathbb{G}_T + 6\,\mathbb{F}
\end{aligned}
$$

- **Setup**: Setup generates the following public parameters:

$$\mathsf{pp_{uPC}} \leftarrow \mathsf{uPC.Setup}(1^\lambda, d_x), \quad \mathsf{pp_{bPC}} \leftarrow \mathsf{bPC.Setup}(1^\lambda, (d_x, d_y)).$$

- **Common Input**: Commitments $[q_M(Y)]_1$, $[q_L(Y)]_1$, $[q_R(Y)]_1$, $[q_O(Y)]_1$, $[q_C(Y)]_1$, $[S_a(Y)]_1$, $[S_b(Y)]_1$, $[S_c(Y)]_1$ to the circuit polynomials.
- **Prover's Input**: Witness polynomials $(a_i(Y), b_i(Y), c_i(Y))$ for $i \in [n]$.
- **Round 1**: The prover $\mathcal{P}$ commits to aggregated witness.
  1. $\mathcal{P}$ computes packed polynomials $A(X,Y)$, $B(X,Y)$ and $C(X,Y)$ as:
  $$A = \sum_{i\in[n]} \mu_i(X)a_i(Y),\ B = \sum_{i\in[n]} \mu_i(X)b_i(Y),\ C = \sum_{i\in[n]} \mu_i(X)c_i(Y)$$
  2. $\mathcal{P}$ computes commitment $[\,A\,]_{\rm bv} = \mathsf{bPCLB.Com}(\mathsf{pp_{bPC}}, A)$ and similarly commitments $[\,B\,]_{\rm bv}$, $[\,C\,]_{\rm bv}$ for polynomials B, C respectively.
  3. $\mathcal{P}$ sends $[\,A\,]_{\rm bv}$, $[\,B\,]_{\rm bv}$ and $[\,C\,]_{\rm bv}$.
  4. $\mathcal{V}$ sends $\beta, \gamma \leftarrow \mathbb{F}$.
- **Round 2**: Prover commits to auxiliary aggregated witness.
  1. $\mathcal{P}$ computes the polynomials $z_i(Y)$, $i \in [n]$ as in [GWC19].
  2. $\mathcal{P}$ computes packed polynomial $Z(X,Y) = \sum_{i\in[n]} \mu_i(X)z_i(Y)$.
  3. $\mathcal{P}$ computes commitment $[\,Z\,]_{\rm bv} = \mathsf{bPCLB.Com}(\mathsf{pp_{bPC}}, Z)$.
  4. $\mathcal{P}$ sends $[\,Z\,]_{\rm bv}$
  5. $\mathcal{V}$ sends $\alpha \leftarrow \mathbb{F}$.
- **Round 3**: Prover and Verifier execute the GAPP argument.
  1. $\mathcal{P}$ and $\mathcal{V}$ set $G(q_M, q_L, q_R, q_O, q_C, S_a, S_b, S_c, a, b, c, z, \bar{z})$ as in Equation (10).
  2. $\mathcal{P}$ and $\mathcal{V}$ define statement $\mathbf{x} = (\kappa, \theta, \mathbf{h}, \chi)$ as in Section 4.4.
  3. $\mathcal{P}$ and $\mathcal{V}$ execute argument of knowledge (Figure 2) for $\mathbf{x} \in \mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$.
  4. $\mathcal{V}$ accepts if the above argument accepts.

Figure 4: PLONK proof aggregation using the GAPP framework.

Comparison with baseline. Note that computing the aggregated polynomial A, B, C, Z incur effort almost equivalent to computing and committing to the same polynomials when generating n separate proofs. Computing the univariate components of the H polynomial involves computing the form G over the univariate polynomials for each $i \in [n]$, and committing to them for computing commitment to H. Again, this effort is identical to computing and committing to the individual quotient polynomials for the n proof instances. However, when computing separate proofs, the prover additionally needs to compute opening proofs for the quotient polynomial, which incurs additional $O(m)$ cryptographic operations per proof. Existing aggregation schemes such as aPlonk [ABST23] incur further costs of evaluating all the polynomials and committing to their evaluations. By contrast, GAPP protocol only requires opening proofs for the committed bivariate polynomials at a random point (x, y) which incurs only $O(m+n)$ additional cryptographic operations, instead of $O(mn)$. Thus, for large enough n, proof aggregation achieved by our scheme is almost 25-30% faster than naive proof generation. Moreover, we see that our scheme supports highly parallel implementation as the univariate components of the bivariate polynomials can be computed and committed in parallel. More experimental results appear in Figure 3.

5 À la carte Proof System for Non-Uniform Computations

In this section, we show how to use the GAPP framework and the bivariate PCS bPCLB to construct an À la carte proof system for non-uniform computations. As a building block of our construction, we design a new (index) lookup argument for tuples, which we describe first.

5.1 Indexed Lookup Argument for Tuples

A lookup argument allows a prover to show that a vector appears as a subvector in the target vector (a table) given commitments to the respective vectors. An *indexed* lookup argument additionally allows the prover to show that a vector appears in the target vector at positions which are also committed as a vector. We first define indexed lookup for tuples as considered in [DXNT24, CGG+24], and then present a construction based on the bPCLB bivariate PCS. The prior works achieved constant time verification, but required $O(mn)$-sized public parameters and $\widetilde{O}(mn)$ cryptographic operations for the prover. We substantially reduce both of these overheads to $O(m+n)$, while incurring a slightly costlier (logarithmic) verification.

Defining Indexed Lookup for Tuples. Let $\mathbb{K}, \mathbb{H}$ and $\mathbb{V}$ denote the subgroups generated by primitive k^{th}, n^{th} and m^{th} roots of unity in $\mathbb{F}$ respectively. Let $\{\mu_i^{\mathbb{K}}\}_{i\in[k]}$, $\{\mu_i^{\mathbb{H}}\}_{i\in[n]}$ and $\{\mu_i^{\mathbb{V}}\}_{i\in[m]}$ denote the Lagrange basis polynomials for the subgroups $\mathbb{K}$, $\mathbb{H}$ and $\mathbb{V}$ respectively. For a vector $\mathbf{t} \in \mathbb{F}^k$ we define the polynomial encoding of $\mathbf{t}$ over $\mathbb{K}$ as $\mathsf{Enc}_{\mathbb{K}}(\mathbf{t}) = \sum_{i=0}^{k-1} t_i \mu_i^{\mathbb{K}}(X)$. In other words, the encoded polynomial interpolates the vector over the domain $\mathbb{K}$. Encodings $\mathsf{Enc}_{\mathbb{H}}(\cdot)$ and $\mathsf{Enc}_{\mathbb{V}}(\cdot)$ are analogously defined for vectors of size n and m respectively.

We extend the above encoding to vectors of m-tuples. To commit to a vector $\mathbf{T} = (\mathbf{t}_0, \ldots, \mathbf{t}_{k-1})$ of m-tuples, we canonically associate m-tuple $\mathbf{t}_i$ with the polynomial $t_i(Y) = \mathsf{Enc}_{\mathbb{V}}(\mathbf{t}_i) \in \mathbb{F}^{<m}[Y]$. The vector $\mathbf{T}$ is encoded as a bivariate polynomial over $\mathbb{K} \times \mathbb{V}$ given by: $\mathsf{Enc}_{\mathbb{K}\times\mathbb{V}}(\mathbf{T}) = T(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{K}}(X) t_i(Y)$.

We can similarly encode a vector $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_{n-1})$ of m-tuples as bivariate polynomial $A(X,Y)$ given by: $\mathsf{Enc}_{\mathbb{H}\times\mathbb{V}}(\mathbf{A}) = A(X,Y) = \sum_{i=0}^{k-1} \mu_i^{\mathbb{H}}(X) a_i(Y)$.

The committed index lookup relation (defined formally below), asserts that i^{th} m-tuple in vector $\mathbf{A}$ appears as the $\mathbf{u}[i]$-th m-tuple in the table $\mathbf{T}$ given commitments C_T, C_A and C_u to $\mathbf{T}, \mathbf{A}$ and $\mathbf{u}$ respectively.

Definition 5.1 (Indexed Lookup)**.** *For* $\mathsf{pp_{bPC}} \leftarrow \mathsf{bPCLB.Setup}(1^\lambda, d_x, d_y)$, $\mathsf{pp_{uPC}} \leftarrow \mathsf{KZG.Setup}(1^\lambda, d_x)$ *and integers* k, m, n *with* $\max(k,n) < d_x$, $m < d_y$, *we define the relation* $\mathcal{R}^{\mathsf{clookup}}_{\mathsf{pp},k,m,n}$ *consisting of tuples* $(\mathbf{x}, \mathbf{w})$ *where* $\mathbf{x} = (C_A, C_T, C_u) \in \mathbb{G}_T^2 \times \mathbb{G}_1$ *and* $\mathbf{w} = (\mathbf{A}, \mathbf{T}, \mathbf{u}, \tilde{\mathbf{c}}_a, \tilde{\mathbf{c}}_t, \tilde{\mathbf{c}}_u)$ *such that:*

- $\mathbf{A}[\,i\,] = \mathbf{T}[\,\mathbf{u}[i]\,]$ *for all* $i \in [n]$,
- $\mathsf{bPCLB.Open}(\mathsf{pp_{bPC}}, C_A, (n, d_y), A, \tilde{\mathbf{c}}_a) = 1$,
- $\mathsf{bPCLB.Open}(\mathsf{pp_{bPC}}, C_T, (k, d_y), T, \tilde{\mathbf{c}}_t) = 1$ *and*
- $\mathsf{uPC.Open}(\mathsf{pp_{uPC}}, C_u, n, u, \tilde{\mathbf{c}}_u) = 1$.

Here $A(X,Y) = \mathsf{Enc}_{\mathbb{H}\times\mathbb{V}}(\mathbf{A})$, $T(X,Y) = \mathsf{Enc}_{\mathbb{K}\times\mathbb{V}}(\mathbf{T})$ *and* $u(X) = \mathsf{Enc}_{\mathbb{H}}(\mathbf{u})$.

- **Setup**: Setup generates the following public parameters:

$$\mathsf{pp}_{\mathsf{uPC}} \leftarrow \mathsf{KZG.Setup}(1^\lambda, d_x), \quad \mathsf{pp}_{\mathsf{bPC}} \leftarrow \mathsf{bPCLB.Setup}(1^\lambda, (d_x, d_y)).$$

- **Common Input**: Commitments $(C_A, C_T, C_u) \in \mathbb{G}_T^2 \times \mathbb{G}_1$ and integers $k \le d_x$, $n \le d_x$ and $m \le d_y$.
- **Prover's Input**: Vectors $\mathbf{A} = (\mathbf{a}_0, \ldots, \mathbf{a}_{n-1}) \in (\mathbb{F}^m)^n$, $\mathbf{T} = (\mathbf{t}_0, \ldots, \mathbf{t}_{k-1}) \in (\mathbb{F}^m)^k$, $\mathbf{u} \in \mathbb{F}^n$, opening hints $\tilde{\mathbf{c}}_a, \tilde{\mathbf{c}}_t$ and $\tilde{\mathbf{c}_u}$. Polynomials $A(X,Y)$, $T(X,Y)$ and $u(X)$ encoding vectors $\mathbf{A}$, $\mathbf{T}$ and $\mathbf{u}$ respectively.
- **Round 1**: Prover commits to univariate polynomials.
  1. $\mathcal{V}$ sends $\gamma \leftarrow \mathbb{F}$.
  2. $\mathcal{P}$ computes vectors $\mathbf{A}_\gamma = (a_i(\gamma))_{i\in[n]}$, $\mathbf{T}_\gamma = (t_i(\gamma))_{i\in[k]}$ and polynomials $A_\gamma(X)$ and $T_\gamma(X)$ interpolating $\mathbf{A}_\gamma$ and $\mathbf{T}_\gamma$ on $\mathbb{H}$ and $\mathbb{K}$ respectively.
  3. $\mathcal{P}$ sends commitment C_a and C_t to polynomials $A_\gamma(X) = A(X,\gamma)$ and $T_\gamma(X) = T(X,\gamma)$ respectively.
  4. $\mathcal{V}$ sends $x \leftarrow \mathbb{F}$.
- **Round 2**: Prover sends evaluations.
  1. $\mathcal{P}$ sends evaluations $\tilde{a} = A_\gamma(x)$, $\tilde{t} = T_\gamma(x)$.
  2. $\mathcal{V}$ sends $r \leftarrow \mathbb{F}$.
- **Round 3**: Prover proves evaluations.
  1. $\mathcal{P}$ computes proofs

$$\pi \leftarrow \mathsf{KZG.Prove}(\mathsf{pp}_{\mathsf{uPC}}, T_\gamma(X) + rA_\gamma(X), x),$$
$$\pi' \leftarrow \mathsf{bPCLB.Prove}(\mathsf{pp}_{\mathsf{bPC}}, T(X,Y) + rA(X,Y), (x,\gamma))$$

  2. $\mathcal{P}$ sends π and π' to $\mathcal{V}$.
- **Round 4:** $\mathcal{V}$ checks evaluations and lookup argument.
  1. $\mathcal{V}$ computes:

$$b \leftarrow \mathsf{KZG.Verify}(\mathsf{pp}_{\mathsf{uPC}}, C_t + r\cdot C_a, k+n, x, \tilde{t} + r\tilde{a}, \pi)$$
$$b' \leftarrow \mathsf{bPCLB.Verify}(\mathsf{pp}_{\mathsf{bPC}}, C_T + r\cdot C_A, (k+n, m), (x,\gamma), \tilde{t} + r\tilde{a}, \pi')$$

  2. $\mathcal{P}$ and $\mathcal{V}$ execute lookup argument

$$b'' \leftarrow \langle \mathcal{P}_{\mathsf{lk}}(C_a, C_t, C_u; (\mathbf{A}_\gamma, \mathbf{T}_\gamma, \mathbf{u}, *)), \mathcal{V}_{\mathsf{lk}}(C_a, C_t, C_u)\rangle$$

  to check $\mathbf{A}_\gamma \preceq \mathbf{T}_\gamma$.
  3. $\mathcal{V}$ accepts if $b = b' = b'' = 1$, otherwise it rejects.

Figure 5: Argument of knowledge for the relation $\mathcal{R}^{\mathsf{lookup}}_{\mathsf{pp},k,m,n}$.

Our Approach: Tuple Lookup Using bPCLB. Let $A(X,Y), T(X,Y)$ and $u(X)$ be polynomials encoding the vectors $\mathbf{A}$, $\mathbf{T}$ and $\mathbf{u}$ over domains $\mathbb{H}\times\mathbb{V}$, $\mathbb{K}\times\mathbb{V}$ and $\mathbb{K}$ respectively. For $\gamma \in \mathbb{F}$, let $\mathbf{A}_\gamma = (a_0(\gamma), \ldots, a_{n-1}(\gamma))$ and $\mathbf{T}_\gamma = (t_0(\gamma), \ldots, t_{k-1}(\gamma))$ be vectors obtained by evaluating the constituent polynomials at $Y = \gamma$. We note that by Schwartz-Zippel Lemma, for uniform γ, except with negligible probability $(kn/|\mathbb{F}|)$ it holds that: $\mathbf{A}_\gamma[\,i\,] = \mathbf{T}_\gamma[\,\mathbf{u}[\,i\,]\,]\,\forall i \in [n] \iff \mathbf{A}[\,i\,] = \mathbf{T}[\,\mathbf{u}[\,i\,]\,]\,\forall i \in [n]$.

Building on the above observation, and noticing that polynomials $A(X,\gamma)$ and $T(X,\gamma)$ encode the vectors $\mathbf{A}_\gamma$ and $\mathbf{T}_\gamma$ over domains $\mathbb{H}$ and $\mathbb{K}$ respectively, the verifier sends $\gamma \leftarrow \mathbb{F}$ to the prover, who responds with KZG commitments C_a and C_t to the polynomials $A_\gamma(X) = A(X,\gamma)$ and $T_\gamma(X) = T(X,\gamma)$. This reduces the case for general m to that for $m = 1$. The verifier can now check that (C_a, C_t, C_u) is a valid statement in $\mathcal{R}_{\mathsf{pp}_{\mathsf{uPC}},k,n,1}$ using existing lookup arguments such as as [DGP+24, Hab22, GW20, EFG22, CFF+24, BC23] etc.

Additionally, the verifier needs to to check consistency of commitments C_t and C_a with polynomials T and A respectively. To do so, the verifier sends another challenge $x \leftarrow \mathbb{F}$ and executes an argument to check: (i) $A(x) = A(x, \gamma)$, (ii) $T(x) = T(x, \gamma)$. Both the conditions are checked by requesting openings to the univariate and bivariate polynomials at x and (x, γ) respectively. The complete protocol appears in Figure 5.

Lemma 5.1. *Assuming that q-DLOG is hard for the bilinear group generator* BG *and* $(\mathcal{P}_{\mathsf{lk}}, \mathcal{V}_{\mathsf{lk}})$ *is an argument of knowledge for the relation* $\mathcal{R}^{\mathsf{clookup}}_{\mathsf{pp},k,m,n}$ *for* $m = 1$ *under the q-DLOG and algebraic group model (AGM), the protocol in Figure 5 is an argument of knowledge for the relation* $\mathcal{R}^{\mathsf{clookup}}_{\mathsf{pp},k,m,n}$ *in the algebraic group model. It satisfies following efficiency parameters:*

$$\begin{aligned}
|\pi^{\mathsf{lookup}}(k,m,n)| &= |\pi^{\mathsf{lk}}| + |\pi^{\mathsf{bPCLB}}(t,m)| \\
\mathrm{t}_{\mathrm{P}}^{\mathsf{lookup}} &= \mathrm{t}_{\mathrm{P}}^{\mathsf{bPCLB}}(t,m) + \mathrm{t}_{\mathrm{P}}^{\mathsf{lk}}(n,k) + O(t)\,\mathbb{G}_1 + O(mt)\,\mathbb{F} \\
\mathrm{t}_{\mathrm{V}}^{\mathsf{lookup}} &= \mathrm{t}_{\mathrm{V}}^{\mathsf{bPCLB}}(t,m) + \mathrm{t}_{\mathrm{V}}^{\mathsf{lk}}(n,k)
\end{aligned}$$

where $t = \max(n,k)$ *and where the superscript* lk *indicates the complexities of the lookup argument* $(\mathcal{P}_{\mathsf{lk}}, \mathcal{V}_{\mathsf{lk}})$.

5.2 "A la-carte" Proof System for Non-Uniform Computation

In this section, we show how to use the GAPP framework and the indexed tuple lookup argument above to construct an À la carte proof system for non-uniform computations. Our approach makes only black-box use of cryptography and further improves over [DXNT24, CGG+24] in terms of efficiency, while being agnostic to the choice of polynomial IOP.

Model for Non-Uniform Computation. Our description of non-uniform computation is closely related to that in non-uniform IVC schemes such as [KS22] and prior work [CGG+24]. We parameterize the computation by a pre-defined family of functions $\mathcal{F} = \{F_0, \ldots, F_{k-1}\}$. For concreteness, we assume that each of the functions can be encoded as at most m PLONK constraints. We also impose minimal uniformity on the public wires of the circuits which does not affect generality. Specifically, we assume that for all $i \in [k]$, the circuit F_i takes s inputs given by $\mathbf{u}_i \in \mathbb{F}^s$, and outputs $\mathbf{v}_i \in \mathbb{F}^s$ possibly taking a non-deterministic input $\mathbf{w}_i$ of arbitrary size. For $n \in \mathbb{N}$, an n-step non-uniform computation over $\mathcal{F}$ consists of n invocations of functions from family $\mathcal{F}$. To allow *input-dependent* invocations of functions, we designate special "selector wires" $\tau_0, \ldots, \tau_{n-1}$ which carry the index (in $[k]$) of the function invoked. Moreover, to capture dependence across steps, we specify a permutation σ of $[(2s+1)n]$ which identifies the $(2s+1)n$ interface wires consisting of inputs, outputs and the selector wires (see Figure 6 for an illustration). The language recognized by the non-uniform computation $(\mathcal{F}, \sigma)$ is defined below:

Given $\mathcal{F}$, integers $m, n, s \in \mathbb{N}$, the language $\mathcal{L}^{\mathsf{nuc}}_{\mathcal{F},\sigma,n,m}$ expressing n-step execution of the non-uniform computation $\mathcal{F}$ with wiring permutation $\sigma : [(2s+1)n] \to [(2s+1)n]$ consists of statements of the form $(\mathbf{w}_{\mathrm{io}}, \mathbf{w}_{\mathrm{int}})$ where

$$\mathbf{w}_{\mathrm{io}} = \big(\tau_0 || \mathbf{u}_0 || \mathbf{v}_0 || \cdots || \tau_{n-1} || \mathbf{u}_{n-1} || \mathbf{v}_{n-1}\big),\ \mathbf{w}_{\mathrm{int}} = \big(\mathbf{w}_0 || \cdots || \mathbf{w}_{n-1}\big)$$

such that $\mathbf{v}_i = F_{\tau_i}(\mathbf{u}_i, \mathbf{w}_i)$ for all $i \in [n]$, and $\mathbf{w}_{\mathrm{io}}[\,j\,] = \mathbf{w}_{\mathrm{io}}[\,\sigma(j)\,]$ for all $j \in [(2s+1)n]$. We wires $(\tau_0, \ldots, \tau_{n-1})$ select a particular function from the family at each step. The wires in vector $\mathbf{w}_{\mathrm{io}}$ are assumed to be *interface* wires, which are *shorted* using the permutation σ, whereas the

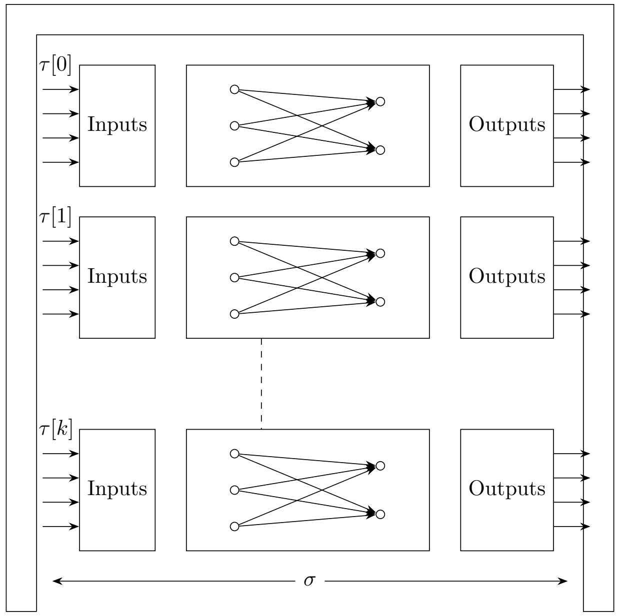

Figure 6: A non-uniform circuit is realized by piecing together different circuits for each step in an input defined manner. Here the selector wires $\tau[i]$ determine the circuit chosen for step i. The connections between inputs and outputs of different circuits are modeled using the permutation σ. The internal wires of the circuits are disjoint.

wires in $\mathbf{w}_{\text{int}}$ are exclusive to a specific step of computation. In typical applications, we expect activation wire τ_{i+1} to be computed as part of the output $\mathbf{v}_i$ (which is shorted to τ_{i+1} using σ). In the non-uniform IVC scheme [KS22], the activation τ_{i+1} is specified as $\tau_{i+1} = \varphi(\tau_i || \mathbf{u_i} || \mathbf{w}_i)$ for some efficiently computable function φ. In our setting, we assume φ is implemented as part of each circuit F_i. We additionally allow modeling "global" structure among interface wires using the permutation σ.

We now instantiate our scheme for proving n-step non-uniform computation as described above, using the PLONK PIOP. For all $j \in [k]$, let F_j be given by following vector of PLONK circuit polynomials:

$$\left(q_M^j(Y), q_L^j(Y), q_R^j(Y), q_O^j(Y), q_C^j(Y), S_a^j(Y), S_b^j(Y), S_c^j(Y)\right) \in (\mathbb{F}[Y])^8$$

Let T_M, T_L, T_R, T_O, T_C, T_a, T_b and T_c denote the tables (vectors) of polynomials $\left(q_M^j(Y)\right)_{j\in[k]}$, …, $\left(S_c^j(Y)\right)_{j\in[k]}$ respectively. Let $\mathcal{T}_M, \ldots, \mathcal{T}_c$ denote the (trusted) commitments to the tables T_M, …, T_a obtained as bPCLB commitments of packed polynomials below (see Section 5.1)

$$T_M(X, Y) = \sum_{j=0}^{k-1} \mu^{\mathbb{K}}(X) q_M^j(Y), \ldots, T_c(X, Y) = \sum_{j=0}^{k-1} \mu^{\mathbb{K}}(X) S_c^j(Y)$$

Let $a_i(Y), b_i(Y)$ and $c_i(Y)$ denote the witness polynomials for the circuit F_{τ_i}. The polynomials a_i, b_i and c_i interpolate the witness (left, right and output wires) for each of the m constraints in the circuit F_{τ_i}. We assume that $m > 2s + 1$ and all the interface wires for each step, namely $\tau_i, \mathbf{u}_i$ and $\mathbf{v}_i$ appear as the first $2s+1$ left wires. Thus, we have for all $i \in [q]$: $a_i(\nu^0) = \tau_i$, $a_i(\nu^{j+1}) = \mathbf{u}_i[j]$ and $a_i(\nu^{s+1+j}) = \mathbf{v}_i[j]$ for $j \in [s]$. We now present the argument of knowledge for the language $\mathcal{L}^{\mathsf{nuc}}_{\mathcal{F},\sigma,n,m}$. The protocol below consists of three main parts:

- First, the prover uses the tuple lookup argument to prove that correct circuit polynomials are used for each step, i.e, they correspond to selector wires $\tau_0, \ldots, \tau_{n-1}$.
- Next, the prover constructs an aggregate proof that witness polynomials (a_i, b_i, c_i) satisfy the PLONK constraints for F_{τ_i} for all $i \in [n]$.
- Finally, the prover proves that interface wires satisfy the permutation constraints imposed by σ.

Setup and Inputs. The setup phase of the protocol generates the public parameters for polynomial commitment schemes uPC and bPC:

$$\mathsf{pp}_{\mathsf{uPC}} \leftarrow \mathsf{KZG.Setup}(1^\lambda, d_x), \quad \mathsf{pp}_{\mathsf{bPC}} \leftarrow \mathsf{bPCLB.Setup}(1^\lambda, (d_x, d_y))$$

It also generates commitments $\mathbf{C}_{\mathcal{F}} = (\mathcal{T}_M, \mathcal{T}_L, \ldots, \mathcal{T}_c)$ to the functions in the family $\mathcal{F}$ as defined earlier. The prover's input consists of $(\mathbf{w}_{\text{io}}, \mathbf{w}_{\text{int}}) \in \mathcal{L}^{\mathsf{nuc}}_{\mathcal{F},\sigma,n,m}$.

Interactive Protocol. The interactive protocol between the prover ($\mathcal{P}$) and honest verifier ($\mathcal{V}$) proceeds as:

- *Commit Witness and Circuit Polynomials*: $\mathcal{P}$ computes commitments $[\,A\,]_{\text{bv}}, [\,B\,]_{\text{bv}}$ and $[\,C\,]_{\text{bv}}$ to packed witness polynomials $A(X,Y)$, $B(X,Y)$ and $C(X,Y)$ as described previously. $\mathcal{P}$ also computes packed polynomials corresponding to circuit polynomials activated at each step.
  - Specifically, $\mathcal{P}$ computes polynomials

$$q_M(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) q_M^{\tau_i}(Y),\ q_L(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) q_L^{\tau_i}(Y),$$
$$q_R(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) q_R^{\tau_i}(Y),\ q_O(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) q_O^{\tau_i}(Y),$$
$$q_C(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) q_C^{\tau_i}(Y),\ S_a(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) S_a^{\tau_i}(Y),$$
$$S_b(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) S_b^{\tau_i}(Y),\ S_c(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) S_c^{\tau_i}(Y)$$

  - $\mathcal{P}$ sends commitments $[\,A\,]_{\text{bv}}, [\,B\,]_{\text{bv}}, [\,C\,]_{\text{bv}}, [\,q_M\,]_{\text{bv}}, \ldots, [\,S_c\,]_{\text{bv}}$.
- *Check correctness of circuit polynomials.* $\mathcal{V}$ checks that the correct polynomials corresponding to vector $(\tau_0, \ldots, \tau_{n-1})$ have been looked up from respective tables.
  - $\mathcal{P}$ starts by sending commitment C_τ to a polynomial $\tau(X)$ which interpolates the vector $(\tau_0, \ldots, \tau_{n-1})$ over $\mathbb{H}$.
  - $\mathcal{V}$ sends a challenge $\chi \leftarrow \mathbb{F}$ to batch the lookup proofs.
  - $\mathcal{V}$ computes:

$$C_a = \left\langle (1, \chi, \ldots, \chi^7), ([\,q_M\,]_{\text{bv}}, [\,q_L\,]_{\text{bv}}, \ldots, [\,S_c\,]_{\text{bv}}) \right\rangle$$
$$C_t = \left\langle (1, \chi, \ldots, \chi^7), (\mathcal{T}_M, \mathcal{T}_L, \mathcal{T}_R, \mathcal{T}_O, \mathcal{T}_C, \mathcal{T}_a, \mathcal{T}_b, \mathcal{T}_c) \right\rangle$$

- $\mathcal{P}$ and $\mathcal{V}$ execute an argument of knowledge to show $(C_a, C_t, C_\tau) \in \mathcal{R}^{\mathsf{clookup}}_{\mathsf{pp},k,m,n}$.

- *Check correctness of selector polynomial.* To establish the correctness of the polynomial $\tau(X)$, $\mathcal{P}$ and $\mathcal{V}$ execute an argument (Lemma 4.2, Figure 2) to check that $\mu_0^{\mathbb{V}}(Y)(A(X,Y) - \tau(X)) = 0$ over $\mathbb{H} \times \mathbb{V}$. The identity ensures that $\tau(\omega^i) = A(\omega^i, \nu^0) = a_i(\nu^0) = \tau_i$, and thus $\tau(X)$ correctly interpolates the purported vector $(\tau_0, \ldots, \tau_{n-1})$ committed in $[\,A\,]_{\text{bv}}$.

- *Check aggregated circuit proofs.* The protocol now proceeds similar to the one for aggregation of PLONK proofs in Section **??**.
  - $\mathcal{V}$ starts by sending challenges $\beta, \gamma \leftarrow \mathbb{F}$.
  - $\mathcal{P}$ computes polynomial $z_i(Y)$ for each $i \in [n]$ according to the PLONK protocol treating
  $$\big(q_M^{\tau_i}(Y), q_L^{\tau_i}(Y), q_R^{\tau_i}(Y), q_O^{\tau_i}(Y), q_C^{\tau_i}(Y), S_a^{\tau_i}(Y), S_b^{\tau_i}(Y), S_c^{\tau_i}(Y)\big)$$
  as the circuit polynomials for the step i.
  - $\mathcal{P}$ sends commitment $[\,Z\,]_{\text{bv}}$ to $Z(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) z_i(Y)$.
  - $\mathcal{V}$ sends $\alpha \leftarrow \mathbb{F}$.
  - $\mathcal{P}$ and $\mathcal{V}$ execute an argument of knowledge to check that $(\kappa, \theta, \mathbf{h}, \mathbf{C})$ is a valid statement in $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ by setting $\mathbf{C} = \big(\,[\,q_M\,]_{\text{bv}}, [\,q_L\,]_{\text{bv}}, [\,q_R\,]_{\text{bv}}, [\,q_O\,]_{\text{bv}}, [\,q_C\,]_{\text{bv}}, [\,S_a\,]_{\text{bv}}, [\,S_b\,]_{\text{bv}}, [\,S_c\,]_{\text{bv}}, [\,A\,]_{\text{bv}}, [\,B\,]_{\text{bv}}, [\,C\,]_{\text{bv}}, [\,Z\,]_{\text{bv}}\big)$, G as the multivariate polynomial in Equation (10), $\mathbf{h}$, κ and θ as in the argument in Section 4.4.

- *Check wiring constraints.* To check that interface wires satisfy the permutation constraints imposed by σ, $\mathcal{P}$ and $\mathcal{V}$ execute the protocol described subsequently in Section 5.3. $\mathcal{V}$ accepts if all the checks pass, and rejects otherwise.

Lemma 5.2. *Assuming that q-DLOG is hard for the bilinear group generator* BG*, the above protocol is an argument of knowledge for the langugage* $\mathcal{L}^{\mathsf{nuc}}_{\mathcal{F},\sigma,n,m}$ *in the algebraic group model with following efficiency parameters:*

$$\begin{aligned} |\pi| &= |\pi^{\mathsf{agg}}(n,m)| + |\pi^{\mathsf{lookup}}(k,n,m)| \\ t_P &= t_P^{\mathsf{agg}}(n,m) + t_P^{\mathsf{lookup}}(k,n,m) + O(mn)\,\mathbb{F} + O(mn)\,\mathbb{G}_1 \\ t_V &= t_V^{\mathsf{agg}}(n,m) + t_V^{\mathsf{lookup}}(k,n,m) \end{aligned}$$

Proof. The proof essentially follows from the properties of arguments of knowledge for $\mathcal{R}^{\mathsf{GAPP}}_{\mathsf{pp},G,n,m}$ and $\mathcal{R}^{\mathsf{clookup}}_{\mathsf{pp},k,m,n}$. □

5.3 Generalized Grand-Product

Consider a bivariate polynomial $A(X,Y)$ with $\deg_X(A) < n$ and $\deg_Y(m) < m$, where we assume that A encodes a set of wires $\mathbf{a} = (a_0, \ldots, a_{nm-1})$ such that $A(\omega^i, \nu^j) = a_{mi+j}$. Given a permutation $\sigma : [nm] \to [nm]$, we present a bivariate PIOP, that checks $a_{\sigma(i)} = a_i$ for all $i \in [nm]$. We define the following polynomials for the identity permutation and σ:

$$S_{\mathsf{id}}(X,Y) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (mi+j)\mu_i^{\mathbb{H}}(X) \cdot \mu_j^{\mathbb{V}}(Y),$$

$$S_\sigma(X,Y) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \sigma(mi+j)\mu_i^{\mathbb{H}}(X) \cdot \mu_j^{\mathbb{V}}(Y) \tag{12}$$

Our PIOP relies on the following observation.

Lemma 5.3. *Let* $\mathbf{a} = (a_0, \ldots, a_{nm-1})$ *be a vector of length* nm *interpolated by the polynomial* A *over the domain* $\mathbb{H} \times \mathbb{V}$ *as above. Let* $\sigma : [nm] \to [nm]$ *be a permutation. Then, the vector* $\mathbf{a}$ *satisfies* $a_i = a_{\sigma(i)}$ *for all* $i \in [mn]$ *if and only if with high probability over the choice* $\beta, \gamma \leftarrow \mathbb{F}$*, there exist polynomials* $P(X,Y), U(X,Y) \in \mathbb{F}[X,Y]$ *and* $Q(X), R(X) \in \mathbb{F}[X]$ *such that:*

$$\begin{aligned}
P(X,Y)(\gamma + \beta S_\sigma(X,Y) + A(X,Y)) &\equiv_{\mathbb{H}\times\mathbb{V}} \gamma + \beta S_{\mathsf{id}}(X,Y) + A(X,Y) \\
U(X,\nu Y)(\mu_0^{\mathbb{V}}(Y)Q(X) - \mu_0^{\mathbb{V}}(Y) + 1) &\equiv_{\mathbb{H}\times\mathbb{V}} U(X,Y)P(X,Y) \\
\mu_0^{\mathbb{H}}(X)(R(X) - 1) &\equiv_{\mathbb{H}} 0 \\
R(\omega X) - R(X)Q(X) &\equiv_{\mathbb{H}} 0
\end{aligned}$$

Here $\equiv_S$ *denotes that the equality holds over set* S*.*

Proof. First, let us assume that the identities are true for some polynomials P, U, Q and R, given the pre-specified polynomials A, S_{id} and S_σ and uniformly sampled challenges $\beta, \gamma \leftarrow \mathbb{F}$. Now, putting $X = \omega^0$ in the third identity implies $R(\omega^0) = 1$. Putting $X = \omega^i$ for $i \in [n]$ in the last equation, implies $R(\omega^{i+1}) = R(\omega^i)Q(\omega^i)$ and thus

$$1 = \frac{R(\omega^n)}{R(\omega^0)} = Q(\omega^0) \cdot Q(\omega^1) \cdots Q(\omega^{n-1}) \tag{13}$$

Now, from the second identity, for all $i \in [n]$, we have by substituting $Y = \nu^0, \ldots, \nu^{m-1}$,

$$\begin{aligned}
U(\omega^i, \nu) \cdot Q(\omega^i) &= U(\omega^i, \nu^0) \cdot P(\omega^i, \nu^0) \\
U(\omega^i, \nu^2) \cdot 1 &= U(\omega^i, \nu) \cdot P(\omega^i, \nu) \\
&\vdots \\
U(\omega^i, \nu^m) \cdot 1 &= U(\omega^i, \nu^{m-1}) \cdot P(\omega^i, \nu^{m-1})
\end{aligned}$$

Multiplying and observing that $U(\omega^i, \nu^j)$ terms cancel off, we are left with $Q(\omega^i) = \prod_{j=0}^{m-1} P(\omega^i, \nu^j)$. Now, from Equation (13), we have

$$\prod_{i=0}^{n-1} \prod_{j=0}^{m-1} P(\omega^i, \nu^j) = 1 \tag{14}$$

Now, from the first identity, we have by substituting $X = \omega^i$ and $Y = \nu^j$,

$$P(\omega^i, \nu^j) = \frac{\gamma + \beta S_{\mathsf{id}}(\omega^i, \nu^j) + A(\omega^i, \nu^j)}{\gamma + \beta S_\sigma(\omega^i, \nu^j) + A(\omega^i, \nu^j)}$$

Then from Equation (14), we have:

$$\prod_{i=0}^{n-1} \prod_{j=0}^{m-1} \left(\frac{\gamma + \beta S_{\mathsf{id}}(\omega^i, \nu^j) + A(\omega^i, \nu^j)}{\gamma + \beta S_\sigma(\omega^i, \nu^j) + A(\omega^i, \nu^j)} \right) = 1$$

Since, the above holds for uniformly sampled $\beta, \gamma \leftarrow \mathbb{F}$, with high probability, we have the polynomial identity:

$$\prod_{i=0}^{n-1} \prod_{j=0}^{m-1} (X + (mi + j)Y + a_{mi+j}) = \prod_{i=0}^{n-1} \prod_{j=0}^{m-1} (X + \sigma(mi + j)Y + a_{mi+j})$$

and thus $a_{mi+j} = a_{\sigma(mi+j)}$ for all $(i,j) \in [n] \times [m]$. This proves one direction of the "if and only if" claim. The other direction is straightforward, where the polynomials P, U, Q and R can be constructed according to the preceding proof. $\square$

Argument of Knowledge for Checking Wiring Constraints. Lemma 5.3 implies a simple argument of knowledge for checking that witness encoded by polynomial $A \in \mathbb{F}[X,Y]$ satisfies wiring constraints given by permutation σ. Let $\mathsf{pp}_\sigma = (C_{\mathsf{id}}, C_\sigma)$ denote honestly generated commitments to polynomials $S_{\mathsf{id}}, S_\sigma$ as defined earlier. Let pp denote all the public parameters $(\mathsf{pp}_{\mathsf{uPC}}, \mathsf{pp}_{\mathsf{bPC}}, \mathsf{pp}_\sigma)$. We define $\mathcal{R}^{\mathsf{perm}}_{\mathsf{pp},\sigma}$ to be the relation consisting of pairs (C_A, A) such that

$$\mathsf{bPC.Open}(pp, A(X,Y), (d_x, d_y), C_A, \tilde{C}_A) = 1$$

and $a_k = a_{\sigma(k)}$ for all $k \in [nm]$, where $a_{mi+j} = A(\omega^i, \nu^j)$ for $(i,j) \in [n] \times [m]$. The protocol is outlined below:

Setup and Inputs. The setup consists of parameters pp as described above. The common input consists of (pp, C_A) where C_A is the commitment of the prover's polynomial. The prover also knows knows the witness polynomial $A(X,Y)$, the underlying witness vector $\mathbf{a} \in \mathbb{F}^{nm}$ and other publicly specified polynomials such as S_{id} and S_σ.

Prover Commits Auxiliary Polynomials. The verifier $\mathcal{V}$ initiates the protocol by sending $\beta, \gamma \leftarrow \mathbb{F}$. The prover $\mathcal{P}$ constructs polynomials $P(X,Y)$, $U(X,Y)$, $Q(X), R(X)$ in Lagrange basis as:

$$p_i(Y) = \sum_{j=0}^{m-1} \mu_j^{\mathbb{V}}(Y) \left(\frac{\gamma + \beta(mi+j) + a_{mi+j}}{\gamma + \beta\sigma(mi+j) + a_{\sigma(mi+j)}} \right),\ u_i(Y) = \sum_{j=0}^{m-1} \mu_j^{\mathbb{V}}(Y) \prod_{k=0}^{j-1} p_i(\nu^k)$$

$$P(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) p_i(Y),\ U(X,Y) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) u_i(Y)$$

$$Q(X) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) p_i(\mu^{m-1}),\ R(X) = \sum_{i=0}^{n-1} \mu_i^{\mathbb{H}}(X) \prod_{j=0}^{i-1} Q(\omega^j)$$

The prover sends commitments to polynomials P, U, Q and R using bPCLB as the bivariate PCS and KZG as the univariate PCS.

Prover and Verifier execute Bivariate PIOP: $\mathcal{P}$ and $\mathcal{V}$ execute PIOP given by identities in Lemma 5.3 using the protocols in Section 4.

References

[ABST23] Miguel Ambrona, Marc Beunardeau, Anne-Laure Schmitt, and Raphael R. Toledo. aPlonK: Aggregated PlonK from multi-polynomial commitment schemes. In Junji Shikata and Hiroki Kuzuno, editors, *IWSEC 23*, volume 14128 of *LNCS*, pages 195–213. Springer, Cham, August 2023.

[AC20] Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Cham, August 2020.

[AFG$^+$16] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. *Journal of Cryptology*, 29(2):363–421, April 2016.

[AST24] Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: SNARKs for virtual machines via lookups. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 3–33. Springer, Cham, May 2024.

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

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 77–110. Springer, Singapore, December 2023.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences*, 37(2):156–189, 1988.

[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 Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Berlin, Heidelberg, May 2016.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349, 2012.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Berlin, Heidelberg, August 2013.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCG+18] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 595–626. Springer, Cham, December 2018.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Berlin, Heidelberg, March 2013.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Cham, May 2020.

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 65–97. Springer, Cham, December 2021.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, Cham, April 2023.

[CFF+24] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: Improvements, extensions and applications zero-knowledge decision trees. In *Public-Key Cryptography*, page 337–369, 2024.

[CGG+24] Arka Rai Choudhuri, Sanjam Garg, Aarushi Goel, Sruthi Sekar, and Rohit Sinha. Sublonk: Sublinear prover plonk. *Proc. Priv. Enhancing Technol.*, pages 314–335, 2024.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020.

[DGP+24] Moumita Dutta, Chaya Ganesh, Sikhar Patranabis, Shubh Prakash, and Nitin Singh. Batching-efficient RAM using updatable lookup arguments. Cryptology ePrint Archive, Report 2024/840, 2024.

[DXNT24] Zijing Di, Lucas Xia, Wilson D. Nguyen, and Nirvan Tyagi. MuxProofs: Succinct arguments for machine computation from vector lookups. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part V*, volume 15488 of *LNCS*, pages 236–265. Springer, Singapore, December 2024.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763, 2022.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018.

[For87] Lance Fortnow. The complexity of perfect zero-knowledge (extended abstract). In Alfred Aho, editor, *19th ACM STOC*, pages 204–209. ACM Press, May 1987.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

[GGHAK22] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose Σ-protocols for disjunctions. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 458–487. Springer, Cham, May / June 2022.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013.

[GHAKS23] Aarushi Goel, Mathias Hall-Andersen, Gabriel Kaptchuk, and Nicholas Spooner. Speed-stacking: Fast sublinear zero-knowledge proofs for disjunctions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 347–378. Springer, Cham, April 2023.

[GK22] Ariel Gabizon and Dmitry Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. Cryptology ePrint Archive, Report 2022/1447, 2022.

[GMN22] Nicolas Gailly, Mary Maller, and Anca Nitulescu. SnarkPack: Practical SNARK aggregation. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 203–229. Springer, Cham, May 2022.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986.

[GPS25] Chaya Ganesh, Sikhar Patranabis, and Nitin Singh. Samaritan: Linear-time prover SNARK from new multilinear polynomial commitments. Cryptology ePrint Archive, Paper 2025/419, 2025.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Berlin, Heidelberg, December 2010.

[GW20] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

[Hab22] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530, 2022.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992.

[KS22] Abhiram Kothapalli and Srinath Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758, 2022.

[KS24] Abhiram Kothapalli and Srinath T. V. Setty. HyperNova: Recursive arguments for customizable constraint systems. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 345–379. Springer, Cham, August 2024.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Cham, August 2022.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010.

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Cham, November 2021.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Berlin, Heidelberg, March 2012.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Berlin, Heidelberg, December 2013.

[LXZ+24] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In *IEEE Symposium on Security and Privacy, SP 2024*, pages 1777–1793. IEEE, 2024.

[Mic94] Silvio Micali. Cs proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 436–453. IEEE, 1994.

[NMT] Shen Noether, Adam Mackenzie, and Monero Core Team. Ring confidential transactions. `https://lab.getmonero.org/pubs/MRL-0005.pdf`.

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

[PK22] Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Report 2022/957, 2022.

[pot] Perpetual powers of tau. `https://github.com/privacy-scaling-explorations/perpetualpowersoftau`.

[SCP+22] Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. Hyperproofs: Aggregating and maintaining proofs in vector commitments. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 3001–3018. USENIX Association, August 2022.

[STW24] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with Lasso. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 180–209. Springer, Cham, May 2024.

[YHH+23] Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. Batchman and robin: Batched and non-batched branching for interactive ZK. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1452–1466. ACM Press, November 2023.

[YZRM24] Xiao Yang, Chengru Zhang, Mark Ryan, and Gao Meng. Multivariate multipolynomial commitment and its applications. Cryptology ePrint Archive, Paper 2024/827, 2024.

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3121–3134. ACM Press, November 2022.

[ZGK+22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Report 2022/1565, 2022.

A Inner Product Arguments and Compressed Sigma Protocols

Compressed Sigma Protocol Framework. We briefly describe the compressed sigma protocol (CSP) framework of [AC20] using the abstraction of a *doubly homomorphic commitment* defined in [BMM+21]. To ease the discussion, we define a deterministic commitment scheme below, the generalization to a hiding commitment scheme is straightforward.

Definition A.1 ([BMM+21])**.** *Let* $(\mathsf{CM}, \mathsf{Setup})$ *be a* computationally binding *commitment scheme with message space* $\mathcal{M}$*, key space* $\mathcal{K}$ *and commitment space* $\mathcal{C}$*. We say that* $(\mathsf{CM}, \mathsf{Setup})$ *is* doubly homomorphic *if* $(\mathcal{M}, +)$*,* $(\mathcal{K}, +)$ *and* $(\mathcal{C}, +)$ *are abelian groups such that for all* $\mathsf{ck}, \mathsf{ck}' \in \mathcal{K}$ *and* $m, m' \in \mathcal{M}$*, we have:*

$$\mathsf{CM}(\mathsf{ck} + \mathsf{ck}', m + m') = \mathsf{CM}(\mathsf{ck}, m) + \mathsf{CM}(\mathsf{ck}', m) + \mathsf{CM}(\mathsf{ck}, m') + \mathsf{CM}(\mathsf{ck}', m') \tag{15}$$

Inner Product Commitment. We consider doubly homomorphic commitment schemes obtained from bilinear maps $\otimes : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ for groups $\mathcal{K}, \mathcal{M}$ and $\mathcal{C}$ with prime order p. For $n \in \mathbb{N}$, we can extend the map $\otimes$ to the inner product $\langle \, , \rangle_\otimes : \mathcal{K}^n \times \mathcal{M}^n \to \mathcal{C}$ between $\mathbb{Z}_p$-vector spaces $\mathcal{K}^n$ and $\mathcal{M}^n$ given by $\langle \mathbf{a} \, , \, \mathbf{b} \rangle_\otimes = \sum_{i=0}^{n-1}(a_i \otimes b_i)$. We call the doubly homomorphic commitment scheme $\mathsf{CM} : \mathcal{K}^n \times \mathcal{M}^n \to \mathcal{C}$ defined by $\mathsf{CM}(\mathsf{ck}, \mathbf{w}) = \langle \mathsf{ck} \, , \, \mathbf{w} \rangle_\otimes$ as an *inner product commitment.* Our notion of inner product commitment is a slight specialization of the same notion in [BMM+21], where we eschew full generalization to more concretely illustrate our folding techniques.

Linear Form. For $\mathcal{K} = \mathbb{Z}_p$, the scalar multiplication $\chi \otimes w = \chi \cdot w$ is a bilinear operator from $\mathbb{Z}_p \times \mathcal{M}$ to $\mathcal{M}$. In keeping with terminology used in CSPs, we call the resulting inner product commitment $\langle\, ,\, \rangle : \mathbb{Z}_p^n \times \mathcal{M}^n \to \mathcal{M}$ given by $\langle \mathsf{ck}\, ,\, \mathbf{w} \rangle = \sum_{i=0}^{n-1} \chi_i \cdot w_i$, with $\mathsf{ck} = (\chi_0, \ldots, \chi_{n-1}) \in \mathbb{Z}_p^n$ and $\mathbf{w} = (w_0, \ldots, w_{n-1}) \in \mathcal{M}^n$ as the *linear form* on $\mathcal{M}^n$.

CSP for Inner Product Commitments Let $\mathcal{K}$ and $\mathcal{M}$ and $\mathcal{C}$ be additive abelian groups of prime order p. Let $(\mathsf{Setup}, \mathsf{CM})$ denote the inner product commitment with message space $\mathcal{M}^*$, key space $\mathcal{K}^*$ and commitment space $\mathcal{C}$ given by $\mathsf{CM}(\mathsf{ck}, \mathbf{w}) = \langle \mathsf{ck}, \mathbf{w} \rangle_\otimes$ for $|\mathsf{ck}| = |\mathbf{w}|$. Otherwise, we define $\mathsf{CM}(\mathsf{ck}, \mathbf{w}) = \perp$. We now define the relation one proves using CSP.

Definition A.2. *Let $\mathcal{K}, \mathcal{M}, \mathcal{C}$ and CM be as above. For $n \in \mathbb{N}$, let $\mathcal{R}^{\mathsf{CSP}}_{\mathsf{CM},n}$ be the relation consisting of pairs $(\mathbf{x}, \mathbf{w})$ where $\mathbf{x} = (\mathsf{ck}, \mathbf{a}, C, v)$ with $\mathsf{ck} \in \mathcal{K}^n$, $\mathbf{a} \in \mathbb{Z}_p^n$, $C \in \mathcal{C}$, $v \in \mathcal{M}$ and $\mathbf{w} \in \mathcal{M}^n$ satisfying $\langle \mathsf{ck}, \mathbf{w} \rangle_\otimes = C$ and $\langle \mathbf{a}, \mathbf{w} \rangle = v$.*

In protocol π_{csp} to prove knowledge of $\mathbf{w} \in \mathcal{M}^n$ such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}^{\mathsf{CSP}}_{\mathsf{CM},n}$, where $\mathbf{x} = (\mathsf{ck}, \mathbf{a}, C, v)$ as in Definition A.2, the prover and verifier execute the following *folding step*:

- The prover splits the witness vector $\mathbf{w}$ into two equal sized vectors $\mathbf{w}_L = (w_0, \ldots, w_{n/2-1})$ and $\mathbf{w}_R = (w_{n/2}, \ldots, w_{n-1})$. It similarly computes $(\mathsf{ck}_L, \mathsf{ck}_R)$ and $(\mathbf{a}_L, \mathbf{a}_R)$ by splitting commitment key ck and the linear form $\mathbf{a}$ respectively. It then computes "cross terms" $A = \langle \mathsf{ck}_L\, ,\, \mathbf{w}_R \rangle_\otimes$, $A' = \langle \mathsf{ck}_R\, ,\, \mathbf{w}_L \rangle_\otimes$, $u = \langle \mathbf{a}_L\, ,\, \mathbf{w}_R \rangle$ and $u' = \langle \mathbf{a}_R\, ,\, \mathbf{w}_L \rangle$. It sends A, A', u, u' to the verifier.
- The verifier sends a uniform challenge $x_0 \leftarrow \mathbb{F}$.
- The prover computes new witness $\mathbf{w}_1 = \mathbf{w}_L + x_0^{-1} \cdot \mathbf{w}_R$.
- Prover and Verifier compute: $\mathsf{ck}_1 = \mathsf{ck}_L + x_0 \cdot \mathsf{ck}_R$, $\mathbf{a}_1 = \mathbf{a}_L + x_0 \cdot \mathbf{a}_R$, $C_1 = x_0 \cdot A' + C + x_0^{-1} \cdot A$, $v_1 = x_0 \cdot u' + v + x_0^{-1} u$.
- Prover and Verifier recursively run the argument of knowledge for showing $(\mathbf{x}_1, \mathbf{w}_1) \in \mathcal{R}^{\mathsf{CSP}}_{\mathsf{CM},n/2}$ where $\mathbf{x}_1 = (\mathsf{ck}_1, \mathbf{a}_1, C_1, v_1)$.

After $\log n$ rounds of the above compression step, the final commitment keys $\mathsf{ck}_{\log n}$ and $\mathbf{a}_{\log n}$ have size 1, in which case the prover sends the witness $w_{\log n}$ (of size 1) for the final relation. The verifier simply checks $\mathsf{ck}_{\log n} \otimes \mathbf{w}_{\log n} = C_{\log n}$ and $\mathbf{a}_{\log n} \cdot \mathbf{w}_{\log n} = v_{\log n}$. Note that we view size 1 vectors as scalars here. While the recursive folding yields logarithmic argument size, in general the verifier incurs $O(n)$ effort, as it is required to fold the commitment keys for the commitment and the linear form.

CSP with Logarithmic Verifier. To achieve logarithmic verification in the CSP framework, prior works such as [BMM+21, GMN22] have considered structured commitment keys. In this case, the verifier delegates the folding of the commitment keys to the prover, and can efficiently check if the final commitment key is correctly computed. The prior work has considered commitment keys with monomial structure i.e, $\mathsf{ck} = (\chi_0, \ldots, \chi_{n-1})$ with $\chi_i = x^i$ for some $x \leftarrow \mathbb{F}$. Moreover, folding of structured commitment keys which are encodings of monomials x^i in a group, e.g $x^i \cdot g$ for a group generator g can also be verified in logarithmic time. In this work, we show that folding of commitment keys structured as Lagrange basis polynomials for the subgroup consisting of n roots of unity can be verified in $O(\log n)$ time.