Proofs for Inner Pairing Products and Applications

Benedikt Bünz[1], Mary Maller[2], Pratyush Mishra[3], Nirvan Tyagi[4] and Psi Vesely[5]

[1] Stanford University
[2] Ethereum Foundation
[3] UC Berkeley
[4] Cornell University
[5] UC San Diego

Abstract. We present a generalized inner product argument and demonstrate its applications to pairing-based languages. We apply our generalized argument to prove that an inner pairing product is correctly evaluated with respect to committed vectors of n source group elements. With a structured reference string (SRS), we achieve a logarithmic-time verifier whose work is dominated by $6 \log n$ target group exponentiations. Proofs are of size $6 \log n$ target group elements, computed using $6n$ pairings and $4n$ exponentiations in each source group.

We apply our inner product arguments to build the first polynomial commitment scheme with succinct (logarithmic) verification, $O(\sqrt{d})$ prover complexity for degree d polynomials (not including the cost to evaluate the polynomial), and a SRS of size $O(\sqrt{d})$. Concretely, this means that for $d = 2^{28}$, producing an evaluation proof in our protocol is $76\times$ faster than doing so in the KZG commitment scheme, and the CRS in our protocol is $1000\times$ smaller: 13MB vs 13GB for KZG.

As a second application, we introduce an argument for aggregating n Groth16 zkSNARKs into an $O(\log n)$ sized proof. Our protocol is significantly faster ($> 1000\times$) than aggregating SNARKs via recursive composition: we aggregate $\sim 130,000$ proofs in 25 minutes, versus 90 proofs via recursive composition. Finally, we further apply our aggregation protocol to construct a low-memory SNARK for machine computations that does not rely on recursive composition. For a computation that requires time T and space S, our SNARK produces proofs in space $\tilde{\mathcal{O}}(S + T)$, which is significantly more space efficient than a monolithic SNARK, which requires space $\tilde{\mathcal{O}}(S \cdot T)$.

1 Introduction

An inner product argument proves that an inner product relation holds between committed vectors. In this work, we present a new construction of inner product arguments for pairing-based languages that yields a logarithmic time verifier — a significant improvement over the linear time verifier of previous work. We use our new inner product argument to build (1) a new polynomial commitment scheme that achieves novel asymptotic characteristics of succinct verification and opening

proofs that can be computed in time square root of the polynomial degree as well as a square root sized SRS; and (2) a new approach for aggregation of Groth16 general-purpose SNARKs [Gro16] useful for verifiable computation, avoiding the expensive costs of recursive proving circuits. We provide an open-source Rust implementationof all our protocols and applications and benchmark them against the state of the art. Our benchmarks show that the asymptotic improvements translate to significant practical gains.

Inner product arguments. Inner product arguments (IPA) are core components of many primitives, including zero-knowledge proofs and polynomial and vector commitment schemes [BCC+16; BBB+18; WTS+18; LMR19; BGH19; BCMS20]. Despite the fact that the inner product arguments constructed in these works largely share the same core strategy as the original protocol in [BCC+16], they all spend significant effort in reproving security to accommodate for minor changes (introduced for efficiency and/or application-specific purposes). This repeated effort adds significant overhead in the process of auditing the security of inner product arguments, and enables errors to slip through unnoticed. Our first contribution is an abstraction of previous work into a generalized inner product argument (GIPA). While the techniques in GIPA are not novel, they do provide a unified view of all prior work, enabling simpler exposition and simpler security proofs. In particular, this means that our single security proof suffices to prove the security of all prior GIPA instantiations [BCC+16; BBB+18; LMR19], as well as the protocols introduced in this paper.

We additionally prove security for the non-interactive variant of GIPA in a generalization of the algebraic group model [FKL18], which we dub the *algebraic commitment model*. Because GIPA is a public-coin protocol, it can be transformed to a non-interactive argument using the Fiat–Shamir heuristic, and it is this variant that is used in applications— non-interactive Bulletproofs secures almost 2 billion USD of Monero [O'L18]. However, due to a technicality about modeling random oracles in recursive arguments (the generic transformation leads to a super-polynomial extractor), prior works provided no satisfactory security proof for these non-interactive variants. Our security proof remedies this oversight, and we envisage that our techniques may be useful in proving the security of other non-interactive and recursive protocols [BFS20].

Reducing verification cost. Making use of the high level GIPA blueprint, our second contribution is a protocol for reducing the verifier cost for specific inner product arguments over pairing-based languages. For a committed vector length of n, we reduce the verifier cost from $\mathcal{O}(n)$ for existing protocols [LMR19], to $\mathcal{O}(\log n)$, which is an exponential improvement. To do this, we introduce a new pairing-based commitment scheme with structured keys and prove its security. We then exploit a special structure of the "homomorphic collapsing" execution of GIPA (first observed in [BGH19]) with our commitment scheme. In particular, the outsourced computation is reduced to opening a KZG polynomial commitment scheme. We rely on a trusted setup that is updatable [GKM+18] and can be used for languages of different sizes (up to some maximum bound specified by the SRS).

Equipped with our new logarithmic-time verifier for inner products over pairing-based languages, we next turn to apply our techniques to two applications: (1) polynomial commitments, and (2) SNARK aggregation.

Polynomial commitments. Polynomial commitment (PC) schemes [KZG10] are commitment schemes specialized to work with polynomials. A committer outputs a short commitment to a polynomial, and then later may convince a verifier of correctness of an evaluation of that committed polynomial at any point via a short evaluation proof, or "opening". PC schemes have been used to reduce communication and computation costs in a vast breadth of applications including proofs of storage and replication [XYZW16; Fis18], anonymous credentials [CDHK15; FHS19], verifiable secret sharing [KZG10; BDK13], and zero-knowledge arguments [WTS+18; MBKM19; Set20; GWC19; XZZ+19; CHM+20].

In this work, we use a combination of inner product arguments in order to build a pairing-based polynomial commitment scheme that requires a universal structured reference string of size only $\sqrt{d}$ when committing to degree d polynomials, and where proving an evaluation claim only requires $\mathcal{O}(\sqrt{d})$ cryptographic operations (i.e., group/pairing operations not including scalar computation). We achieve this while maintaining constant-sized commitments, $\mathcal{O}(\log d)$-sized evaluation proofs, and $\mathcal{O}(\log d)$ verifier time.

This compares to a linear sized CRS for the widely used KZG [KZG10] commitment scheme. Concretely, this means that for polynomial of degree 2^{22}, KZG requires a large SRS of size $\sim$ 400MB. This can cause deployment hurdles in applications in decentralized systems, as this SRS needs to be stored by every prover. For example, in SNARKs relying on polynomial commitments [GWC19; CHM+20], the degree of the polynomial is roughly the size of the circuit, which can be large [BCG+14; WZC+18]. A large SRS also has a non-trivial impact on security [GGW18]. In contrast, the SRS of our protocol has size 3MB, which is over 130$\times$ smaller, making deployment much easier.

Furthermore, as noted above, computing an evaluation proof requires only $\mathcal{O}(\sqrt{d})$ cryptographic operations, which is much better than KZG, which requires $\mathcal{O}(d)$ cryptographic operations. This is important for applications such as vector commitments [LY10] and proofs of space [Fis19], where a polynomial is committed to just once, but the commitment is opened at many different evaluation points.

SNARK aggregation. A SNARK aggregation protocol takes as input many SNARK proofs and outputs a single aggregated proof that can be verified more quickly than individually verifying each SNARK. This is useful for applications where the batch of proofs will be verified many times by different clients. For example, this is the case in applications that aim to improve the scalability of decentralized blockchains by using SNARKs to prove the correctness of state transitions [Whi; BMRS20].

We use our inner product arguments to design an aggregation protocol for Groth16 [Gro16] SNARKs that enjoys the following efficiency properties when aggregating n proofs: (a) aggregation requires $\mathcal{O}(n)$ cryptographic operations,

(b) the aggregated proof has size $\mathcal{O}(\log n)$, and (c) verification requires $\mathcal{O}(\log n)$ cryptographic operations, and $\mathcal{O}(n)$ field operations.

Our protocol offers asymptotic and concrete improvements over prior approaches that aggregate proofs via *recursive composition*. In more detail, these approaches create (another) SNARK for the circuit that contains n copies of the Groth16 verifier circuit [BCTV14a; BCG+20]. This entails constructing arithmetic circuits for computing pairings, which is expensive (for example, computing a pairing on the BLS12-377 curve requires $\sim 15,000$ constraints [BCG+20]). In contrast, our protocol "natively" works with pairing-based languages. This results in the following efficiency savings: (a) our protocol does not have to reason about arithmetic circuits for computing pairings, (b) our protocol does not have to compute FFTs, which require time $\mathcal{O}(n \log n)$, and (c) our protocol does not require special cycles or chains of curves [BCTV14a; BCG+20]. Put together, these savings allow us to aggregate proofs over $\sim 1400\times$ faster than the recursive approach. Furthermore, our protocol requires the verifier to only perform $\mathcal{O}(n)$ field operations, as opposed to $\mathcal{O}(n)$ cryptographic operations for the recursive approach.

Low-memory SNARKs for machine computations. We leverage our aggregation protocol to construct a *low-memory SNARK* for (non-deterministic) machine computations. In more detail, if for a machine M, checking an execution transcript requires space S and time T, then our SNARK prover takes space $\tilde{\mathcal{O}}(S + T)$ to produce a proof for that execution. In comparison, constructing a monolithic proof for the entire computation at once requires space $\tilde{\mathcal{O}}(S \cdot T)$, whereas the only other solution for constructing low-memory SNARKs for machine computations requires recursive composition of proofs [BCCT13], which is concretely expensive.

Summary of contributions.

- We provide a unifying generalization of inner product arguments, identifying and formalizing the appropriate *doubly-homomorphic* commitment property.
- We prove security of the non-interactive Fiat-Shamir transform of this protocol, implying security for the entire family of protocols.
- We provide a new set of inner product arguments for pairing-based languages that improve verifier efficiency from linear to logarithmic by introducing a trusted structured setup.
- We construct a new polynomial commitment scheme with constant-sized commitments, opening time square root in the degree and square root sized CRS. The opening verifier runs in logarithmic time and opening proofs are logarithmic in size.
- We design an aggregator for Groth16 [Gro16] pairing-based SNARKs that produces an aggregated proof of logarithmic size. We apply our aggregator to construct a low-memory SNARK for machine computations *without relying on recursive composition*.
- We implement a set of Rust libraries that realize our inner product argument protocols and applications from modular and generic components. We

polynomial commitment	communication complexity				transparent setup	time complexity		
	CRS	commitments	openings	$d = 2^{20}$		Commit	Open	Verify
Kate et al.[KZG10]	d $\mathbb{G}_1$	1 $\mathbb{G}_1$	1 $\mathbb{G}_1$	96b	no	d $\mathbb{G}_1$	d $\mathbb{G}_1$	1 $P, \mathbb{G}_1$
Bulletproofs [BBB+18]	d $\mathbb{G}_1$	1 $\mathbb{G}_1$	$\log(d)$ $\mathbb{G}_1$	1.3 KB	yes	d $\mathbb{G}_1$	d $\mathbb{G}_1$	d $\mathbb{G}_1$
Hyrax [WTS+18]	$\sqrt{d}$ $\mathbb{G}_1$	$\sqrt{d}$ $\mathbb{G}_1$	$\log(d)$ $\mathbb{G}_1$	33 KB	yes	d $\mathbb{G}_1$	$\sqrt{d}$ $\mathbb{G}_1$	$\sqrt{d}$ $\mathbb{G}_1$
DARKs [BFS20]	d $\mathbb{G}_U$	1 $\mathbb{G}_U$	$\log(d)$ $\mathbb{G}_U$	8.6 KB	yes	d $\mathbb{G}_U$	$d \log(d)$ $\mathbb{G}_U$	$\log(d)$ $\mathbb{G}_U$
Virgo [ZXZS20]	1	1 $\mathbb{H}$	$\log(d)^2$ $\mathbb{H}$	183 KB	yes	$d \log(d)$ $\mathbb{H}$	$d \log(d)$ $\mathbb{H}$	$\log(d)^2$ $\mathbb{H}$
Groth [Gro11]	$\sqrt[3]{d}$ $\mathbb{G}_2$	$\sqrt[3]{d}$ $\mathbb{G}_T$	$\sqrt[3]{d}$ $\mathbb{G}_1$	25 KB	yes	d $\mathbb{G}_1$	$\sqrt[3]{d}$ $\mathbb{G}_1$	$\sqrt[3]{d}$ P
This work	$\sqrt{d}$ $\mathbb{G}_2$	1 $\mathbb{G}_T$	$\log(d)$ $\mathbb{G}_T$	2.5 KB	no	d $\mathbb{G}_1$	$\sqrt{d}$ P	$\log(d)\mathbb{G}_T$

Table 1: Efficiency comparisons for polynomial commitment schemes. All numbers are given asymptotically. We use $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ to represent groups in a bilinear map, P to represent pairings, $\mathbb{G}_U$ to represent groups of unknown order, and $\mathbb{H}$ to represent hash functions. For simplicity we only specify the dominant costs e.g., if there are d $\mathbb{G}_1$ and d $\mathbb{G}_2$ group exponentiations we simple write d $\mathbb{G}_2$. Column 5 is the expected size of one commitment plus one opening proof at $d = 2^{20}$ over a BN256 curve.

evaluate our implementation, and find that our PC scheme is over 14× faster to open than a KZG commitment [KZG10] for polynomials of degree 10^6, and that our aggregation scheme aggregates over 1400× faster than the alternative 2-chain approach.

Related work. Lai, Malavolta, and Ronge [LMR19] introduced an inner product argument for pairing based languages. Their scheme runs over a transparent setup and is secure under the SXDH assumption. Their work improves on Groth and Sahai Proofs [GS08] which are a method to prove pairing-based languages under zero-knowledge without reducing to NP. Their proving costs are dominated by a linear number of pairings, their proof sizes are logarithmic and their verifier running costs are dominated by a linear number of group exponentiations. Our pairing based IPA's have much lower verification costs but we use a trusted setup. Our generalized IPA argument can be used to greatly simplify the security proofs for their Theorems 3.2, 4.1, 4.2 and 4.3, and we prove security of a non-interactive variant in the algebraic commitment model.

In Table 1, we compare the efficiency of various polynomial commitment schemes. [KZG10] introduced a pairing based polynomial commitment scheme with constant sized proofs. Their scheme is secure under an updatable setup in the algebraic group model. Groth [Gro11] designed a pairing based "batch product argument" secure under SXDH. This argument that can be seen as a form of polynomial commitment scheme and our two-tiered polynomial commitment techniques were inspired by this work. Under discrete-logarithm assumptions, Bayer and Groth designed a zero-knowledge proving system to show that a committed value is the correct evaluation of a known polynomial [BG13]. Both the prover and verifier need only compute a logarithmic number of group exponentiations, however verifier costs are linear in the degree of the polynomial. Wahby et al. proved that it is possible to use the inner product argument of Bulletproofs [BBB+18] to build a polynomial commitment scheme [WTS+18]. Bowe et al. [BGH19] argued that the inner product argument of Bulletproofs is

also highly aggregatable, to the point where aggregated proofs can be verified using a one off linear cost and an additional logarithmic factor per proof. Attema and Cramer [AC20] recently provided an orthogonal generalization of the inner product argument. They show that the inner product argument can be seen as a black box compression mechanism for sigma protocols and show that it can be used as a proof system for secret shared data.

Polynomial commitment schemes have also been constructed using Reed-Solomon codes [ZXZS20]. These commitments use highly efficient symmetric key primitives, however the protocols that use them require soundness boosting techniques that result in large constant overheads. Bünz et al.[BFS20] designed a polynomial commitment scheme in groups of unknown order such as RSA groups or class groups with efficient verifier time and small proof sizes. However, it requires super-linear commitment and prover time. Asymptotically, our scheme positions itself competitively among state-of-the-art PCs (see Table 1). In terms of concrete efficiency, the trusted setup scheme of Kate et al. [KZG10] allows for constant proof sizes and verifier time (versus our logarithmic results), whereas our protocol offers quadratic improvements to opening efficiency and the maximum degree polynomial supported by a SRS of a given size.

Prior aggregatable SNARKs have relied on efficiently expressing SNARK verifiers as as arithmetic circuits [BCTV14b; BCG+13]. For pairing based SNARKs this was achieved through the use of pairing-friendly cycles [BCTV14a] or two-chains [BCG+20]. Known cycles and two-chains for the 128-bit security level require roughly 768-bit curves, which are $\sim 10\times$ more expensive than the roughly 384-bit curves used when recursion is not necessary. Bowe et al. introduce a novel approach to recursive SNARKs that works with cycles of standard (non-pairing) curves [BGH19]. Bünz et al. [BCMS20] generalize and formalize this approach. Chiesa et al. build a post-quantum recursive SNARK [COS20]. For all of these approaches we expect to significantly improve on prover time because we do not rely on expensive NP reductions.

Subsequent work. Prior (full) versions of this work included an additional polynomial commitment construction based on GIPA that only requires an unstructured reference string. In this construction, the prover computes $\mathcal{O}(\sqrt{d})$ pairings and exponentiations, the opening proof consists of $\mathcal{O}(\log(d))$ group elements, and the verifier performs $\mathcal{O}(\sqrt{d})$ exponentiations for degree d univariate polynomials. Recent subsequent work [Lee20] introduced a new PC scheme (called Dory) that builds on, and improves upon, our unstructured-setup construction. The key improvement is that the verifier time of this scheme is $\mathcal{O}(\log d)$, which is achieved by cleverly switching the commitment key in every round of the GIPA protocol and folding the old commitment key into the committed vector. This is possible when GIPA is instantiated with a bilinear group as the key space of the commitment to one vector is the message space of the commitment to the other vector, and vice versa. It is therefore possible to combine keys and messages homomorphically. However, log-verification costs of Dory are concretely more costly than our log-verification structured-setup PC scheme ($\approx 6\times$): at $d = 2^{20}$,

Dory's opening proofs are 18KB and computed in 6 seconds, while our scheme has proofs of size 2.5KB computed in 1 second.

Further subsequent works have applied our inner product arguments to aggregate vector commitment opening proofs [SCP+21], construct incrementally-verifiable computation without recursion [TFBT21], and aggregate SNARKs in blockchain settings using existing trusted setups [GMN21].

2 Technical Overview

2.1 Generalized Inner Product Argument

The first contribution of our paper is a *generalized inner product argument* we denote GIPA. At a high level, our protocol generalizes the protocols of [BCC+16; BBB+18] as follows. The protocols of [BCC+16; BBB+18] enable proving the correctness of inner products of scalar vectors committed via the Pedersen commitment scheme [Ped92]. Our protocol generalizes their techniques to enable proving the correct computation of a large class of inner products between vectors of group and/or field elements committed to using (possibly distinct) *doubly homomorphic commitments*. We explain in more detail below.

Starting point: inner product arguments. The inner product argument (IPA) by [BCC+16] enables a prover to convince a verifier that two committed vectors (using Pedersen vector commitments) have a publicly known inner product. It does this by elegantly rescaling the committed vectors to half their size in each round. In each round the verifier sends a random challenge, which the prover uses to take a linear combination of the right and left half of the committed vectors, and they both rescale the commitment keys accordingly.

After $\log_2 m$ such reduction step the prover simply opens the commitment and the verifier checks that the product relation holds. In Bulletproofs [BBB+18] the authors improve on the IPA by committing to the two vectors and the scalar in a single commitment, while maintaining the halving structure of the argument. This enables sending just two commitments per round.

We observe that the same argument structure works for a much wider class of commitment schemes. In particular we require only that the commitment scheme is binding and has the homomorphic properties that enable the rescaling step. This property is that the commitment scheme is doubly homomorphic, i.e., homomorphic over the messages and the commitment keys.

Doubly homomorphic commitments. At a high level, a doubly homomorphic commitment scheme is a binding commitment scheme $(\mathsf{Setup}, \mathsf{CM})$ where the key space $\mathcal{K}$, message space $\mathcal{M}$, and commitment space $\mathcal{C}$ form abelian groups of the same size such that $\mathsf{CM}((\mathsf{ck}_1 + \mathsf{ck}_2); (M_1 + M_2)) = \mathsf{CM}(\mathsf{ck}_1, M_1) + \mathsf{CM}(\mathsf{ck}_1; M_2) + \mathsf{CM}(\mathsf{ck}_2, M_1) + \mathsf{CM}(\mathsf{ck}_2, M_2)$.

The Pedersen commitment $\mathsf{CM}(\boldsymbol{g}, \boldsymbol{a}) \to \prod_i g_i^{a_i}$ is the doubly homomorphic commitment used in Bulletproofs. Lai, Malavolta, and Ronge [LMR19] used a doubly homomorphic commitment for bilinear groups where the committed vectors consist of group elements in a bilinear group: $\mathsf{CM}(\boldsymbol{v}, \boldsymbol{v'}, \boldsymbol{w}, \boldsymbol{w'}; \boldsymbol{A}, \boldsymbol{B}) \to \prod_i e(v_i, A_i) e(B_i, w_i), \prod_i e(v'_i, A_i) e(B_i, w'_i)$.

In some of our protocols the verifier already has access to one of the committed vectors. For instance, in the polynomial commitment scheme the verifier can simply compute the vector consisting of the monomials of the evaluation point. Such protocols are also captured by our abstraction since the identity commitment is doubly homomorphic. In the actual protocols, the prover doesn't send any scalings of these vectors, and the verifier simply computes them directly.

Inner products. Building on our generalization of commitment schemes that work for inner product arguments, GIPA also generalizes the types of inner products that can be proven between committed vectors. It can be used not only to show inner products between field elements, but for arbitrary inner product maps $\langle \cdot, \cdot \rangle$ that are bilinear, i.e., for which $\langle \boldsymbol{a}+\boldsymbol{b}, \boldsymbol{c}+\boldsymbol{d} \rangle = \langle \boldsymbol{a}, \boldsymbol{c} \rangle + \langle \boldsymbol{a}, \boldsymbol{d} \rangle + \langle \boldsymbol{b}, \boldsymbol{c} \rangle + \langle \boldsymbol{b}, \boldsymbol{d} \rangle$. It immediately follows our generalized argument works for bilinear pairings. We apply GIPA to three different inner products:

$$\begin{aligned}
&\langle \cdot, \cdot \rangle : \mathbb{G}_1^m \times \mathbb{G}_2^m \mapsto \mathbb{G}_T, \ \langle \boldsymbol{A}, \boldsymbol{B} \rangle = \textstyle\prod_{i=0}^{m-1} e(A_i, B_i) \\
&\langle \cdot, \cdot \rangle : \mathbb{G}_1^m \times \mathbb{F}^m \mapsto \mathbb{G}_1, \ \ \langle \boldsymbol{A}, \boldsymbol{b} \rangle = \textstyle\prod_{i=0}^{m-1} A_i^{b_i} \\
&\langle \cdot, \cdot \rangle : \mathbb{F}^m \times \mathbb{F}^m \mapsto \mathbb{F}, \quad \ \langle \boldsymbol{a}, \boldsymbol{b} \rangle = \textstyle\sum_{i=0}^{m-1} a_i b_i
\end{aligned}$$

We refer to the first inner product as the inner pairing product.

Security proof. We prove both the interactive and the non-interactive variant of GIPA to be knowledge-sound. The interactive security proof shows the $(k_1, \ldots, k_r)$-special soundness of GIPA protocols, which implies knowledge-soundness via a recent result of Attema and Cramer [AC20] (previous interactive security proofs showed only witness-extended emulation). In particular, we reduce the security of any GIPA instantiation to the binding of its commitment scheme.

We also prove knowledge-soundness of the non-interactive version of GIPA given by the Fiat-Shamir transform. It is known from folklore that applying the Fiat-Shamir transformation to a r-round interactive argument of knowledge with negligible soundness error yields a non-interactive argument of knowledge in the random oracle model where the extractor $\mathcal{E}$ runs in time $O(t^r)$ for an adversary that performs at most $t = \text{poly}(\lambda)$ random oracle queries. GIPA has $\log m$ rounds for $m = \text{poly}(\lambda)$ so this transformation yields a super-polynomial extractor. Given this, we directly prove the security of the non-interactive argument in the algebraic commitment model, a generalization of the algebraic group model [FKL18] for doubly-homomorphic commitments. In essence, whenever the prover outputs a commitment he must also give an opening to it with respect to some linear combination of commitment keys; the commitment schemes we consider can be shown to achieve this model in their own respective algebraic group models. Our security proof yields an efficient linear-time extractor and negligible knowledge-soundness. Given the generality of GIPA this also yields the first tight security analysis of non-interactive Bulletproofs [BCC+16; BBB+18] and the many related protocols [LMR19; BGH19; BCMS20].

TIPP and MIPP. Generically GIPA protocols have logarithmic communication but linear verifier time as computing the final commitment key takes a linear number of operations. We introduce TIPP, a logarithmic verifier variant for the

inner pairing product and MIPP for the multi-exponentiation inner product.[6] These schemes use universal and updatable structured references string as commitment keys. Their commitments are based on that of Abe et al. [AFG+16], where given a commitment key $(v_0, v_1) \in \mathbb{G}_2$ the commitment to $(A_0, A_1) \in \mathbb{G}_1^2$ is given by $e(A_0, v_0)e(A_1, v_1)$, and the KZG polynomial commitment [KZG10].

Instead of the verifier having to compute the verification key itself, we leverage a recent insight by Bowe, Grigg, and Hopwood [BGH19]. The final commitment key in GIPA can be viewed as a polynomial commitment to a degree m polynomial that can be verified in $\log m$ time. Using the structured setup we can outsource computing the commitment key to the prover. The verifier simply verifies that the commitment key was computed correctly. This amounts to evaluating the polynomial at a random point and checking a KZG [KZG10] polynomial commitment proof.

2.2 Applications

We show how to use instantiations of our generalized inner product argument to obtain interesting applications: a polynomial commitment scheme where computing evaluation proofs for polynomials of degree d requires only $O(\sqrt{d})$ cryptographic operations, and a protocol for aggregating n Groth16 SNARKs [Gro16] to produce an aggregate proof of size $O(\log n)$ and verifiable in time $O(\log n)$.

2.2.1 Polynomial commitment

Following Groth [Gro11] we use two-tiered homomorphic commitments: i.e. commitments to commitments. Suppose we wish to commit to a polynomial

$$f(X, Y) = f_0(Y) + f_1(Y)X + \ldots + f_{m-1}(Y)X^{m-1} = \textstyle\sum_{i=0}^{m-1} f_i(Y)X^i.$$

We can view this polynomial in matrix form

$$f(X, Y) = (1, X, X^2, \ldots, X^{m-1}) \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \ldots & a_{0,\ell-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \ldots & a_{1,\ell-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \ldots & a_{2,\ell-1} \\ \vdots & & & \ddots & \vdots \\ a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \ldots & a_{m-1,\ell-1} \end{pmatrix} \begin{pmatrix} 1 \\ Y \\ Y^2 \\ \ldots \\ Y^{\ell-1} \end{pmatrix}$$

One first computes commitments $A_0, \ldots, A_{m-1}$ to $f_0(Y), \ldots, f_{m-1}(Y)$. Next one commits to the commitments $A_0, \ldots, A_{m-1}$.

On receiving an opening challenge (x, y) the prover evaluates the first tier at x to obtain a commitment A to $f(x, Y)$. This is done using MIPP. The prover then opens the second tier commitment A at y in order to obtain $\nu = f(x, y)$. This is done using a KZG univariate polynomial commitment scheme [KZG10]. To apply our prover efficient polynomial commitment scheme to univariate polynomials, commit to $f(X, X^n)$ and open at (x, x^n).

Note that for $m \approx \ell \approx \sqrt{d}$ both the MIPP and the KZG commitment are only of square root size. This results in a square root reference string. In order

[6] We actually introduce two variants of MIPP: MIPP_u, where both the vectors are committed, and MIPP_k where the verifier already knows the exponent, but it's of a structured form.

to achieve square root prover time (in addition to evaluating the polynomial) the prover needs to store the $A_0, \ldots, A_{m-1}$ when committing to the polynomial. Using these values the resulting MIPP can be opened in $O(m) = O(\sqrt{d})$ time.

2.2.2 SNARK aggregation and proofs of machine computation

Pairing-based SNARKs such as Groth16 can be proven and verified using only algebraic operations (e.g., field operations, group operations and pairings). This means we can aggregate by applying TIPP to the Groth16 verifier equations, such that whenever TIPP verifies the aggregator must have seen some verifying proof. In particular, to aggregate n Groth 16 proofs $\{(A_i, B_i, C_i)\}_{i=1}^n \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$, one first computes commitments to the A_i, B_i, C_i values. Then the aggregator computes $\prod_{i=1}^n e(A_i, B_i)^{r^{2i}}$ and $\prod_{i=1}^n C_i^{r^{2i}}$ for some random value r and proves these are correct using our pairing based arguments. Finally the verifier checks that these values satisfy a randomized version of the Groth16 verifier equations. Overall the verifier only performs one field multiplication per instance and $O(\log(n))$ cryptographic operations for the TIPP protocol.

Low-memory SNARKs for machine computation. We make use of the SNARK aggregation protocol to build a low-memory SNARK that does not rely on recursive computation. Our approach proceeds by producing an individual Groth16 proof for each machine step and aggregating these individual proofs. The key observation is that due to the structure of the intermediate computation state, i.e., the output of one computation step becomes the input to the next, we can speed up the verifier's work from linear in the number of computation steps to logarithmic with an additional inner product commitment to the intermediate states. See Section 8 for details.

2.3 Implementation

We implement a set of Rust libraries that realize our inner product argument protocols and applications. Our libraries consists of a number of modular and generic components: (a) a generic interface for inner products, and instantiations for scalar products, multi-scalar multiplication, and pairing products; (b) a generic interface for doubly-homomorphic commitments, with instantiations for Pedersen commitments, the commitments of [AFG+16], and trivial identity commitments; (c) a generic implementation of GIPA that relies on the above interfaces, and instantiations for the various concrete inner products and corresponding commitments; and (d) implementations of our polynomial commitment scheme and our aggregation scheme for Groth16 proofs. See Sections 6 and 7 for evaluation details.

3 Notation

We denote by $[n]$ the set $\{1, \ldots, n\} \subseteq \mathbb{N}$. We use $\boldsymbol{a} = [a_i]_{i=1}^n$ as a short-hand for the vector $(a_1, \ldots, a_n)$, and $[\boldsymbol{a}_i]_{i=1}^n = [[a_{i,j}]_{j=1}^m]_{i=1}^n$ as a short-hand for the vector $(a_{1,1}, \ldots, a_{1,m}, \ldots, a_{n,1}, \ldots, a_{n,m})$; $|\boldsymbol{a}|$ denotes the number of entries in $\boldsymbol{a}$. We analogously define $\{a_i\}_{i=1}^n$ with respect to sets instead of vectors. If x is a binary string then $|x|$ denotes its bit length. For a finite set S, let $x \xleftarrow{\$} S$ denote that x

is an element sampled uniformly at random from S. We also write $x \xleftarrow{\$} \mathsf{A}()$ to denote when an algorithm A samples and uses randomness in the computation of x.

Inner pairing product notation. We introduce some special notation related to our inner pairing product argument, some of which is borrowed from the Pedersen inner product introduced in [BBB+18]. We write group operations as multiplication. For a scalar $x \in \mathbb{F}$ and vector $\boldsymbol{A} \in \mathbb{G}^n$, we let $\boldsymbol{A}^x = (A_1^x, \ldots, A_n^x) \in \mathbb{G}^n$, and for a vector $\boldsymbol{x} = (x_0, \ldots, x_{m-1}) \in \mathbb{F}^n$ we let $\boldsymbol{A}^{\boldsymbol{x}} = (A_0^{x_0}, \ldots, A_{m-1}^{x_{m-1}})$. For a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$ and pair of source group vectors $\boldsymbol{A} \in \mathbb{G}_1^n,\ \boldsymbol{B} \in \mathbb{G}_2^n$ we define $\boldsymbol{A} * \boldsymbol{B} = \prod_{i=1}^n e(A_i, B_i)$. For two vectors $\boldsymbol{A}, \boldsymbol{A'} \in \mathbb{G}^n$ we let $\boldsymbol{A} \circ \boldsymbol{A'} = (A_0 A'_0, \ldots, A_{m-1} A'_{m-1})$.

Let $\boldsymbol{A} \| \boldsymbol{A'} = (A_0, \ldots, A_{n-1}, A'_0, \ldots, A'_{m-1})$ be the concatenation of two vectors $\boldsymbol{A} \in \mathbb{G}^n$ and $\boldsymbol{A'} \in \mathbb{G}^m$. To denote slices of vectors given $\boldsymbol{A} \in \mathbb{G}^n_{\cdot}$ and $0 \leq \ell < n-1$ we write $\boldsymbol{A}_{[:\ell]} = (A_0, \ldots, A_{\ell-1}) \in \mathbb{G}^\ell$ and $\boldsymbol{A}_{[\ell:]} = (A_\ell, \ldots, A_{n-1}) \in \mathbb{G}^{n-\ell}$.

Languages and relations. We write $\{(\mathbb{x}) : p(\mathbb{x})\}$ to describe a polynomial-time language $\mathcal{L} \subseteq \{0,1\}^*$ decided by the polynomial-time predicate $p(\cdot)$. We write $\{(\mathbb{x}; \mathbb{w}) : p(\mathbb{x}, \mathbb{w})\}$ to describe a NP relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ between instances $\mathbb{x}$ and witnesses $\mathbb{w}$ decided by the polynomial-time predicate $p(\cdot, \cdot)$.

Security notions. We denote by $\lambda \in \mathbb{N}$ a security parameter. When we state that $n \in \mathbb{N}$ for some variable n, we implicitly assume that $n = \mathrm{poly}(\lambda)$. We denote by $\mathrm{negl}(\lambda)$ an unspecified function that is *negligible* in λ (namely, a function that vanishes faster than the inverse of any polynomial in λ). When a function can be expressed in the form $1 - \mathrm{negl}(\lambda)$, we say that it is *overwhelming* in λ. When we say that algorithm $\mathcal{A}$ is an *efficient* we mean that $\mathcal{A}$ is a family $\{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of non-uniform polynomial-size circuits. If the algorithm consists of multiple circuit families $\mathcal{A}_1, \ldots, \mathcal{A}_n$, then we write $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$.

Arguments of knowledge and Commitments. We use several standard notions in this paper such as interactive arguments of knowledge and commitments. For completeness, we include their definitions in the full version [BMM+19].

4 Generalized Inner Product Argument (GIPA)

We now generalize the inner product argument (IPA) from [BCC+16; BBB+18] to work for all "doubly homomorphic" inner product commitments. The generalized inner product argument (GIPA) protocol is described with respect to a doubly homomorphic inner product commitment and an inner product map defined over its message space. All of the inner pairing product arguments in this paper as well as the discrete-log inner product argument from [BCC+16; BBB+18] can be described as instantiations of GIPA, sometimes with non-black-box optimizations that do not work generally. The generalized version enables us to simplify the proof of security of the specific instantiations presented in the rest of the paper and provides a "compiler" that lets the reader plug in their own computationally binding "inner product commitment" to obtain a new inner product argument (of knowledge).

Protocol intuition. The protocol works by reducing the instance from size m to $m/2$ each round. As an intuition, we will show how to reduce an instance with

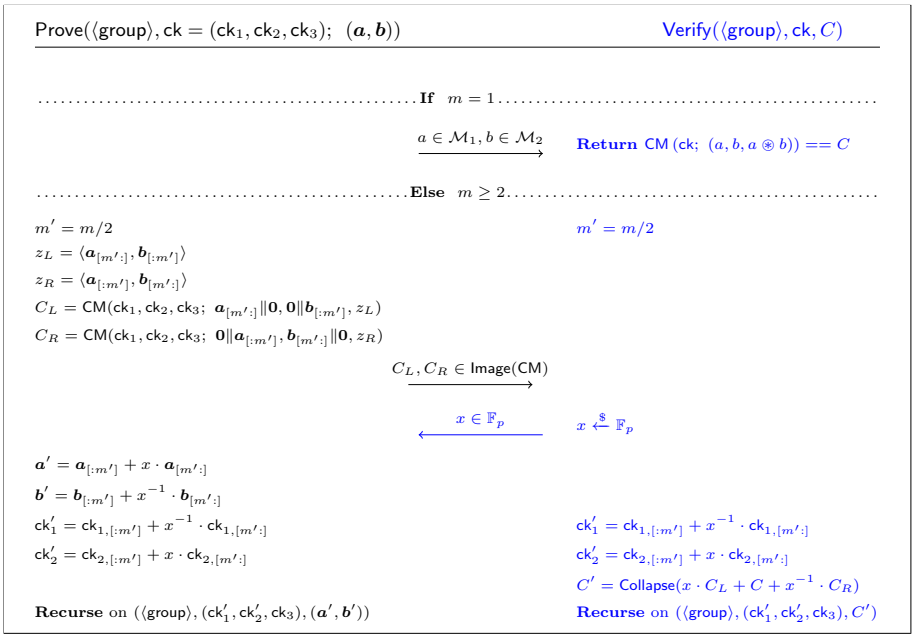

Fig. 1: Generalized inner product argument. Cases are based on the length m of the message (and correspondingly commitment key) vectors. Here, $\boldsymbol{0}$ is the vector containing m' sequential group identity elements for the appropriate group.

2 expensive mappings $\circledast$ to an instance with just a single $\circledast$. Given a_1, a_2, b_1, b_2 a prover wants to convince a verifier that $(a_1 \circledast b_1) + (a_2 \circledast b_2) = c$ for an expensive map $\circledast$. To do this the prover sends cross terms $l = a_1 \circledast b_2$ and $r = a_2 \circledast b_1$. The verifier then sends a challenge x. Note that for $a' = x \cdot a_1 + a_2$ and $b' = x^{-1} \cdot b_1 + b_2$ we have that $a' \circledast b' = x \cdot l + c + x^{-1} \cdot r$. Since the prover has to commit to the cross terms l and r before knowing x, if x is uniformly sampled from a sufficiently large space then checking this latter equation implies that $c = (a_1 \circledast b_1) + (a_2 \circledast b_2)$ with overwhelming probability.

GIPA extends this idea to work for committed vectors $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{b}_1, \boldsymbol{b}_2$. It relies on *doubly* homomorphic commitments with a commitment key ck where $\mathsf{CM}(\mathsf{ck}, \boldsymbol{a}) = \mathsf{CM}(x^{-1} \cdot \mathsf{ck}, x \cdot \boldsymbol{a})$.

4.1 Doubly homomorphic commitments

We can apply GIPA over any commitment scheme which is "doubly-homomorphic." For example, consider the Pedersen commitment scheme:

$$\frac{\mathsf{Setup}(1^\lambda) \to \mathsf{ck}}{\text{Return } (g_1, \ldots, g_m) \xleftarrow{\$} \mathbb{G}} \qquad \frac{\mathsf{CM}(\mathsf{ck}, \boldsymbol{a}) \to c}{\text{Return } g_1^{a_1} \cdots g_m^{a_m}}$$

This scheme allows us to commit to elements in the message space $\mathcal{M} = \mathbb{F}_p^m$ under commitment keys in the key space $\mathcal{K} = \mathbb{G}^m$ for a group $\mathbb{G}$ of prime order p. We denote the key space (i.e., the image of the setup algorithm) by $\mathcal{K}$. The commitment space is additively homomorphic because for all $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{M}$ and $\boldsymbol{g} \in \mathcal{K}$ we have that $\boldsymbol{g}^{\boldsymbol{a}} \cdot \boldsymbol{g}^{\boldsymbol{b}} = \boldsymbol{g}^{\boldsymbol{a}+\boldsymbol{b}}$. The key space is also homomorphic because

for all $\boldsymbol{g}, \boldsymbol{w} \in \mathcal{K}$ and $\boldsymbol{a} \in \mathcal{M}$ we have that $\boldsymbol{g}^{\boldsymbol{a}} \cdot \boldsymbol{w}^{\boldsymbol{a}} = (\boldsymbol{g} \circ \boldsymbol{w})^{\boldsymbol{a}}$. Thus, we consider the Pedersen commitment scheme to be *doubly-homomorphic* (i.e., homomorphic in both the commitment space and the key space).

Definition 1 (Doubly homomorphic commitment scheme). *A commitment scheme* $(\mathsf{Setup}, \mathsf{CM})$ *is doubly homomorphic if* $(\mathcal{K}, +)$, $(\mathcal{M}, +)$ *and* $(\mathsf{Image}(\mathsf{CM}), +)$ *define abelian groups such that for all* $\mathsf{ck}, \mathsf{ck}' \in \mathcal{K}$ *and* $M, M' \in \mathcal{M}$ *it holds that*

1. $\mathsf{CM}(\mathsf{ck}; M) + \mathsf{CM}(\mathsf{ck}; M') = \mathsf{CM}(\mathsf{ck}; M_1 + M')$
2. $\mathsf{CM}(\mathsf{ck}; M) + \mathsf{CM}(\mathsf{ck}'; M) = \mathsf{CM}(\mathsf{ck} + \mathsf{ck}'; M)$

Observe that if CM is doubly homomorphic then for all $x \in \mathbb{Z}_p$ it holds that $\mathsf{CM}(x \cdot \mathsf{ck}; M) = \mathsf{CM}(\mathsf{ck}; x \cdot M)$.

4.2 Inner Product

We consider inner products as bilinear maps from two equal-dimension vector spaces over two groups to a third group.

Definition 2 (Inner product map). *A map* $\circledast : \mathcal{M}_1 \times \mathcal{M}_2 \to \mathcal{M}_3$ *from two groups of prime order* p *to a third group of order* p *is an inner product map if for all* $a, b \in \mathcal{M}_1$ *and* $c, d \in \mathcal{M}_2$ *we have that*

$$(a + b) \circledast (c + d) = a \circledast c + a \circledast d + b \circledast c + b \circledast d$$

Given an inner product $\circledast$ *between groups we define the inner product between vector spaces* $\langle , \rangle : \mathcal{M}_1^m \times \mathcal{M}_2^m \to \mathcal{M}_3$ *to be* $\langle \boldsymbol{a}, \boldsymbol{b} \rangle := \sum_{i=1}^{m} a_i \circledast b_i$

We use three different inner products in this paper. For the Pedersen commitment described above we have that $\circledast$ is multiplication between elements in $\mathbb{F}_p$ and $\langle , \rangle$ is the dot product. In TIPP we have that $\circledast : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and $A \circledast B = e(A, B)$. In this case we refer to the resulting protocols as *inner pairing product* arguments. In MIPP we use the inner product $\circledast : \mathbb{G} \times \mathbb{F} \to \mathbb{G}$ and $A \circledast b = A^b$, a multiexponentiation inner product.

Inner product commitment. We further define an inner product commitment which consists of a doubly homomorphic commitment with a message space that is the Cartesian product of three message subspaces and an inner product that maps the first two message subspaces to the third. For GIPA the committed vectors and commitment keys halve in every round. If the commitments are constant sized, we can add commitments of different length. If not, we need to assume that the commitment key has a collapsing property such that additions of commitments are still well defined: Concretely we require that there exists a collapsing function $\mathsf{Collapse}$ to reduce the size of commitments with repeated entries. For example consider a commitment scheme with commitment key $[g_1, g_2, g_3, g_4] \in \mathbb{G}^4$ that commits a message vector with repeated entries, $[a_1, a_2, a_1, a_2] \in \mathbb{F}^4$ as $[g_1^{a_1}, g_2^{a_2}, g_3^{a_1}, g_4^{a_2}]$. Then, we can define a collapsing function that outputs the shorter commitment $[(g_1 g_3)^{a_1}, (g_2 g_4)^{a_2}]$ under a compressed commitment key $[g_1 g_3, g_2 g_4] \in \mathbb{G}^2$.

Definition 3 (Inner product commitment). *Let* $(\mathsf{Setup}, \mathsf{CM})$ *be a doubly homomorphic commitment with message space* $\mathcal{M} = \mathcal{M}_1^m \times \mathcal{M}_2^m \times \mathcal{M}_3$ *and key space* $\mathcal{K} = \mathcal{K}_1^m \times \mathcal{K}_2^m \times \mathcal{K}_3$ *defined for all* $m \in [2^j]_{j\in\mathbb{N}}$*, where* $|\mathcal{M}_i| = |\mathcal{K}_i| = p$ *is prime for* $i \in [3]$*. Let* $\circledast : \mathcal{M}_1 \times \mathcal{M}_2 \to \mathcal{M}_3$*. We call* $((\mathsf{Setup}, \mathsf{CM}), \circledast)$ *an inner product commitment if there exists an efficient deterministic function* $\mathsf{Collapse}$ *such that for all* $m \in [2^j]_{j\in\mathbb{N}}$*,* $M \in \mathcal{M}$*, and* $\mathsf{ck}, \mathsf{ck}' \in \mathcal{K}$ *such that* $\mathsf{ck}_3 = \mathsf{ck}'_3$ *it holds as*

$$\mathsf{Collapse}\left(\mathsf{CM}\left(\begin{array}{l|l}\mathsf{ck}_1 \| \mathsf{ck}'_1 & M_1 \| M_1 \\ \mathsf{ck}_2 \| \mathsf{ck}'_2 & M_2 \| M_2 \\ \mathsf{ck}_3 & M_3\end{array}\right)\right) = \mathsf{CM}\left(\begin{array}{l|l}\mathsf{ck}_1 + \mathsf{ck}'_1 & M_1 \\ \mathsf{ck}_2 + \mathsf{ck}'_2 & M_2 \\ \mathsf{ck}_3 & M_3\end{array}\right) \quad .$$

We refer to the requirement above as the collapsing property.

Let $((\mathsf{Setup}, \mathsf{CM}), \circledast)$ be a binding inner product commitment as defined above. In Fig. 1 we present a generalized inner product argument defined for all $m \in [2^j]_{j\in\mathbb{N}}$. We prove that this protocol is an argument (resp., proof) of knowledge when instantiated with a computationally (resp., statistically) binding inner product commitment. The proof of the following theorem is presented in the full version [BMM+19].

Theorem 1 (GIPA knowledge-soundness). *If* $((\mathsf{Setup}, \mathsf{CM}), \circledast)$ *is a computationally (resp., perfectly) binding inner product commitment, then* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$*, where* CM *and* $\circledast$ *instantiate the* Prove *and* Verify *algorithms presented in Fig. 1, has perfect completeness and computational (resp., statistical) knowledge-soundness for the relation*

$$\mathcal{R}_{\mathsf{IPA}} = \left\{ \begin{array}{r} \left(\mathsf{ck} \in \mathcal{K}_1^m \times \mathcal{K}_2^m \times \mathcal{K}_3 \; C \in \mathsf{Image}(\mathsf{CM}); \boldsymbol{a} \in \mathcal{M}_1^m, \boldsymbol{b} \in \mathcal{M}_2^m\right) : \\ C = \mathsf{CM}\left(\mathsf{ck};\ (\boldsymbol{a}, \boldsymbol{b}, \langle \boldsymbol{a}, \boldsymbol{b}\rangle)\right) \end{array} \right\} .$$

Non-interactive argument. In order to turn the public-coin interactive argument into a non-interactive proof we rely on the Fiat–Shamir heuristic. This results in all challenges being generated from a cryptographic hash function instead of by a verifier. The proof for the following theorem is presented in the full version [BMM+19].

Theorem 2. *If* $((\mathsf{Setup}, \mathsf{CM}), \circledast)$ *is a computationally (resp., perfectly) binding inner product commitment then in the algebraic group model and modeling* Hash *as a random oracle* $\mathsf{FS}(\mathsf{GIPA})$ *is a non-interactive argument of knowledge against an efficient* t*-query adversary in the random oracle model.*

Efficiency. Let m be a power of 2 and $\ell = \log_2 m$, the number of rounds in the GIPA protocol. The prover communication consists of 2ℓ commitments, 1 $\mathcal{M}_1$ element, and 1 $\mathcal{M}_2$ element. When the commitment scheme used is constant-sized, an instantiation of GIPA produces log-size proof. The prover makes 2 commitments to $(m+1)$-element messages in the first round, 2 commitments to $(m/2+1)$-element messages in the second, and 2 commitments to $(m/2^{i-1}+1)$-element messages in the i-th. It holds that $2 \cdot \sum_{i=1}^{\ell} \left(\frac{m}{2^{i-1}} + 1\right) = 4m + 2\ell - 4 \approx 4m$. So we say the prover commits to a total of $4m$ elements. Before computing

these commitments, however, the prover first must compute the z_L and z_R inner products, similarly requiring $2m$ invocations of $\circledast$ on $4m$ elements. Upon receiving the 2 commitments sent each round, the verifier uses them along with the challenge x_i it sampled that round to compute C', requiring 2ℓ multiplications in $\mathsf{Image}(\mathsf{CM})$.

The prover and verifier each compute ck' in each round, requiring $2m$ multiplications in $\mathcal{K}$. Some extensions of the GIPA protocol we'll introduce later use trusted setups to produced structured commitment keys. In these protocols, the verifier doesn't compute ck' themself in each round, but instead is sent the final rescaling $\mathsf{ck} \in \mathcal{K}_1 \times \mathcal{K}_2 \times \mathcal{K}_3$ that can be seen as a polynomial commitment in the verifiers challenges because of how the commitment key was structured. The verifier asks for an opening at a random point, which they can check with a small constant number of multiplications and pairings, and $O(\ell)$ field operations. This technique achieves a log-time verifier.

The prover alone computes $\boldsymbol{a}'$ and $\boldsymbol{b}'$, requiring m multiplications in each of $\mathcal{M}_1$ and $\mathcal{M}_2$. In some instantiations of GIPA, one or both of the vectors in $\mathcal{M}_1$ and $\mathcal{M}_2$ are included in full in the public input (i.e., the commitment performs the identity map on these inputs). In this case the verifier computes $\boldsymbol{a}'$ and/or $\boldsymbol{b}'$ themself.

4.3 Instantiation

GIPA can be instantiated with different commitments and inner product maps. In Bulletproofs [BBB+18] it is instantiated with the generalized Pedersen commitment defined above, where $\mathcal{K} = \mathbb{G}^m \times \mathbb{G}^m \times \mathbb{G}^m$, $\mathcal{M} = \mathbb{F}_p^m \times \mathbb{F}_p^m \times \mathbb{F}_p$, and $\circledast$ is the field addition operation. The reader can verify the commitment is a binding, doubly-homomorphic commitment scheme if the DL assumption holds for $\mathbb{G}$.

As a second example, in [LMR19] GIPA is instantiated for the inner pairing product $a \circledast b \equiv e(a, b)$ using the public-coin setup commitment scheme

$$\mathsf{CM}((\boldsymbol{v}, \boldsymbol{w}, \mathbf{1});\ (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{A} * \boldsymbol{B})) = (\boldsymbol{A} * \boldsymbol{v}, \boldsymbol{w} * \boldsymbol{B}, \boldsymbol{A} * \boldsymbol{B})\ .$$

Parts of the commitment may be computable directly from inputs to the verifier. For efficiency reasons the prover would not have to transmit that part of the commitment. We can formulate instantiations of GIPA for the inner pairing product map and the identity commitment scheme, which is perfectly (and thus statistically) binding.

An improvement on [LMR19]. GIPA also directly yields an improvement to the protocol presented in [LMR19] for proving knowledge of committed vectors of source group elements such that their inner pairing product is a public target group element. Replacing Lai et al.'s commitment scheme with [AFG+16] results in a 2 times faster prover and verifier for the relation while retaining the same proof size and assumptions.

5 Log-time verifier inner pairing product arguments

We present three inner product protocols that build on GIPA with the use of a trusted setup. Informally, these protocols prove the following relations:

(1) TIPP: An inner pairing product argument that proves $Z \in \mathbb{G}_T$ is the inner pairing product between committed vectors $\boldsymbol{A} \in \mathbb{G}_1^m$ and $\boldsymbol{B} \in \mathbb{G}_2^m$.

(2) MIPP_u: An unknown-exponent multiexponentiation inner product argument that proves $U \in \mathbb{G}_1$ is the multiexponentiation product between committed vectors $\boldsymbol{A} \in \mathbb{G}_1^m$ and $\boldsymbol{b} \in \mathbb{F}^m$.

(3) MIPP_k: A known-exponent multiexponentiation inner product argument that proves $U \in \mathbb{G}_1$ is the multiexponentiation inner product between a committed vector $\boldsymbol{A} \in \mathbb{G}_1^m$ and an uncommitted vector $\boldsymbol{b} = [b^i]_{i=0}^{m-1}$ for $b \in \mathbb{F}$.

Our arguments achieve log-time verification by building on a recent observation about inner product arguments by Bowe, Grigg, and Hopwood [BGH19]. A specially structured commitment scheme allows the prover to send the final commitment key and a succinct proof (as a KZG polynomial opening) of its correctness, which is verified via a log-time evaluation of the polynomial and two pairings.

5.1 Inner product commitments with structured setup

We construct inner product commitments for our arguments that are structured-key variants of the pairing-based commitment for group elements introduced by Abe et al. in [AFG+16] and of the Pedersen commitment for field elements [Ped92]. The setup algorithms for the inner product arguments are input a security parameter λ and a max instance size $m \in \{2^n\}_{n \in \mathbb{Z}^+}$. A type 3 bilinear group description $\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}_3(1^\lambda)$ is sampled. The structured setup proceeds by sampling random trapdoor elements $\alpha, \beta \xleftarrow{\$} \mathbb{F}$, and constructing the prover and verifier keys (SRS) as follows for generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$:

$$(\langle \mathsf{group} \rangle, \mathsf{pk} = (\left[g^{\alpha^i}\right]_{i=0}^{2m-2}, \left[h^{\beta^i}\right]_{i=0}^{2m-2}), \mathsf{vk} = (g^\beta, h^\alpha)) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, m)$$

The inner product commitment keys are derived by taking the even powers from the prover SRS as $\boldsymbol{w} = \left[g^{\alpha^{2i}}\right]_{i=0}^{m-1}$ and $\boldsymbol{v} = \left[h^{\beta^{2i}}\right]_{i=0}^{m-1}$. They are used as keys for the following inner product commitments. Observe that the vector commitment components of these inner product commitments are simply the structured-key variants of [AFG+16] and [Ped92]. The inner product values U, Z and the known vector $\boldsymbol{b}$ are committed to as the identity with keys initialized to 1.

(1) TIPP: $\mathsf{CM}_{\mathsf{TIPP}}((\boldsymbol{v}, \boldsymbol{w}, 1_{\mathbb{G}_T});\ \boldsymbol{A}, \boldsymbol{B}, Z) := (\boldsymbol{A} * \boldsymbol{v}, \boldsymbol{w} * \boldsymbol{B}, Z)$

(2) MIPP_u: $\mathsf{CM}_{\mathsf{MIPP}\text{-}u}((\boldsymbol{v}, \boldsymbol{w}, 1_{\mathbb{G}_T});\ \boldsymbol{A}, \boldsymbol{b}, U) := (\boldsymbol{A} * \boldsymbol{v}, \boldsymbol{w}^{\boldsymbol{b}}, U)$

(3) MIPP_k: $\mathsf{CM}_{\mathsf{MIPP}\text{-}k}((\boldsymbol{v}, \boldsymbol{1}_{\mathbb{F}}, 1_{\mathbb{G}_T});\ \boldsymbol{A}, \boldsymbol{b}, U) := (\boldsymbol{A} * \boldsymbol{v}, \boldsymbol{b}, U)$

It follows directly from the q-ASDBP assumption (see full version [BMM+19]) that these commitments are binding with respect to both the commitment key and the proving SRS. Note that the commitment keys only use even powers of trapdoor elements. This is to prevent an adversary from using (g^β, h^α) to find collisions in the commitment scheme—observe that $e(g, h^\alpha) \cdot e(g^\alpha, h^{-1}) = 1_{\mathbb{G}_T}$. The proving SRS requires all powers in order to compute the succinct KZG

polynomial opening proofs for the final commitment keys. This is the reason for our introduction of a new security assumption.

KZG polynomial commitments. As mentioned, we make use of the KZG polynomial commitment scheme [KZG10] which commits to polynomials of some max degree n. For polynomial $f(X) = \sum_{i=0}^{n-1} a_i X^i$ where $\boldsymbol{a} = [a_i]_{i=0}^{n-1}$, the commitment is computed with an analogously-structured trapdoor commitment key $\boldsymbol{ck} = \left[g^{\alpha^i}\right]_{i=0}^{n-1}$ as $\mathsf{KZG.CM}(\langle \mathsf{group} \rangle, \boldsymbol{ck}, \boldsymbol{a}) = \boldsymbol{ck}^{\boldsymbol{a}}$.

To open a point (x, y) where $y = f(x)$, KZG uses the polynomial remainder theorem which says $f(x) = y \Leftrightarrow \exists q(X) : f(X) - y = q(X)(X - x)$. The proof is just a KZG commitment to the quotient polynomial $q(X)$ where if $q(X)$ has coefficients $\boldsymbol{b}$, then $\mathsf{KZG.Open}(\langle \mathsf{group} \rangle, \boldsymbol{ck}, \boldsymbol{a}, x) = \boldsymbol{ck}^{\boldsymbol{b}}$. The verifier key consists of h^α, and the verifier runs $\mathsf{KZG.Verify}(\langle \mathsf{group} \rangle, h^\alpha, C, W, x, y)$ for commitment C and opening W and checks that $e(Cg^{-y}W^x, h) = e(W, h^\alpha)$.

5.2 Final commitment keys

Recall in GIPA, the verifier is required to perform a logarithmic amount of work to verify the final commitments C_L and C_R, using the challenges of each round of recursion to transform the commitments homomorphically. Assuming the commitments are of constant size this means that the verifier can efficiently check that these values are correct. However, the verifier must also perform a linear amount of work in rescaling the commitment key ck. Thus to achieve logarithmic verification time, when instantiating GIPA we need to avoid having the verifier rescale the commitment keys. We do this by outsourcing the work of rescaling the commitment keys to the prover.

The prover will compute the final commitment keys and then prove that they are well-formed, i.e., that they are exactly what the verifier would have computed in an unmodified instantiation of GIPA. Recall, we have structured our commitment keys as $\boldsymbol{w} = \left[g^{\alpha^{2i}}\right]_{i=0}^{m-1}$ and $\boldsymbol{v} = \left[h^{\beta^{2i}}\right]_{i=0}^{m-1}$. Without loss of generality, we will present the approach inspired by techniques from [BGH19] with respect to proving well-formedness of the final commitment key for $\boldsymbol{w} \in \mathbb{G}_1$; the techniques will apply analogously to $\boldsymbol{v} \in \mathbb{G}_2$.

In each round of GIPA, the commitment key is homomorphically rescaled by the round challenge x as:

$$\boldsymbol{w}' = \boldsymbol{w}_{[:m/2]} \circ \boldsymbol{w}_{[m/2:]}^{x} = \left[g^{\alpha^{2i}(1+x\alpha^{m+2i})}\right]_{i=0}^{m/2-1}.$$

Repeating this rescaling over $\ell = \log m$ recursive rounds with challenges $\boldsymbol{x} = [x_j]_{j=0}^{\ell}$, we claim (and show using an inductive argument in the full version [BMM+19]) that the final commitment key w takes the form:

$$w = g^{\prod_{j=0}^{\ell}\left(1+x_{\ell-j}\alpha^{2^{j+1}}\right)}.$$

We can then view this final commitment key w as a KZG polynomial commitment to the polynomial $f_w(X)$ defined below (and analogously v as the commitment

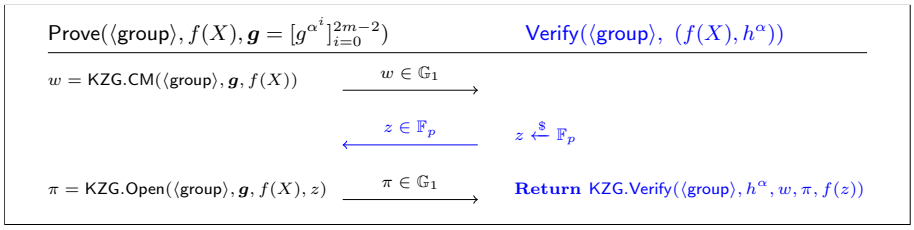

Fig. 2: The argument (of $\mathcal{R}_{ck}$) used to allow a prover to prove well-formedness of the final structured commitment key. The final commitment key w is interpreted as a KZG polynomial commitment that the prover must open at a random point. Shown for $w \in \mathbb{G}_1$, but holds analogously for $v \in \mathbb{G}_2$.

to $f_v(X)$):

$$f_w(X) = \prod_{j=0}^{\ell} \left(1 + x_{\ell-j} X^{2^{j+1}}\right) \qquad f_v(X) = \prod_{j=0}^{\ell} \left(1 + x_{\ell-j}^{-1} X^{2^{j+1}}\right)$$

Thus, to prove the well-formedness of the final commitment keys, the prover will prove the following relation $\mathcal{R}_{\mathsf{ck}}$ making direct use of the KZG polynomial opening proof. Again, without loss of generality, the relation is presented with respect to the final commitment key $w \in \mathbb{G}_1$.

$$\mathcal{R}_{\mathsf{ck}} = \left\{ \left(\langle\mathsf{group}\rangle, w \in \mathbb{G}_2, f(X), h^\alpha \; ; \boldsymbol{g} = [g^{\alpha^i}]_{i=0}^{2m-2} \right) : w = g^{f(\alpha)} \right\}$$

Our protocol for proving $\mathcal{R}_{\mathsf{ck}}$ is given in Fig. 2. At a high level, the verifier produces a challenge point $z \in \mathbb{F}$. If the prover can provide a valid KZG opening proof of $f_w(z)$ for commitment w, then the verifier accepts. We formally prove the security of this argument system in the full version [BMM+19]in the algebraic group model.

5.3 TIPP: Inner pairing product

The TIPP protocol allows a prover to show that for $T, U, Z \in \mathbb{G}_T$, they know $\boldsymbol{A} \in \mathbb{G}_1$ and $\boldsymbol{B} \in \mathbb{G}_2$ such that T and U are pairing commitments to $\boldsymbol{A}$ and $\boldsymbol{B}$, and Z is the inner pairing product $Z = \boldsymbol{A} * \boldsymbol{B}$.

This description is not quite general enough to cover the needs of our applications, such as batch verification. For example, to check that m pairing equations are simultaneously satisfied (i.e., that $[Z_i = e(A_i, B_i)]_{i=0}^{m-1}$), it is not sufficient to prove that $\Pi_{i=0}^{m-1} e(A_i, B_i) = \Pi_{i=0}^{m-1} Z_i$. Rather, instead you must prove the inner pairing product of a random linear combination defined by verifier challenge $r \in \mathbb{F}$: $\Pi_{i=0}^{m-1} e(A_i, B_i)^{r^i} = \Pi_{i=0}^{m-1} Z_i^{r^i}$.

We support this by modifying the TIPP relation to include the linear combination challenge r. For notational simplicity, we will use powers of two (matching that of our commitment keys) and define a public vector of field elements $\boldsymbol{r} = [r^{2i}]_{i=0}^{2m-2}$. The prover first commits to T and U, and then the verifier send a

random field element r. Thus, the TIPP relation is captured formally as follows:

$$\mathcal{R}_{\mathsf{TIPP}} = \left\{ \left(\begin{array}{c} \langle \mathsf{group} \rangle,\ g^{\beta} \in \mathbb{G}_1,\ h^{\alpha} \in \mathbb{G}_2,\ T, U, Z \in \mathbb{G}_T,\ r \in \mathbb{F}\ ; \\ \boldsymbol{w} = [g^{\alpha^{2i}}]_{i=0}^{m-1}, \boldsymbol{A} \in \mathbb{G}_1^m,\ \boldsymbol{v} = [h^{\beta^{2i}}]_{i=0}^{m-1}, \boldsymbol{B} \in \mathbb{G}_2^m, \\ \boldsymbol{r} = [r^{2i}]_{i=0}^{m-1} \in \mathbb{F}^m \\ T = \boldsymbol{A} * \boldsymbol{v}\ \wedge\ U = \boldsymbol{w} * \boldsymbol{B}\ \wedge\ Z = \boldsymbol{A}^{\boldsymbol{r}} * \boldsymbol{B} \end{array} \right) : \right\} .$$

Observe that if $T = \boldsymbol{A} * \boldsymbol{v}$ is a commitment to $\boldsymbol{A}$, then $T = \boldsymbol{A}^{\boldsymbol{r}} * \boldsymbol{v}^{\boldsymbol{r}^{-1}}$ is a commitment to $\boldsymbol{A}^{\boldsymbol{r}}$ under the commitment key $\boldsymbol{v}^{\boldsymbol{r}^{-1}}$. Intuitively, the argument proceeds by having the prover act as if it is working with a rescaled commitment key $\boldsymbol{v}' = \boldsymbol{v}^{\boldsymbol{r}^{-1}}$. TIPP runs the GIPA protocol with $\mathsf{CM}_{\mathsf{TIPP}}$ where the collapsing function is defined as the identity, $\mathsf{Collapse}_{\mathsf{id}}(C) = C$, over message $(\boldsymbol{A}^{\boldsymbol{r}}, \boldsymbol{B}, Z = \boldsymbol{A}^{\boldsymbol{r}} * \boldsymbol{B})$ and commitment key $(\boldsymbol{v}' = \boldsymbol{v}^{\boldsymbol{r}^{-1}}, \boldsymbol{w}, 1_{\mathbb{G}_T})$. Since all components of the commitment are compact, the identity collapsing function is sufficient.

Lastly, since the protocol is run over a rescaled commitment key $\boldsymbol{v}'$, the polynomial with which the prover proves the well-formedness of the final commitment key is also rescaled. It is as follows (derived in the full version [BMM+19]):

$$f_v'(X) = \prod_{j=0}^{\ell} \left(1 + x_{\ell-j}^{-1} (rX)^{2^{j+1}}\right)$$

A full description of the protocol is given in Figure 3. Because the protocol is public-coin, we can transform the interactive argument into a non-interactive proof using the Fiat-Shamir heuristic. In later sections, we may overload $\mathsf{TIPP.Prove}$ and $\mathsf{TIPP.Verify}$ as their non-interactive counterparts in which the prover will output a proof π that will be taken as an additional input by the verifier. This will be the case for MIPP_u and MIPP_k as well.

Communication and time complexity. Table 2 gives an overview of the communication and time complexity of our inner product protocols. Here we provide accounting for TIPP. The prover SRS consists of $2m$ elements in $\mathbb{G}_1$ and $2m$ elements in $\mathbb{G}_2$. The SRS consists only of monomials and therefore is updatable. The verifier's SRS consists of the group description, 1 elements in $\mathbb{G}_1$ and 1 elements in $\mathbb{G}_2$.

We calculate the prover computation. Our recursive argument requires $\log(m)$ rounds. The left and right commitments at each recursive round of GIPA require a total of $6m$ pairings to compute: $3m$ in the first round, $\frac{3m}{2}$ in the second round, and $\frac{3m}{2^{j-1}}$ in the j-th round. Homomorphically rescaling the commitment keys $(\boldsymbol{v}, \boldsymbol{w})$ and the messages $(\boldsymbol{A}, \boldsymbol{B})$ require a total of $2m$ exponentiations in each source group. The prover for the final commitment key costs $2m$ group exponentiations in each source group (for each commitment key). In total this sums to $6m$ pairings, $4m$ $\mathbb{G}_1$ exponentiations and $4m$ $\mathbb{G}_2$ exponentiations.

Regarding proof size, we have $6\log(m)$ $\mathbb{G}_T$ elements from the recursive argument, 1 $\mathbb{G}_1$ element and 1 $\mathbb{G}_2$ element from the final openings, and 2 $\mathbb{G}_1$ elements and 2 $\mathbb{G}_2$ elements from the final commitment key argument (i.e., w, v, and their proofs of correctness).

The verifier computes 7 pairings: 3 from the recursive argument and 4 from the final commitment key argument. Homomorphically rescaling the commitments in

$\mathsf{TIPP.Prove}(\langle\mathsf{group}\rangle, \mathsf{pk} = (\left[g^{\alpha^i}\right]_{i=0}^{2m-2}, \left[h^{\beta^i}\right]_{i=0}^{2m-2}), (T, U, Z, r), (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{w}, \boldsymbol{v}, \boldsymbol{r}))$
$\leftrightarrow \mathsf{TIPP.Verify}(\langle\mathsf{group}\rangle, \mathsf{vk} = (g^\beta, h^\alpha), (T, U, Z, r)) :$

1. Prover rescales $\boldsymbol{A}$ and $\boldsymbol{v}$ with respect to linear combination challenge r:

$$\boldsymbol{A}' = \boldsymbol{A}^{\boldsymbol{r}} \qquad \boldsymbol{v}' = \boldsymbol{v}^{\boldsymbol{r}^{-1}}.$$

Run GIPA:

2. Prover and verifier run GIPA with $\mathsf{CM}_{\mathsf{TIPP}}$ and $\mathsf{Collapse}_{\mathsf{id}}$ with some minor changes:

$$\mathsf{GIPA}_{\mathsf{CM\text{-}TIPP}}.\mathsf{Prove}(\langle\mathsf{group}\rangle, (\boldsymbol{v}', \boldsymbol{w}, 1_{\mathbb{G}_T}), (\boldsymbol{A}', \boldsymbol{B})) \leftrightarrow \mathsf{GIPA}_{\mathsf{CM\text{-}TIPP}}.\mathsf{Verify}(\langle\mathsf{group}\rangle, \cdot, (T, U, Z))$$

   (a) The verifier does not take as input a commitment key, and does not perform commitment key rescalings during GIPA execution. The verifier takes as output the final commitment C, the final message values (A, B), and the recursive round challenges $\boldsymbol{x} = [x_j]_{j=0}^{\log m}$.
   (b) The prover stores the recursive round challenges $\boldsymbol{x}$ and the final commitment keys $(v, w) = (\mathsf{ck}_1, \mathsf{ck}_2)$.
   (c) The prover sends the final commitment keys (v, w) to the verifier.

Prove well-formedness of final commitment keys:

3. Define the following polynomials for $\ell = \log m$:

$$f_w(X) = \prod_{j=0}^{\ell}\left(1 + x_{\ell-j}X^{2^{j+1}}\right) \qquad f'_v(X) = \prod_{j=0}^{\ell}\left(1 + x_{\ell-j}^{-1}(rX)^{2^{j+1}}\right)$$

4. Prover and verifier run the argument from Figure 2 for each final commitment key v and w:

$$\mathsf{CK.Prove}(\langle\mathsf{group}\rangle, f_w(X), \left[g^{\alpha^i}\right]_{i=0}^{2m-2}) \leftrightarrow \mathsf{CK.Verify}(\langle\mathsf{group}\rangle, (w, f_w(X), h^\alpha))$$

$$\mathsf{CK.Prove}(\langle\mathsf{group}\rangle, f'_v(X), \left[h^{\beta^i}\right]_{i=0}^{2m-2}) \leftrightarrow \mathsf{CK.Verify}(\langle\mathsf{group}\rangle, (v, f'_v(X), g^\beta))$$

5. Verifier returns 1 if the above arguments accept and if $\mathsf{CM}_{\mathsf{TIPP}}((v, w, 1_{\mathbb{G}_T}); (A, B, e(A, B))) == C$.

Fig. 3: TIPP argument of knowledge for inner pairing product between committed vectors.

	communication complexity		time complexity	
	$\lvert\mathrm{SRS}\rvert$	$\lvert\pi\rvert$	Prove	Verify
TIPP	$2m\ \mathbb{G}_1 + 2m\ \mathbb{G}_2$	$6\log m\ \mathbb{G}_T + 3\ \mathbb{G}_1 + 3\ \mathbb{G}_2$	$4m\ \mathbb{G}_1 + 4m\ \mathbb{G}_2 + 6m\ P\ 7\ P + 6\log m\ \mathbb{G}_T$	
MIPP_u	$m\ \mathbb{G}_1 + 2m\ \mathbb{G}_2$	$2\log m\ \mathbb{G}_T + 3\ \mathbb{G}_1 + 2\ \mathbb{G}_2 + 1\ \mathbb{F}$	$3m\ \mathbb{G}_1 + 3m\ \mathbb{G}_2 + 2m\ P\ 6\ P + 2\log m\ \mathbb{G}_T$	
MIPP_k	$2m\ \mathbb{G}_2$	$2\log m\ \mathbb{G}_T + 1\ \mathbb{G}_1 + 2\ \mathbb{G}_2$	$m\ \mathbb{G}_1 + 3m\ \mathbb{G}_2 + 2m\ P\ \ 4\ P + 2\log m\ \mathbb{G}_T + \log m\ \mathbb{F}$	

Table 2: Efficiency table for TIPP, MIPP_k, and MIPP_u. The verifier keys are succinct.

the recursive argument requires $6\log(m)$ exponentiations in $\mathbb{G}_T$. The verifier also computes $f(z)$ in the final commitment key argument which costs $2\ell = 2\log_2(m)$ field multiplications and additions.

Security. Here we prove soundness for TIPP in the algebraic group model.

Theorem 3 (Computational knowledge-soundness TIPP). *The protocol defined in Section 5.3 for the NP relation $\mathcal{R}_{\mathsf{TIPP}}$ has computational knowledge-soundness against algebraic adversaries under the q-ASDBP and $2q$-SDH assumptions.*

Proof. The commitment scheme $\mathsf{CM}((\boldsymbol{v}', \boldsymbol{w}, 1), (\boldsymbol{A}', \boldsymbol{B}, Z)) = (\boldsymbol{A}' * \boldsymbol{v}',\ \boldsymbol{w} * \boldsymbol{B}, Z) = (T, U, Z)$ is doubly homomorphic: the key space $\mathbb{G}_2^m \times \mathbb{G}_1^m \times \mathbb{F}$ is homomorphic under $\mathbb{G}_2$ multiplication, $\mathbb{G}_1$ multiplication, and $\mathbb{F}$ addition. The message

space $\mathbb{G}_1^m \times \mathbb{G}_2^m \times \mathbb{G}_T$ is homomorphic under the respective group multiplications. The commitment space $\mathbb{G}_T \times \mathbb{G}_T \times \mathbb{G}_T$ is homomorphic under $\mathbb{G}_T$ multiplication. All groups have prime order p for $p > 2^\lambda$. The commitment scheme is also binding by the q-ASDBP assumption. This means that the commitment scheme is an inner product commitment. Thus either the adversary convinces the verifier of incorrect w, v, or by Theorem 1 an adversary that breaks knowledge-soundness can extract a valid m-ASDBP instance. An algebraic adversary that convinces a verifier of incorrect w, v can extract a valid $2m$-SDH instance by the security of $\mathcal{R}_{\mathsf{ck}}$ (Equation 5.2). □

5.4 MIPP_u: Multiexponentiation with unknown field vector

In the MIPP_u protocol, a prover demonstrates knowledge for pairing commitment $T \in \mathbb{G}_T$ and KZG commitment $B \in \mathbb{G}_2$ of $\boldsymbol{A} \in \mathbb{G}_1^m$ as the opening of T and $\boldsymbol{b} \in \mathbb{F}^m$ as the opening of B where $U = \prod_{i=0}^{m-1} A_i^{r^{2i} b_i}$ for a public field element r. The public field element r, as in Section 5.3, allows the argument to be used for random linear combinations. The MIPP_u relation is captured formally as follows:

$$\mathcal{R}_{\mathsf{MIPP}\text{-}u} = \left\{ \left(\begin{array}{c} \langle \mathsf{group} \rangle,\ g^\beta \in \mathbb{G}_1,\ h^\alpha \in \mathbb{G}_2,\ T \in \mathbb{G}_T,\ B, U \in \mathbb{G}_1,\ r \in \mathbb{F}\ ; \\ \boldsymbol{w} = [g^{\alpha^{2i}}]_{i=0}^{m-1}, \boldsymbol{A} \in \mathbb{G}_1^m, \boldsymbol{v} = [h^{\beta^{2i}}]_{i=0}^{m-1}, \boldsymbol{b} \in \mathbb{F}^m, \\ \boldsymbol{r} = [r^{2i}]_{i=0}^{m-1} \in \mathbb{F}^m \\ T = \boldsymbol{A} * \boldsymbol{v} \ \wedge\ B = \boldsymbol{w}^{\boldsymbol{b}} \ \wedge\ U = \boldsymbol{A}^{\boldsymbol{r} \circ \boldsymbol{b}} \end{array} \right) : \right\} .$$

The MIPP_u argument proceeds analogously to TIPP if using the inner product commitment $\mathsf{CM}_{\mathsf{MIPP}\text{-}u}$ where k_U is initialized to $1_{\mathbb{G}_T}$:

$$\mathsf{CM}_{\mathsf{MIPP}\text{-}u}((\boldsymbol{v}, \boldsymbol{w}, k_U);\ \boldsymbol{A}, \boldsymbol{b}, U) := (\boldsymbol{A} * \boldsymbol{v}, \boldsymbol{w}^{\boldsymbol{b}}, k_U U)$$

However, we make a small optimization by replacing the above commitment scheme with a modified scheme $\mathsf{CM}'_{\mathsf{MIPP}\text{-}u}$ with a commitment size consisting only of one element in $\mathbb{G}_T$ (concretely ∼25% reduction in size). Recall, the proof includes a logarithmic number of commitments, so cutting the commitment size by 25% more or less cuts the proof size by the same proportion.

Using $\mathsf{CM}'_{\mathsf{MIPP}\text{-}u}$ adds two additional random group elements $\hat{h}_1, \hat{h}_2 \xleftarrow{\$} \mathbb{G}_2$ to the prover key and verifier key $(\mathsf{pk}, \mathsf{vk})$ during setup. After setting (T, B, U, r), the verifier samples values $(c_1, c_2) \xleftarrow{\$} \mathbb{F}$ and sends them to the prover. The prover and verifier then each set $\hat{h}'_1 = \hat{h}_1^{c_1}$ and $\hat{h}'_2 = \hat{h}_2^{c_2}$. The values $\hat{h}'_1$ and $\hat{h}'_2$ become part of the commitment key for the following inner product commitment:

$$\mathsf{CM}'_{\mathsf{MIPP}\text{-}u}((\boldsymbol{v}, \boldsymbol{w}, (\hat{h}'_1, \hat{h}'_2));\ \boldsymbol{A}, \boldsymbol{b}, U) := (\boldsymbol{A} || \boldsymbol{w}^{\boldsymbol{b}} || U) * (\boldsymbol{v} || \hat{h}'_1 || \hat{h}'_2)$$

The prover then proceeds analogously to TIPP. First, running GIPA with $\mathsf{CM}'_{\mathsf{MIPP}\text{-}u}$ with the identity collapsing function over message $(\boldsymbol{A}^{\boldsymbol{r}}, \boldsymbol{b}, U = \boldsymbol{A}^{\boldsymbol{r} \circ \boldsymbol{b}})$ and commitment key $(\boldsymbol{v}' = \boldsymbol{v}^{\boldsymbol{r}^{-1}}, \boldsymbol{w}, (\hat{h}'_1, \hat{h}'_2))$. The verifier runs with commitment $C = T \cdot e(B, \hat{h}'_1) \cdot e(U, \hat{h}'_2$. The final commitment keys w and v are proved with respect to the same polynomials $f_w(X)$ and $f'_v(X)$.

A full description of the protocol is given in the full version [BMM+19]. Soundness follows for algebraic adversaries from the q-ASDBP and the q-SDH assumptions and the algorithm is proven secure in the full version [BMM+19].

5.5 MIPP_k: Multiexponentiation with known field vector

In the MIPP_k protocol a prover demonstrates knowledge of $\boldsymbol{A} \in \mathbb{G}_1^m$ such that $\boldsymbol{A}$ commits to pairing commitment T under $\boldsymbol{v}$ and $U = \boldsymbol{A}^{\boldsymbol{b}}$ for a public vector $\boldsymbol{b} = [b^i]_{i=0}^{m-1}$ for $b \in \mathbb{F}$. The MIPP_k relation is captured formally as follows:

$$\mathcal{R}_{\mathsf{MIPP\text{-}}k} = \left\{ \left(\begin{array}{c} \langle \mathsf{group} \rangle,\ g^\beta \in \mathbb{G}_1,\ T \in \mathbb{G}_T,\ U \in \mathbb{G}_1,\ b \in \mathbb{F}\ ; \\ \boldsymbol{A} \in \mathbb{G}_1^m, \boldsymbol{v} = [h^{\beta^{2i}}]_{i=0}^{m-1}, \boldsymbol{b} \\ T = \boldsymbol{A} * \boldsymbol{v}\ \wedge\ U = \boldsymbol{A}^{\boldsymbol{b}}\ \wedge \boldsymbol{b} = [b^i]_{i=0}^{m-1} \end{array} \right) : \right\} .$$

For the known vector multiexponentiation inner product, we use an inner product commitment that commits to the vector $\boldsymbol{b}$ as itself using a key $\boldsymbol{k_b}$ initialized to $\mathbf{1}_{\mathbb{F}}$. Since the commitment is no longer compact, we use a collapsing function that collapses the vector by adding the first and second halves. This provides the required homomorphic properties of Definition 3.

$$\mathsf{CM}_{\mathsf{MIPP\text{-}}k}((\boldsymbol{v}, \boldsymbol{k_b}, 1_{\mathbb{G}_T});\ \boldsymbol{A}, \boldsymbol{b}, U) := (\boldsymbol{A} * \boldsymbol{v}, [k_{b,i} b_i]_{i=0}^{m-1}, U)$$
$$\mathsf{Collapse}_{\mathsf{MIPP\text{-}}k}(C = (C_A, \boldsymbol{C_b}, C_U)) = (C_A, [C_{b,i} + C_{b,(i+\frac{m}{2})}]_{i=0}^{\frac{m}{2}-1}, C_U)$$

If we were to run GIPA naively with this commitment, the proof size would be linear in the length of $\boldsymbol{b}$. However, we can use a similar to trick to how we calculate the final commitment keys (Section 5.2). Instead of sending the commitment to the rescaled message $\boldsymbol{b}$ at each recursive round, we observe that rescaling the structured vector $\boldsymbol{b}$ leads to a closed-form expression of the final b' message using recursive challenges $\boldsymbol{x} = [x_j]_{j=0}^{\log m} : b' = \prod_{j=0}^{\ell} \left(1 + x_{\ell-j}^{-1} b^{2^j}\right)$. This value b' can be computed in $\log m$ time by the verifier and allows for the prover to omit the commitment to $\boldsymbol{b}$, bringing the proof size back to logarithmic in m.

In addition, as in Section 5.4 for MIPP_u, we provide an optimized inner product commitment scheme $\mathsf{CM}'_{\mathsf{MIPP\text{-}}k}$ with commitment size equal to one element of $\mathbb{G}_T$ (when using the above trick to omit $\boldsymbol{b}$). The commitment $\mathsf{CM}'_{\mathsf{MIPP\text{-}}u}$ adds one additional random group element $\hat{h} \xleftarrow{\$} \mathbb{G}_2$ to the prover key and verifier key $(\mathsf{pk}, \mathsf{vk})$ during setup. After setting (T, U, b), the verifier samples value $c \xleftarrow{\$} \mathbb{F}$ and sends it to the prover. The prover and verifier then each set $\hat{h}' = \hat{h}^c$. The value $\hat{h}'$ becomes part of the commitment key for the following inner product commitment:

$$\mathsf{CM}'_{\mathsf{MIPP\text{-}}k}((\boldsymbol{v}, \boldsymbol{k_b}, \hat{h}');\ \boldsymbol{A}, \boldsymbol{b}, U) := ((\boldsymbol{A}||U) * (\boldsymbol{v}||\hat{h}'), [k_{b,i} b_i]_{i=0}^{m-1})$$
$$\mathsf{Collapse}'_{\mathsf{MIPP\text{-}}k}(C = (C_{A||U}, \boldsymbol{b} = [b^i]_{i=0}^{m-1})) = (C_{A||U}, [C_{b,i} + C_{b,(i+\frac{m}{2})}]_{i=0}^{\frac{m}{2}-1})$$

A full description of the protocol is given in the full version [BMM+19]. Soundness follows for algebraic adversaries from the q-ASDBP and the q-SDH assumptions and the algorithm is proven secure in the full version [BMM+19].

6 Log-time verifier polynomial commitments with square root SRS

In this section we introduce a polynomial commitment (PC) scheme with a square root sized SRS and opening time, and logarithmic proof sizes and verifier time. We use a two-tiered homomorphic commitment algorithm similar to the

one from [Gro11] but with structured keys. We first describe how our PC can be used for bivariate polynomials, and then present a simple way to use it for univariate polynomials as well. In the full version [BMM+19], we show how these polynomial commitments can be made hiding for zero-knowledge applications.

Two-tiered inner product commitment. We describe a two-tiered inner product commitment for bivariate polynomials. It is based on the [Gro11] two tiered commitment. We use the structured-key variant of the [AFG+16] commitment introduced in Section 5.1 to commit to the KZG commitments [KZG10]. A brief description of KZG commitments was also given in Section 5.1. We describe our polynomial commitment in Figure 4.

To commit to a polynomial $f(X,Y) = \sum_{j=0}^{m-1} f_j(Y)X^j$ given commitment key $ck = (\boldsymbol{g}, \boldsymbol{v}, \hat{h})$, the committer computes m KZG polynomial commitments $\boldsymbol{A} = [A_j]_{j=0}^{m-1}$ to y-polynomials $\boldsymbol{f} = [f_j(Y)]_{j=0}^{m-1}$ where say $f_j(Y)$ has coefficients $\boldsymbol{a}_j = [a_{i,j}]_{i=0}^{\ell-1}$: $A_j = \mathsf{KZG.CM}(\boldsymbol{g}, \boldsymbol{a}_j) = \boldsymbol{g}^{\boldsymbol{a}_j} = g^{\sum_{i=0}^{\ell-1} a_{i,j}\alpha^i}$. The committer then computes the pairing commitment [AFG+16] to the KZG commitments

$$T = \boldsymbol{A} * \boldsymbol{v} = \prod_{j=0}^{m-1} e(A_j, v_i) = \prod_{j=0}^{m-1} e(A_j, h^{\beta^{2j}}) \ .$$

Thus, $T = e(g,h)^{\sum_{i,j=0}^{\ell-1,m-1} a_{i,j}\alpha^i\beta^{2j}}$, and this commitment is binding under the q-ASDBP assumption and the q-SDH assumption.

Two-tiered opening. Our opening algorithm proves a commitment T to a polynomial $f(X,Y)$ evaluates to ν at a point $(x,y) \in \mathbb{F}^2$. We proceed in three steps. First the prover produces an opening for an outer tier partial evaluation $U = f(x,Y) = \prod_{i=0}^{m-1} A_i^{x^i}$ for a point $x \in \mathbb{F}$. Observe that U is a KZG commitment to the univariate polynomial $f(x,Y) = \sum_{j=0}^{\ell-1}(\sum_{i=0}^{m-1} a_{i,j}x^i)Y^j$. Second the prover produces a MIPP_k proof (see Section 5.5) that U is the inner product of the opening to T and the vector $\boldsymbol{x} = (1, x, \ldots, x^{m-1})$. Third the prover produce a KZG proof that ν is the evaluation of U at y. The prover returns U and the two proofs. The verifier simply checks the two proofs.

Theorem 4. *If there exists a bilinear group sampler* $\mathsf{SampleGrp_3}$ *that satisfies the* q-ASDBP *assumption in* $\mathbb{G}_2$ *and the* q-SDH *assumption, then the protocol in Fig. 4 is a polynomial commitment scheme with computational extractability against algebraic adversaries.*

Note that computing the partial opening U takes $m\ell$ $\mathbb{G}_1$ exponentiations if computing from scratch. Instead, if the KZG commitments to the y-polynomials $\boldsymbol{A}$ are given as input, U can be computed with only m $\mathbb{G}_1$ exponentiations. Thus, we pass $\boldsymbol{A}$, which was already computed during commitment, as auxiliary data to the opening algorithm to facilitate our square root degree opening time.

Supporting univariate polynomials. If we have a univariate polynomial, then we set $\ell m = d$ for d the degree of $f(X)$ and $f_i(Y) = a_{i\ell} + a_{i\ell+1}Y + \ldots + a_{(i+1)\ell-1}Y^{\ell-1} = \sum_{j=0}^{\ell-1} a_{i\ell+j}Y^j$. Observe now that $p(X,Y) = \sum_{i=0}^{m-1} f_i(Y)X^i$ is such that $p(X^\ell, X) = f(X)$ Thus we commit to $f(X)$ by committing to $p(X,Y)$.

$\underline{\mathsf{Setup}(1^\lambda, \ell, m):}$
$\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}_3(1^\lambda)$
$\hat{h} \xleftarrow{\$} \mathbb{G}_2; \quad \alpha, \beta \xleftarrow{\$} \mathbb{F}$
$\boldsymbol{g} \leftarrow [g^{\alpha^i}]_{i=0}^{\ell-1}$
$\boldsymbol{v} \leftarrow [h^{\beta^{2i}}]_{i=0}^{m-1}$
$\mathsf{ck} \leftarrow (\langle \mathsf{group} \rangle, \boldsymbol{g}, \boldsymbol{v}, \hat{h})$
$\mathsf{ek} \leftarrow (\langle \mathsf{group} \rangle, \boldsymbol{g}, [h^{\beta^i}]_{i=0}^{2m-2}, \hat{h})$
$\mathsf{vk} \leftarrow (\langle \mathsf{group} \rangle, g^\beta, h^\alpha, \hat{h})$
Return $(\mathsf{ck}, \mathsf{vk})$

$\underline{\mathsf{CM}(\mathsf{ck}, f(X,Y)):}$
$[A_j]_{j=0}^{m-1} \leftarrow \prod_{i=0}^{\ell-1} g_i^{a_{i,j}}$
$T \leftarrow \prod_{j=0}^{m-1} e(A_j, v_j)$
Return T

$\underline{\mathsf{Open}(\mathsf{ek}, T, (x,y), \nu, f(X,Y), [A_j]_{j=0}^{m-1})}$
$(\langle \mathsf{group} \rangle, \boldsymbol{g}, \mathsf{pk}, \hat{h}) \leftarrow \mathsf{ek}$
$U \leftarrow \prod_{j=0}^{m-1} A_j^{x^i}$
$\pi_1 \leftarrow \mathsf{MIPP}_k.\mathsf{Prove}(\langle \mathsf{group} \rangle, (\mathsf{pk}, \hat{h}), (T, U, x), (\boldsymbol{A}, \boldsymbol{v}, \boldsymbol{x}))$
$\pi_2 \leftarrow \mathsf{KZG.Open}(\langle \mathsf{group} \rangle, \boldsymbol{g}, f(x, Y), \nu)$
Return (U, π_1, π_2)

$\underline{\mathsf{Check}(\mathsf{vk}, (T, (x,y), \nu), (U, \pi_1, \pi_2))}$
$b_1 \leftarrow \mathsf{MIPP}_k.\mathsf{Verify}(\langle \mathsf{group} \rangle, (g^\beta, \hat{h}), (T, U, x), \pi_1))$
$b_2 \leftarrow \mathsf{KZG.Verify}(\langle \mathsf{group} \rangle, h^\alpha, U, \pi_2, y)$
Return $b_1 \wedge b_2$

Fig. 4: A two-tiered inner product commitment.

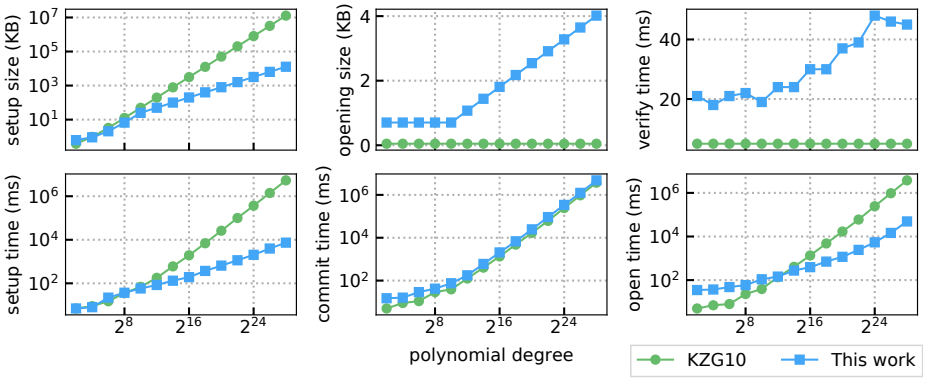

Fig. 5: Measured performance of the IPA polynomial commitment.

To evaluate $f(X)$ at x the prover evaluates the first tier at x^ℓ and the second at x. If $\ell \approx m$ then we have square root values $f_i(X)$ which each have degree square root in d. Hence our IPP arguments are ran over a square root number of commitments, which is what makes our verifier complexity and SRS size square root.

Evaluation. In Figure 5, we compare the performance of our polynomial commitment scheme against the state-of-the-art KZG commitment scheme. In optimizing the IPA commitment scheme, we found that the MIPP_k proof was more expensive than the KZG proof. Therefore, it makes sense to skew the split of the polynomial so the MIPP_k proof is over a smaller vector than the KZG proof. We found a skew of $\kappa = 16$ to be optimal, leading to a split of $m = \frac{\sqrt{d}}{\kappa}$ and $\ell = \kappa\sqrt{d}$; this explains the hitch in the plots until the optimal tradeoff is able to be made at $d = 2^{10}$.

Both KZG and our IPA produce commitments of constant size (a single $\mathbb{G}_1$ element for KZG and a single $\mathbb{G}_T$ element for IPA). The differences are that KZG allows for constant opening proof sizes and constant verifier time (versus our logarithmic opening sizes and verifier time), whereas IPA allows for square root opening time and SRS size (compared to the linear complexity of KZG).

These asymptotic differences result in significant concrete tradeoffs between the two schemes. As expected, the IPA commitment, while expensive for low degree polynomials due to overhead of the inner product argument, quickly becomes much faster to compute opening proofs with breakeven degree being $d \approx 2000$; at $d = 10^6$, IPA is 14× faster, and at $d = 250 \times 10^6$ is 80× faster. Similar savings are made with respect to prover SRS size. For degree 10^6, IPA requires an SRS of size 800KB, 60× smaller than the 50MB SRS required by KZG. In contrast, the IPA verifier time and opening size grow logarithmically and thus do not get too large; verifier time remains below 50ms even for polynomials of degree $d = 250 \times 10^6$, and opening proof size remains below 4KB.

7 Aggregating SNARK proofs

We now discuss how the inner pairing product can be used to verify that n independently generated SNARK proofs on independent instances can be aggregated to a $O(\log(n))$ sized proof. While zk-SNARKs have constant-sized proofs and verifiers, in many settings, such as blockchains, a verifier needs to read and verify many proofs created by independent provers. We show how an untrusted aggregator can use inner product arguments to aggregate these proofs into a small logarithmic sized proof. The verifiers only need to check the aggregated proof to be convinced of the existence of the underlying pairing-based SNARKs. We show our approach is concretely much faster than existing approaches relying on recursive composition and expensive pairing-friendly cycles of elliptic curves.

To date the most efficient zkSNARK is due to Groth [Gro16]; it consists of 3 group elements and requires checking a single pairing product equation to verify. We thus choose to describe our methods with respect to [Gro16], but note that they apply more generally to pairing-based SNARKs that do not use random oracles [GM17; PGHR13]. We first provide some background on the [Gro16] SNARK, focusing on the verifier and not the prover, for it is the verification equations that we aim to prove are satisfied.

[Gro16] Background. We recall the following facts about the [Gro16] SNARK: The verification key is of the form:

$$\mathsf{vk} := (p = g^{\rho}, q = h^{\tau}, [s_j = g^{(\beta u_j(x) + \alpha v_j(x) - w_j(x))}]_{j=1}^{\ell}, d = h^{\delta}) \ .$$

Here $\rho, \tau, \delta, x \in \mathbb{F}$ are secrets generated (and discarded) during the generation of the proving and verification keys, ℓ is the statement size, and $u_j(X), v_j(X), w_j(X)$ are public polynomials that together with δ define a circuit representation of the computation being checked. The proof is of the form $\pi := (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$. On input a verification key vk, an NP instance $\mathbb{x} := (a_1, \ldots, a_\ell) \in \mathbb{F}^\ell$, and a proof $\pi = (A, B, C)$, the verifier checks that $e(A, B) = e(p, q) \cdot e(\prod_{j=1}^{\ell} s_j^{a_j}, h) \cdot e(C, d)$.

A part of the [Gro16] trusted setup is *circuit-specific*, i.e., the s_j values constructed from $u_j(X), v_j(X), w_j(X)$ polynomials and d. Our protocol supports

$\mathsf{Setup}(\langle \mathsf{group} \rangle, [\mathsf{vk}_i]_{i=0}^{n-1})$:

1. Construct commitment keys and prover and verifier keys. Note commitment keys $\boldsymbol{w} = \left[g^{\alpha^{2i}}\right]_{i=0}^{m-1}$ and $\boldsymbol{v} = \left[h^{\beta^{2i}}\right]_{i=0}^{m-1}$ are included in $\mathsf{pk}_{\mathsf{IPP}}$. Sample $\alpha, \beta \xleftarrow{\$} \mathbb{F}$:

$$(\mathsf{pk}_{\mathsf{IPP}} = (\left[g^{\alpha^i}\right]_{i=0}^{2m-2}, \left[h^{\beta^i}\right]_{i=0}^{2m-2}), \mathsf{vk}_{\mathsf{IPP}} = (g^{\beta}, h^{\alpha})) \xleftarrow{\$} \mathsf{IPP.Setup}(m; \alpha, \beta)$$

2. Commit to circuit-specific elements of verification keys, $\mathsf{vk}_i = (p, q, [s_{i,j}]_{j=1}^{\ell}, d_i)$:
   (a) Commit to $\boldsymbol{d} = [d_i]_{i=0}^{n-1}$: $C_d \leftarrow \mathsf{CM}(\boldsymbol{w}, \boldsymbol{d}) = \boldsymbol{w} * \boldsymbol{d}$.
   (b) For each $j \in [\ell]$, commit to $\boldsymbol{s}_j$: $C_{s,j} \leftarrow \mathsf{CM}(\boldsymbol{v}, \boldsymbol{s}_j = [s_{i,j}]_{i=0}^{n-1}) = \boldsymbol{s}_j * \boldsymbol{v}$.
3. Return $(\mathsf{pk}_{\mathsf{agg}} = (\mathsf{pk}_{\mathsf{IPP}}, [\mathsf{vk}_i]_{i=0}^{n-1}, [C_{s,j}]_{j=1}^{\ell}, C_d, \boldsymbol{d}), \mathsf{vk}_{\mathsf{agg}} = (\mathsf{vk}_{\mathsf{IPP}}, p, q, [C_{s,j}]_{j=1}^{\ell}, C_d))$.

$\mathsf{Agg}(\mathsf{pk}_{\mathsf{agg}}, [(\mathbb{x}_i = [a_{i,j}]_j^{\ell}, \pi_i)]_{i=0}^{n-1})$:
$(\pi, r) \leftarrow \mathsf{AggHelper}(\mathsf{pk}_{\mathsf{agg}}, [(\mathbb{x}_i, \pi_i)]_{i=0}^{n-1}, \bot)$
Return π

$\mathsf{Verify}(\mathsf{vk}_{\mathsf{agg}}, [\mathbb{x}_i = [a_{i,j}]_j^{\ell}]_{i=0}^{n-1}, \pi_{\mathsf{agg}})$:
$[Z_{s,j}]_{j=1}^{\ell} \leftarrow [\prod_{i=0}^{n-1} s_{i,j}^{a_{i,j} r^{2i}}]_j^{\ell}$
$(b, r) \leftarrow \mathsf{VerHelper}(\mathsf{vk}_{\mathsf{agg}}, [Z_{s,j}]_j^{\ell}, \pi_{\mathsf{agg}}, \bot)$
Return b

Fig. 6: Aggregation of Groth16 SNARKs. The helper subprotocols for aggregation and verification are given in the full version [BMM+19].

aggregating proofs over different circuits that share the non-circuit-specific part of their trusted setup, i.e., the p, q elements in the verification key.

Our aggregation protocol. Our aggregation protocol is described in Fig. 6. Given n instances $[[a_{i,j}]_{i=0}^{n-1}]_{j=1}^{\ell}$, proofs $[\pi_i = (A_i, B_i, C_i)]_{i=0}^{n-1}$, and circuit-specific verification keys $[[s_{i,j}]_j^{\ell}, d_i]_{i=0}^{n-1}$, verifying the pairing product equation for each proof π_i individually requires performing $3n$ pairings and $n\ell$ exponentiations. To reduce this computation to a single pairing product equation, the verifier can take a random linear combination between all equations. That is, the verifier samples $r \xleftarrow{\$} \mathbb{F}$, sets $\boldsymbol{r} = (1, r^2, \ldots, r^{2n-2})$ and then checks whether

$$\prod_{i=0}^{n-1} e((A_i)^{r^{2i}}, B_i) = e(p, q)^{\sum_i^{n-1} r^{2i}} \cdot e(\prod_{j=1}^{\ell} \prod_{i=0}^{n-1} s_{i,j}^{a_{i,j} r^{2i}}, h) \cdot e(\prod_{i=0}^{n-1} C_i^{r^{2i}}, d_i) \ .$$

If this equation holds, then with overwhelming probability each individual verification holds. It therefore suffices to check this one pairing product instead of checking all SNARKs individually.

We make use of two inner products arguments to prove that the above check succeeds. At a high level, the prover commits to $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$. First, the TIPP protocol is used to prove the evaluation of $\boldsymbol{A}^{\boldsymbol{r}} * \boldsymbol{B} = Z_{AB}$. The verifier must check Z_{AB} against the expected evaluation of the right-hand side of the above pairing product equation. To further help the verifier, a second evaluation of TIPP is used to prove the evaluation of $\boldsymbol{C}^{\boldsymbol{r}} * \boldsymbol{d} = Z_{Cd}$, where $\boldsymbol{d}$ is derived from the circuit-specific verification keys. The verifier then completes by evaluating and checking:

$$Z_{AB} = e(p, q)^{\frac{r^{2n}-1}{r^2-1}} \cdot e(\prod_{j=1}^{\ell} \prod_{i=0}^{n-1} s_{i,j}^{a_{i,j} r^{2i}}, h) \cdot Z_{Cd} \ ,$$

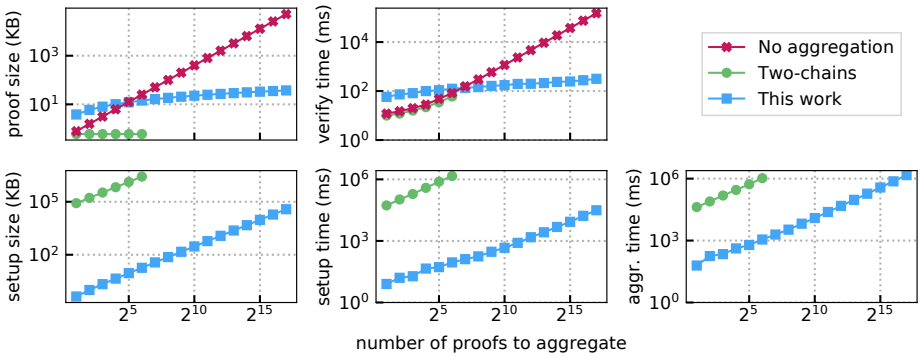

Fig. 7: Measured performance of TIPP aggregation of SNARK proofs compared to the cost of proving a one layer of recursion inside a SNARK.

which requires only two pairings, $\ell+2$ exponentiations, and $O(\ell \cdot n)$ field operations. If aggregating over the same circuit, the circuit-specific setup of $\boldsymbol{d}$ is not needed and the protocol can be simplified to use MIPP_k instead of TIPP to derive Z_{Cd}.

Evaluation. In Figure 7, we compare the performance for aggregating SNARK proofs using (a) our aggregation protocol, (b) using recursive SNARKs over a 2-chain [BCG+20], and (c) not aggregating at all (i.e., sending all proofs individually). The 2-chain approach proves inside another SNARK that each of the aggregated SNARKs is valid. The verification time for no aggregation consists of a single batched pairing check.

While our protocol does not produce constant-sized proofs, it does reduce setup size and aggregation time greatly. For example, when aggregating 64 proofs, our protocol is 900× faster than the 2-chain approach. Furthermore, the 2-chain approach is unable to scale further as it consumed too much memory. In fact, in the time it takes 2-chain approach to aggregate 64 proofs, our protocol can aggregate 65,000 proofs into a 35 kB proof that takes 300 ms to verify.

8 Low-memory SNARKs for machine computations

We now show how to leverage our aggregation protocol in Section 7 to construct a *low-memory SNARK* for (non-deterministic) machine computations. A machine computation M consists of applying a sequence of operations $M = (\mathsf{op}_0, \ldots, \mathsf{op}_T)$ (from some operation set OpSet) over a fixed number of registers. We model that M can read and write to an external memory (of size S) using techniques for online memory checking [BCCT13; BCGT13; BEG+91] in which the memory is represented as a Merkle tree. In this case, an arithmetic circuit P_i for each operation op_i can be built such that $|P_i| = \text{polylog}(S)$, given that op_i itself has complexity $\text{polylog}(S)$ and makes at most $\text{polylog}(S)$ reads or writes to memory. Taking this approach, we provide Theorem 5, which states that if a machine computation M executes using memory S over T operation steps, then our SNARK prover takes time $\tilde{\mathcal{O}}(\max_i(|P_i|) \cdot T)$ and space $\tilde{\mathcal{O}}(\max_i(|P_i|) + T + S)$ to produce a proof for that execution.

In comparison, constructing a monolithic proof for the entire computation at once requires the same time, but incurs a space usage of $\tilde{\mathcal{O}}(\sum_i(|P_i|) \cdot T + S)$.

The only other solution for constructing low-memory SNARKs for machine computations requires recursive composition of proofs [BCCT13]. Recursive composition achieves a further improved space usage of $\tilde{\mathcal{O}}(\max_i(|P_i|) + S)$, but the time to prove, while asymptotically equivalent to the previous solutions, is concretely very expensive.

Definition 4 (Machine relation). *For a machine M with step operations $[\mathsf{op}_i]_{i=0}^{T-1}$, the NP relation $\mathcal{R}_M$ is the set of of instance-witness pairs $(x, [\omega]_i^{T-1})$, such that M accepts $(x, [\omega]_i^{T-1})$ after the T step operations are applied.*

Theorem 5. *Let $\mathcal{R}_M$ be a machine relation for some machine M with step operations $[\mathsf{op}_i]_{i=0}^{T-1}$ that can be represented with arithmetic circuits $[P_i]_i^{T-1}$, and $\mathsf{op}_i \in \mathsf{OpSet}$ for all i. Then there exists a SNARK for $\mathcal{R}_M$ where*

(1) *Setup takes time $\mathcal{O}(T + \sum_{\mathsf{op} \in \mathsf{OpSet}}(|P_{\mathsf{op}}|))$.*

(2) *Proving takes time $\tilde{\mathcal{O}}(\max_i(|P_i|) \cdot T)$ and uses space $\tilde{\mathcal{O}}(\max_i(|P_i|) + T + S)$, where S is the space required to compute M.*

(3) *Proof size is $\mathcal{O}(\log(T))$ and verification takes time $\mathcal{O}(\log(T))$.*

Overview of solution. We first introduce some notation. The full details of our protocol are given in the full version [BMM+19]. Machine M operates over a fixed set of ℓ registers. The statement for each operation circuit P_i consists of 2ℓ elements: ℓ input registers $[a_{i,j}]_{j=1}^{\ell}$ and ℓ output registers $[b_{i,j}]_{j=1}^{\ell}$. The circuit verifies that the output registers are valid with respect to applying the operation on the input registers. Importantly, the output registers of an operation are passed as the input registers to the next operation in sequence:

$$[b_{i,j}]_{j=1}^{\ell} = [a_{i+1,j}]_{j=1}^{\ell}\,.$$

The verifier does not need to be aware of all of the values the registers take on during intermediate steps of execution. Instead, it need only verify that the above "sequential" pattern of registers is present in the proofs for each operation step. This is the key observation we take advantage of to produce a log-time verifier.

As a strawman, consider the solution of proving an individual Groth16 SNARK for each operation step and aggregating using the protocol of Section 7. To verify, the verifier must receive and perform scalar computations over all of the intermediate statements, incurring linear proof size and verification time.

Instead, in our solution, the prover commits to all of the intermediate statements and proves to the verifier that they follow the sequential structure, i.e., the second half of the statement for proof i is the first half of the statement for proof $i+1$. The verifier can verify this in time ℓ with knowledge of only the initial register state $[a_{0,j}]_j^{\ell}$ and the final register state $[b_{T-1,j}]_j^{\ell}$. The prover commits to the inputs and outputs of all statements, $\boldsymbol{a}_j$, $\boldsymbol{b}_j$, to $C_{a,j}, C_{b,j}$. The prover then proves the sequential pattern between $\boldsymbol{a}_j$ and $\boldsymbol{b}_j$ holds, namely that the vectors are offset by one:

$$\begin{array}{l} a_{0,j}\ a_{1,j}\ \ldots\ a_{T-1,j} \\ \quad\ b_{0,j}\ \ldots\ b_{T-2,j}\ b_{T-1,j} \end{array}$$

The prover does this by homomorphically shifting the commitment to $\boldsymbol{b}_j$ using challenge r and taking the difference between the two vector commit-

ments $C_{a,j}C_{b,j}^{-1/r^2}$, then providing a KZG opening proof that it opens to $a_{0,j} - b_{n-1,j}r^{2T-2}$ when evaluated on r. Lastly, the prover uses the commitments to precompute a part of the final pairing product verification check to help the verifier avoid the linear scalar computations. The prover computes and proves using MIPP_u the multiexponentiation inner products, $Z_{s,j} = \boldsymbol{s}_j^{\boldsymbol{a}_j \circ \boldsymbol{r}}$ and $Z_{s,\ell+j} = \boldsymbol{s}_{\ell+j}^{\boldsymbol{b}_j \circ \boldsymbol{r}}$ for $\boldsymbol{s}_j$ derived from circuit-specific verification keys. The verifier then completes the verification by checking the following pairing product equation:

$$Z_{AB} = e(p,q)^{\frac{r^{2n}-1}{r^2-1}} \cdot e(\prod_{j=1}^{2\ell} Z_{s,j}, h) \cdot Z_{Cd} \ ,$$

which requires only two pairings and $O(\ell)$ group operations. Our solution may be adapted to provide greater efficiency in the case of repeated application of a single step operation.

References

[AC20] T. Attema and R. Cramer. "Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics". In: CRYPTO '20.

[AFG+16] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. "Structure-Preserving Signatures and Commitments to Group Elements". In: *J. Cryptology* (2016).

[BBB+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: S&P '18.

[BCC+16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: EUROCRYPT '16.

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: STOC '13.

[BCG+13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge". In: CRYPTO '13.

[BCG+14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: SP '14.

[BCG+20] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. "ZEXE: Enabling Decentralized Private Computation". In: S&P '20.

[BCGT13] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. "Fast Reductions from RAMs to Delegatable Succinct Constraint Satisfaction Problems". In: ITCS '13.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. "Proof-Carrying Data from Accumulation Schemes". In: TCC '20.

[BCTV14a] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: CRYPTO '14.

[BCTV14b] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture". In: USENIX Security '14.

[BDK13] M. Backes, A. Datta, and A. Kate. "Asynchronous Computational VSS with Reduced Communication Complexity". In: CT-RSA '13.

[BEG+91] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. "Checking the correctness of memories". In: FOCS '91.

[BFS20] B. Bünz, B. Fisch, and A. Szepieniec. "Transparent SNARKs from DARK Compilers". In: EUROCRYPT '20.

[BG13] S. Bayer and J. Groth. "Zero-Knowledge Argument for Polynomial Evaluation with Application to Blacklists". In: EUROCRYPT '13.

[BGH19] S. Bowe, J. Grigg, and D. Hopwood. "Halo: Recursive Proof Composition without a Trusted Setup". ePrint 2019/1021.

[BMM+19] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely. "Proofs for Inner Pairing Products and Applications". ePrint 2019/1177.

[BMRS20] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. "Coda: Decentralized Cryptocurrency at Scale". ePrint 2020/352.

[CDHK15] J. Camenisch, M. Dubovitskaya, K. Haralambiev, and M. Kohlweiss. "Composable and Modular Anonymous Credentials: Definitions and Practical Constructions". In: ASIACRYPT '15.

[CHM+20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: EUROCRYPT '20.

[COS20] A. Chiesa, D. Ojha, and N. Spooner. "Fractal: Post-quantum and Transparent Recursive Proofs from Holography". In: EUROCRYPT '20.

[FHS19] G. Fuchsbauer, C. Hanser, and D. Slamanig. "Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials". In: *J. Crypto.* (2019).

[Fis18] B. Fisch. "PoReps: Proofs of Space on Useful Data". ePrint 2018/678.

[Fis19] B. Fisch. "Tight Proofs of Space and Replication". In: CRYPTO '19.

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. "The Algebraic Group Model and its Applications". In: CRYPTO '18.

[GGW18] K. Gurkan, A. Gabizon, and Z. Williamson. "Cheon's attack and its effect on the security of big trusted setups". `https://ethresear.ch/t/cheons-attack-and-its-effect-on-the-security-of-big-trusted-setups/6692`.

[GKM+18] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs". In: CRYPTO '18.

[GM17] J. Groth and M. Maller. "Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs". In: CRYPTO '17.

[GMN21] N. Gailly, M. Maller, and A. Nitulescu. "SnarkPack: Practical SNARK Aggregation". ePrint 2021/529.

[Gro11] J. Groth. "Efficient Zero-Knowledge Arguments from Two-Tiered Homomorphic Commitments". In: ASIACRYPT '11.

[Gro16] J. Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: EUROCRYPT '16.

[GS08] J. Groth and A. Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: EUROCRYPT '08.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. "PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge". ePrint 2019/953.

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: ASIACRYPT '10.

[Lee20] J. Lee. "Dory: Efficient, Transparent arguments for Generalised Inner Products and Polynomial Commitments". ePrint 2020/1274.

[LMR19] R. W. F. Lai, G. Malavolta, and V. Ronge. "Succinct Arguments for Bilinear Group Arithmetic: Practical Structure-Preserving Cryptography". In: CCS '19.

[LY10] B. Libert and M. Yung. "Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs". In: TCC '10.

[MBKM19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings". In: CCS '19.

[O'L18] R. O'Leary. "Monero to Become First Billion-Dollar Crypto to Implement 'Bulletproofs' Tech". `https://www.coindesk.com/monero-to-become-first-billion-dollar-crypto-to-implement-bulletproofs-tech`.

[Ped92] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: CRYPTO '91.

[PGHR13] B. Parno, C. Gentry, J. Howell, and M. Raykova. "Pinocchio: Nearly Practical Verifiable Computation". In: S&P '13.

[SCP+21] S. Srinivasan, A. Chepurnoy, C. Papamanthou, A. Tomescu, and Y. Zhang. "Hyperproofs: Aggregating and Maintaining Proofs in Vector Commitments". ePrint 2021/599.

[Set20] S. Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: CRYPTO '20.

[TFBT21] N. Tyagi, B. Fisch, J. Bonneau, and S. Tessaro. "Client-Auditable Verifiable Registries". ePrint 2021/627.

[Whi] B. Whitehat. *Roll up: Scale Ethereum with SNARKs.* URL: `https://github.com/barryWhiteHat/roll_up`.

[WTS+18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: S&P '18.

[WZC+18] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica. "DIZK: A Distributed Zero Knowledge Proof System". In: USENIX Security '18.

[XYZW16] J. Xu, A. Yang, J. Zhou, and D. S. Wong. "Lightweight Delegatable Proofs of Storage". In: ESORICS '16.

[XZZ+19] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. "Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation". In: CRYPTO '19.

[ZXZS20] J. Zhang, T. Xie, Y. Zhang, and D. Song. "Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof". In: S&P '20.