Proofs for Inner Pairing Products and Applications

Benedikt Bünz[1]
benedikt@cs.stanford.edu

Mary Maller[2]
mary.maller@ethereum.org

Pratyush Mishra[3]
pratyush@berkeley.edu

Noah Vesely[3]
psi@berkeley.edu

Abstract

We present a generalized inner product argument and demonstrate its applications to pairing-based languages. As a first instantiation, we introduce a statistically sound proof for the product of n pairings given public source group elements. This protocol enables outsourcing many pairing equation checks to an untrusted prover. Proofs are $2\log n$ target group elements, computed using $2n$ pairings and n exponentiations in each source group. There is no setup. The verifier work is dominated by two multi-exponentiations of size n which require time $O(n/\log n)$. Asymptotically, verification is thus faster than computing the n Miller loops that dominate the pairing product computation; indeed, our implementation demonstrates a $8\times$ speedup for a million pairings.

Next, we apply our generalized argument to proving that an inner pairing product is correctly evaluated with respect to committed vectors of source group elements. With a structured reference string (SRS) we achieve a log-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 show how this argument can be used to aggregate n Groth16 zkSNARKs into an $O(\log n)$ proof, presenting a more efficient alternative to recursive composition of SNARKs.

Using a combination of our techniques, we build a polynomial commitment scheme with logarithmic communication and $O(\sqrt{d})$ prover complexity for degree d polynomials (not including the cost to evaluate the polynomial). With a public-coin setup the verifier runs in $O(\sqrt{d})$ time, and with an SRS the verifier runs in $O(\log d)$ time.

This work was completed while at [1]Stanford University, [2]Ethereum Foundation, and [3]UC Berkeley.

1 Introduction

Pairing-based cryptography constructions include efficient signatures [BLS01], zero-knowledge proofs [GS08; GGPR13], anonymous credentials [BCKL08], and more. These protocols rely on pairings that enable a verifier to directly check bilinear relations between committed secrets. Concretely, given g^a and g^b a verifier can directly check that a group element C is equal to g^{ab}. Thus, using a bilinear pairing, one can check that a *quadratic* equation in unknown variables is satisfied. It has been shown that this suffices to build NIZK arguments in the plain model [GOS06], SNARKs with constant sized verifiers [PGHR13], rerandomizable signature schemes [PS16], and many more primitives. Not only are pairing-based arguments of theoretical interest, they are widely used in practice and there are standardization efforts to help the efforts of developers [Lan08].

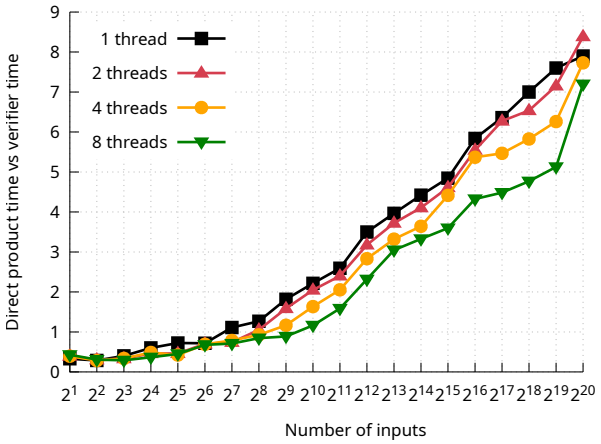
In this work we explore how a generalized inner product argument (GIPA) can be applied to pairing based languages in order to: (1) outsource pairing equations to a more powerful prover; (2) demonstrate that committed source group elements are in pairing based languages; (3) build a polynomial commitment scheme with constant-sized commitments and efficient openings. Our GIPA protocol is a generalization of the inner product argument for discrete logarithm relations [BCCGP16; BBBPWM18].

SIPP. Our first instantiation of the GIPA protocol is a statistical inner pairing product proof (SIPP). A prover runs SIPP to create a log-sized, publicly verifiable proof of the correct computation of an inner pairing product with respect to public source group elements. The protocol can be used to outsource n arbitrary pairings to a prover. The verifier performs a single pairing and one variable-base multi-exponentiation of size n in each source group. Group exponentiations are cheaper than pairings and existing multi-exponentiation algorithms require only $\mathcal{O}(n/\log(n))$ exponentiations. Hence, SIPP asymptotically reduces the number of cryptographic operations performed by the verifier. In Fig. 1 we demonstrate that SIPP achieves concrete verifier improvements: our verifier is faster than directly computing the pairing product at 128 pairings. Further, the verifier savings increases with the number of inputs, empirically validating the asymptotic improvements. For a million pairings the verifier is more than $8\times$ faster. We also see that, as expected, the prover runs in approximately 3-$4\times$ the time of direct computation. In Section 9, we use SIPP to build a new aggregate signature for BLS [BLS01] with a faster verifier than previous results [BGLS03; RY07; BDN18].

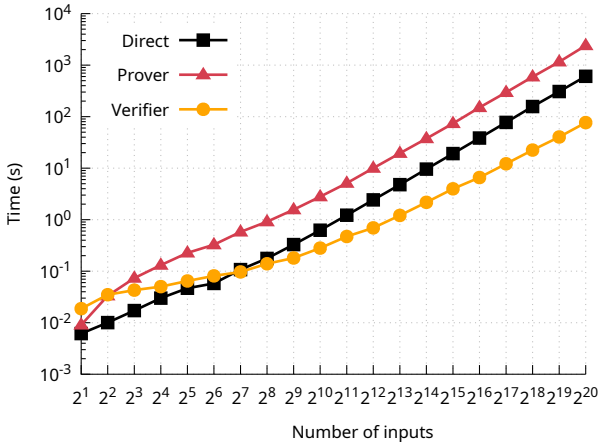
TIPP. Our second instantiation of GIPA is a trusted setup inner pairing product (TIPP) argument of knowledge used demonstrate that certain pairing relations hold between committed group elements.TIPP has logarithmic proof size and verifier time, and linear prover time (all with small constants). Extrapolating from the results of our IPP implementation, concrete prover time will be only 9–$12\times$ direct computation time. In Section 8 we use TIPP to show how an untrusted party can aggregate n Groth16 SNARKs [Gro16] into a single log-sized proof with a log-time verifier.

Comparison to using pairing-friendly cycles and SNARKs. An alternative to using SIPP or TIPP is to prove the same relations by using NP-reductions and a general purpose SNARK. Asymptotically, SIPP and TIPP have logarithmic proof sizes versus the constant-size proofs of SNARKs such as [Gro16], but our protocols have much faster prover time and greater compatibility with existing systems.

Efficiently expressing pairing based languages as arithmetic circuits for general purpose SNARKs requires the use of pairing-friendly cycles [BCTV14] or two-chains [BCGMMW20]. Known cycles and two-chains for the 128-bit security level require roughly 780-bit curves, versus the roughly 380-bit curves used when recursion is not necessary. More efficient pairing-friendly cycles seem elusive at best, with negative results on their existence for some curve families [CCW19]. Since almost no deployed systems are using pairing-friendly cycles or two-chains, our protocols also have the added benefit of direct applicability to many systems using pairing-based cryptography.



(a) Ratio of the time to compute a direct pairing product to the time to verify a SIPP for that computation.



(b) Running time of the SIPP algorithms compared to the time to directly compute the pairing product.

Figure 1: Measured performance of SIPP compared to the cost of directly computing the pairing product on the efficient BLS12 − 377 elliptic curve [BCGMMW20]. Experiments were performed on a machine with an Intel Xeon 6136 CPU at 3.0 GHz.

Even with pairing-friendly cycles the reduction of the statements to an arithmetic circuit (or R1CS), the NP-language for general purpose SNARKs, is very expensive. A single pairing operation requires about $15,000$ R1CS constraints to express. For the most efficient SNARKs such as [Gro16], proving time then scales quasi-linearly with respect to this (much larger) set of constraints. This stands in stark contrast to the small constant overhead of our protocol.

1.1 Applications

Polynomial commitments. Polynomial commitment (PC) schemes [KZG10] are commitment schemes specialized to work with polynomials. A committer outputs a short commitment to polynomial, and then later may convince a verifier of correctness of an evaluation of that committed polynomial at any point via a short 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], zero-knowledge arguments [WTSTW18; MBKM19; Gab19; Set19; GWC19; XZZPS19; CHMMVW20]. We use a combination of inner product arguments in order to build a pairing based inner product argument with constant-sized commitments, logarithmic-sized openings, and a square root reference string.

Our PC scheme uses a Groth [Gro11] style two-tiered homomorphic commitment and supports both univariate and bivariate polynomials. We demonstrate how to instantiate it both with a public-coin setup, achieving square root verifier time, and with an updatable SRS [GKMMM18], achieving log-time verification. The transparent variant is secure in the plain model under the standard SXDH assumption, and our trusted setup scheme is secure in the algebraic group model (AGM) [FKL18]. One of the main advantages to our scheme is the time to produce opening proofs: for univariate polynomials opening costs are square root in the degree of the polynomial; for bivariate polynomials opening costs are linear in the degree of one variable. The sublinear opening time is particularly relevant in applications where the commitment is opened many

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 [BBBPWM18]	$d\ \mathbb{G}_1$	$1\ \mathbb{G}_1$	$\log(d)\ \mathbb{G}_1$	1.3 KB	yes	$d\ \mathbb{G}_1$	$d\ \mathbb{G}_1$	$d\ \mathbb{G}_1$
Hyrax [WTSTW18]	$\sqrt{d}\ \mathbb{G}_1$	$\sqrt{d}\ \mathbb{G}_1$	$\log(d)\ \mathbb{G}_1$	33 KB	yes	$d\ \mathbb{G}_1$	$\sqrt{d}\ \mathbb{G}_1$	$\sqrt{d}\ \mathbb{G}_1$
DARKs [BFS19]	$d\ \mathbb{G}_U$	$1\ \mathbb{G}_U$	$\log(d)\ \mathbb{G}_U$	8.6 KB	yes	$d\ \mathbb{G}_U$	$d\log(d)\ \mathbb{G}_U$	$\log(d)\ \mathbb{G}_U$
Virgo [ZXZS19]	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	$d\ \mathbb{G}_1$	$\sqrt[2]{3}{d}\ \mathbb{G}_1$	$\sqrt[3]{d}\ P$
This work transparent	$\sqrt{d}\ \mathbb{G}_2$	$1\ \mathbb{G}_T$	$\log(d)\ \mathbb{G}_T$	4.6 KB	yes	$d\ \mathbb{G}_1$	$\sqrt{d}\ P$	$\sqrt{d}\ \mathbb{G}_2$
This work structured	$\sqrt{d}\ \mathbb{G}_2$	$1\ \mathbb{G}_T$	$\log(d)\ \mathbb{G}_T$	4.1 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.

times. This is the case for vector commitment and proof of space applications. Computing a commitment itself is linear in the number of non-zero coefficients of the polynomial.

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. Smaller SRS size can have consequences not only for storage and setup efficiencies but also for security. Indeed, it was recently noted by Gurk et al. [GGW18] that Cheon's attack on q-type assumptions [Che10] can degrade the security of some SNARK schemes over BLS12-381 from the advertised 128 bits of security to 114 bits of security.

Aggregating pairing-based SNARKs. We design an aggregator for Groth16 [Gro16] pairing-based SNARKs based on TIPP. The aggregated proof has logarithmic size. The verifier computation depends on the cost of uploading the instances in addition to a logarithmic number of target group exponentiations. We thus obtain a single layer of recursive proofs (i.e., a proof of proofs) that does not depend on inefficient pairing-friendly cycles or two-chains, nor expensive NP reductions.

Aggregating BLS signatures. We apply our SIPP protocol to BLS signatures [BLS01]. Boneh et al. observed that BLS signatures can be efficiently aggregated [BGLS03], however, their aggregated signatures require the verifier to compute one pairing per distinct message. Using SIPP, we derive a new aggregate signature for BLS where verficiation requires computing a single pairing and a multi-exponentiation in each source group of size equal to the number of messages being verified. The tradeoff is our aggregate signatures are logarithmic versus the constant-size aggregate signatures of [BGLS03]. The aggregation algorithm is computed offline by an untrusted party who requires no secrets.

1.2 Related work

Lai, Malavolta, and Ronge [LMR19] an inner product argument for pairing based languages. Their scheme runs over a transparent setup, is secure under the SXDH assumption and they discuss the applications of their argument to zero-knowledge proofs. We present a detailed efficiency analysis of their scheme compared this work in Table 2.

Groth and Sahai [GS08] introduced a method to prove pairing-based languages under zero-knowledge without reducing to NP (or alternatively under witness indistinguishably with smaller proofs). The group

elements are committed to under either a perfectly binding commitment key or a perfectly hiding commitment key (and the prover cannot distinguish which) and depending on which key is used the protocol is either perfectly sound or perfectly zero-knowledge. This approach has since been improved by Escala and Groth [EG14] and by Ghadafi et al. [GSW10]. Blazy et al. [BFIJSV10] noted that it is possible to batch verify pairing equations that share a source group element. However, their results do not extend to the setting where both source group elements are different. Our work can be used to aggregate pairing equations where the source group elements differ. GS proofs are secure in the standard model under standard assumptions. Thus their linear sized proofs and verifier time are optimal [GW11], whereas our smaller proofs can only be obtained because we are in the random oracle model. Nonetheless, this means that GS proofs can be used for applications that require straight-line extractors whereas ours cannot. We also note that, unlike GS proofs [BCCKLS09], our proofs are not re-randomizable.

	communication complexity		transparent setup	time complexity	
	|CRS|	$\|\pi\|$		Prove	Verify
[LMR19]	$2m\ \mathbb{G}_1 + 2m\ \mathbb{G}_2$	$6 \log m\ \mathbb{G}_T$	yes	$3m\ \mathbb{G}_1 + 3m\ \mathbb{G}_2 + 10m\ P$	$2m\ \mathbb{G}_1 + 2m\ \mathbb{G}_2 + 6 \log m\ \mathbb{G}_T$
IPP	$m\ \mathbb{G}_1 + \ m\ \mathbb{G}_2$	$6 \log m\ \mathbb{G}_T$	yes	$2m\ \mathbb{G}_1 + 2m\ \mathbb{G}_2 + \ 6m\ P$	$m\ \mathbb{G}_1 + \ m\ \mathbb{G}_2 + 6 \log m\ \mathbb{G}_T$
TIPP	$2m\ \mathbb{G}_1 + 2m\ \mathbb{G}_2$	$6 \log m\ \mathbb{G}_T$	no	$4m\ \mathbb{G}_1 + 4m\ \mathbb{G}_2 + \ 6m\ P$	$6 \log m\ \mathbb{G}_T$
SIPP	—	$2 \log m\ \mathbb{G}_T$	yes	$m\ \mathbb{G}_1 + \ m\ \mathbb{G}_2 + \ 2m\ P$	$m\ \mathbb{G}_1 + \ m\ \mathbb{G}_2 + 2 \log m\ \mathbb{G}_T$

Table 2: Efficiency comparison between the inner pairing product protocol of Lai et al. [LMR19], versus our protocols IPP, TIPP, and SIPP. Small constants have been ommitted. SIPP applies over a modified relation where the verifier knows all the provers inputs.

Abe et al. proved that one can commit to group elements in asymmetric bilinear groups under the double pairing assumption [AFGHO16] and that such techniques are helpful for building structure preserving signatures (i.e. signatures where messages are group elements) in the standard model. This work uses their commitment scheme.

In Table 1 we compare the efficiency of various polynomial commitment schemes. Kate et al.[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. 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 to build a polynomial commitment scheme [WTSTW18]. 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.

Polynomial commitment schemes have also been constructed using Reed-Solomon codes [ZXZS19]. 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.[BFS19] 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.

2 Technical Overview

Generalized Inner Product Argument. At the heart of this work is a generalized inner product argument. We generalize the techniques of [BCCGP16; BBBPWM18] to apply to commitment schemes that are doubly homomorphic i.e.

$$\mathsf{CM}((\mathsf{ck}_1 + \mathsf{ck}_2); (M_1 + M_2)) = \mathsf{CM}(\mathsf{ck}_1, M_1) + \mathsf{CM}(\mathsf{ck}_1, M_2) + \mathsf{CM}(\mathsf{ck}_2, M_1) + \mathsf{CM}(\mathsf{ck}_2, M_2)$$

and inner products that are bilinear i.e.

$$\langle \boldsymbol{a} + \boldsymbol{b}, \boldsymbol{c} + \boldsymbol{d} \rangle = \langle \boldsymbol{a}, \boldsymbol{c} \rangle + \langle \boldsymbol{a}, \boldsymbol{d} \rangle + \langle \boldsymbol{b}, \boldsymbol{c} \rangle + \langle \boldsymbol{b}, \boldsymbol{d} \rangle.$$

Not coincidently we see that our generalized argument is highly applicable to bilinear maps. We apply GIPA to three different inner products

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

SIPP and TIPP. We use our first inner product to obtain SIPP with respect to the identity commitment. We also use our first inner product to obtain TIPP with respect to a commitment scheme that has a structured commitment key. The actual commit algorithm works the same as that of Abe et al.[AFGHO16]: 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 $e(A_0, v_0)e(A_1, v_1)$. However by using a structured setup we get that the components computed by the verifier in GIPA are highly structured; indeed they correspond to a KZG [KZG10] polynomial commitment to a polynomial that Bowe et al. [BGH19] reason can be computed in logarithmic time. Thus we outsource the verifier work to the prover. The prover demonstrates their honest intent by opening the KZG commitment to the correct evaluation (which the verifier computes themselves).

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

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

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 the polynomials $f_0(Y), \ldots, f_{m-1}(Y)$. Next one commits to the commitments: $T = \mathsf{CM}(A_0, \ldots, A_{m-1})$.

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 a multiexponentiation IPP argument (MIPP). The prover then opens the second tier commitment A at y in order to obtain $\mathsf{eval} = f(x, y)$. This is done using a univariate polynomial commitment scheme: in the transparent version we use Bulletproofs [BBBPWM18] and in the structured setup version we use KZG. See Figure 2 for a depiction.

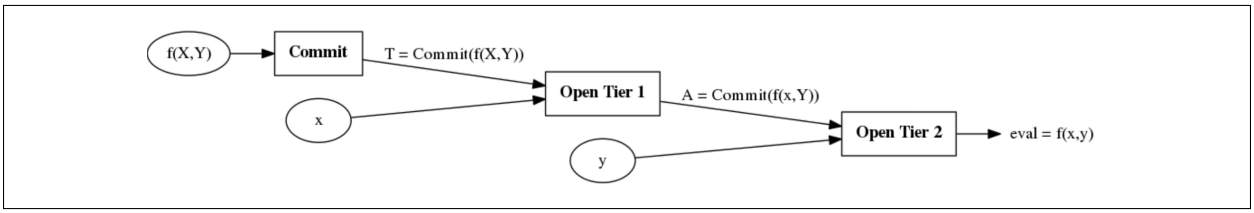


Figure 2: Our polynomial commitment scheme for $f(X, Y)$.

3 Preliminaries

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.

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 [BBBPWM18]. 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 $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$ (see Appendix A.1) 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 $\boldsymbol{A} \in \mathbb{G}^n$ and $0 \leq \ell < n - 1$ we write

$$\boldsymbol{A}_{[:\ell]} = (A_0, \ldots, A_{\ell-1}) \in \mathbb{G}^\ell \quad \text{and} \quad \boldsymbol{A}_{[\ell:]} = (A_\ell, \ldots, A_{n-1}) \in \mathbb{G}^{n-\ell} \ .$$

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 Appendix A.

4 Outsourcing inner pairing products

In this section we introduce an interactive, RBR sound inner pairing product (SIPP) proof system that produces proofs of pairing products. SIPP requires no setup and is public-coin. The verifier trades off computing the n pairings directly for n exponentiations (or two multi-exponentiations of size n) in the source groups. The prover computes just $2n$ pairings and sends $2\log(n)$ target group proof elements.

4.1 Construction

In this section we present our inner pairing product proof SIPP for the membership in the language $\mathcal{L}_{\mathsf{SIPP}}$ defined by

$$\mathcal{L}_{\mathsf{SIPP}} = \{(\boldsymbol{A} \in \mathbb{G}_1^m,\ \boldsymbol{B} \in \mathbb{G}_2^m,\ Z \in \mathbb{G}_T)\ :\ Z = \boldsymbol{A} * \boldsymbol{B}\} \quad .$$

Without loss of generality assume m is a power of two. Our proof is defined by a recursive protocol that in each round "folds" vectors $\boldsymbol{A}, \boldsymbol{B}$ into new vectors $\boldsymbol{A}', \boldsymbol{B}'$ of length $m' = m/2$ such that $e(\boldsymbol{A}', \boldsymbol{B}') = \boldsymbol{Z}'$. Both the prover and verifier independently do this folding using a verifier generated challenge.

First the prover commits to a pair of target group elements Z_L, Z_R that the verifier uses to scale the Z from the last round to a new target group element Z'. In the final round of recursion where $m' = 1$ the verifier simply checks the pairing equation $e(A', B') = Z'$.

In more detail, the prover and verifier start with vectors $\boldsymbol{A} \in \mathbb{G}_1^m$ and $\boldsymbol{B} \in \mathbb{G}_2^m$ and a claimed inner pairing product $Z \in \mathbb{G}_T$. The prover wishes convince the verifier that $Z = \boldsymbol{A} * \boldsymbol{B}$ by engaging in $\log(m)$ rounds of a recursive protocol. The prover and verifier begin each round by each setting $m' = m/2$. The prover first computes

$$Z_L = \boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']} \quad \text{and} \quad Z_R = \boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]} \ ,$$

sending them as commitments to the verifier. The verifier samples $x \xleftarrow{\$} \mathbb{F}$ and sends x to the prover. The prover and the verifier then each set

$$Z' = Z_L^x \cdot Z \cdot Z_R^{x^{-1}} \quad \text{and} \quad \boldsymbol{A}' = \boldsymbol{A}_{[m':]}^x \circ \boldsymbol{A}_{[:m']} \quad \text{and} \quad \boldsymbol{B}' = \boldsymbol{B}_{[m':]}^{x^{-1}} \circ \boldsymbol{B}_{[:m']} \ .$$

The protocol then recurses on $(\boldsymbol{A}, \boldsymbol{B}, Z) = (\boldsymbol{A}', \boldsymbol{B}', Z')$ until the final round, where $m = 2$. In that round after computing $(A', B', Z') \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_T$ the verifier accepts the proof if $e(A', B') = Z'$ and otherwise rejects it.

Informally, the security of the protocol relies on the fact that in each round if $Z' = \boldsymbol{A}' * \boldsymbol{B}'$, then it holds with overwhelming probability that $Z = \boldsymbol{A} * \boldsymbol{B}$. This follows from the DeMillo-Lipton-Schwartz–Zippel lemma: the prover must commit to the coefficients of a polynomial that is then evaluated at a random point; as elaborated on in the proof of round by round soundness (Theorem 4.2), the probability

$$Z' = \boldsymbol{A}' * \boldsymbol{B}' \ \Leftrightarrow\ Z_L^x \cdot Z \cdot Z_R^{x^{-1}} = \left(\boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']}\right)^x \cdot \boldsymbol{A} * \boldsymbol{B} \cdot \left(\boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]}\right)^{x^{-1}}$$

for $x \xleftarrow{\$} \mathbb{F}$ is negligible if Z_L, Z, and Z_R are not the claimed values. Applying this technique recursively, we defer the actual pairing check until $m' = 1$ and the verifier need only compute a single pairing.

Pseudocode. We give pseudocode for our protocol in Fig. 3.

4.2 Efficiency

We provide a detailed breakdown of the communication and computation complexity of the protocol in Table 2. There are $\log(n)$ rounds, wherein each the prover sends 2 target group elements Z_L, Z_R, making the total proof size $2\log(n)$ target group elements.

In the first round the prover requires n pairings to compute Z_L, Z_R. This halves to $n/2$ in the second and $n/2^i$ in the i-th. Over $\log(n)$ rounds the prover computes $2n$ pairings in total. The verifier computes just one pairing in the final round. Over $\log(n)$ rounds the prover and verifier each compute a total of n exponentiations in each of the source groups to compute $\boldsymbol{A}'$ and $\boldsymbol{B}'$. To compute Z' the verifier then computes a total of $2\log(n)$ exponentiations in $\mathbb{G}_T$.

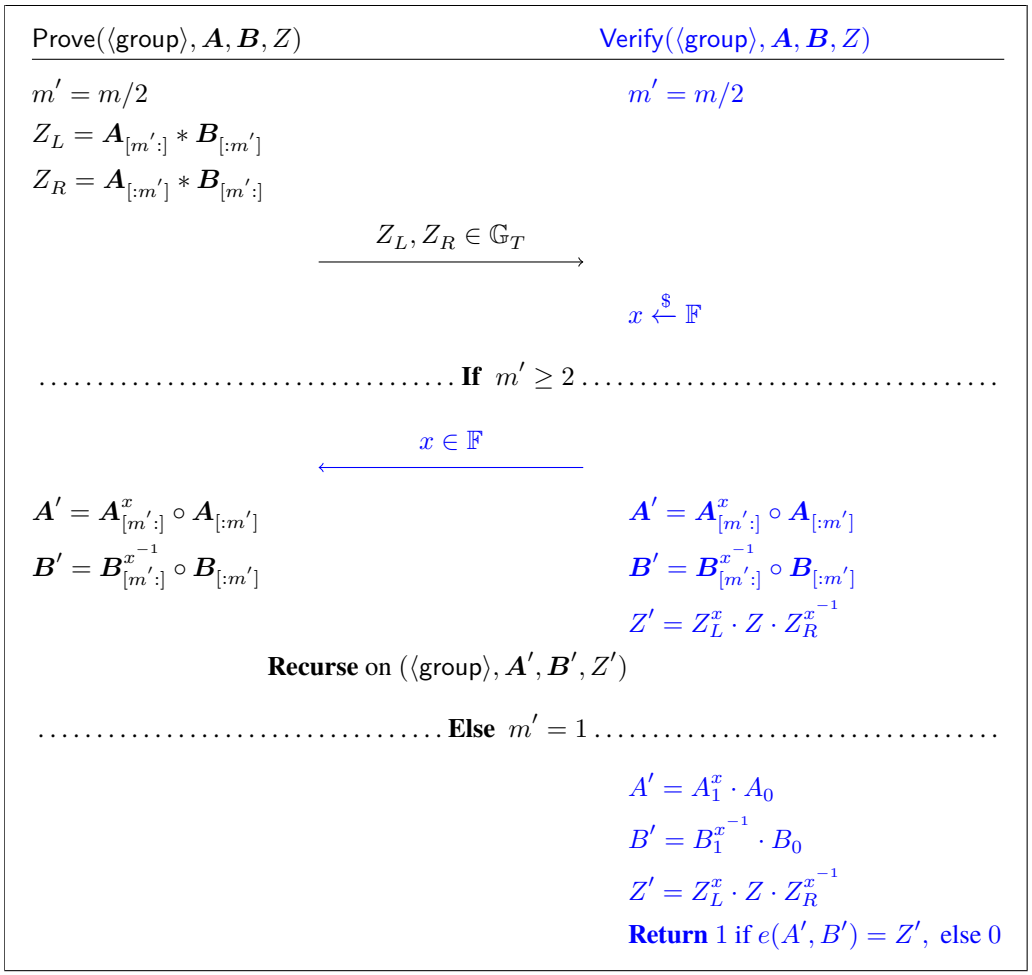


Figure 3: A RBR sound protocol for outsourcing pairings.

4.3 Security

We prove the following two theorems showing that SIPP satisfies perfect completeness and RBR soundness in Appendix B.1.

Theorem 4.1 (Perfect completeness of SIPP)**.** *The interactive protocol defined by Fig. 3 for the language $\mathcal{L}_{\mathsf{SIPP}}$ has perfect completeness (Definition A.7).*

Theorem 4.2 (Round-by-round soundness of SIPP)**.** *The interactive protocol defined by Fig. 3 for the language $\mathcal{L}_{\mathsf{SIPP}}$ has round-by-round soundness (Definition A.8).*

4.4 Reducing pairings to a pairing product

SIPP can also be used to outsource n arbitrary pairings checks $e(A_i, B_i) = e(C_i, D_i)$ using a random linear combination. For this verifier samples a random $r \xleftarrow{\$} \mathbb{F}_p$ and reduces the statement to an inner pairing product of length $2n$: $\prod_{i=1}^{n} e(A_i^{r^i}, B_i) e(C_i^{-r^i}, D_i) = 1$. Using the SIPP protocol to outsource this product

the verifier can delay the exponentiation by r^i. The verifier cost is still dominated by a multi-exponentiation in $\mathbb{G}_1$ and one in $\mathbb{G}_2$. The direct verification would either have to compute the $2n$ pairings directly or use $2n$ exponentiations in $\mathbb{G}_1$ and one pairing product.

4.5 Implementation

We implemented the SIPP protocol in Rust, based on efficient elliptic curve and finite field libraries.[1] Our implementation utilizes Pippenger's fast multi-exponentiation algorithm [Pip80] to speed up the verifier's computation. We also implement standard techniques for direct computation of pairing products [MJ16, Section 11.4.2]. For example we delay the final exponentiation so the pairing cost is dominated by the Miller loop computations.

We evaluated our implementation on a machine with an Intel Xeon 6136 CPU at 3.0 GHz. Our experiments relied on the efficient BLS12-377 elliptic curve [BCGMMW20]. As noted in Fig. 1, our evaluation demonstrates that using the SIPP verifier becomes faster than direct computation the pairing product at roughly 128 pairings. Our experiments also validate the asymptotic superiority of the verifier: the gap between direct computation and verification widens as the number of pairings increases. For example, at 2^{20} pairings, our verifier is roughly $8\times$ faster than directly computing the pairing product.

5 Generalized Inner Product Argument

We now generalize the inner product argument (IPA) from [BCCGP16; BBBPWM18] to work for all so called doubly-homomorphic inner product commitments. The generalized protocol (GIPA) is described in terms of an inner product commitment. All of the inner pairing product protocols in this paper including SIPP as well as the IPA from [BCCGP16; BBBPWM18] are variants of GIPA instantiated with different inner product commitments. The generalized version enables us to simply proof security of the specific instantiations presented in the rest of the paper.

Protocol intuition. The SIPP protocol gives a good intuition for how GIPA works. The protocol works by reducing an instance of size $2m$ to one of size m. As an intuition we will show how to reduce an instance with 2 expensive multiplications $\circledast$ to one 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 bilinear 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, checking the second instance implies that $c = (a_1 \circledast b_1) + (a_2 \circledast b_2)$.

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 $\mathsf{CM}(\mathsf{ck}, \boldsymbol{a}) = \mathsf{CM}(x^{-1} \cdot \mathsf{ck}, x \cdot \boldsymbol{a})$.

Doubly homomorphic commitment. We can apply the generalized inner product argument (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 message space is additively homomorphic because for all $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{M}$ and $\boldsymbol{g} \in \mathcal{K}$ we

[1] https://www.github.com/scipr-lab/zexe

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 message space and the key space.

Definition 5.1. *A commitment scheme* $(\mathsf{Setup}, \mathsf{CM})$ *(see Definition A.3) is doubly homomorphic if* $(\mathcal{K}, +)$, $(\mathcal{M}, +)$ *and* $(\mathsf{Image}(\mathsf{CM}), +)$ *define abelian groups such that for all* $\mathsf{ck}_1, \mathsf{ck}_2 \in \mathcal{K}$ *and* $M_1, M_2 \in \mathcal{M}$ *it holds that*

1. $\mathsf{CM}(\mathsf{ck}_1, M_1) + \mathsf{CM}(\mathsf{ck}_1, M_2) = \mathsf{CM}(\mathsf{ck}_1, M_1 + M_2)$
2. $\mathsf{CM}(\mathsf{ck}_1, M_1) + \mathsf{CM}(\mathsf{ck}_2, M_1) = \mathsf{CM}(\mathsf{ck}_1 + \mathsf{ck}_2, M_1)$

Note that if a prime p divides the order of $\mathcal{M}$ and $\mathcal{K}$ then for all $x \in \mathbb{F}_p$ it holds that

$$\mathsf{CM}(x \cdot \mathsf{ck}, M_1) = \mathsf{CM}(\mathsf{ck}, x \cdot M_1).$$

Inner Product. We consider inner products that map two vectors to a group in a manner that satisfies bilinearity.

Definition 5.2. *A map* $\circledast : \mathcal{M}_1 \times \mathcal{M}_2 \mapsto \mathcal{M}_T$ *from two prime order groups to a third prime order group is an inner product 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* $\langle , \rangle : \mathcal{M}_1^m \times \mathcal{M}_2^m \mapsto \mathcal{M}_T$ *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 SIPP and the other pairing based protocols we have that $\circledast : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ and $A \circledast B = e(A, B)$. In this case we refer to the resulting protocols as *inner pairing product* proofs. We will also use the inner product $\circledast : \mathbb{G}_1 \times \mathbb{F} \mapsto \mathbb{G}_1$ and $A \circledast b = A^b$.

We further define an inner product commitment which consists of three doubly homomorphic commitments and an inner product that maps the first two message spaces to the third.

Definition 5.3 (Inner product commitment)**.** *Let* CM_{IP} *be a doubly homomorphic commitment with message space* $\mathcal{M} = (\mathcal{M}_1^m, \mathcal{M}_2^m, \mathcal{M}_T)$ *and key space* $\mathcal{K} = (\mathcal{K}_1^m, \mathcal{K}_2^m, \mathcal{K}_T)$ *defined for all* $m \in [2^i]_{i \in \mathbb{N}}$ *such that* $\mathcal{M}_i$ *and* $\mathcal{K}_i$ *are groups of prime order* p *for* $i \in \{1, 2, T\}$*. If there is an inner product map* $\circledast : \mathcal{M}_1 \times \mathcal{M}_2 \to \mathcal{M}_3$ *then the commitment is an* inner product commitment*.*

Let $\mathsf{CM}((\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3); (M_1, M_2, \langle M_1, M_2 \rangle)$ be a binding inner product commitment as defined above. In Fig. 4 we present the generalized inner product argument where we assume that the dimension of the message spaces is a power of two.

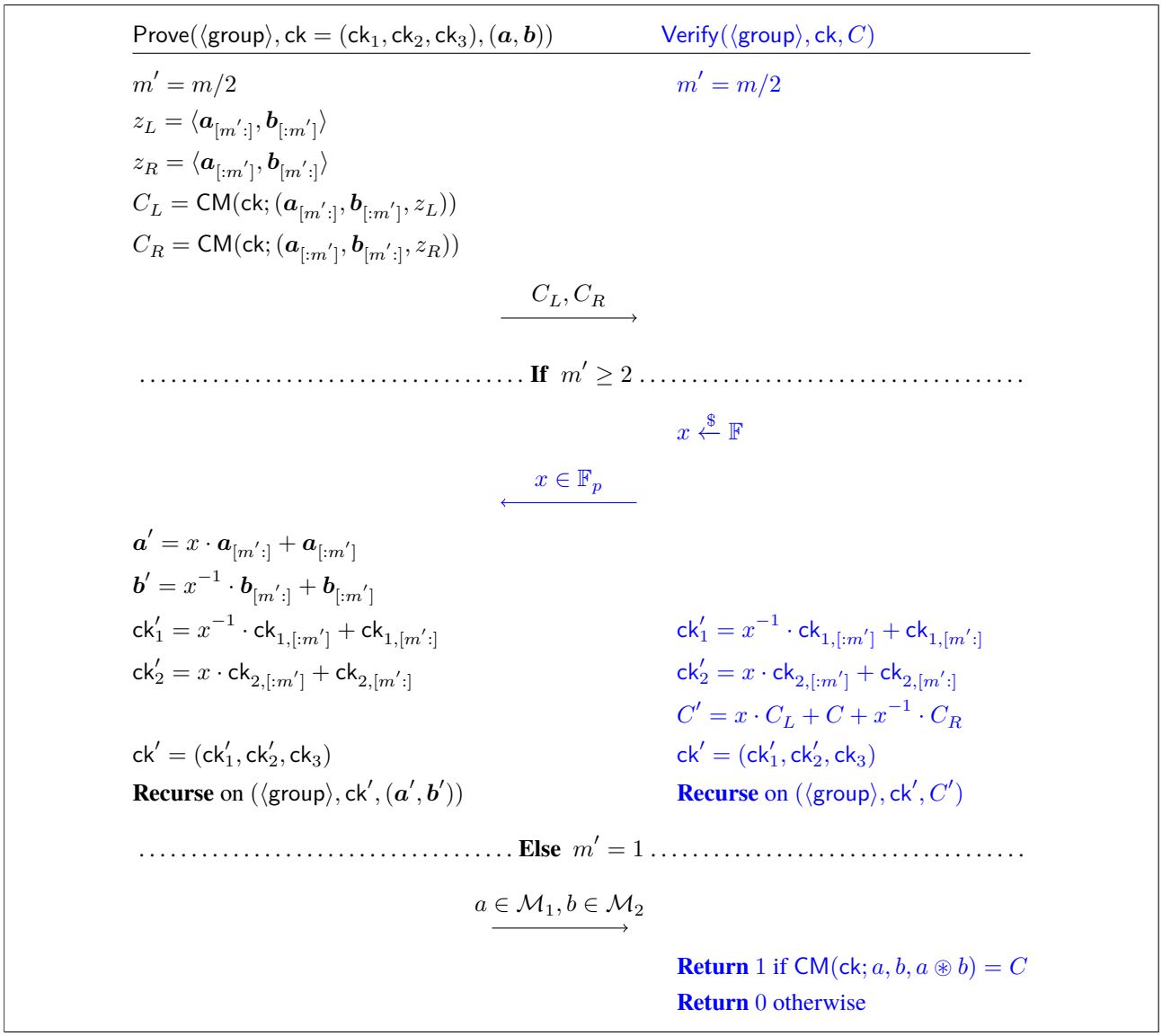


Figure 4: Generalized inner product argument.

We now show that the protocol is an argument of knowledge given a binding inner product commitment

Theorem 5.4 (Security generalized GIPA)**.** *If* $(\mathsf{Setup}, \mathsf{CM})$ *is a binding inner product commitment with* $\mathcal{M}$ *of size* $p > 2^\lambda$ *then the protocol presented in Fig. 4 has perfect completeness and computational witness-extended emulation (Definition A.1) as an argument system for the relation*

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

We prove Theorem 5.4 in Appendix B.2.

Efficiency. The communication complexity of the protocol is $2\log_2(m)$ commitments as GIPA has $\log_2(m)$ rounds. Note that in some applications part of the commitment can be computed from the verifier's input and don't need to be transmitted.

The prover produces commitments for vectors of length $m/2, m/4, m/8, \ldots$. The total length of all committed vectors is $4 \cdot n$. The verifier computes C' in each round. In total this costs $2\log_2(n)$ scalar multiplications in $\mathsf{Image}(\mathsf{CM})$. The prover and verifier also both compute ck' in each round. In total this costs $2 \cdot n$ scalar multiplications in $\mathcal{K}$. Using a technique introduced in [BBBPWM18] the verifier can use a single large multi-exponentiation in $\mathcal{K}$ to compute the final ck. In Section 6 we show how a structured reference string can reduce the verification time to something that is just logarithmic in n. As seen in Section 4 the verifier only performs a single $\circledast$ operation. This can be useful in pairing settings where this operation is significantly more expensive than scalar multiplications in $\mathcal{K}$.

Non-interactive GIPA. Using the Fiat-Shamir heuristic for logarithmic round public coin protocols[2] the GIPA protocol can be made non-interactive and publicly verifiable. To do this the challenge x is generated from a hash function Hash applied to the transcript instead of from the verifier. The verifier then checks that all challenges were computed correctly.

5.1 Instantiating GIPA

GIPA can be instantiated with different commitments. In Bulletproofs [BBBPWM18] it is instantiated with a Pedersen commitment: $\mathsf{CM}(\boldsymbol{g}, \boldsymbol{h}; \boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{g}^{\boldsymbol{a}}\boldsymbol{h}^{\boldsymbol{b}}u^{\langle \boldsymbol{a},\boldsymbol{b}\rangle}$ for $\boldsymbol{g} \in \mathbb{G}^m, \boldsymbol{h} \in \mathbb{G}^m, u \in \mathbb{G}$ and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_p^m$ for a group $\mathbb{G}$ of prime order p. The commitment is binding if the discrete logarithm assumption holds for $\mathbb{G}$. As a second example, in LMR [LMR19] GIPA is instantiated with a pairing commitment

$$\mathsf{CM}(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{w}_1, \boldsymbol{w}_2; \boldsymbol{A}, \boldsymbol{B}) = ((\boldsymbol{A} * \boldsymbol{v}_1) \cdot (\boldsymbol{w}_1 * \boldsymbol{B}),\ (\boldsymbol{A} * \boldsymbol{v}_2) \cdot (\boldsymbol{w}_2 * \boldsymbol{B}),\ \boldsymbol{A} * \boldsymbol{B})$$

As pointed out 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. This allows us to formulate SIPP as an instantiation of GIPA. The commitment is essentially the identity function and is perfectly binding.

SIPP Commitment. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ be a bilinear group. We denote the inner pairing product by

$$* : \mathbb{G}_1^m \times \mathbb{G}_2^m \to \mathbb{G}_T,\ \boldsymbol{A} * \boldsymbol{B} = \prod_{i=1}^{m} e(A_i, B_i).$$

We specify a commitment $(\mathsf{SIPP.Setup}, \mathsf{SIPP.CM})$ which is an inner product commitment. The message space is $\mathsf{SIPP}.\mathcal{M} = \mathbb{G}_1^m \times \mathbb{G}_2^m \times \mathbb{G}_T$ and homomorphic under (the respective) group multiplication. The key space is $\mathbb{F}^m \times \mathbb{F}^m \times \mathbb{F}$ and is homomorphic under addition. The commitment algorithm works as follows:

$$\mathsf{SIPP.CM}((\boldsymbol{a}, \boldsymbol{b}, z), (\boldsymbol{A}, \boldsymbol{B}, Z)) = \left((A_0^{a_0}, \ldots, A_{m-1}^{a_{m-1}}), (B_0^{b_0}, \ldots, B_{m-1}^{m-1}), Z^z\right)$$

In our instantiation $\mathsf{Setup}(1^\lambda)$ is a deterministic algorithm that simply sets all keys to be the multiplicative identity 1.

Observe that this commitment scheme is perfectly binding - indeed it is injective and this directly implies perfect binding. As a direct corollary of Theorem 5.4 and we can reprove that SIPP is an interactive proof.

Corollary 5.5. SIPP *is an interactive proof with perfect completeness and statistical witness extended emulation.*

Proof. The commitment scheme $(\mathsf{SIPP.Setup}, \mathsf{SIPP.CM})$ is a perfectly binding inner product commitment scheme. Thus security follows from Theorem 5.4. □

[2]In the random oracle model the heuristic is secure for constant-round protocols and for multi-round protocols satisfying *soundness against restoration attacks* functions [FS86; BCS16; CCHLRR18]

IPP an improvement on [LMR19]. GIPA also directly yields an improvement to the protocol presented in [LMR19]. Their commitment uses a commitment key of length $4 \cdot m$ to commit to 2 vectors of length m. Instead we can use the commitment scheme of [AFGHO16] which is also secure under SXDH but uses only half of the commitment key. We define the IPP commitment scheme with respect to $\mathcal{M} = \mathbb{G}_1^m \times \mathbb{G}_2^m \times \mathbb{G}_T$, $\mathcal{K} = \mathbb{G}_2^m \times \mathbb{G}_1^m \times \mathbb{F}$, and $\mathsf{Image}(\mathsf{CM}) = \mathbb{G}_T^3$ as follows:

$$\frac{\mathsf{IPPSetup}(1^\lambda) \mapsto \mathsf{ck}}{\langle \mathsf{group} \rangle \xleftarrow{\$} \mathsf{SampleGrp_3}(1^\lambda)} \quad \frac{\mathsf{IPPCM}(\mathsf{ck};\ \boldsymbol{A}, \boldsymbol{B}, Z) \mapsto c}{\text{return } (\boldsymbol{A} * \boldsymbol{v}, \boldsymbol{w} * \boldsymbol{B}, Z)}$$
$$\boldsymbol{w}, \boldsymbol{v} \xleftarrow{\$} \mathbb{G}_1^m \times \mathbb{G}_2^m$$
$$\text{return } (\langle \mathsf{group} \rangle, \boldsymbol{v}, \boldsymbol{w}, 1)$$

We see that the message space and the key space and the commitment space are all doubly homomorphic. Thus this commitment is an inner product commitment. Using the commitment to instantiate GIPA we get an inner pairing product proof or IPP. This protocol has the same proof size [LMR19] and the smaller commitment key reduces the prover and verifier time. In Section 6 we show how a structured setup can further reduces the verifier time to be only logarithmic in the length of the committed vectors.

6 Efficienctly verifiable TIPP argument with an SRS

We present a protocol for showing that two committed vectors of group elements $\boldsymbol{A} \in \mathbb{G}_1^m$ and $\boldsymbol{B} \in \mathbb{G}_2^m$ have a certain pairing product $C = \boldsymbol{A} * \boldsymbol{B} \in \mathbb{G}_T$. This is done using an inner pairing product argument which uses significantly less communication than communicating $\boldsymbol{A}, \boldsymbol{B}$. First, the prover and verifier reduce the m equations to one using a random linear combination. The prover commits to all A_i, B_i and the verifier generates a challenge $r \leftarrow \mathbb{F}$. The prover then convinces the verifier that $\prod_{i=1}^m e(A_i^{r^{2i}}, B_i) = Z$. This can be done efficiently using an instantiation of GIPA, called trusted inner pairing product (TIPP). We construct TIPP using an updatable reference string [GKMMM18] and a recent observation by Bowe et al.[BGH19] we obtain a verifier that can efficiently check that $\mathsf{ck}_{\mathsf{final}}$ was computed correctly. We show in Section 8 how TIPP can be applied to aggregate pairing based SNARKs such as the super-efficient Groth16 [Gro16] SNARKs.

6.1 A doubly homomorphic commitment with a structured key

We design a commitment scheme for our aggregation argument that makes use of structured generators. This commitment scheme can be seen as an extension of the pairing based commitment scheme introduced in [AFGHO16]. We commit to one vector of group elements in $\mathbb{G}_1$ and one vector of group elements in $\mathbb{G}_2$. The commitment is simply the pairing product of the group elements with a vector of structured group elements in the other source group (i.e., $T \leftarrow \boldsymbol{A} * \boldsymbol{v}$ and $U \leftarrow \boldsymbol{w} * \boldsymbol{B}$ for $\boldsymbol{w} \in \mathbb{G}_1^m, \boldsymbol{v} \in \mathbb{G}_2^m$ defined in the SRS). The structure of these SRS elements is carefully chosen to conserve the binding property of the commitment scheme.

The setup algorithm samples pair and $\alpha, \beta \in \mathbb{F}$ and returns the commitment key

$$\mathsf{ck} = (\mathsf{pair},\ \boldsymbol{w} = \left[g^{\alpha^{2i}}\right]_{i=0}^{m-1} \text{ and } \boldsymbol{v} \leftarrow \left[h^{\beta^{2i}}\right]_{i=0}^{m-1}).$$

The commitment algorithm $\mathsf{CM} : \mathbb{G}_1^m \times \mathbb{G}_2^m \mapsto \mathbb{G}_T^2$ behaves as follows

$$\mathsf{CM}(\boldsymbol{v}, \boldsymbol{w};\ \boldsymbol{A}, \boldsymbol{B}) := (\boldsymbol{A} * \boldsymbol{v}, \boldsymbol{w} * \boldsymbol{B}) = (T, U)\ .$$

The proving SRS contains additional values

$$g^{\beta}, h^{\alpha}, \{g^{\alpha^i}, h^{\beta^i}\}_{i=0}^{2m-2}.$$

It follows directly from the q-ASDBP assumption (Assumption 2) that these commitments are binding with respect to both the commitment key and the proving SRS. Observe that we do not use odd powers of α in $\mathbb{G}_1$, i.e. $g^{\alpha^{2i+1}}$. This is to prevent the value h^{α} given in the SRS being used to find collisions in the commitment (note that $e(g, h^{\alpha}) \cdot e(g^{\alpha}, h^{-1}) = 1_{\mathbb{T}}$).

6.2 Construction

The TIPP relation. 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 commitments to $\boldsymbol{A}$ and $\boldsymbol{B}$, and Z is the inner pairing product $Z = \boldsymbol{A}^{\boldsymbol{r}} * \boldsymbol{B}$ with respect to some public vector of field elements $\boldsymbol{r}$. More formally, the relation $\mathcal{R}_{\mathsf{pair}}$ is defined by

$$\left\{ \begin{array}{c} \left(\ \langle \mathsf{group} \rangle, \ \boldsymbol{w} \in \mathbb{G}_1^m, \ \boldsymbol{v} \in \mathbb{G}_2^m, \ T, U, Z \in \mathbb{G}_T, \ r \in \mathbb{F} \ ; \quad \boldsymbol{A} \in \mathbb{G}_1^m, \ \ \boldsymbol{B} \in \mathbb{G}_2^m \ \right) : \\ T = \boldsymbol{A} * \boldsymbol{v} \ \wedge \ U = \boldsymbol{w} * \boldsymbol{B} \ \wedge \ \{A'_i = A_i^{r^{2i}}\}_{i=0}^{m-1} \ \wedge \ Z = \boldsymbol{A}' * \boldsymbol{B} \end{array} \right\},$$

where $\langle \mathsf{group} \rangle$ is the group description and $\boldsymbol{v}, \boldsymbol{w}$ form the reference string. Looking ahead, the inclusion of the $\boldsymbol{r}$ into the relation facilitates more simple use of our argument as a subroutine when building aggregators. In such constructions (e.g., the SNARK proof aggregator we build in Section 8) many pairing equations must be collapsed into one, and $\boldsymbol{r}$ is set to the vector of exponents chosen by the verifier to do so. 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 $\boldsymbol{v}^{\boldsymbol{r}^{-1}}$. When using our arguments to build aggregators as we are describing, the aggregator first sends T, U, then the verifier chooses $\boldsymbol{r}$, and then the aggregator and verifier engage in the rest of the protocol using $\boldsymbol{v}' = \boldsymbol{v}^{\boldsymbol{r}^{-1}}$.

Below we describe our algorithm for proving TIPP. Soundness follows for algebraic adversaries from the q-ASDBP and the q-SDH assumptions and the algorithm is proven secure in Theorem 6.1.

Setup: The TIPP setup algorithm takes as input $\alpha, \beta \in \mathbb{F}$ and outputs the proving key

$$\mathsf{srs.p} \leftarrow \left(\ \mathsf{pair}, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^{2m-2}}, h^{\beta}, \ldots, h^{\beta^{2m-2}} \ \right)$$

and the verifying key $\mathsf{srs.v} \leftarrow (\mathsf{pair}, g^{\beta}, h^{\alpha})$.

Initialise: The prover and verifier set the TIPP instance to be $T = A * \boldsymbol{v}$, $U = \boldsymbol{w} * \boldsymbol{B}$, $Z = \boldsymbol{A}^{\boldsymbol{r}} * \boldsymbol{B}$, r for $\boldsymbol{r} = (1, r^2, \ldots, r^{2m-2})$. The prover and verifier rescale the commitment key $\boldsymbol{v}' = \boldsymbol{v}^{\boldsymbol{r}^{-1}}$ such that $v'_i = v_i^{r^{-1}}$. The prover now aims to convince the verifier that they know an opening $\boldsymbol{A}', \boldsymbol{B}$ to T, U under the commitment key $(\boldsymbol{v}', \boldsymbol{w})$ such that $Z = \boldsymbol{A}' * \boldsymbol{B}$.

Recurse: The prover and verifier run the GIPA protocol from Fig. 4 with respect to the doubly homomorphic commitment

$$\mathsf{CM}((\boldsymbol{v}', \boldsymbol{w}, 1), (\boldsymbol{A}', \boldsymbol{B}, Z)) = (\boldsymbol{A}' * \boldsymbol{v}', \ \boldsymbol{w} * \boldsymbol{B}, Z) = (T, U, Z).$$

We explain in more detain below. For ease of exposition we reset $\boldsymbol{A}' = \boldsymbol{A}$ and $\boldsymbol{v}' = \boldsymbol{v}$. In each round we "fold" our input vectors $\boldsymbol{A}, \boldsymbol{B}$ and our commitment keys $\boldsymbol{v}, \boldsymbol{w}$ into a new vectors $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}, \boldsymbol{w}$ of length $m' = m/2$ such that $T' = \boldsymbol{A}' * \boldsymbol{v}'$, $U' = \boldsymbol{w}' * \boldsymbol{B}'$ and $Z' = \boldsymbol{A}' * \boldsymbol{B}'$. The prover first computes

$$T_L = (\boldsymbol{A}_{[m':]} * \boldsymbol{v}_{[m':]}),\ U_L = \boldsymbol{w}_{[m':]} * \boldsymbol{B}_{[:m']},\ Z_L = \boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']}$$

and

$$T_R = (\boldsymbol{A}_{[:m']} * \boldsymbol{v}_{[:m']}),\ U_R = \boldsymbol{w}_{[:m']} * \boldsymbol{B}_{[m':]},\ Z_R = \boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]}$$

sending them as commitments to the verifier. The verifier samples $x \xleftarrow{\$} \mathbb{F}$ and sends x to the prover. The prover and the verifier then each set

$$Z' = Z_L^x \cdot Z \cdot Z_R^{x^{-1}} \quad .$$

and the prover sets

$$\boldsymbol{A}' = \boldsymbol{A}_{[m':]}^x \circ \boldsymbol{A}_{[:m']},\ \boldsymbol{B}' = \boldsymbol{B}_{[m':]}^{x^{-1}} \circ \boldsymbol{B}_{[:m']}, \boldsymbol{v}' = \boldsymbol{v}_{[m':]}^{x^{-1}} \circ \boldsymbol{v}_{[:m']},\ \boldsymbol{w}' = \boldsymbol{w}_{[m':]}^x \circ \boldsymbol{w}_{[:m']}$$

The protocol then recurses on $(\boldsymbol{A}, \boldsymbol{B}, Z) = (\boldsymbol{A}', \boldsymbol{B}', Z')$ until the final round, where $m = 1$.

Final Round: In that round the prover sends an opening $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$, the final commitment keys $w \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and proves to the verifier using the protocol in Fig. 5 that (w, v) have been computed correctly. The verifier accepts the proof if it is convinced that w, v are well formed and if

$$T = e(A, v),\ U = e(w, B),\ Z = e(A, B)$$

and otherwise rejects it.

6.2.1 Final Commitment Keys

Having run the recursive argument for TIPP, we next require a proving system for the relation

$$\mathcal{R}_{\mathsf{ck}} = \left((\mathsf{srs}, r, (w, v), \boldsymbol{x};\ \cdot)\ :\ \begin{array}{l} w = g^{f_w(\alpha)} \text{ for } f_w(X) = \prod_{j=0}^{\ell} \left(x_{\ell-j} + X^{2^{j+1}} \right), \\ v = h^{f_v(\beta)} \text{ for } f_v(X) = \prod_{j=0}^{\ell} \left(x_{\ell-j}^{-1} + r^{-1} X^{2^{j+1}} \right) \end{array} \right)$$

For this, we make direct use of the polynomial commitment scheme of Kate et al. [KZG10]. Our protocol for proving R_{ck} is given in Fig. 5. There are three rounds: (1) the prover sends the claimed evaluations w and v; (2) the verifier sends a random challenge z; and (3) the prover sends an argument that should convince the verifier if and only if w and v are correct.

We introduce the polynomials

$$f_w(X) = \prod_{j=0}^{\ell} \left(x_{\ell-j} + X^{2^{j+1}} \right) \text{ and } f_v(X) = \prod_{j=0}^{\ell} \left(x_{\ell-j}^{-1} + r^{-1} X^{2^{j+1}} \right) \quad .$$

We show in Proposition B.1 that when w and v are computed correctly we have that $w = g^{f_w(\alpha)}$ and $v = h^{f_v(\beta r^{-1})}$. Our outsourcing argument can thus be seen as a form of polynomial commitment in which we open w, v at a random point and check that the evaluation is as expected.

$\underline{\mathsf{Setup}((\mathsf{pair}, m); \alpha, \beta)}$
$\mathsf{srs.p} \leftarrow \left(\mathsf{pair}, \{g^{\alpha^i}, h^{\beta^i}\}_{i=0}^{2m-2}\right)$
$\mathsf{srs.v} \leftarrow (\mathsf{pair}, g^{\beta}, h^{\alpha})$
return $(\mathsf{srs.p}, \mathsf{srs.v})$

$\underline{\mathsf{Prove}(\mathsf{srs.p}, r, \boldsymbol{x}, z)}$
$f_w(X) \leftarrow \prod_{j=0}^{\ell}(x_{\ell-j} + X^{2^{j+1}})$
$f_v(X) \leftarrow \prod_{j=0}^{\ell}(x_{\ell-j}^{-1} + (r^{-1}X)^{2^{j+1}})$
$P \leftarrow g^{(f_w(\alpha)-f_w(z))/(\alpha-z)}$
$Q \leftarrow h^{(f_v(\beta)-f_v(z))/(\beta-z)}$
return (P, Q)

$\underline{\mathsf{ComputeFinal}(\mathsf{srs.p}, r, \boldsymbol{x})}$
$w \leftarrow g^{\prod_{j=0}^{k}(x_j+\alpha^{2^{j+1}})}$
$v \leftarrow h^{\prod_{j=0}^{k}(x_j^{-1}+(r^{-1}\beta)^{2^{j+1}})}$
return (w, v)

$\underline{\mathsf{Verify}(\mathsf{srs.v}, (r, \boldsymbol{x}, w, v), z, (P, Q))}$
check $e(wg^{-f_w(z)}P^z, h) = e(P, h^{\alpha})$
check $e(g, vh^{-f_v(z)}) = e(g^{\beta-z}, Q)$
return 1 if all checks pass, else return 0

Figure 5: The arguments required by the outsourcing protocol for the relation R_{ck} which determines that the format of the final commitment keys in the verifier's IPP protocol have been computed correctly. First the prover runs $\mathsf{ComputeFinal}(\mathsf{srs.p}, r, \boldsymbol{x})$ where $\boldsymbol{x}$ consists of the verifier's challenges in the IPP protocol. On input of (w, v) the verifier sends a random challenge z. The prover runs Prove to get a proof of correctness (P, Q). Finally, the verifier checks that (P, Q) is a verifying proof for (w, v) with respect to the challenge z.

Below we describe our protocol for $\mathcal{R}_{\mathsf{ck}}$ which is given formally in Fig. 5. We formally prove the security of this argument system in Lemma B.2 in the algebraic group model.

Step 1. The prover sends the claimed evaluations $w \in \mathbb{G}_1$, $v \in \mathbb{G}_2$. The verifier responds with a random challenge $z \in \mathbb{F}$.

Step 2. The prover sets

$$\phi_w(X) = \frac{f_w(X) - f_w(z)}{X - z} \qquad \phi_v(X) = \frac{f_v(X) - f_v(z)}{X - z}$$

and returns $P = g^{\phi_w(\alpha)}$ and $Q = h^{\phi_v(\beta)}$ to the verifier.

Step 3. The verification algorithm accepts if and only if

$$e\left(w \cdot g^{-f_w(z)}P^z, h\right) = e\left(h^{\alpha}, P\right) \quad \wedge \quad e\left(g, v \cdot h^{-f_v(z)}\right) = e\left(g^{\beta-z}, Q\right) \ .$$

6.3 Efficiency

We provide a detailed breakdown of the communication and computation complexity of the protocol in Table 2. The provers 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 T_L, T_R, U_L, U_R, Z_L, Z_R values 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. The new $\boldsymbol{v}', \boldsymbol{w}'$ require m exponentiations in $\mathbb{G}_1$ and m in $\mathbb{G}_2$. The new $\boldsymbol{A}', \boldsymbol{B}'$

values require an additional m exponentiations in each source group. The prover computes P using $2m$ group exponentiations in $\mathbb{G}_1$ and Q using $2m$ group exponentiations in $\mathbb{G}_2$. 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 recusive argument, and 2 $\mathbb{G}_1$ elements and 2 $\mathbb{G}_2$ elements from the final-generator argument (including 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. Computing the T', U', Z' values in the recursive argument requires $6\log(m)$ exponentiations in $\mathbb{G}_T$. They also compute $f_w(z)$ and $f_v(z)$ in the final commitment key argument which costs $\ell = \log_2(m)$ field multiplications and additions.

Theorem 6.1 (Computational witness-extended emulation of TIPP)**.** *The protocol defined in Section 6.2 for the NP relation $\mathcal{R}_{\mathsf{pair}}$ has computational witness-extended emulation (Definition A.1) against algebraic adversaries under m-ASDBP and $2m$-SDH.*

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 m-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 5.4 an adversary that breaks witness-extended emulation 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 Lemma B.2. □

7 Pairing-based polynomial commitment schemes

In this section we introduce a polynomial commitment scheme that is built using a combination of generalized inner product arguments. We discuss how to instantiate our scheme both with a transparent setup and with a structured setup. We also, in Section 7.6, demonstrate how to make the commitments hiding for zero-knowledge applications. Our polynomial commitment scheme supports univariate and bivariate polynomials.

We first describe the two-tiered commitment scheme of Groth [Gro11]. Second we describe how to evaluate the first tier of the commitment using a multiexponentiation inner pairing product argument. Third we describe how to evaluate the second tier of the commitment using a discrete-log inner product argument. Finally we put the two tiers together and provide the full evaluation algorithm in Section 7.4. At each step we first describe our transparent variant and we second describe our structured setup variant.

In Section 7.6 we discuss how to extend our scheme to achieve a hiding commitment with a zero-knowledge evaluation.

7.1 Commitment algorithm

7.1.1 Transparent variant

In Figure 6 we describe our polynomial commitment scheme, which is derived from the generalized two-tiered commitment scheme from Groth [Gro11]. Let $f(X, Y)$ be a polynomial of degree $m-1$ in X and $\ell-1$ in Y. Our commitment key then consists of ℓ randomly chosen generators in $\mathbb{G}_1$ and m randomly chosen generators in $\mathbb{G}_2$.

$$\mathsf{ck} = (g_0, \ldots, g_{\ell-1}) \in \mathbb{G}_1^{\ell}\ ,\ (v_0, \ldots, v_{m-1}) \in {\mathbb{G}_2}^{m}\ .$$

To commit to a polynomial $f(X, Y) = \sum_{i=0}^{m-1} f_i(Y)X^i$ the committer first computes m generalized Pedersen commitments $A_0, ..., A_{m-1}$ to $f_0(Y), \ldots, f_{\ell-1}(Y)$. We set

$$A_i = \mathsf{PedersenCommit}(a_{i,0}, \ldots, a_{i,\ell-1}) = g_0^{a_{i,0}} \ldots g_{\ell-1}^{a_{i,\ell-1}}.$$

where $f_i(Y) = \sum_{j=0}^{\ell-1} a_{i,j}Y^j$. The committer then computes the pairing commitment to the Pedersen commitments

$$T = \mathsf{PairingCommit}(A_0, \ldots, A_{m-1}) = \prod_{i=0}^{m-1} e(A_i, v_i).$$

Thus

$$T = e(g_0^{a_{0,0}} \ldots g_{\ell-1}^{a_{0,\ell-1}}, v_0) \cdots e(g_0^{a_{m-1,0}} \ldots g_{\ell}^{a_{m-1,\ell-1}}, v_{m-1})$$

and this commitment is binding under the q-DBP assumption and the DL assumption.

$\underline{\mathsf{PC.Setup}(1^\lambda, \ell, m)}$:
$\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp_3}(1^\lambda)$
$g_0, \ldots, g_\ell \xleftarrow{\$} \mathbb{G}_1$
$v_0, \ldots, v_{m-1} \xleftarrow{\$} \mathbb{G}_2$
$\mathsf{ck} \leftarrow (\langle \mathsf{group} \rangle, \boldsymbol{g}, \boldsymbol{v}, h)$
Return ck

$\underline{\mathsf{PC.CM}(\mathsf{ck}, f(X, Y))}$:
for $0 \le i \le m-1$:
  $A_i \leftarrow \prod_{j=0}^{\ell-1} g_j^{a_{i,j}}$
$T \leftarrow \prod_{i=0}^{m-1} e(A_i, v_i)$
Return T

Figure 6: A polynomial commitment scheme with a transparent setup.

7.1.2 Structured setup variant

Our structured variation of the commitment scheme in Figure 7 runs identically to the transparent variation, except that the commitment key is structured. We set

$$\mathsf{ck} = \begin{array}{l} (g_0, \ldots, g_{\ell-1}) = (g, g^{\alpha}, \ldots, g^{\alpha^{\ell-1}}) \in \mathbb{G}_1^{\ell} \\ (v_0, \ldots, v_{m-1}) = (h, h^{\beta^2}, \ldots, h^{\beta^{2m-2}}) \in {\mathbb{G}_2}^{m}. \end{array}$$

To commit to a polynomial $f(X, Y) = \sum_{i=0}^{m-1} f_i(Y)X^i$ the committer computes m KZG polynomial commitments $A_0, ..., A_{m-1}$ to $f_0(Y), \ldots, f_{m-1}(Y)$:

$$A_i = \mathsf{KZGCommit}(a_{i\ell}, \ldots, a_{(i+1)\ell-1}) = g_0^{a_{i\ell}} \ldots g_{\ell-1}^{a_{(i+1)\ell-1}} = g^{\sum_{j=0}^{\ell-1} a_{i\ell+j}\alpha^j}.$$

The committer then computes the pairing commitment to the KZG commitments

$$T = \mathsf{PairingCommit}(A_0, \ldots, A_{m-1}) = \prod_{i=0}^{m-1} e(A_i, v_i) = \prod_{i=0}^{m-1} e(A_i, h^{\beta^{2i}})$$

Thus

$$T = e(g,h)^{\sum_{i,j=0}^{m-1,\ell-1} a_{i,j}\alpha^j\beta^{2i}}$$

and this commitment is binding under the q-ASDBP assumption and the q-SDH assumption.

$\underline{\mathsf{PC.Setup}(1^\lambda, \ell, m)}$:
$\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp_3}(1^\lambda)$
$\alpha, \beta \xleftarrow{\$} \mathbb{F}$
$g_0, \ldots, g_\ell \leftarrow g, g^\alpha, \ldots, g^{\alpha^{\ell-1}}$
$v_0, \ldots, v_{m-1} \leftarrow h, h^{\beta^2}, \ldots, h^{\beta^{2m-2}}$
$\mathsf{ck} \leftarrow (\langle \mathsf{group} \rangle, \boldsymbol{g}, \boldsymbol{v})$
Return ck

$\underline{\mathsf{PC.CM}(\mathsf{ck}, f(X,Y))}$:
for $0 \le i \le m-1$:
  $A_i \leftarrow \prod_{j=0}^{\ell-1} g_j^{a_{i,j}}$
$T \leftarrow \prod_{i=0}^{m-1} e(A_i, v_i)$
Return T

Figure 7: A polynomial commitment scheme with a structured setup.

7.2 Tier 1 evaluation from a multiexponentiation IPP argument

In the first layer of our evaluation algorithm the prover is required to demonstrate that given a commitment T, a group element $A \in \mathbb{G}_1$, and an evaluation point x, the prover knows $\boldsymbol{A}$ such that T is a pairing commitment to $\boldsymbol{A}$ and $A = \prod_{i=0}^{m-1} A_i^{x^i}$. To do this we make use of a more generalized multiexponentiation IPP algorithm (MIPP): for any given commitment

$$T = \boldsymbol{A} * \boldsymbol{v} = e(A_0, v_0) \cdots e(A_{m-1}, v_{m-1})$$

and vector of field elements $\boldsymbol{b} \in \mathbb{F}^m$, the multiexponentiation argument demonstrates that

$$A = \langle \boldsymbol{A}, \boldsymbol{b} \rangle = A_0^{b_0} \cdots A_{m-1}^{b_{m-1}}$$

has been correctly computed.

$$\mathcal{R}_{\mathsf{MIPP}} = \left\{ \begin{array}{l} \Big(\langle \mathsf{group} \rangle, \boldsymbol{v} \in \mathbb{G}_2^{m+1},\ \ (T \in \mathbb{G}_T,\ \ A \in \mathbb{G}_1,\ \ \boldsymbol{b} \in \mathbb{F}^m)\ ; \quad \boldsymbol{A} \in \mathbb{G}_1^m \Big) : \\ \quad T = \boldsymbol{A} * \boldsymbol{v} \ \wedge\ A = \langle \boldsymbol{A}, \boldsymbol{b} \rangle \end{array} \right\}$$

7.2.1 Transparent variant

Below we describe our algorithm for proving $\mathsf{MIPP}_{\mathsf{trans}}$. Soundness follows from the q-DBP assumption and is proven in Lemma B.3.

Setup: The transparent MIPP setup outputs the commitment key $\boldsymbol{v}$ as well as a random value $\hat{h} \in \mathbb{G}_2$.

Initialize: The verifier sends a random challenge c. The prover and verifier set $Z = T \cdot e(A, \hat{h}^c)$. The prover now aims to convince the verifier that they know an opening $\boldsymbol{A} \| A$ to Z under the commitment key $\boldsymbol{v} \| \hat{h}^c$ such that $A = \boldsymbol{A}^{\boldsymbol{b}}$.

Recurse: The prover and verifier run the GIPA protocol from Figure 4 with respect to the commitment

$$\mathsf{CM}((\boldsymbol{v}, \boldsymbol{1}, \hat{h}^c), (\boldsymbol{A}, \boldsymbol{b}, A)) = ((\boldsymbol{A}||A) * (\boldsymbol{v}||\hat{h}^c),\ \boldsymbol{b}) = (Z, \boldsymbol{b}).$$

We explain in more detain below. For ease of exposition we reset $\hat{h} = \hat{h}^c$. In each round we "fold" our input vectors $\boldsymbol{A}, \boldsymbol{b}$ and our commitment key $\boldsymbol{v}$ into a new vectors $\boldsymbol{A}', \boldsymbol{b}', \boldsymbol{v}'$ of length $m' = m/2$ such that $Z' = (\boldsymbol{A}' * \boldsymbol{v}') \cdot e(A', \hat{h})$ for $A' = \langle \boldsymbol{A}, \boldsymbol{b} \rangle$. The prover first computes

$$\begin{aligned} Z_L &= (\boldsymbol{A}_{[m':]} * \boldsymbol{v}_{[m':]}) \cdot e(\langle \boldsymbol{A}_{[m':]}, \boldsymbol{b}_{[m':]} \rangle, \hat{h}) \\ Z_R &= (\boldsymbol{A}_{[m':]} * \boldsymbol{v}_{[:m']}) \cdot e(\langle \boldsymbol{A}_{[:m']}, \boldsymbol{b}_{[:m']} \rangle, \hat{h}) \ , \end{aligned}$$

sending them as commitments to the verifier. The verifier samples $x \xleftarrow{\$} \mathbb{F}$ and sends x to the prover. The prover and the verifier then each set

$$Z' = Z_L^x \cdot Z \cdot Z_R^{x^{-1}} \quad \text{and} \quad \boldsymbol{b}' = x^{-1} \boldsymbol{b}_{[m':]} + \boldsymbol{b}_{[:m']} \quad \text{and} \quad \boldsymbol{v}' = \boldsymbol{v}_{[m':]}^{x^{-1}} \circ \boldsymbol{v}_{[:m']} \ .$$

and the prover sets

$$\boldsymbol{A}' = \boldsymbol{A}_{[m':]}^{x} \circ \boldsymbol{A}_{[:m']}$$

The protocol then recurses on $(\boldsymbol{A}, \boldsymbol{b}, Z) = (\boldsymbol{A}', \boldsymbol{b}', Z')$ until the final round, where $m = 1$.

Final Round: In that round the prover sends an evaluation $A \in \mathbb{G}_1$ the verifier accepts the proof if $Z = e(A, v)e(A^b, \hat{h}) = e(A, v\hat{h}^b)$ and otherwise rejects it.

Efficiency The prover must compute the Z_L, Z_R values requiring a total of $2m$ pairings. The new generators $\boldsymbol{v}'$ require m $\mathbb{G}_2$ exponentiations. The new $\boldsymbol{A}'$ values require an additional m $\mathbb{G}_1$ exponentiations. Regarding proof size, we have $2\log(m)$ $\mathbb{G}_T$ elements from the recusive argument. The verifier computes 1 pairing in the recursive protocol. Computing the Z' values in the recursive argument requires $2\log(m)$ exponentiations in $\mathbb{G}_T$. Computing the rescaled generators $\boldsymbol{v}'$ costs m $\mathbb{G}_2$ exponentiations.

7.2.2 Structured setup variant

Our algorithm for the structured setup variant of $\mathsf{MIPP}_{\mathsf{srs}}$ is proven secure against algebraic adversaries in Lemma B.4. The protocol uses the same techniques as described in Section 6: first we run the transparent version of the MIPP argument with respect to structured generators; second the prover provides the verifier with the final commitment keys; and third the prover provides a correctness argument for the final commitment keys.

Setup: The structured MIPP setup takes as input β and outputs

$$(g^{\beta}, \{h^{\beta^i}\}_{i=0}^{2m-2}, \hat{h}).$$

Of these values the verifier only requires $g^{\beta}, \hat{h}$.

Initialize: The initialisation process is the same as for the transparent MIPP. The verifier sends a random challenge c. The prover and verifier set $Z = T \cdot e(A, \hat{h}^c)$. The prover now aims to convince the verifier that they know an opening $\boldsymbol{A}||A$ to Z under the commitment key $\boldsymbol{v}||\hat{h}^c$ such that $A = \langle \boldsymbol{A}, \boldsymbol{b} \rangle$.

Recurse: The prover and verifier run the GIPA protocol from Figure 4 with respect to the commitment

$$\mathsf{CM}((\boldsymbol{v}, \boldsymbol{1}, \hat{h}^c), (\boldsymbol{A}, \boldsymbol{b}, A)) = ((\boldsymbol{A}||A) * (\boldsymbol{v}||\hat{h}^c),\ \boldsymbol{b}) = (Z, \boldsymbol{b}).$$

This structured recursion process for computing the final (A, b, Z) values run the same as for our transparent argument except that the verifier does not rescale the generators $\boldsymbol{v}$.

Final Round: In the final round the prover sends

$$v = h^{f_v(\beta)} \text{ for } f_v(Y) := \prod_{j=0}^{\ell} \left(x_{\ell-j}^{-1} + Y^{2^{j+1}} \right)$$

and by Proposition B.1 we have that v is equal to the final commitment key in the recursion. The verifier checks that $Z' = e(A, v)e(A^b, \hat{h})$. The prover then shows that v is computed correctly using our argument for R_{ck} given in Section 6. Specifically verifier then sends the prover a random challenge z. The prover sets $f_v(Y) = \prod_{j=0}^{\ell} \left(x_{\ell-j}^{-1} + Y^{2^{j+1}} \right)$ and sends a KZG proof

$$Q = h^{q(\beta)} \text{ for } q(X) = \frac{f_v(X) - f_v(z)}{X - z}$$

that v has been computed correctly. The verifier then checks that

$$e(g, vh^{-f_v(z)}) = e(g^{\beta - z}, Q)$$

and returns true if both the recursion and this check verifies. By Theorem B.2 we have that an algebraic adversary that convinces the verifier of a false v breaks the q-SDH assumption.

Efficiency Our setup consists of $2m$ $\mathbb{G}_2$ values and 1 $\mathbb{G}_1$ value (and the group description). In terms of prover computation, our recursive argument requires $2m$ pairings, m $\mathbb{G}_1$ exponentiations and m $\mathbb{G}_2$ exponentiations. Our proof that the final commitment key is correctly formed requires $2m$ $\mathbb{G}_2$ exponentiations. Regarding proof size, we have $2\log(m)$ $\mathbb{G}_T$ elements from the recusive argument and 2 $\mathbb{G}_2$ elements for setting the final commitment key (including v). The verifier computes 1 pairing in the recursive protocol. Computing the Z' values in the recursive argument requires $2\log(m)$ exponentiations in $\mathbb{G}_T$. Verifying the final commitment key v costs 2 pairings, and 1 $\mathbb{G}_1$ and 1 $\mathbb{G}_2$ exponentiations. Computing the final b' values costs m $\mathbb{F}$ multiplications (in our evaluation proofs we will show that for some values of $\boldsymbol{b}$ this value can be computed in logarithmic time).

7.3 Tier 2 evaluation from univariate polynomial commitment schemes

In our evaluation proofs we are given a commitment T to $f(X, Y)$. Evaluating this commitment at (x, y) requires two steps: in Section 7.2 we discussed our first tier for evaluating T at x to a commitment A to $f(x, Y)$; in this section we discuss our second tier for evaluating the commitment A to $f(x, y)$ at y.

7.3.1 Tranparent variant

In our transparent variant we have that our polynomial commitment A to $f(x, Y) = \sum_{j=0}^{\ell-1} a_j Y^j$ is given by

$$A = g_0^{a_0} \dots g_{\ell-1}^{a_{\ell-1}}$$

i.e., a generalized Pedersen commitment. To evaluate this commitment we thus have prover and verifier run (minor modification of) the discrete-log inner product argument from [BBBPWM18] for proving the following relation.

$$\mathcal{R}_{\mathsf{DL}} = \left\{ \begin{array}{c} \Big(\langle \mathsf{group} \rangle,\ \boldsymbol{g} \in \mathbb{G}_1^\ell,\ (A \in \mathbb{G}_1, \boldsymbol{b}, \mathsf{eval} \in \mathbb{F}^{\ell+1});\quad \boldsymbol{a}, \in \mathbb{F}^\ell\Big) : \\ A = \boldsymbol{g}^{\boldsymbol{a}} \quad \wedge \quad \mathsf{eval} = \langle \boldsymbol{a}, \boldsymbol{b} \rangle \end{array} \right\}$$

We now describe our argument for $\mathcal{R}_{\mathsf{DL}}$ which is proven secure in Lemma B.5.

Setup: The transparent univariate polynomial commitment setup outputs the commitment key $\boldsymbol{g}$ and an additional random value $u \in \mathbb{G}_1$.

Initialize: The prover sends eval to the verifier. The verifier sends a random challenge c. The prover and verifier set $P = A \cdot u^{c \cdot \mathsf{eval}}$. The prover now aims to convince the verifier that they know a vector $\boldsymbol{a}$ such that $(\boldsymbol{a}, \mathsf{eval})$ is an evaluation to P under the commitment key $(\boldsymbol{g}, u^c)$ such that $\mathsf{eval} = \langle \boldsymbol{a}, \boldsymbol{b} \rangle$.

Recurse: The prover and verifier run the GIPA protocol from Figure 4 with respect to the commitment

$$\mathsf{CM}((\boldsymbol{g}, \boldsymbol{1}, u^c), (\boldsymbol{a}, \boldsymbol{b}, y)) = (\boldsymbol{g}^{\boldsymbol{a}} u^y,\ \boldsymbol{b}) = (P, \boldsymbol{b}).$$

We explain in more detail below. For ease of notation we reset $u = u^c$. In each round "fold" the input vectors $\boldsymbol{a}, \boldsymbol{b}$ and commitment key $\boldsymbol{g}$ into a new vectors $\boldsymbol{a}', \boldsymbol{b}', \boldsymbol{g}'$ of length $m' = m/2$ such that $P' = \boldsymbol{g}'^{\boldsymbol{a}'} u^{\mathsf{eval}'}$ for $\mathsf{eval}' = \langle \boldsymbol{a}', \boldsymbol{b}' \rangle$. The prover first computes

$$P_L = \boldsymbol{g}_{[m':]}^{\boldsymbol{a}_{[:m']}} \cdot u^{\langle \boldsymbol{a}_{[:m']}, \boldsymbol{b}_{[m':]} \rangle} \quad \text{and} \quad P_R = \boldsymbol{g}_{[:m']}^{\boldsymbol{a}_{[m':]}} \cdot u^{\langle \boldsymbol{a}_{[m':]}, \boldsymbol{b}_{[:m']} \rangle}\ ,$$

sending them as commitments to the verifier. The verifier samples $x \xleftarrow{\$} \mathbb{F}$ and sends x to the prover. The prover and the verifier then each set

$$P' = P_L^x \cdot P \cdot P_R^{x^{-1}} \quad \text{and} \quad \boldsymbol{b}' = x^{-1} \boldsymbol{b}_{[m':]} + \boldsymbol{b}_{[:m']} \quad \text{and} \quad \boldsymbol{g}' = \boldsymbol{g}_{[m':]}^{x^{-1}} \circ \boldsymbol{g}_{[:m']}\ .$$

and the prover sets

$$\boldsymbol{a}' = x \boldsymbol{a}_{[m':]} + \boldsymbol{a}_{[:m']}.$$

The protocol then recurses on $(\boldsymbol{a}, \boldsymbol{b}, P) = (\boldsymbol{a}', \boldsymbol{b}', P')$ until the final round, where $m = 1$.

Final Round: In the final round the prover sends an evaluation a. The verifier accepts the proof if $P = g^a u^{a \cdot b}$ and otherwise rejects it.

Efficiency The prover must compute the P_L, P_R values requiring a total of 2ℓ $\mathbb{G}_1$ exponentiations. Computing the new generators $\boldsymbol{g}'$ require ℓ $\mathbb{G}_1$ exponentiations. In total this sums to 3ℓ $\mathbb{G}_1$ exponentiations. Regarding proof size, we have $2\log(\ell)$ $\mathbb{G}_1$ elements from the recusive argument. The verifier computes the rescaled generators $\boldsymbol{g}'$ using ℓ $\mathbb{G}_1$ exponentiations.

7.3.2 Structured setup variant

In our structured setup variant we have that our polynomial commitment A to $f(x, Y) = \sum_{j=0}^{\ell-1} a_j Y^j$ is given by

$$A = g_0^{a_0} \dots g_{\ell-1}^{a_{\ell-1}} = g^{\sum_{j=0}^{\ell-1} a_j \alpha^j}$$

i.e., a KZG polynomial commitment. To evaluate this commitment we thus have prover and verifier run the KZG algorithm for proving the following relation.

$$\mathcal{R}_{\mathsf{KZG}} = \left\{ \begin{array}{c} \left(\langle \mathsf{group} \rangle, \ \{g^{\alpha^i}\}_{i=0}^{\ell-1} \in \mathbb{G}_1^\ell, \ y, \mathsf{eval} \in \mathbb{F}^2; \quad \boldsymbol{a} \in \mathbb{F}^\ell \right) : \\ A = \boldsymbol{g}^{\boldsymbol{a}} \quad \wedge \quad \mathsf{eval} = \sum_{j=0}^{\ell-1} a_j y^j \end{array} \right\}$$

The KZG polynomial commitment scheme is extractable by Proposition B.1. Here we quickly recap the prover and verifier algorithms.

Prove: For the KZG evaluation argument, the prover computes $\Omega(Y) = \frac{f(Y)-f(y)}{Y-y}$ and sends $W = g^{\Omega(y)}$ to the verifier.

Verify: On input (A, y, eval) and proof W the verifier checks that

$$e(Ag^{-\mathsf{eval}} W^y, h) = e(W, h^\alpha) \ ,$$

and returns 1 if and only if this check passes.

7.4 Evaluation algorithm

We now demonstrate how to put together a multiexponentiation argument and a univariate polynomial commitment scheme from Sections 7.2 and 7.3 in order to evaluate our commitments

$$T = \prod_{i=0}^{m-1} e\left(\prod_{j=0}^{\ell-1} g_j^{a_{i,j}}, v_i \right) \ .$$

to polynomials

$$f(X, Y) = \sum_{i,j=0}^{m-1,\ell-1} a_{i,j} X^i Y^j = \sum_{i=0}^{m-1} f_i(Y) X^i \ .$$

We prove the following Theorem in Appendix B.4

Theorem 7.1 (Transparent Polynomial Commitment is Extractable)**.** *If there exists a bilinear group sampler* SampleGrp *that satisfies the* q-DBP *assumption in* $\mathbb{G}_2$ *and the* DL *assumption, then the scheme in Figure 8 is a polynomial commitment scheme that achieves extractability.*

Theorem 7.2 (Structured Polynomial Commitment is Extractable)**.** *If there exists a bilinear group sampler* SampleGrp *that satisfies the* q-DBP *assumption in* $\mathbb{G}_2$ *and the* q-SDH *assumption, then the scheme in Figure 9 is a polynomial commitment scheme that achieves extractability against algebraic adversaries.*

$\underline{\mathsf{EvalSetup}(\mathsf{ck})}$:
$u \xleftarrow{\$} \mathbb{G}_1^{\ell+1}$
$\hat{h} \xleftarrow{\$} \mathbb{G}_2^{m+1}$
return $\mathsf{crs} = (\mathsf{ck}, u, \hat{h})$

$\underline{\mathsf{EvalProve}(\mathsf{crs}, (T, (x, y), \mathsf{eval}), (f(X, Y), \boldsymbol{A})}$:
$A \leftarrow \prod_{i=0}^{m-1} A_i^{x^i}$
$\boldsymbol{a}' \leftarrow (\sum_{i=0}^{m-1} a_{i,0} x^i, \ldots, \sum_{i=0}^{m-1} a_{i,\ell-1} x^i)$
Run $\mathsf{MIPP}_{\mathsf{trans}}(\langle \mathsf{group} \rangle, (\boldsymbol{v}, \hat{h}), (T, A, \boldsymbol{x});\ \boldsymbol{A})$ with the verifier.
Run $\mathsf{DL}(\langle \mathsf{group} \rangle, (\boldsymbol{g}, u), (A, \boldsymbol{y}, \mathsf{eval});\ \boldsymbol{a}')$ with the verifier.

$\underline{\mathsf{EvalVerify}(\mathsf{crs}, (T, (\boldsymbol{x}, \boldsymbol{y}), \mathsf{eval}))}$:
Run $\mathsf{MIPP}_{\mathsf{trans}}(\langle \mathsf{group} \rangle, (\boldsymbol{v}, \hat{h}), (T, A, \boldsymbol{x}))$ with the prover
Run $\mathsf{DL}(\langle \mathsf{group} \rangle, (\boldsymbol{g}, u), (A, \boldsymbol{y}, \mathsf{eval}))$ with the prover
Return 1 if both inner products verify; else 0.

Figure 8: A PC evaluation scheme with a transparent setup. The EvalProve and EvalVerify algorithms are ran with the input $\boldsymbol{x} = (1, x, \ldots, x^{m-1})$ and $\boldsymbol{y} = (1, y, \ldots, y^{\ell-1})$. Evaluating costs do not include the cost to compute the polynomial evaluation.

7.4.1 Transparent Variant

We show how our transparent prover evaluates polynomial commitments in Figure 8. To evaluate a polynomial commitment T to $\mathsf{eval} = f(X, Y)$, the prover first computes a value $A = \prod_{i=0}^{m-1} A_i^{x^i} \in \mathbb{G}_1$, which is sent to the verifier. To convince the verifier that they have correctly computed A, the prover uses the MIPP argument from Section 7.2.1. The prover then computes a polynomial $f(x, Y) = \sum_{i=0}^{\ell-1} x^i f_i(Y)$ and uses the univariate polynomial commmitment argument from Section 7.3.1 to evaluate A to $f(x, y)$ at y. We then have that $\mathsf{eval} = f(x, y)$ by the soundness of the MIPP argument and the univariate polynomial evaluation argument. Efficiency results are presented in Table 3.

	Communication Complexity		
	|CRS|	$\|\pi\|$	|CM|
Transparent	$(\ell+1)\mathbb{G}_1, (m+1)\mathbb{G}_2$	$(2\log(\ell)+1)\mathbb{G}_1, 2\log(m)\mathbb{G}_T$	$1\ \mathbb{G}_T$
Structured	$(\ell+1)\mathbb{G}_1, (2m+2)\mathbb{G}_2$	$2\mathbb{G}_1, 2\mathbb{G}_2, 2\log(m)\mathbb{G}_T$	$1\mathbb{G}_T$

	Time Complexity		
	CM	Eval	Check
Transparent	$m\ell\mathbb{G}_1, m\mathbb{P}$	$(m+3\ell)\mathbb{G}_1, m\mathbb{G}_2, 2m\mathbb{P}$	$\ell\mathbb{G}_1, m\mathbb{G}_2, 2\log(m)\mathbb{G}_T, 1\mathbb{P}$
Structured	$m\ell\mathbb{G}_1, m\mathbb{P}$	$(m+2\ell)\mathbb{G}_1, 3m\mathbb{G}_2, 2m\mathbb{P}$	$3\mathbb{G}_1, 2\mathbb{G}_2, 2\log(m)\mathbb{G}_T, 5\mathbb{P}$

Table 3: Computational and communication complexity for the transparent polynomial commitment scheme in Figure 8 and for the structured setup polynomial commitment scheme in Figure 9. Here CRS includes the combined size of the commitment key and the common/structured reference strings. Here (m, ℓ) are the degrees of X and Y respectively. For univariate polynomials, $m \approx \ell \approx \sqrt{n}$. Opening computation does not include the cost to compute the polynomial evaluation.

7.4.2 Structured setup variant

$\underline{\mathsf{EvalSetup}(\mathsf{ck};\ \alpha, \beta)}$:
$\hat{h} \xleftarrow{\$} \mathbb{G}_2$
$\mathsf{srs.p} \leftarrow \left(\begin{array}{l} \mathsf{pair}, \mathsf{ck}, \{g^{\alpha^j}\}_{j=0}^{\ell-1}, \\ \hat{h}, \{h^{\beta^i}\}_{i=0}^{2m-2} \end{array} \right)$
$\mathsf{srs.v} \leftarrow (\mathsf{pair}, g^{\beta}, h^{\alpha}, \hat{h})$
return $(\mathsf{srs.p}, \mathsf{srs.v})$

$\underline{\mathsf{EvalProve}(\mathsf{srs.p}, (T, (\boldsymbol{x}, y), \mathsf{eval}), (f(X,Y), \boldsymbol{A})}$:
$A \leftarrow \prod_{i=0}^{m-1} A_i^{x^i}$
$\boldsymbol{a}' \leftarrow (\sum_{i=0}^{m-1} a_{i,0} x^i, \ldots, \sum_{i=0}^{m-1} a_{i,\ell-1} x^i)$
Run $\mathsf{MIPP}_{\mathsf{srs}}(\langle \mathsf{group} \rangle, (\boldsymbol{v}, \{h^{\beta^i}\}_{i=0}^{2m-2}, \hat{h}), (T, A, y);\ \boldsymbol{A})$ with the verifier.
Run $\mathsf{KZG}(\langle \mathsf{group} \rangle, \boldsymbol{g}, (A, y, \mathsf{eval});\ \boldsymbol{a}')$ with the verifier.

$\underline{\mathsf{EvalVerify}(\mathsf{crs}, (T, (x, y), \mathsf{eval}))}$:
Run $\mathsf{MIPP}_{\mathsf{srs}}(\langle \mathsf{group} \rangle, (g^{\beta}, \hat{h}), (T, A, x))$ with the prover
Run $\mathsf{KZG}(\langle \mathsf{group} \rangle, h^{\alpha}, (A, y, \mathsf{eval}))$ with the prover
Return 1 if both inner products verify; else 0.

Figure 9: A PC opening scheme with a structured setup.

We show how our structured setup prover evaluates polynomial commitments in Figure 9. To evaluate a polynomial commitment T to $\mathsf{eval} = f(X, Y)$, the prover first computes a value $A = \prod_{i=0}^{m-1} A_i^{x^i} \in \mathbb{G}_1$, which is sent to the verifier. To convince the verifier that they have correctly computed A, the prover uses the MIPP argument from Section 7.2.2. The prover then computes a polynomial $f(x, Y) = \sum_{i=0}^{\ell-1} x^i f_i(Y)$ and uses the KZG univariate polynomial commmitment argument from Section 7.3.2 to evaluate A to $f(x, y)$ at y. We then have that $\mathsf{eval} = f(x, y)$ by the soundness of the MIPP argument and the univariate polynomial evaluation argument.

Remark 7.3. In the MIPP argument as described, the verifier is required to recursively rescale compute a quantity $\boldsymbol{b} = (1, x, \ldots, x^{m-1})$ by the verifiers challenges. For efficiency purposes, one can instead delay computing the final value b until the end of the recursion. The final value b has the form

$$b = \prod_{j=0}^{k} (x_{k-j}^{-1} + b^{2^j})$$

where x_{k-j} is the $(k-j)$th verifier challenge. We can see this from Lemma B.1 since in each round $\boldsymbol{b}' = x^{-1}\boldsymbol{b}_{[:m']} + \boldsymbol{b}_{[m':]}$ is scaled by the same quantities as $\boldsymbol{v}$. This quantity can thus be computed in logarithmic time.

7.5 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 $p(X, Y)$. 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 squareroot values $f_i(X)$ which each have degree squareroot in d. Hence our IPP arguments are ran over a squareroot number of commitments, which is what makes our verifier complexity squareroot in the transparent setting.

7.6 A hiding polynomial commitment scheme

The inner product arguments we have described thus far are, by default, not hiding. An attacker can easily distinguish one polynomial from another, for example by computing the commitment themselves (our commitment algorithm is deterministic). When instantiating zero-knowledge arguments that use polynomial commitments, this is potentially problematic. For example, Marlin [CHMMVW20] only achieves zero-knowledge when it is instantiated using a hiding polynomial commitment scheme.

Fortunately there exists a simple generic approach to transforming homomorphic polynomial commitment schemes from non-hiding to hiding [BFS19]. First the initial commitment T to the polynomial $f(X, Y)$ needs to be randomised so as to make it hiding. Upon evaluation, the prover sends: the evaluation $f(x, y)$; a commitment R to a fully random polynomial $r(X, Y)$ (with the same degree as $f(X, Y)$); and the evaluation $r(x, y)$ to the verifier. The verifier returns a challenge c. The prover then proves that $T^c R$ evaluates to $c \cdot f(x, y) + r(x, y)$. Hoffmann et al. [HKR19] optimise this method for the prover by observing that it suffices for $r(X, Y)$ to have only a logarithmic number of non-zero coefficients.

We specify the hiding variation on our polynomial commitment schemes in Figure 10 and prove security in Lemma B.6.

Hiding Commitment: Our commitment key contains an additional value $v_m \in \mathbb{G}_2$. Our prover commits to $f(X, Y)$ by sampling $\tau \xleftarrow{\$} \mathbb{F}$ and setting

$$T = \mathsf{PC.CM}(\mathsf{ck}, f(X, Y)) \cdot e(g_0^{\tau}, v_m) = e(g_0^{\tau}, v_m) \prod_{i=0}^{m-1} e\left(\prod_{j=0}^{\ell-1} g_j^{a_{i,j}}, v_i\right).$$

Observe that this commitment is perfectly hiding because there is one group element and one randomiser.

Zero-Knowledge Evaluation Argument: Upon receiving an evaluation instance (x, y) for the commitment T, the prover samples

$$r(X, Y) = \sum_{i,j=0}^{m-1,\ell-1} r_{i,j} X^i Y^j$$

for $r_{i,j}$ sampled randomly from $\mathbb{F}$ and computes

$$R = \mathsf{PC.CM}(\mathsf{ck}, r(X, Y)).$$

The prover sends $(R, r(x, y))$ to the verifier. The verifier returns $c \xleftarrow{\$} \mathbb{F}$. The prover sends $\rho' = c\tau + \rho$ to the verifier. The prover now demonstrates to the verifier that $T^c \cdot R \cdot e(g_0^{-\rho'}, v_m)$ evaluates to $c \cdot f(x, y) + r(x, y)$.

$\underline{\mathsf{HidingPC.Setup}(1^\lambda, \ell, m)}$
return $\mathsf{PC.Setup}(1^\lambda, \ell, m+1)$

$\underline{\mathsf{ZKEvalSetup}(\mathsf{ck};\ \alpha, \beta)}$
return $\mathsf{EvalSetup}(\mathsf{ck};\ \alpha, \beta)$

$\underline{\mathsf{ZKEvalProve}(\mathsf{crs}, (T, (x,y), \mathsf{eval}), (f(X,Y), \boldsymbol{A}, \tau))}$
sample $r(X,Y) \xleftarrow{\$} \mathbb{F}[X,Y]$
sample $\rho \xleftarrow{\$} \mathbb{F}$
$R \leftarrow \mathsf{HidingPC.CM}(\mathsf{ck}, r(X,Y); \rho)$

send $(R, r(x,y))$ to the verifier
receive challenge $c \in \mathbb{F}$
send $\rho' = \tau c + \rho$ to the verifier

$T' \leftarrow T^c \cdot R \cdot e(g_0^{-\rho'}, v_m)$
$\mathsf{eval}' \leftarrow c \cdot f(x,y) + r(x,y)$
$f'(X,Y) = c \cdot f(X,Y) + r(X,Y)$
$\boldsymbol{A}' \leftarrow \boldsymbol{A}^c \circ \boldsymbol{R}$
run $\mathsf{EvalProve}(\mathsf{crs}, (T', (x,y), \mathsf{eval}'), (f'(X,Y), \boldsymbol{A}'))$

$\underline{\mathsf{HidingPC.CM}(\mathsf{ck}, v, f(X,Y))}$
$\tau \xleftarrow{\$} \mathbb{F}$
$T \leftarrow \mathsf{PC.CM}(\mathsf{ck}, f(X,Y))$
$T \leftarrow T \cdot e(g_0^\tau, v_m)$
return T

$\underline{\mathsf{ZKEvalVerify}(\mathsf{crs}, (T, (x,y), \mathsf{eval}))}$
receive (R, r) from the prover
send challenge $c \in \mathbb{F}$
receive ρ' from the prover

$T' \leftarrow T^c \cdot R \cdot e(g_0^{\rho'}, v_m)$
$\mathsf{eval}' \leftarrow c \cdot \mathsf{eval} + r$
run $\mathsf{EvalProve}(\mathsf{crs}, (T', (x,y), \mathsf{eval}'))$

Figure 10: Our hiding extension to our polynomial commitment schemes. Here ck and crs are taken to be the commitment keys and common reference strings from either the transparent or the structured setup polynomial commitment schemes. Additionally the prover inputs $\boldsymbol{A}$ and $\boldsymbol{R}$ are commitments to $f_0(Y), \ldots, f_{m-1}(Y)$ and $r_0(Y), \ldots, r_{m-1}(Y)$ respectively.

Optimisation. We specify an optimisation by Hoffmann et al. [HKR19] for reducing the number of non-zero coefficients in $r(X,Y)$. This optimisation is important because otherwise computing the commitment R would require $m\ell$ $\mathbb{G}_1$ operations, increasing our prover time not only practically but also asymptotically. Let $\mathbb{M}_m = \{0\} \cup \{2^k, 2^k+1\}_{k=0}^{m-1}$ and $\mathbb{M}_\ell = \{0\} \cup \{2^k, 2^k+1\}_{k=0}^{\ell-1}$ and observe that these sets have logarithmic size. Rather than sampling $r(X,Y)$ from the full polynomial space, instead sample

$$r_{i,j} = \begin{cases} r \xleftarrow{\$} \mathbb{F} \text{ for } (i,j) \in \{(0,1),(0,2),(0,3)\} \\ r \xleftarrow{\$} \mathbb{F} \text{ for } i \in \mathbb{M}_m \text{ and } j = 6 \\ r \xleftarrow{\$} \mathbb{F} \text{ for } i = 6 \text{ and } j \in \mathbb{M}_\ell \\ 0 \text{ otherwise} \end{cases}$$

and set $r(X,Y) = \sum_{i,j} r_{i,j} X^i Y^j$. We have chosen $r_{i,6}$ because 6 is the first value not in $\mathbb{M}_\ell$ (and similarly for $r_{6,j}$).

8 Aggregating Groth16 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 zero-knowledge Succinct Non-interactive ARguments of Knowledge (zkSNARKs) 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 the TIPP protocol run by an untrusted aggregator can be used 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. The protocol can be made non-interactive and publicly verifiable using the Fiat-Shamir transform. This results in a logarithmic sized aggregation of n SNARKs without the need for expensive pairing-friendly cycles of elliptic curves.

To date the most efficient zkSNARK is due to Groth [Gro16] and consists of 3 group elements and 1 verification equation that requires three pairings to check. We thus choose to describe our methods with respect to Groth16, but note that they apply more generally to pairing based SNARKs that do not use random oracles. We present only the Groth16 verifier and not the prover, for it is the verification equations that we aim to prove are satisfied.

The verifiers SRS is given by

$$e(g^{\rho}, h^{\tau}), [S_i = g^{(\beta u_i(x) + \alpha v_i(x) - w_i(x))/\gamma}]_{i=0}^{\ell}, h^{\gamma}, h^{\delta}$$

for secret $\rho, \tau, \gamma, \delta, x \in \mathbb{F}$ and $u_i(X), v_i(X), w_i(X)$ public polynomials that define the relation. Here we differ slightly from the notation in Groth16 to avoid overloading our TIPP notation.

Given an instance $\mathbb{x} = (a_0, \ldots, a_\ell)$ and a proof $\pi = (A, B, C)$, the Groth16 verifier checks that

$$e(A, B) = e(g^{\rho}, h^{\tau}) \cdot e(\prod_{i=0}^{\ell} S_i^{a_i}, h^{\gamma}) \cdot e(C, h^{\delta})$$

and returns 1 if and only if the check passes. Given n instances $[[a_{i,j}]_{i=0}^{\ell}]_{j=0}^{n-1}$ and proofs $[(A_j, B_j, C_j)]_{j=0}^{n-1}$, checking each equation separately requires performing $3n$ pairings and exponentiations. To reduce this computation to a single verification, 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

$$\boldsymbol{A}^{-\boldsymbol{r}} * \boldsymbol{B} = e(g^{\rho \sum_j^{n-1} r^{2j}}, h^{\tau}) \cdot e(\prod_{i=0}^{\ell} S_i^{\sum_{j=0}^{n-1} a_{i,j} r^{2j}}, h^{\gamma}) \cdot e(\prod_{j=0}^{n-1} C_j^{r^{2j}}, h^{\delta}) \ , \tag{1}$$

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.

Our aggregation protocol thus works as follows:

Step 1. The setup algorithm from Figure 5 outputs srs.P and srs.V where srs.P includes the commitment keys

$$w_i = g^{\alpha^{2j}} \text{ and } v_j = h^{\beta^{2j}} \text{ for } 0 \leq j \leq n-1$$

Step 2. The prover computes commitments $T = \boldsymbol{A} * \boldsymbol{v}$ and $U = \boldsymbol{w} * \boldsymbol{B}$ and $V = \boldsymbol{C} * \boldsymbol{v}$ and sends (T, U, V) to the verifier. The verifier responds with a challenge r. The prover sends back $Z = \boldsymbol{A}^{\boldsymbol{r}} * \boldsymbol{B}$ and $C = \prod_{j=0}^{n-1} C_j^{r^{2j}}$ (where $\boldsymbol{r} = (1, r^2, \ldots, r^{2n-2})$).

Step 3. The prover and the verifier engage in TIPP for proving that (T, U, Z) is a valid instance of $\mathcal{R}_{\mathsf{pair}}$ by running the TIPP protocol in Section 6. The prover and the verifier engage in $\mathsf{MIPP}_{\mathsf{srs}}$ for proving that $(V, C, \boldsymbol{r})$ is a valid instance of $\mathcal{R}_{\mathsf{MIPP}}$ by running the $\mathsf{MIPP}_{\mathsf{srs}}$ protocol in Section 7.2.2. If either of these protocols fail then the verifier returns 0.

Step 4. The verifier checks that

$$Z = e(g^{\rho \sum_j^{n-1} r^{2j}}, h^{\tau}) \cdot e(\prod_{i=0}^{\ell} S_i^{\sum_{j=0}^{n-1} a_{i,j} r^{2j}}, h^{\gamma}) \cdot e(V, h^{\delta})$$

and returns 1 if the check passes and 0 otherwise.

9 An aggregate signature scheme based on BLS

Boneh, Lynn, and Shacham introduced the BLS signature scheme in [BLS01]. Boneh, Gentry, Lynn, and Shacham later extended this scheme by showing how to accomplish offline aggregation of signatures and keys in [BGLS03]. This is different from aggregate Schnorr signatures [BR93], which require signers to remain online throughout the signing process. In this section we describe an alternative aggregate signature scheme based on BLS, where the verifier is required to compute just one pairing for any number of signatures. Previous aggregate signature schemes based on BLS including [BGLS03; RY07; BDN18] have constant-sized ($\mathcal{O}_\lambda(1)$) aggregate signatures and require computing $n + 1$ pairings to verifying an aggregate signature over n distinct messages. Our scheme trades off space for time, requiring the verifier compute just 1 pairing and one n-sized multi-exponentiation in each of the source groups at the cost of a logarithmic-sized signature.

The basic BLS signature scheme is given in Fig. 11. Our description is given over Type III bilinear groups as opposed to the original scheme which was described only over the less efficient Type II bilinear groups.

$\underline{\mathsf{Setup}(1^\lambda)}$	$\underline{\mathsf{KeyGen}(\langle \mathsf{group} \rangle)}$
return $\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}_3(1^\lambda)$	$\mathsf{sk} \xleftarrow{\$} \mathbb{F}$
	$\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
	return $(\mathsf{pk}, \mathsf{sk})$
$\underline{\mathsf{Sign}(\langle \mathsf{group} \rangle, \mathsf{sk}, M)}$:	$\underline{\mathsf{Verify}(\langle \mathsf{group} \rangle, \mathsf{pk}, M, \sigma)}$
$h \in \mathbb{G}_2 \leftarrow \mathsf{RO}_2(M)$	$h \in \mathbb{G}_2 \leftarrow \mathsf{RO}_2(M)$
return $\sigma \leftarrow h^{\mathsf{sk}}$	return $e(g, \sigma) = e(\mathsf{pk}, h)$

Figure 11: The BLS signature scheme where $\mathsf{RO}_2 : \{0,1\}^* \to \mathbb{G}_2$ is a random oracle.

9.1 Construction

We introduce a pair of algorithms $(\mathsf{AggSign}, \mathsf{VerifyAgg})$ that extends the BLS signature scheme into a aggregate signature scheme. These algorithms make use $\mathsf{SIPP.Prove}$ and $\mathsf{SIPP.Verify}$, respectively, as subroutines. The aggregator $\mathsf{AggSign}$ is given a group description $\langle \mathsf{group} \rangle$, a list of public keys $[\mathsf{pk}_i]_{i=1}^n$, a set of distinct messages $\{M_i\}_{i=1}^n$, and a list of signatures $[\sigma_i]_{i=1}^n$. The aggregator begins by computing $\sigma_A = \prod_i \sigma_i$. Next, they use $\mathsf{SIPP.Prove}$ to produce a proof π that

$$e(g^{-1}, \sigma_A) \cdot \prod_{i=1}^{n} e(\mathsf{apk}_i, \mathsf{RO}_2(M_i)) = 1 \ .$$

The aggregator returns $\Sigma \leftarrow (\sigma_A, \pi)$.

The verifier running $\mathsf{VerifyAgg}$ is given group description $\langle \mathsf{group} \rangle$, public keys $[\mathsf{pk}_i]_{i=1}^n$, distinct messages $\{M_i\}_{i=1}^n$, and aggregate signature Σ. The verifier then has the full inputs it needs to run $\mathsf{SIPP.Verify}$, and outputs the result of checking the inner pairing product proof.

Pseudocode. We present pseudocode for our BLS aggregation protocol in Fig. 12.

$\underline{\mathsf{AggSign}(\langle \mathsf{group} \rangle, [\mathsf{pk}_i]_{i=1}^n, \{M_i\}_{i=1}^n, [\sigma_i]_{i=1}^n) :}$
$(h_1, \ldots, h_n) \leftarrow (\mathsf{RO}_2(M_i), \ldots, \mathsf{RO}_2(M_n))$
$\sigma_A \leftarrow \prod_{i=1}^n \sigma_i$
$\pi \leftarrow \mathsf{SIPP.Prove}(\langle \mathsf{group} \rangle, g^{-1} \| \mathbf{apk}, \sigma_A \| \boldsymbol{h}, 1)$
$\Sigma \leftarrow (\sigma_A, \pi)$
return Σ

$\underline{\mathsf{VerifyAgg}(\langle \mathsf{group} \rangle, [\mathsf{pk}_i]_{i=1}^n, \{M_i\}_{i=1}^n, \Sigma) :}$
$(\sigma_A, \pi) \leftarrow \Sigma$
$(h_1, \ldots, h_n) \leftarrow (\mathsf{RO}_2(M_i), \ldots, \mathsf{RO}_2(M_n))$
return $\mathsf{SIPP.Verify}(\langle \mathsf{group} \rangle, g^{-1} \| \mathbf{apk}, \sigma_A \| \boldsymbol{h}, 1, \pi)$

Figure 12: Aggregation and verification algorithms for an aggregate signature scheme.

9.2 Efficiency

Verifying n signatures on n different messages using $\mathsf{VerifyAgg}$ requires computing n hashes and then running $\mathsf{SIPP.Verify}$. Running $\mathsf{AggSign}$ requires n hash evaluations and a call to $\mathsf{SIPP.Prove}$. Ignoring small constant terms, the prover computes a total of $2n$ pairings, n exponentiations in each source group, and n hashes. The verifier computes a single pairing, n exponentiations in each source group, $2\log(n)$ target group exponentiations, and n hashes. The aggregate signatures consists of $2\log(n)$ elements in $\mathbb{G}_T$ and 1 element in $\mathbb{G}_2$.

References

[AFGHO16] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. "Structure-Preserving Signatures and Commitments to Group Elements". In: *J. Cryptology* 29.2 (2016), pp. 363–421.

[BB08] D. Boneh and X. Boyen. "Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups". In: *J. Cryptology* 21.2 (2008), pp. 149–177.

[BBBPWM18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 315–334.

[BCCGP16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *Proceedings of the 35th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '16. 2016, pp. 327–357.

[BCCKLS09] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. "Randomizable Proofs and Delegatable Anonymous Credentials". In: *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '09. 2009, pp. 108–125.

[BCGMMW20] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. "ZEXE: Enabling Decentralized Private Computation". In: *Proceedings of the 2020 IEEE Symposium on Security and Privacy*. S&P '20. 2020.

[BCKL08] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. "P-signatures and noninteractive anonymous credentials". In: *Proceedings of the 5th Theory of Cryptography Conference*. TCC '08. 2008, pp. 356–374.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: *Proceedings of the 34th Annual International Cryptology Conference*. CRYPTO '14. Extended version at `http://eprint.iacr.org/2014/595`. 2014, pp. 276–294.

[BDK13] M. Backes, A. Datta, and A. Kate. "Asynchronous Computational VSS with Reduced Communication Complexity". In: *Proceedings of the Cryptographers' Track at the RSA Conference 2013*. CT-RSA 2013. 2013, pp. 259–276.

[BDN18] D. Boneh, M. Drijvers, and G. Neven. "Compact Multi-signatures for Smaller Blockchains". In: *Proceedings of the 24th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '18. 2018, pp. 435–464.

[BFIJSV10] O. Blazy, G. Fuchsbauer, M. Izabachène, A. Jambert, H. Sibert, and D. Vergnaud. "Batch Groth-Sahai". In: *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*. ACNS '10. 2010, pp. 218–235.

[BFS19] B. Bünz, B. Fisch, and A. Szepieniec. *Transparent SNARKs from DARK Compilers*. Cryptology ePrint Archive, Report 2019/1229. 2019.

[BG13] S. Bayer and J. Groth. "Zero-Knowledge Argument for Polynomial Evaluation with Application to Blacklists". In: *Proceedings of 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques,the Advances in Cryptology*. EUROCRYPT '13. 2013, pp. 646–663.

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

[BGLS03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps". In: *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*. 2003, pp. 416–432.

[BLS01] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing". In: *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '01. 2001, pp. 514–532.

[BR93] M. Bellare and P. Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. 1993, pp. 62–73.

[CCHLRR18] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, and R. D. Rothblum. *Fiat-Shamir From Simpler Assumptions*. Cryptology ePrint Archive, Report 2018/1004. 2018.

[CCW19] A. Chiesa, L. Chua, and M. Weidner. "On Cycles of Pairing-Friendly Elliptic Curves". In: *SIAM Journal on Applied Algebra and Geometry* 3.1 (2019), pp. 175–192.

[CDHK15] J. Camenisch, M. Dubovitskaya, K. Haralambiev, and M. Kohlweiss. "Composable and Modular Anonymous Credentials: Definitions and Practical Constructions". In: *Proceedings of the 21st International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '15. 2015, pp. 262–288.

[Che10] J. H. Cheon. "Discrete Logarithm Problems with Auxiliary Inputs". In: *Journal of Cryptology* 23.3 (2010), pp. 457–476.

[CHMMVW20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020.

[EG14] A. Escala and J. Groth. "Fine-Tuning Groth-Sahai Proofs". In: *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*. PKC '14. 2014, pp. 630–649.

[FHS19] G. Fuchsbauer, C. Hanser, and D. Slamanig. "Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials". In: *Journal of Cryptology* 32.2 (2019), pp. 498–546.

[Fis18] B. Fisch. *PoReps: Proofs of Space on Useful Data*. Cryptology ePrint Archive, Report 2018/678. 2018.

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. "The Algebraic Group Model and its Applications". In: *Proceedings of the 38th Annual International Cryptology Conference*. CRYPTO '18. 2018, pp. 33–62.

[FS86] A. Fiat and A. Shamir. "How to prove yourself: practical solutions to identification and signature problems". In: *Proceedings of the 6th Annual International Cryptology Conference*. CRYPTO '86. 1986, pp. 186–194.

[Gab19] A. Gabizon. *Improved prover efficiency and SRS size in a Sonic-like system*. Cryptology ePrint Archive, Report 2019/601. 2019.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '13. 2013, pp. 626–645.

[GGW18] K. Gurk, 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`. 2018.

[GKMMM18] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs". In: *Proceedings of the 38th Annual International Cryptology Conference*. CRYPTO '18. 2018, pp. 698–728.

[GOS06] J. Groth, R. Ostrovsky, and A. Sahai. "Perfect Non-interactive Zero Knowledge for NP". In: *Proceedings of the 25th Annual International Conference on Advances in Cryptology*. EUROCRYPT '06. 2006, pp. 339–358.

[GPS08] S. D. Galbraith, K. G. Paterson, and N. P. Smart. "Pairings for cryptographers". In: *Discrete Applied Mathematics* 156.16 (2008), pp. 3113–3121.

[Gro11] J. Groth. "Efficient Zero-Knowledge Arguments from Two-Tiered Homomorphic Commitments". In: *Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '11. 2011, pp. 431–448.

[Gro16] J. Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Proceedings of the 35th Annual International Conference on Theory and Applications of Cryptographic Techniques*. EUROCRYPT '16. 2016, pp. 305–326.

[GS08] J. Groth and A. Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: *Proceedings of the 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '08. 2008, pp. 415–432.

[GSW10] E. Ghadafi, N. P. Smart, and B. Warinschi. "Groth-Sahai Proofs Revisited". In: *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*. PKC '10. 2010, pp. 177–192.

[GW11] C. Gentry and D. Wichs. "Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions". In: *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*. STOC '11. 2011, pp. 99–108.

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

[HKR19] M. Hoffmann, M. Klooß, and A. Rupp. “Efficient Zero-Knowledge Arguments in the Discrete Log Setting, Revisited”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’19. 2019, pp. 2093–2110.

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT ’10. 2010, pp. 177–194.

[Lan08] H. Lane. “Draft standard for identity-based publickey cryptography using pairings”. In: *IEEE P1636* 3 (2008), p. D1.

[LMR19] R. W. F. Lai, G. Malavolta, and V. Ronge. *Succinct Arguments for Bilinear Group Arithmetic: Practical Structure-Preserving Cryptography*. Cryptology ePrint Archive, Report 2019/969. 2019.

[MBKM19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. “Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*. CCS ’19. 2019, pp. 2111–2128.

[MJ16] N. E. Mrabet and M. Joye. *Guide to Pairing-Based Cryptography*. 2016.

[PGHR13] B. Parno, C. Gentry, J. Howell, and M. Raykova. “Pinocchio: Nearly Practical Verifiable Computation”. In: *Proceedings of the 34th IEEE Symposium on Security and Privacy*. S&P ’13. 2013, pp. 238–252.

[Pip80] N. Pippenger. “On the Evaluation of Powers and Monomials”. In: *SIAM Journal on Computing* 9.2 (1980), pp. 230–250.

[PS16] D. Pointcheval and O. Sanders. “Short Randomizable Signatures”. In: *Topics in Cryptology - CT-RSA 2016 - The Cryptographers’ Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*. CT-RSA ’16. 2016, pp. 111–126.

[RY07] T. Ristenpart and S. Yilek. “The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks”. In: *Proceedings of the 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’07. 2007, pp. 228–245.

[Set19] S. Setty. *Spartan: Efficient and general-purpose zkSNARKs without trusted setup*. Cryptology ePrint Archive, Report 2019/550. 2019.

[WTSTW18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. “Doubly-Efficient zkSNARKs Without Trusted Setup”. In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P ’18. 2018, pp. 926–943.

[XYZW16] J. Xu, A. Yang, J. Zhou, and D. S. Wong. “Lightweight Delegatable Proofs of Storage”. In: *Proceedings of the 21st European Symposium on Research in Computer Security*. ESORICS ’16. 2016, pp. 324–343.

[XZZPS19] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. “Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation”. In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO ’19. 2019, ???–???

[ZXZS19] J. Zhang, T. Xie, Y. Zhang, and D. Song. *Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof*. Cryptology ePrint Archive, Report 2019/1482. 2019.

BEGINNING OF SUPPLEMENTARY MATERIAL

A Preliminaries

A.1 Bilinear groups

The cryptographic primitives that we construct in this paper rely on cryptographic assumptions about bilinear groups. We formalize these via a *bilinear group sampler*, which is an efficient algorithm $\mathsf{SampleGrp}$ that given a security parameter λ (represented in unary), outputs a tuple $\langle \mathsf{group} \rangle = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups with order divisible by the prime $q \in \mathbb{N}$, g generates $\mathbb{G}_1$, h generates $\mathbb{G}_2$, and $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a (non-degenerate) bilinear map.

Galbraith et al. distinguish between three types of bilinear group samplers in [GPS08]. Type I groups have $\mathbb{G}_1 = \mathbb{G}_2$ and are known as *symmetric* bilinear groups. Types II and III are *asymmetric* bilinear groups, where $\mathbb{G}_1 \neq \mathbb{G}_2$. Type II groups have an efficiently computable homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, while Type III groups do not have an efficiently computable homomorphism in either direction. Certain assumptions are provably false w.r.t. certain group types (e.g., SXDH only holds for Type III groups), and in general in this work we assume we are working with working with a Type III groups. We will write $\mathsf{SampleGrp}_3$ to explicitly denote a bilinear group sampler that outputs Type III groups.

A.1.1 Arguments of knowledge

We define interactive arguments of knowledge $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for NP relations, following Bootle et al. [BCCGP16]. Our definitions are identical except that we additionally define a probabilistic setup algorithm

$$\mathsf{Setup}(\mathcal{R};\ \tau) \mapsto \mathsf{crs}$$

that on input of a relation output by a relation generator $\mathsf{RGen}(1^\lambda)$ outputs a common reference string used as input by the prover and verifier. We denote $\mathsf{Setup}(\mathcal{R}) \xrightarrow{\$} \mathsf{crs}$ to mean that the setup algorithm first samples τ randomly and then computes $\mathsf{Setup}(\mathcal{R};\ \tau)$. The crs includes a description of the relation.

In the arguments we consider a prover Prove and a verifier Verify, both of which are probabilistic polynomial time interactive algorithms. The transcript produced by Prove and Verify when interacting on inputs s and t is denoted by $\mathsf{tr} \leftarrow \langle \mathsf{Prove}(s), \mathsf{Verify}(t) \rangle$. We write $\langle P(s), V(t) \rangle = b$ depending on whether the verifier rejects, $b = 0$, or accepts, $b = 1$.

Perfect completeness means the honest prover can always convince a verifer to accept $\mathbb{x} \in \mathcal{L}(\mathcal{R})$ provided they know a witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$.

Knowledge-Soundness. Bootle et al.'s notion of knowledge soundness is called computational witness-extended emulation. The idea is that for all efficient provers there exists an emulator that with overwhelming probability produces the same argument, and when that argument is accepting, the emulator also extracts a valid witness (again, with overwhelming probability).

Definition A.1 (Computational witness-extended emulation)**.** *The argument system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for a relation generator* $\mathsf{RGen}(1^\lambda)$ *has witness-extended emulation if for all deterministic, efficient provers* $\mathcal{P}^*$ *there exists an efficient emulator* $\mathcal{E}$ *such that for all pairs of efficient interactive adversaries* $\mathcal{A}_1, \mathcal{A}_2$ *it*

holds that

$$\left| \Pr\left[\mathcal{A}_1(\mathsf{tr}) = 1 \;\middle|\; \begin{array}{r} \mathcal{R} \xleftarrow{\$} \mathsf{RGen}(1^\lambda),\ \mathsf{crs} \leftarrow \mathsf{Setup}(\mathcal{R}) \\ (\mathbb{x}, \mathsf{st}) \leftarrow \mathcal{A}_2(\mathsf{crs}) \\ \mathsf{tr} \leftarrow \langle \mathcal{P}^*(\mathsf{st}),\ \mathsf{Verify} \rangle (\mathsf{crs}, \mathbb{x}) \end{array} \right] \right.$$

$$\left. - \Pr\left[\begin{array}{c} \mathcal{A}_1(\mathsf{tr}) = 1 \\ \wedge \\ (\mathsf{tr} \text{ is accepting} \Rightarrow (\mathbb{x}, \mathbb{w}) \in \mathcal{R}) \end{array} \;\middle|\; \begin{array}{r} \mathcal{R} \xleftarrow{\$} \mathsf{RGen}(1^\lambda),\ \mathsf{crs} \leftarrow \mathsf{Setup}(\mathcal{R}) \\ (\mathbb{x}, \mathsf{st}) \leftarrow \mathcal{A}_2(\mathsf{crs}) \\ (\mathsf{tr}, \mathbb{w}) \leftarrow \mathcal{E}^{\mathcal{O}}(\mathsf{crs}, \mathbb{x}) \end{array} \right] \right| \leq \mathrm{negl}(\lambda) \ ,$$

where tr *is the transcript of communication between* $\mathcal{P}^*$ *and* Verify*, the transcript oracle is given by* $\mathcal{O} = \langle \mathcal{P}^*(\mathsf{st}),\ \mathcal{V} \rangle (\mathsf{crs}, \mathbb{x})$*, and permits rewinding to a specific point and resuming with fresh randomness for the verifier from this point onwards.*

In the definition st can be interpreted as the state of $\mathcal{P}^*$, including its randomness. Whenever $\mathcal{P}^*$ creates a valid argument in state st, then with all but negligible probability $\mathcal{E}$ can extract a witness. This makes IPP an argument of knowledge.

By resuming with *fresh randomness* we mean that the local random tape of the verifier is replaced by a string of some prescribed length chosen uniformly at random. Therefore, it is possible that after rewinding a verifier issues the same challenge. This happens with negligible probability and in our proof of witness-extended emulation for IPP we do not deal with this case. Instead, we construct an extractor that on input a statement and a $(n_1, \ldots, n_k)$-tree of accepting transcripts (see Appendix A.4), outputs a corresponding witness with overwhelming probability, and then we apply Lemma A.9.

Zero-Knowledge. Bootle et al. define an argument as honest verifier zero-knowledge (HVZK) if there exists an efficient simulator that can simulate transcripts generated between an honest prover and an honest verifier. The Fiat-Shamir transform turns an interactive public-coin HVZK argument into a non-interactive argument with zero-knowledge even against malicious verifier. The simulator is modelled as having access to the verifier's challenges in advance.

Definition A.2 (Perfect honest verifier zero-knowledge)**.** *An argument* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for* RGen *has honest verifier zero-knowledge (HVZK) if there exists a probabilistic polynomial time simulator* $\mathcal{S}$ *such that for all interactive non-uniform polynomial time adversaries* $(\mathcal{A}_0, \mathcal{A}_1)$ *and for all* $\mathcal{R} \xleftarrow{\$} \mathsf{RGen}(1^\lambda)$ *and* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(\mathcal{R})$

$$\begin{aligned} &\Pr\left[\begin{array}{c} (\mathbb{x}, \mathbb{w}) \in \mathcal{R} \\ \wedge \\ \mathcal{A}_1(\mathsf{tr}) = 1 \end{array} \;\middle|\; \begin{array}{c} (\mathbb{x}, \mathbb{w}, \rho) \xleftarrow{\$} \mathcal{A}_0(\mathsf{crs}); \\ \mathsf{tr} \leftarrow \langle \mathsf{Prove}(\mathsf{crs}, \mathbb{x}, \mathbb{w}), \mathsf{Verify}(\mathsf{crs}, \mathbb{x};\ \rho) \rangle \end{array} \right] \\ &= \Pr\left[\begin{array}{c} (\mathbb{x}, \mathbb{w}) \in \mathcal{R} \\ \wedge \\ \mathcal{A}_1(\mathsf{tr}) = 1 \end{array} \;\middle|\; \begin{array}{c} (\mathbb{x}, \mathbb{w}, \rho) \xleftarrow{\$} \mathcal{A}_0(\mathsf{crs}); \\ \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{crs}, \mathbb{x}, \rho) \rangle \end{array} \right] \end{aligned}$$

where ρ *is the public coin randomness used by the verifier.*

A.2 Commitment Scheme

A commitment scheme for a message space $\mathcal{M}$ consists of a 2-tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{CM})$ that behave as follows:

- $\mathsf{Setup}(1^\lambda, \mathsf{aux}) \xrightarrow{\$} \mathsf{ck}$: a set-up algorithm that, given a security parameter λ (represented in unary) and an auxiliary input aux, outputs a commitment key ck.
- $\mathsf{CM}(\mathsf{ck}, M; r) \mapsto c$: a commitment algorithm that, given a commitment key ck and a message $M \in \mathcal{M}$, outputs a commitment c.

When the commitment scheme is deterministic we write $\mathsf{CM}(\mathsf{ck}, M)$. We denote $\mathsf{CM}(\mathsf{ck}, M) \xrightarrow{\$} c$ to mean that the commitment algorithm first samples r randomly and then computes $\mathsf{CM}(\mathsf{ck}, M; r)$.

We require that a commitment scheme satisfies binding i.e. no adversary can open the same commitment to two different messages.

Definition A.3 (Binding commitment)**.** *A commitment scheme* $(\mathsf{Setup}, \mathsf{CM})$ *is* binding *if for all polynomial time adversaries* $\mathcal{A}$ *the following probability is negligible in* λ

$$\Pr\left[\begin{array}{c|c} M_0, M_1 \in \mathcal{M} \wedge M_0 \neq M_1 \\ \wedge \\ \mathsf{CM}(\mathsf{ck}, M_0; r_0) = \mathsf{CM}(\mathsf{ck}, M_1; r_1) \end{array} \middle| \begin{array}{c} (\mathsf{ck}, \mathcal{M}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathit{aux}) \\ (M_0, r_0, M_1, r_1) \xleftarrow{\$} \mathcal{A}(\mathsf{ck}, \mathcal{M}) \end{array}\right]$$

We say the commitment is perfectly binding *if the above probability is exactly* 0*.*

Some of our commitment schemes must also satisfy hiding i.e. the commitment scheme is randomised such that no adversary can distinguish which of two messages a commitment contains.

Definition A.4 (Hiding commitment)**.** *A commitment scheme* $(\mathsf{Setup}, \mathsf{CM})$ *is perfectly* hiding *if for all polynomial time adversaries* $(\mathcal{A}_0, \mathcal{A}_1)$

$$\Pr\left[\begin{array}{c|c} M_0, M_1 \in \mathcal{M} \\ \wedge \\ b = b' \end{array} \middle| \begin{array}{c} b \xleftarrow{\$} \{0,1\},\ (\mathsf{ck}, \mathcal{M}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathit{aux}), \\ (M_0, M_1) \xleftarrow{\$} \mathcal{A}_0(\mathsf{ck}, \mathcal{M}),\ c \xleftarrow{\$} \mathsf{CM}(\mathsf{ck}, M_b), \\ b' \xleftarrow{\$} \mathcal{A}_2(c) \end{array}\right] = \frac{1}{2}$$

We say the commitment is perfectly binding *if the above probability is exactly* 0*.*

A.3 Polynomial Commitment Scheme

We define a polynomial commitment scheme following the definition of [BFS19], as a commitment scheme with message space $\mathbb{F}^{\leq d}[X]$, i.e. polynomials over $\mathbb{F}$ of degree bounded by d which is polynomial in λ. The commitment additionally has an interactive argument of knowledge $(\mathsf{EvalSetup}, \mathsf{EvalProve}, \mathsf{EvalVerify})$ for showing that the committed polynomial was correctly evaluated at a point.

A polynomial commitment scheme of a 5-tuple of efficient algorithms $\mathsf{PC.Setup}$, $\mathsf{PC.CM}$, $\mathsf{EvalSetup}$, $\mathsf{EvalProve}$, $\mathsf{EvalVerify}$) such that:

- $\mathsf{PC.Setup}(\mathbb{F}, d) \xrightarrow{\$} (\mathsf{ck})$: a set-up algorithm that, given a field and a maximum degree, outputs a commitment key and a proving key.
- $\mathsf{PC.CM}(\mathsf{ck}, f(X); r) \leftarrow c$: a commitment algorithm that, given a commitment key ck and a polynomial $f(X) \in \mathbb{F}[X]$ of maximum degree d, outputs a commitment c.
- $(\mathsf{EvalSetup}, \mathsf{EvalProve}, \mathsf{EvalVerify})$ is an interactive argument of knowledge with respect to the relation set

$$\mathcal{R}_{\mathsf{eval}} = \left\{ (\mathsf{ck}, c, x, \mathsf{eval};\ r,\ f(X) \in \mathbb{F}[X]) : \begin{array}{c} c = \mathsf{PC.CM}(\mathsf{ck}, f(X);\ r) \\ \wedge\ \deg(f(X)) \leq d \\ \wedge\ f(x) = \mathsf{eval} \end{array} \right\}$$

When the polynomial commitment scheme is deterministic we write $\mathsf{PC.CM}(\mathsf{ck}, f(X))$. We denote $\mathsf{PC.CM}(\mathsf{ck}, f(X)) \xrightarrow{\$} c$ to mean that the commitment algorithm first samples r randomly and then computes $\mathsf{CM}(\mathsf{ck}, f(X); r)$. If the polynomial is multivariate then we say that the degree of the polynomial in each variable must be less than d, and we require that d^μ for μ variables is polynomial in the security parameter.

Definition A.5 (Extractable)**.** *A polynomial commitment* $\mathsf{PC} = (\mathsf{PC.Setup}, \mathsf{PC.CM}, \mathsf{EvalSetup}, \mathsf{EvalProve}, \mathsf{EvalVerify})$ *is extractable if* $(\mathsf{PC.Setup}, \mathsf{PC.CM})$ *is a binding commitment scheme and if* $(\mathsf{EvalSetup}, \mathsf{EvalProve}, \mathsf{EvalVerify})$ *satisfies witness extended emulation.*

Definition A.6 (Hiding)**.** *A polynomial commitment* $\mathsf{PC} = (\mathsf{PC.Setup}, \mathsf{PC.CM}, \mathsf{EvalSetup}, \mathsf{EvalProve}, \mathsf{EvalVerify})$ *is hiding if* $(\mathsf{PC.Setup}, \mathsf{PC.CM})$ *is a hiding commitment scheme and if* $(\mathsf{EvalSetup}, \mathsf{EvalProve}, \mathsf{EvalVerify})$ *satisfies honest verifier zero-knowledge.*

A.3.1 Round-by-round sound proofs

We first define interactive proof systems $(\mathsf{Prove}, \mathsf{Verify})$ for polynomial-time languages. In general proof systems must satisfy two properties: completeness and soundness. Perfect completeness means that for every $\mathbb{x} \in \mathcal{L}$, the honest prover will always convince the honest verifier to accept. For SIPP we prove a notion of soundness called round-by-round (RBR) soundness, introduced by Canetti et al. in [CCHLRR18]. We say an interactive proof system is RBR sound if there is a well-defined *state* (depending on the transcript so far), and that some of these states are "doomed." RBR soundness requires that once a partial transcript results in a doomed state, then for any possible next prover message, with overwhelming probability over the verifier's next message, the state of this more complete transcript will still be doomed. We can then use Theorem 5.8 from [CCHLRR18], which states that if you replacing the verifier challenges in a RBR sound proof with a suitable choice of hash function RBR , then the resulting proof system is an adaptively sound non-interactive proof system *in the plain model*.

Definition A.7 (Perfect completeness)**.** *The proof system* $(\mathsf{Prove}, \mathsf{Verify})$ *for a language* $\mathcal{L}$ *has perfect completeness if for all instances* $\mathbb{x} \in \mathcal{L}$ *it holds that*

$$\Pr\left[\langle \mathsf{Verify}, \mathsf{Prove}\rangle(1^\lambda, \mathbb{x}) = 1\right] = 1 \ .$$

Our definition of RBR soundness comes from [CCHLRR18, Definition 5.3].

Definition A.8 (Round-by-round (RBR) soundness)**.** *Let* $\Pi = (\mathsf{Prove}, \mathsf{Verify})$ *be a* $2r$*-message public coin interactive proof system for a language* $\mathcal{L}$*. Denote by* tr *the protocol transcript so far. For any* $\mathbb{x} \in \{0,1\}^*$ *and prefix* tr*, let* $\mathsf{Verify}(\mathbb{x}, \mathsf{tr})$ *denote the distribution of the next message (or output) in the transcript.*

We say that Π *has is round-by-round sound if there exists a deterministic (not necessarily efficiently computable) function* st *that takes as input an instance* $\mathbb{x}$ *and a transcript prefix* tr *and outputs either* 1 *or* 0 *such that the following properties hold:*

1. *If* $\mathbb{x} \notin \mathcal{L}$ *then* $\mathsf{st}(\mathbb{x}, \emptyset) = 0$*, where* $\emptyset$ *is the empty transcript.*
2. *If* $\mathsf{st}(\mathbb{x}, \mathsf{tr}) = 0$ *for a transcript prefix* tr*, then for every PPT adversary* $\mathcal{A}$ *it holds that*

$$\Pr\left[\mathsf{st}(\mathbb{x}, \mathsf{tr}\|\alpha\|\beta) = 1 \;\middle|\; \alpha \xleftarrow{\$} \mathcal{A}(\mathbb{x}, \mathsf{tr}); \beta \leftarrow \mathsf{Verify}(\mathbb{x}, \mathsf{tr}\|\alpha)\right] \leq \mathrm{negl}(1^\lambda)$$

3. *For any full transcript (i.e., a transcript for which the verifier halts)* tr*, if* $\mathsf{st}(\mathbb{x}, \mathsf{tr}) = 0$*, then* $\mathsf{Verify}(\mathbb{x}, \mathsf{tr}) = 0$*.*

A.4 Forking lemma

Suppose that we have a $(2k+1)$-move public-coin argument with k challenges $x_1, \ldots, x_k$ is sequence. Let $n_i \geq 1$ for $1 \leq i \leq k$. Consider $\prod_{i=1}^{k} n_i$ accepting transcripts with challenges in the following tree format. The tree has depth k and $\prod_{i=1}^{k} n_i$ leaves. The root of the tree is labeled with the statement. Each node of depth $i < k$ has exactly n_k children, each labeled with a *distinct* value for the ith challenge x_i. This can be referred to as a $(n_1, \ldots, n_k)$-tree of accepting transcripts. In the following lemma it is assumed each challenge $x_1, \ldots, x_k$ is uniformly sampled from a super-polynomial space in the security parameter λ.

The following forking lemma is used in our proof that IPP has computational witness-extended emulation. It is a slightly modified version of the forking lemma from [BCCGP16, Lemma 1], which requires that the malicious prover used to produce the transcripts is computationally bounded, and allows the extractor to fail with negligible probability. This modified statement is directly implied by the proof in [BCCGP16].

Lemma A.9 (Forking lemma)**.** *Let* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a* $(2k+1)$*-move, public-coin interactive protocol. Let* $\mathcal{E}$ *be an efficient witness extraction algorithm that succeeds with overwhelming probability in extracting a witness from an* $(n_1, \ldots, n_k)$*-tree of accepting transcripts produced by an efficient malicious prover interacting with an honest verifier. Assume that* $\prod_{i=1}^{k} n_i$ *is bounded above by a polynomial in the security parameter* λ*. Then* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *has computational witness-extended emulation.*

A.5 Cryptographic assumptions

We use the q-DBP Assumption of Abe et al., which is secure uder the SXDH assumption [AFGHO16]:

Assumption 1 (q-Double pairing assumption (q-DBP))*.* We say the q-double pairing assumption holds in $\mathbb{G}_2$ relative to $\mathsf{SampleGrp}_3$ if for any efficient $\mathcal{A}$ and for all $q > 2 \in \mathbb{N}$:

$$\Pr\left[\begin{array}{c|r} (A_1, \ldots, A_q) \neq \mathbf{1}_{\mathbb{G}_1} & \langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}_3(1^\lambda) \\ \wedge & (h_1, \ldots, h_q) \xleftarrow{\$} \mathbb{G}_2 \\ 1_{\mathbb{G}_T} = \prod_{i=1}^{q} e(A_i, h_i) & (A_1, \ldots, A_q) \xleftarrow{\$} \mathcal{A}(\langle \mathsf{group} \rangle, h_1, \ldots, h_q) \end{array}\right] \leq \mathrm{negl}(\lambda) \ .$$

In our TIPP protocol we use a commitment scheme with structured parameters. Our commitment scheme with structured parameters is secure under an assumption which we dub the computational q-Auxiliary Structured Double Pairing assumption.

Assumption 2 (q-Auxiliary Structured Double Pairing assumption (q-ASDBP))*.* We say the q-ASDBP assumption holds relative to $\mathsf{SampleGrp}_3$ if for any efficient algorithm $\mathcal{A}$

$$\Pr\left[\begin{array}{c|r} (A_0, \ldots, A_{q-1}) \neq \mathbf{1}_{\mathbb{G}_1} & \langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}_3(1^\lambda) \\ \wedge & \beta \xleftarrow{\$} \mathbb{F} \\ 1_{\mathbb{G}_T} = \prod_{i=1}^{q-1} e(A_i, h^{\beta^{2i}}) & (A_0, \ldots, A_{q-1}) \xleftarrow{\$} \mathcal{A}(\langle \mathsf{group} \rangle, g, g^\beta, h, h^\beta, \ldots, h^{\beta^{q-1}}) \end{array}\right]$$

is negligible.

More specifically, we refer to this as the q-$\mathsf{ASDBP}_{\mathbb{G}_2}$ assumption and also define its dual, the q-$\mathsf{ASDBP}_{\mathbb{G}_1}$ assumption, by swapping $\mathbb{G}_1$ and $\mathbb{G}_2$ in the definition above. Observe that in this assumption it is essential that the adversary does not see g^{β^2}.

Lemma A.10. *The* q-ASDBP *assumption holds in the generic group model.*

Proof. Suppose $\mathcal{A}$ is an adversary that on input $g, g^{\beta}, h, h^{\beta}, \ldots, h^{\beta^{q-1}}$, outputs $A_0, \ldots, A_{q-1}$ such that

$$\prod_{i=0}^{q-1} e(A_i, h^{\beta^{2i}}) = 1.$$

Then it's GGM extractor outputs $\{a_{2i}, a_{2i+1}\}$ such that $A_i = g^{a_{2i}+a_{2i+1}\beta}$ and

$$\sum_{i=0}^{q-1}(a_{2i} + a_{2i+1}X)X^{2i} = 0.$$

Thus

$$a_0 + a_1X + a_2X^2 + a_3X^3 + \cdots + a_{2q-2}X^{2q-2} + a_{2q-1}X^{2q-1} = 0.$$

As a result, $a_i = 0$ for all $0 \leq i \leq 2q-1$ and $A_i = g^0 = 1_{\mathbb{G}_1}$. □

We use Boneh and Boyen's q-SDH assumption [BB08] when proving the security of our TIPP scheme.

Assumption 3 (q-Strong Diffie-Hellman assumption (q-SDH)). We say the q-Strong Diffie-Hellman assumption holds relative to $\mathsf{SampleGrp}_3$ if for any efficient algorithm $\mathcal{A}$

$$\Pr\left[\begin{array}{c|r} A = h^{\frac{1}{\beta-a}} & \langle \mathsf{group}\rangle \leftarrow \mathsf{SampleGrp}_3(1^{\lambda}) \\ & \gamma \xleftarrow{\$} \mathbb{F} \\ & (a, A) \leftarrow \mathcal{A}(\langle \mathsf{group}\rangle, g, h, g^{\beta}, h^{\beta}, \cdots, g^{\beta^q}, h^{\beta^q}) \end{array}\right] \leq \mathrm{negl}(\lambda) \ .$$

B Deferred proofs

In this section we present various proofs promised in the main body of this work.

B.1 Proofs of Theorems 4.1 and 4.2

First, we show that SIPP has perfect completeness (Theorem 4.1).

Proof. We prove perfect completeness by showing that in each round if $Z = \boldsymbol{A} * \boldsymbol{B}$ then $Z' = \boldsymbol{A}' * \boldsymbol{B}'$. We show this by simply rewriting Z' as follows:

$$\begin{aligned}
Z' &= Z_L^x Z Z_R^{x^{-1}} \\
&= \left(\boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']}\right)^x \cdot \boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[:m']} \cdot \boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[m':]} \cdot \left(\boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]}\right)^{x^{-1}} \\
&= \left(\boldsymbol{A}_{[m':]}^x \circ \boldsymbol{A}_{[:m']}\right) * \boldsymbol{B}_{[:m']} \cdot \left(\boldsymbol{A}_{[:m']}^{x^{-1}} \circ \boldsymbol{A}_{[m':]}\right) * \boldsymbol{B}_{[m':]} \\
&= \left(\boldsymbol{A}_{[m':]}^x \circ \boldsymbol{A}_{[:m']}\right) * \boldsymbol{B}_{[:m']} \cdot \left(\boldsymbol{A}_{[:m']} \circ \boldsymbol{A}_{[m':]}^x\right) * \boldsymbol{B}_{[m':]}^{x^{-1}} \\
&= \left(\boldsymbol{A}_{[m':]}^x \circ \boldsymbol{A}_{[:m']}\right) * \left(\boldsymbol{B}_{[m':]}^{x^{-1}} \circ \boldsymbol{B}_{[:m']}\right) \\
&= \boldsymbol{A}' * \boldsymbol{B}'
\end{aligned}$$

□

Next, we prove that SIPP is RBR sound (Theorem 4.2).

Proof. We begin by defining a state function st that takes an instance $\mathbb{x} := (\boldsymbol{A}, \boldsymbol{B}, Z)$ and a (possibly empty or partial) transcript tr, and uses the transcript, up to the last verifier message $\beta_i = x_i$, to compute $\boldsymbol{A}', \boldsymbol{B}'$ and Z' just as the verifier would. If $\boldsymbol{A}' * \boldsymbol{B}' = Z'$, then $\mathsf{st}(\mathbb{x}, \mathsf{tr})$ outputs 1, else 0. Property (1) of RBR soundness follows from the fact that when $\mathsf{tr} = \emptyset$ then st checks whether $\mathbb{x} \in \mathcal{L}$ directly, i.e., if $\mathbb{x} \notin \mathcal{L}$ then $\mathsf{st}(\mathbb{x}, \emptyset) = 0$. Property (3) holds because when tr is a complete transcript the state function carries out the exact same decision procedure as the verifier, i.e., $\mathsf{st}(\mathbb{x}, \mathsf{tr}) = \mathcal{V}(\mathbb{x}, \mathsf{tr})$. Now it is just left to show property (2) holds.

Recall that property (2) requires that if $\mathsf{st}(\mathbb{x}, \mathsf{tr}) = 0$ for a transcript prefix tr where the adversarial prover $\mathcal{A}$ is about to move, it holds that

$$\Pr\left[\mathsf{st}(\mathbb{x}, \mathsf{tr}\|\alpha\|\beta) = 1 \;\middle|\; \alpha \xleftarrow{\$} \mathcal{A}(\mathbb{x}, \mathsf{tr}); \beta \leftarrow \mathsf{Verify}(\mathbb{x}, \mathsf{tr}\|\alpha)\right] \leq \mathrm{negl}(1^\lambda)$$

If $\mathsf{st}(\mathbb{x}, \mathsf{tr}) = 0$ for some partial transcript, then by definition of st for the $\boldsymbol{A}, \boldsymbol{B}$, and Z the protocol recursed on in the latest round it holds that $\boldsymbol{A} * \boldsymbol{B} \neq Z$.

Suppose that $\mathcal{A}$ outputs the next message $\alpha = (Z_L, Z_R)$. Denote by $z_L, z, z_R, \boldsymbol{a}$, and $\boldsymbol{b}$ the discrete logarithms of $Z_L, Z, Z_R, \boldsymbol{A}$, and $\boldsymbol{B}$, respectively. Consider the Laurent polynomial

$$f(X) = z_L X + z + z_R X^{-1} - \sum_{i=1}^{m'} \left(a_{m'+i} b_i X^{-1} + a_i b_i + a_{m'+i} b_{m'+i} + a_i b_{m'+i} X \right) \quad .$$

Evaluated at a point $x \xleftarrow{\$} \mathbb{F}$ it holds that $f(X) = 0$ with probability at most $2/|\mathbb{F}|$, regardless of prover message $\alpha = (z_L, z_R)$. This probability follows from the observation that $f(X) \in \mathbb{F}[X, Y^{-1}] \cong \mathbb{F}[X, Y]/(XY - 1)$. Using the isomorphism that maps $f(X)$ to $f(X, Y)$ our claim then follows from the DeMillo-Lipton-Schwartz–Zippel Lemma. Evaluated $f(X)$ at x corresponds exactly to the check done in the exponent when checking $Z_L^x \cdot Z \cdot Z_R^{x^{-1}} = \boldsymbol{A}' * \boldsymbol{B}'$, where as observed in the completeness proof it holds that

$$\boldsymbol{A}' * \boldsymbol{B}' = \left(\boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']} \right)^x \cdot \boldsymbol{A} * \boldsymbol{B} \cdot \left(\boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]} \right)^{x^{-1}} \quad .$$

Then we have that

$$\begin{aligned} &\Pr\left[\mathsf{st}(\mathbb{x}, \mathsf{tr}\|\alpha\|\beta) = 1 \;\middle|\; \alpha \xleftarrow{\$} \mathcal{A}(\mathbb{x}, \mathsf{tr}); \beta \leftarrow \mathsf{Verify}(\mathbb{x}, \mathsf{tr}\|\alpha)\right] \\ &= \Pr\left[\boldsymbol{A} * \boldsymbol{B} \neq Z \wedge f(\beta) = 0 \;\middle|\; (Z_L, Z_R) \xleftarrow{\$} \mathcal{A}(\mathbb{x}, \mathsf{tr}); \beta \xleftarrow{\$} \mathbb{F}\right] \leq \tfrac{2}{|\mathbb{F}|} \end{aligned} \quad ,$$

where $\boldsymbol{A}, \boldsymbol{B}$, and Z are the scalings of the instance elements in latest round as defined by tr. We conclude that the RBR soundness error of SIPP is at most $\frac{2}{|\mathbb{F}|}$. Thus the probability that an adversary breaks RBR soundness is bounded by $\frac{2\log(m)}{|\mathbb{F}|}$. □

B.2 Proof of Theorem 5.4

We proof that GIPA has perfect completeness and witness extended emulation of CM is a binding inner product commitment.

Proof. **Completeness.** Completeness follows directly from the doubly-homomorphic properties of $\mathsf{CM}(\mathsf{ck};)$. The double homomorphism implies a distributive property between keys and messages. For ease of notation we set

$$\mathsf{ck}_i = \mathsf{CM}((\mathsf{ck}_{i,[:m']};,) \text{ and } \mathsf{ck}'_i = \mathsf{CM}((\mathsf{ck}_{i,[m':]};,)$$

for $i = 1, 2, 3$ and

$$M_1 = \boldsymbol{a}_{[:m']}, \;\; M_1' = \boldsymbol{a}_{[m':]}, \;\; M_2 = \boldsymbol{b}_{[:m']}, \;\; M_2' = \boldsymbol{b}_{[m':]}$$

First observe that

$$\begin{aligned}
x^{-1} \cdot C_L &= x^{-1} \cdot \mathsf{CM}(\mathsf{ck}_1, \mathsf{ck}_2', \mathsf{ck}_3;\ M_1', M_2, \langle M_1', M_2 \rangle) \\
&= \mathsf{CM}(x^{-1}\mathsf{ck}_1, \mathsf{ck}_2', \mathsf{ck}_3;\ M_1', x^{-1}M_2, \langle M_1', x^{-1}M_2 \rangle)
\end{aligned}$$

$$\begin{aligned}
C &= \mathsf{CM}((\mathsf{ck}_1 \| \mathsf{ck}_1'), (\mathsf{ck}_2 \| \mathsf{ck}_2'), (\mathsf{ck}_3 \| \mathsf{ck}_3');\ (M_1 \| M_1'), (M_2 \| M_2'), \langle M_1, M_2 \rangle + \langle M_1', M_2' \rangle) \\
&= \mathsf{CM}(x^{-1}\mathsf{ck}_1, x\mathsf{ck}_2, \mathsf{ck}_3; xM_1, x^{-1}M_2, \langle xM_1, x^{-1}M_2 \rangle) \ + \ \mathsf{CM}(\mathsf{ck}_1', \mathsf{ck}_2', \mathsf{ck}_3'; M_1', M_2', \langle M_1', M_2' \rangle) \\
&= C_L' + C_R'
\end{aligned}$$

$$\begin{aligned}
x \cdot C_R &= x \cdot \mathsf{CM}(\mathsf{ck}_1', \mathsf{ck}_2, \mathsf{ck}_3;\ M_1, M_2', \langle M_1, M_2' \rangle) \\
&= \mathsf{CM}(\mathsf{ck}_1', x\mathsf{ck}_2, \mathsf{ck}_3;\ xM_1, M_2', \langle xM_1, M_2' \rangle)
\end{aligned}$$

Second observe that

$$\begin{aligned}
x^{-1} \cdot C_L + C_L' &= \mathsf{CM}(x^{-1}\mathsf{ck}_1, \mathsf{ck}_2', \mathsf{ck}_3;\ M_1', x^{-1}M_2, \langle M_1', x^{-1}M_2 \rangle) \\
&\quad + \mathsf{CM}(x^{-1}\mathsf{ck}_1, x\mathsf{ck}_2, \mathsf{ck}_3; xM_1, x^{-1}M_2, \langle xM_1, x^{-1}M_2 \rangle)
\end{aligned}$$

$$x \cdot C_R + C_R' = \mathsf{CM}(\mathsf{ck}_1', x\mathsf{ck}_2, \mathsf{ck}_3;\ xM_1, M_2', \langle xM_1, M_2' \rangle) \ + \ \mathsf{CM}(\mathsf{ck}_1', \mathsf{ck}_2', \mathsf{ck}_3'; M_1', M_2', \langle M_1', M_2' \rangle)$$

Hence

$$\begin{aligned}
&x^{-1} \cdot C_L + C + x \cdot C_R \\
&= \mathsf{CM}\left(\ x^{-1}\mathsf{ck}_1 + \mathsf{ck}_1', x\mathsf{ck}_2 + \mathsf{ck}_2', \mathsf{ck}_3;\ xM_1 + M_1', x^{-1}M_2 + M_2', \langle xM_1 + M_1', x^{-1}M_2 + M_2' \rangle \ \right) \\
&= \mathsf{CM}(\mathsf{ck}', (\boldsymbol{a}', \boldsymbol{b}', \langle \boldsymbol{a}, \boldsymbol{b} \rangle))
\end{aligned}$$

Extraction. We will show that given a tree of valid transcripts for different challenges as described by Lemma A.9 we can build an efficient extractor that extracts either a wittiness or a break of $\mathsf{CM}(\mathsf{ck};)$'s binding property. The extractor reads a, b from the transcript and then runs the following extraction recursively for each round of the protocol.

Formally given $\boldsymbol{a}' \in \mathcal{M}_1^{m'}, \boldsymbol{b}' \in \mathcal{M}_2^{m'}$ and $Z' = \mathsf{CM}(\mathsf{ck}_3, \langle \boldsymbol{a}', \boldsymbol{b}' \rangle) = x^{-1} \cdot C_L + C + x \cdot C_R$ the extractor computes openings to C_L, C and C_R. Given three transcripts in each for different challenges x_1, x_2, x_3 the extractor receives transcripts $(C_L, C, C_R \in \mathbb{T}, \boldsymbol{a}^{(i)} \in \mathcal{M}_1^{m'}, \boldsymbol{b}^{(i)} \in \mathcal{M}_2^{m'})$ and corresponding keys $\mathsf{ck}_A^{(1)} = x_i^{-1} \cdot \mathsf{ck}_{1,[:m']} + \mathsf{ck}_{1,[m':]}, \mathsf{ck}_2^{(i)} = x_i \cdot \mathsf{ck}_{2,[:m']} + \mathsf{ck}_{2,[m':]}$ such that

$$\begin{aligned}
&\mathsf{CM}(\mathsf{ck}_1^{(i)}, \mathsf{ck}_2^{(i)}, \mathsf{ck}_3; \boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)}, \langle \boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)} \rangle)) \\
&= \mathsf{CM}(\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3; (x_i^{-1}\boldsymbol{a}^{(i)} || \boldsymbol{a}^{(i)}), (x_i \boldsymbol{b}^{(i)} || \boldsymbol{b}^{(i)}), \langle \boldsymbol{a}_i, \boldsymbol{b}^{(i)} \rangle) \\
&= x_i^{-1} \cdot C_L + C + x_i \cdot C_R \quad \forall i \in \{1, 2, 3\}
\end{aligned} \tag{2}$$

The extractor now finds a linear combination of x_1, x_2, x_3 such that using this combination we can compute $(\boldsymbol{a}_L, \boldsymbol{a}_C, \boldsymbol{a}_R \in \mathbb{G}_1^m, \boldsymbol{b}_L, \boldsymbol{b}_C, \boldsymbol{b}_R \in \mathbb{G}_2^m)$ and z_L, z_C, z_R such that

$$C_L = \mathsf{CM}(\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3; \boldsymbol{a}_L, \boldsymbol{b}_L, z_L)$$

and similarly $C = \mathsf{CM}(\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3; \boldsymbol{a}_C, \boldsymbol{b}_C, z_C)$ and $C_R = \mathsf{CM}(\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3; \boldsymbol{a}_R, \boldsymbol{b}_R, z_R)$ Now given 4 total transcripts we have that for each transcript

$$\mathsf{CM}(\mathsf{ck}_1^{(i)}, \mathsf{ck}_2^{(i)}, \mathsf{ck}_3; \boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)}, \langle \boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)} \rangle) = x^{-1} \cdot C_L + C + x \cdot C_R$$

Given the extracted values for x_i this implies that either

$$\begin{aligned}
x_i^{-1} \cdot \boldsymbol{a}_{L,[:m']} + \boldsymbol{a}_{C,[:m']} + x_i \cdot \boldsymbol{a}_{R,[:m']} &= x^{-1} \cdot \boldsymbol{a}^{(i)} \\
x_i^{-1} \cdot \boldsymbol{a}_{L,[m':]} + \boldsymbol{a}_{C,[m':]} + x_i \cdot \boldsymbol{a}_{R,[m':]} &= \boldsymbol{a}^{(i)} \\
x_i^{-1} \cdot \boldsymbol{b}_{L,[:m']} + \boldsymbol{b}_{C,[:m']} + x_i \cdot \boldsymbol{b}_{R,[:m']} &= x \cdot \boldsymbol{b}^{(i)} \\
x_i^{-1} \cdot \boldsymbol{b}_{L,[m':]} + \boldsymbol{b}_{C,[m':]} + x_i \cdot \boldsymbol{b}_{R,[m':]} &= \boldsymbol{b}^{(i)} \\
x_i^{-1} \cdot z_L + z_C + x_i \cdot z_R &= \langle \boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)} \rangle
\end{aligned}$$

or we can directly compute a break of the binding property of the commitment scheme. By assumption this happens with at most negligible probability. These equations imply that

$$\begin{aligned}
x_i^{-1} \cdot \boldsymbol{a}_{L,[m':]} + (\boldsymbol{a}_{L,[:m']} - \boldsymbol{a}_{C,[m':]}) + x_i \cdot (\boldsymbol{a}_{C,[:m']} - \boldsymbol{a}_{R,[m':]}) + x_i^2 \cdot \boldsymbol{a}_{R,[:m']} &= 0 \\
x_i^{-2} \cdot \boldsymbol{b}_{L,[:m']} + x_i^{-1} \cdot (\boldsymbol{b}_{C,[:m']} - \boldsymbol{b}_{L,[m':]}) + (\boldsymbol{a}_{R,[:m']} - \boldsymbol{b}_{C,[m':]}) + x_i \cdot \boldsymbol{b}_{R,[m':]} &= 0
\end{aligned}$$

We can view the left hand side as a polynomials in x_i. Since the equality holds for 4 different x_i we know that the polynomials must be the zero polynomials. This then implies that

$$\begin{aligned}
&\boldsymbol{a}_{L,[m':]} = \boldsymbol{a}_{R,[:m']} = \boldsymbol{b}_{L,[:m']} = \boldsymbol{b}_{R,[m':]} = 0 \\
&\boldsymbol{a}_{C,[:m']} = \boldsymbol{a}_{R,[m':]} \\
&\boldsymbol{a}_{C,[m':]} = \boldsymbol{a}_{L,[:m']} \\
&\boldsymbol{b}_{C,[:m']} = \boldsymbol{b}_{L,[m':]} \\
&\boldsymbol{b}_{C,[m':]} = \boldsymbol{b}_{R,[:m']}
\end{aligned}$$

So $\boldsymbol{a}_C = \boldsymbol{a}_{R,[m':]} || \boldsymbol{a}_{L,[:m']}$ and $\boldsymbol{b}_C = \boldsymbol{b}_{L,[m':]} || \boldsymbol{b}_{R,[:m']}$. Finally this means that for all 4 x_i

$$\begin{aligned}
&\langle \boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)} \rangle = \langle x_i \cdot \boldsymbol{a}_{C,[:m']} + \boldsymbol{a}_{C,[m':]}, x_i^{-1} \cdot \boldsymbol{b}_{C,[:m']} + \boldsymbol{b}_{C,[m':]} \rangle \\
&= x_i^{-1} \cdot z_L + z_C + x_i \cdot z_R \implies \\
&x_i^{-1}(\langle \boldsymbol{a}_{C,[m':]}, \boldsymbol{b}_{C,[:m']} \rangle - z_L) + (\langle \boldsymbol{a}_C, \boldsymbol{b}_C \rangle - z_C) + x_i \cdot (\langle \boldsymbol{a}_{C,[:m']}, \boldsymbol{b}_{C,[m':]} \rangle - z_C) = 0
\end{aligned}$$

The last equality implies that $z_C = \langle \boldsymbol{a}_C, \boldsymbol{b}_C \rangle$ which in turn shows that $C = \mathsf{CM}(\mathsf{ck}_1 || \mathsf{ck}_2; \boldsymbol{a}_C, \boldsymbol{b}_C, z_C)$ is in fact an inner product commitment for which the extractor has computed a witness.

The extractor uses at most $4^{\log_2(m)} = m^2$ transcripts and thus runs in polynomial time. If extraction of a witness fails then it will extract a break of the binding property of the commitment. This, however happens with at most negligible probability. This shows that the protocol has witness extended emulation for the relation $\mathcal{R}_{\mathsf{IPA}}$ □

B.3 Deferred proofs from Section 6

In Theorem Theorem 6.1 we prove that TIPP satisfies computational witness-extended emulation. In the proof we use the facts that: (1) the structure of our honestly generated final commitment keys is correct; (2) an algebraic adversary that convinces a verifier of ill-formed final commitment keys can break q-SDH. We prove these two facts in this section in Proposition B.1 and Lemma B.2.

The following proposition demonstrates the correctness of the format of our final commitment keys.

Proposition B.1. *In the* TIPP *protocol, the final commitment key has the structure*

$$\begin{aligned} w &= g^{\prod_{j=0}^{\ell}(x_j+\alpha^{2^{j+1}})} \\ v &= h^{\prod_{j=0}^{\ell}(x_j^{-1}+(r^{-2}\beta)^{2^{j+1}})} \end{aligned}$$

where x_j is the $(\ell-j)$th verifier challenge and $\ell = \log_2(m)$.

Proof. Recall that in each round of IPP, the prover and the verifier compute the new commitment key to equal

$$\boldsymbol{w}' = \boldsymbol{w}_{[:m']}^{x} \circ \boldsymbol{w}_{[m':]} \text{ and } \boldsymbol{v}' = \boldsymbol{v}_{[:m']}^{x^{-1}} \circ \boldsymbol{v}_{[m':]}.$$

We provide an inductive argument.

First observe that if there is only one round, i.e. if $\ell = 0$, then abusing notation to relabel $\boldsymbol{v} = \boldsymbol{v}^{\boldsymbol{r}^{-2}}$, we see that

$$\begin{aligned} w &= w_0^{x_0} w_1 &&= g^{x_0+\alpha^2} &&= g^{\prod_{j=0}^{0}(x_{\ell-j}+\alpha^{2^{j+1}})} \\ v &= v_0^{x_0^{-1}} v_1 &&= h^{x_0^{-1}+(r^{-1}\beta)^2} &&= h^{\prod_{j=0}^{0}(x_{\ell-j}^{-1}+(r^{-1}\beta)^{2^{j+1}})} \end{aligned}$$

and thus the statement holds in the base case.

Next suppose the statement is true for $\ell - 1$. We show that the statement is true for ℓ. On the first round we rescale the commitment key to

$$\begin{aligned} \boldsymbol{w}' &= \boldsymbol{w}_{[:2^\ell]}^{x_\ell} \circ \boldsymbol{w}_{[2^\ell:]} &&= \left(w_0^{x_\ell} w_0^{\alpha^{2^{\ell+1}}}, \ldots, w_{2^\ell-1}^{x_\ell} w_{2^\ell-1}^{\alpha^{2^{\ell+1}}} \right) \\ \boldsymbol{v}' &= \boldsymbol{v}_{[:2^\ell]}^{x_\ell^{-1}} \circ \boldsymbol{v}_{[2^\ell:]} &&= \left(v_0^{x_\ell^{-1}} v_0^{(r^{-1}\beta)^{2^{\ell+1}}}, \ldots, v_{2^\ell-1}^{x_\ell^{-1}} v_{2^\ell-1}^{(r^{-1}\beta)^{2^{\ell+1}}} \right) \end{aligned}$$

We then run the IPP protocol with respect to ℓ and the generators

$$\boldsymbol{w}' = \boldsymbol{w}_{[:2^\ell]}^{x_\ell+\alpha^{2^{\ell+1}}} \text{ and } \boldsymbol{v}' = \boldsymbol{v}_{[:2^\ell]}^{x_\ell^{-1}+\beta^{2^{\ell+1}}}.$$

From our inductive assumption we have that running TIPP on $\boldsymbol{w}_{[:2^\ell]}$ and $\boldsymbol{v}_{[:2^\ell]}$ yields

$$w = g^{\prod_{j=0}^{\ell-1}(x_{\ell-j}+\alpha^{2^{j+1}})} \quad \text{and} \quad v = h^{\prod_{j=0}^{\ell-1}(x_{\ell-j}^{-1}+\beta^{2^{j+1}})}.$$

where we have used that the challenges indexes are shift by 1. Hence the final commitment key has the form

$$\begin{aligned} w' &= w^{x_\ell+\alpha^{2^{\ell+1}}} &&= g^{(x_\ell+\alpha^{2^{\ell+1}})\prod_{j=0}^{\ell-1}(x_{\ell-j}+\alpha^{2^{j+1}})} &&= g^{\prod_{j=0}^{\ell}(x_{\ell-j}+\alpha^{2^{j+1}})} \\ v' &= v^{x_\ell^{-1}+\beta^{2^{\ell+1}}} &&= h^{(x_\ell^{-1}+\beta^{2^{\ell+1}})\prod_{j=0}^{\ell-1}(x_{\ell-j}^{-1}+\beta^{2^{j+1}})} &&= h^{\prod_{j=0}^{k}(x_{\ell-j}^{-1}+\beta^{2^{j+1}})} \end{aligned}$$

as required. □

We show the soundness of our final commitment key argument i.e. we show that an adversarial prover cannot convince an honest verifier.

Lemma B.2. *The protocol defined by Fig. 5 for the language $\mathcal{L}_{\mathsf{ck}}$ is sound in the algebraic group model under the q-SDH assumption.*

Proof. We prove this lemma for w and note that by symmetry we have that soundness also holds for v (the elements in the SRS included for proving v are independent from those used to prove w and thus provide no advantage to an adversary). Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a pair of algorithms that share state. On input $(\mathsf{srs.p}, r, \boldsymbol{x})$ the adversary $\mathcal{A}_0$ returns w and on input z the adversary $\mathcal{A}_1$ returns (P, Q) such that $\mathsf{Verify}(\mathsf{srs.v}, (r, \boldsymbol{x}, w, v), z, (P, Q)) = 1$.

Let $\mathcal{B}$ be an algorithm against q-SDH. Then $\mathcal{B}$, on input of pair, $(g^{\alpha}, \ldots, g^{\alpha^m}, h^{\alpha}, \ldots, h^{\alpha^m})$, behaves as follows.

1. Choose $\beta \xleftarrow{\$} \mathbb{F}$ and run $\mathcal{A}_0$ on the input
$$\mathsf{pair}, g^{\beta}, h^{\alpha}, \{g^{\alpha^i}, h^{\beta^i}\}_{i=0}^{2m-2}.$$
2. When $\mathcal{A}_0$ returns w extract f_0, f_1 such that $w = g^{f_0(\alpha)+f_1(\beta)}$. Set $s(X) = f_0(X) + f_1(\beta)$.
3. Run $\mathcal{A}_1$ on input $z \xleftarrow{\$} \mathbb{F}$ to obtain verifying (P, Q).
4. Compute $P' = g^{\frac{s(\alpha)-s(z)}{\alpha-z}}$.
5. Return $\left(z, (P'P^{-1})^{\frac{1}{s(z)-f_w(z)}}\right)$.

We now argue that if $(\mathcal{A}_0, \mathcal{A}_1)$ succeeds then either w has been computed correctly with overwhelming probability or $\mathcal{B}$ succeeds. First observe that due to the verifiers equation
$$e(wg^{-f_w(z)}P^z, h) = e(P, h^{\alpha}).$$
Also observe that by design
$$e(wg^{-s(z)}, h) = e(P', h^{\alpha-z}).$$
Thus
$$e(g^{f_w(z)-s(z)}, h) = e(P'P^{-1}, h^{\alpha-z})$$
and either $f_w(z) = s(z)$ or
$$(P'P^{-1})^{\frac{1}{s(z)-f_w(z)}} = g^{\frac{1}{\alpha-z}}$$
and $\mathcal{B}$ succeeds.

If $f_w(z) = s(z)$ then with overwhelming probability $f_w(X) = s(X)$. Hence $w = g^{s(\alpha)} = g^{f_w(\alpha)}$ and w has been computed correctly. □

B.4 Proof of Theorems 7.1 and 7.2

Proof. Let $\mathcal{A}$ be an adversary that succeeds at convincing a verifier with non-negligible probability for random evaluation challenges (x, y). By Lemma B.3 (or by Lemma B.4) there exists an extractor that outputs $\boldsymbol{A}$ such that $T = \boldsymbol{A} * \boldsymbol{v}$ and $A = \boldsymbol{A}^{\boldsymbol{x}}$. By Lemma B.5 (or by Lemma B.2) there exists an extractor that outputs $\boldsymbol{a}'$ such that $\mathsf{eval} = \boldsymbol{a}' \cdot \boldsymbol{y}$ and $A = \boldsymbol{g}^{\boldsymbol{a}'}$.

The extractor runs $\mathcal{A}$ on $\max(m, \ell)$ parallel instances of (x, y). They compute a Vandermonde matrix to find $\hat{M} \in \mathbb{F}^{m,\ell}$ which relates the exponents of A_i to the exponents of $\boldsymbol{g}$. With overwhelming probability this matrix is invertible, so they learn $a_{i,0}, \ldots, a_{i,\ell-1}$ such that $A_i = \prod_{j=0}^{\ell-1} g_j^{a_{i,j}}$. They return $f(X, Y) = \sum_{i,j} a_{i,j} X^i Y^j$. Observe that $\mathsf{eval} = \sum_{i=0}^{m-1} (\sum_{j=0}^{\ell} a_{i,j} y^j) x_i$ is the correct evaluation of $f(X, Y)$. □

B.5 Deferred proofs from Section 7

In Theorems Theorem 7.1&Theorem 7.2 we prove that our polynomial commitments schemes are extractable. In this section we prove two Lemma's showing that the MIPP arguments used as subprotocols are extractable, which are used in the proof of our main theorems. We also prove that our hiding variation on the polynomial commitment schemes satisfies honest verifier zero-knowledge i.e. there exists a simulator who can program the verifier queries that is indistinguishable from an honest prover.

Lemma B.3 (Transparent MIPP is Extractable)**.** *If there exists a bilinear group sampler* SampleGrp *that satisfies the* q-DBP *assumption in* $\mathbb{G}_2$ *then the* MIPP *argument from Section 7.2.1 achieves extractability.*

Proof. We first observe that

$$\mathsf{CM}((\boldsymbol{v}, \mathbf{1}, \hat{h}), (\boldsymbol{A}, \boldsymbol{b}, C)) = ((\boldsymbol{A}||C) * (\boldsymbol{v}||\hat{h}),\ \boldsymbol{b})$$

is a doubly homomorphic binding commitment under the q-DBP assumption. Thus by Theorem 5.4, there exists an extractor that outputs $\boldsymbol{A}$ such that $Z = (\boldsymbol{A} * \boldsymbol{v}) \cdot e(\boldsymbol{A}^{\boldsymbol{b}}, \hat{h}^c)$.

Let $\mathcal{A}$ be an adversary that convinces an honest verifier for T. Our extractor for T works as follows: when the adversary sends A, return random c_1. The adversary then runs convinces the generlized IPA verifier for the commitment key $\mathsf{ck} = (\boldsymbol{v}, \mathbf{1}, \hat{h}^{c_1})$. Run the IPA extractor to obtain $\boldsymbol{A}_1$ such that

$$e(A, \hat{h}^{c_1}) \cdot T = (\boldsymbol{A}_1 * \boldsymbol{v}) \cdot e(\boldsymbol{A}_1^{\boldsymbol{b}}, \hat{h}^{c_1}).$$

Repeat the process under a different random c_2 to obtain $\boldsymbol{A}_2$ such that

$$e(A, \hat{h}^{c_2}) \cdot T = (\boldsymbol{A}_2 * \boldsymbol{v}) \cdot e(\boldsymbol{A}_2^{\boldsymbol{b}}, \hat{h}^{c_2}).$$

By the binding property of the commitment scheme we have that

$$\boldsymbol{A}_1 = \boldsymbol{A}_2 \text{ and } (\boldsymbol{A}_1^{\boldsymbol{b}} A^{-1})^{c_1} = (\boldsymbol{A}_1^{\boldsymbol{b}} A^{-1})^{c_2}.$$

Hence $\boldsymbol{A}_1^{\boldsymbol{b}} A^{-1} = 1$ and $T = \boldsymbol{A}_1 * \boldsymbol{v}$. Further, we get that $A = \boldsymbol{A}_1^b$. □

Lemma B.4 (Structured setup MIPP is Extractable)**.** *If there exists a bilinear group sampler* SampleGrp *that satisfies the* q-$ASDBP$ *assumption in* $\mathbb{G}_2$ *and the* q-SDH *assumption then the* MIPP *argument from Section 7.2.2 achieves extractability against algebraic adversaries.*

Proof. Following the argument from Lemma B.3 we see that if the final commitment key v is correctly computed then by the q-$ASDBP$ assumption there exists an extractor that returns $\boldsymbol{A}$ such that $T = \boldsymbol{A} * \boldsymbol{v}$ and $A = \boldsymbol{A}^b$. By Lemma B.2 we see that an algebraic adversary that convinces the verifier of incorrect v can be used to build an adversary against SDH. Putting the two together we see that the scheme is extractable against algebraic adversaries. □

Lemma B.5 (Transparent univariate polynomial commitment scheme is Extractable)**.** *If there exists a bilinear group sampler* $\mathsf{SampleGrp}$ *that satisfies the* DLR *assumption in* $\mathbb{G}_1$ *then the univariate polynomial commitment argument from Section 7.3.1 achieves extractability.*

Proof. We first observe that

$$\mathsf{CM}((\boldsymbol{g}, \mathbf{1}, u), (\boldsymbol{a}, \boldsymbol{b}, \mathsf{eval})) = (\boldsymbol{g}^{\boldsymbol{a}} u^{\mathsf{eval}},\ \boldsymbol{b})$$

is a doubly homomorphic binding commitment under the DLR assumption. Thus by Theorem 5.4, there exists an extractor that outputs $\boldsymbol{a}$ such that $P = \boldsymbol{g}^{\boldsymbol{a}} u^{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}$.

Let $\mathcal{A}$ be an adversary that convinces an honest verifier for A, eval. Our extractor for A works as follows: when the adversary sends A, return random c_1. The adversary then runs convinces the generlized IPA verifier for the commitment key $\mathsf{ck} = (\boldsymbol{g}, \mathbf{1}, u^{c_1})$. Run the IPA extractor to obtain $\boldsymbol{a}_1$ such that

$$u^{c_1 \cdot \mathsf{eval}} \cdot A = \boldsymbol{g}^{\boldsymbol{a}_1} \cdot u^{c_1 \langle \boldsymbol{a}_1, \boldsymbol{b} \rangle}$$

Repeat the process under a different random c_2 to obtain $\boldsymbol{a}_2$ such that

$$u^{c_2 \cdot \mathsf{eval}} \cdot A = \boldsymbol{g}^{\boldsymbol{a}_2} \cdot u^{c_2 \langle \boldsymbol{a}_2, \boldsymbol{b} \rangle}$$

By the binding property of the commitment scheme we have that

$$\boldsymbol{a}_1 = \boldsymbol{a}_2 \text{ and } c_1((\boldsymbol{a}_1 \cdot \boldsymbol{b}) - \mathsf{eval}) = c_2((\boldsymbol{a}_1 \cdot \boldsymbol{b}) - \mathsf{eval}).$$

Hence $\boldsymbol{a}_1 \cdot \boldsymbol{b} - \mathsf{eval} = 1$ and $A = \boldsymbol{g}^{\boldsymbol{a}_1}$. Further, we get that $\mathsf{eval} = \langle \boldsymbol{a}, \boldsymbol{b} \rangle$. □

Lemma B.6. *The polynomial commitment scheme in Fig. 10 is special honest verifier zero-knowledge.*

Proof. First observe that both the prover samples T from a uniformly random distribution and thus the commitment scheme is hiding.

Consider a simulator that knows the verifier responses in advance:

- To simulate a commitment, return $T \xleftarrow{\$} \mathbb{G}_T$.
- To simulate an evaluation of T to eval with respect to the challenge $c \xleftarrow{\$} \mathbb{F}$
  1. Choose $\rho' \xleftarrow{\$} \mathbb{F}$ and $r(X, Y)$ a random polynomial that evaluates at (x, y) to eval.
  2. Set $R' = \mathsf{PC.CM}(\mathsf{ck}, r(X, Y))$ and $R = T^{-c} \cdot R' \cdot e(g_0^{\rho'}, h^m)$. Send $(R, r(x, y))$ to the verifier.
  3. When the verifier sends c, respond with ρ'.
  4. Run the provers evaluation algorithm on $T' = T^c \cdot R \cdot e(g_0^{-\rho'}, v_m) = R'$.

The provers evaluations $(R, r(x, y))$ is masked by $r_{1,2}, r_{1,3}$ and thus is indistinguishable from the verifiers random evaluation. By Lemma 4.6 of Hoffmann et al. [HKR19], $\mathbb{M}_m$ and $\mathbb{M}_\ell$ are masking sets for MIPP and $\mathsf{DL\text{-}IP}$ respectively. Thus the provers recursive arguments Z_L, Z_R, P_L, P_R are distributed uniformly at random and are indistinguishable from the simulators uniformly random recursive arguments. The provers evaluation $A = \mathsf{CM}(\mathsf{ck}, f(x, Y))$ is masked by $r_{1,1}$ and thus is indistinguishable from the simulators random evaluation A. Hence $(\mathsf{PC.Setup}, \mathsf{PC.CM}, \mathsf{EvalSetup}, \mathsf{EvalProve}, \mathsf{EvalVerify})$ satisfies honest verifier zero-knowledge. □