Efficient Proofs for Pairing-Based Languages

Benedikt Bünz[5]
benedikt@cs.stanford.edu

Mary Maller[1,3]
mary.maller@ethereum.org

Noah Vesely[2,3]
psi@berkeley.edu

Abstract

We present two new protocols for proving correct computation of a product of n bilinear pairings. The first protocol is a statistically sound proof with $2\log(n)$ communication, requiring the verifier compute only a single pairing and $2(n+\log(n))$ exponentiations. We show how this *statistical inner pairing product* (SIPP) can be used to create a new aggregate signature scheme based on BLS with logarithmic signature size and verification time that while still linear is concretely faster than previous results. This is because the verifier time of SIPP is linear only in source group exponentiations, which when computed as multi-exponentiations take significantly less time than computing the Miller loops that dominate the pairing product computation directly.

The second protocol proves that the inner pairing product was evaluated correctly on committed vectors of source group elements, supporting logarithmic aggregation of pairing-based proofs, signatures, etc. without the need for expensive algebraic constructions such as cycles of pairing-friendly elliptic curves. Verifying this *inner pairing product* (IPP) argument requires computing 3 pairings and $2n + 6\log(n)$ exponentiations, and proofs are of size $6\log(n)$ target group elements. As an application, we sketch how IPP can be used to aggregate n Groth16 zkSNARKs into an $\mathcal{O}(\log(n))$ proof.

Both our protocols are public-coin interactive protocols, and thus can be made non-interactive using Fiat-Shamir (such as is done in the construction of our aggregate signatures). Further, neither protocol requires a trusted setup—IPP can be initialized with a public-coin setup, while SIPP requires no setup at all.

Keywords: inner pairing product, bilinear pairings, aggregate signatures, aggregate proofs

This work was completed while at [1]Ethereum Foundation, [2]UC Berkeley, [3]University College London, and [4]Stanford University.

1 Introduction

Pairing-based cryptography constructions include some of the most 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], signature schemes that can be rerandomised [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 provide inner-pairing protocols, inspired by the inner product argument from [BCCGP16; BBBPWM18], that enable a verifier to outsource the verification of pairing equations to a prover. This has two distinct advantages. First, pairing operations are computationally intensive even compared to other cryptographic operations such as exponentiations. Our first protocol SIPP outsources n independent pairings to an untrusted prover who can with small communication convince the verifier that all pairings were computed correctly. The verifier only needs to perform a single pairing and n exponentiations in each source group, which can be batched into a multi-exponentiation to achieve a logarithmic speedup in n over doing each exponentiation individually. For clarity we do not include this optimization in our construction exposition, but we note that this is what provides the real asymptotic win of SIPP over computing a pairing product directly. Preliminary verification benchmarks using libzexe[1] over curve BLS12-377 suggest a speed up of approximately $3.8\times$ for $4,096$ pairings and a $7\times$ for 1 million pairings. The protocol can be made non-interactive and publicly verifiable such that many verifiers can benefit from a single proof.

As an application we introduce an aggregatable signature scheme which is inspired by Boneh, Gentry, Lynn, and Shacham [BGLS03]. Where Boneh et al. observed that BLS signatures can be aggregated, they still require the verifier to compute one pairing per distinct message. We aggregate further using the statistical inner pairing product (SIPP) to prove the result correct. Consequently, our verifier needs to evaluate just one pairing and a linear number of exponentiations, independent of the number of messages signed. The aggregation algorithm, computed by an untrusted party who requires no secrets, is approximately twice as expensive as verifying the BLS signatures directly.

Secondly, many pairing relations are NP relations where it suffices to show that there exist witness group elements such that some public group elements are in a pairing-based language. In this case a prover sends some commitment group elements to the verifier. The prover then shows that these elements satisfy a set of pairing equations via an interactive proof without ever sending the witness group elements. We using a *key homomorphic* commitment scheme, we can show that two committed vectors satisfy an inner pairing product relation (i.e., the product of bilinear pairings between committed equal-length vectors of committed group elements is equal to some public value). This inner pairing product (IPP) protocol has only logarithmic communication complexity and can replace sending a linear number of witness group elements. We sketch how IPP can be used to aggregate n Groth16 [Gro16] pairing-based SNARKs (normally requiring $2n$ group one and n group two elements) with just $6\log(2n)$ target group proof elements. the prover convinces the verifier that these n SNARKs are valid proofs for n instances. The interactive IPP can be made non-interactive and publicly verifiable with the Fiat-Shamir transform. This results in a logarithmic sized proof and enables the first protocol for aggregating SNARKs that does not rely on costly recursive proving techniques.

Both the SIPP and the IPP protocol are structure-preserving (i.e., all public objects including commitments,

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

proof elements, and witnesses are bilinear group elements and functional correctness can be verified by only computing group operations, testing group membership, and evaluating pairing product equations). IPP has a public coin (transparent) setup, while SIPP requires no setup at all.

The inner pairing products are inspired by the inner product argument of Bulletproofs[BCCGP16; BBBPWM18] but applied to bilinear relations between group elements rather than an inner product between committed vectors. The protocols use similar techniques to the concurrent work by Lai et al. [LMR19]. However concretely our protocol uses significantly less communication by being designed for specific relations ($6\log(n)$ and $2\log(n)$ vs. $16\log(n)$).

We note that both protocols could be replaced by general purpose proving techniques such as general purpose SNARKs for NP. Our techniques have two distinct advantages over these general purpose techniques. Firstly, the reduction to an NP statement such as an arithmetic circuit over a finite field is costly and adds orders of magnitude onto the prover complexity. Secondly, our techniques can potentially be composed with these general purpose techniques. For example, instead of directly proving that n pairings are valid the SIPP outsourcing protocol can reduce the verification cost to $2n$ exponentiations. This ladder verification is potentially significantly cheaper to arithmetize and can be more efficiently proven to be correct in a general purpose SNARK.

	SIPP				IPP			
	sizes		time complexity		sizes		time complexity	
	|CRS|	$\|\pi\|$	prover	verifier	|CRS|	$\|\pi\|$	prover	verifier
$\mathbb{G}_1$	—	—	n	n	n	1	$2n$	n
$\mathbb{G}_2$	—	—	n	n	n	1	$2n$	n
$\mathbb{G}_T$	—	$2\log(n)$	—	$2\log(n)$	—	$6\log(n)$	—	$6\log(n)$
e	—	—	$2n$	1	—	—	$6n$	3

Table 1: Computational and communication complexity of SIPP and IPP where n is the number of pairings in the inner pairing product.

We present a detailed efficiency analysis of IPP in Table 1. Proof sizes are logarithmic and the verifier is required to compute a linear number of exponentiations in the number of pairings. The prover costs are approximately six times the cost of naive computation.

In our aggregate signature scheme, aggregating n signatures requires $2n$ exponentiations in both source groups and $2n$ pairings. The aggregate signature consists of $2\log(n+1)$ elements in $\mathbb{G}_T$ and 1 element in $\mathbb{G}_2$. The verifier computes n exponentiations in each of the source groups, $2\log(n)$ exponentiations in $\mathbb{G}_T$, and 1 pairing. The CRS required to support these computations consists of n elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$.

1.1 Our Techniques

The foundations of this paper are built upon two variants of an inner product argument which we call SIPP and IPP. In both the aim of the prover is to demonstrate that a value

$$Z = e(A_1, B_1) \cdots e(A_m, B_m)$$

has been computed correctly with respect to source group elements $A_1, B_1, \ldots, A_m, B_m$. In our SIPP argument, these source group elements are given in the clear and the verifier merely aims to outsource the work of computing Z. In our IPP argument, the source group elements are hidden inside the commitments

$$T = e(A_1, v_1) \cdots e(A_m, v_m) \quad \wedge \quad U = e(w_1, B_1) \cdots e(w_m, B_m)$$

wrt the commitment key $(\boldsymbol{w}, \boldsymbol{v}) \in \mathbb{G}_1^n \times \mathbb{G}_2^n$, and the aim is to compress the quantity of information being communicated to the verifier (our proof sizes are logarithmic in the target group).

To obtain this argument, we started from Bootle et al.'s inner product argument [BCCGP16] and extended it to the pairing setting. Our inner pairing product argument has logarithmic size, the prover computes a linear number of pairings, and the verifier computes a linear number of exponentiations in the number of group elements being paired. In SIPP the verifier can compute the final linear combination of $\boldsymbol{A}$ and $\boldsymbol{B}$ themselves, and thus we only need to communicate the $2\log(m)$ target group elements associated with computing Z. In IPP the verifier must send an additional $4\log(m)$ target group elements to convince the verifier that they have computed the final linear combination of $\boldsymbol{A}$ and $\boldsymbol{B}$ correctly. We prove IPP secure under the SXDH assumption in the interactive setting.

To obtain our aggregatable digital signatures, we started from the work of Boneh et al. [BGLS03], who noted that Boneh-Lynn-Shacham signatures [BLS01] can be aggregated to reduce the space and time requirements on the verifier. This is useful in blockchain applications where the space and time abilities of the verifier are assumed to be highly limited. However, when the aggregated signature is applied to more than one message, Boneh et al.'s verifier is required to compute one pairing per message. Using our pairing argument, we show how one can instead provide a proof that the pairing equations are satisfied, and thus reduce the computational burden on the verifier.

1.2 Related work

In concurrent work Lai, Malavolta, and Ronge [LMR19] introduce a more generalized inner pairing product argument. We arrive at an argument system, IPP, in Section 4.1 for a relation which is a subset of the relations covered by Lai et al.s argument (see [LMR19, Protocol 3]). The advantage of IPP is that even considering a simplified version of [LMR19] that only covers our simpler relation, IPP requires half the prover and one-quarter verifier time, with a CRS half the size. Both systems achieve identical proof sizes and prove the same security notion of witness-extended emulation under SXDH. Additionally, we apply our inner pairing product arguments to different settings: Lai et al. discuss the applications to zero-knowledge proofs, whereas we discuss the aggregation of BLS signatures.

Groth and Sahai [GS08] that 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.

One alternative method for proving the correct evaluation of a pairing relation is to embed the pairing inside a general purpose circuit and then apply a generalised zero-knowledge proof with sublinear verification time. The main difficulty with this approach is that the prover time is necessarily at least linear in the number of gates, and a boolean representation of a pairing equation would require a substantial number of gates. However, Ben-Sasson et al.[BCTV14] demonstrated that by choosing the field sizes with care, it is possible to embed pairings inside arithmetic circuits using a moderate number of gates. They further show that if one has a pairing-based proving system, then it is possible to recursively prove that previous proofs verify, by ensuring that the field is chosen not only so that it embeds a pairing, but also so that it is the order of a separate pairing-based group. Compared to our approach, these methods yield proof sizes and verifier computation which are constant in the security parameter. However, each pairing requires tens of thousands of gates, putting a considerable burden on the prover. As such we believe our approaches complement each other: one could use our argument to reduce the number of pairings that the verifier must compute, and Ben-Sasson et al.'s approach to prove that the verifier is satisfied. In doing so one would obtain both a smaller prover and a smaller verifier. Alternatively, if one is not comfortable using setup assumptions, our approach relies on a trustless setup, whereas Ben-Sasson et al.'s approach depends on pairing-based SNARKs, which so far we only know how to build under trusted setup.

Boneh et al.[BGLS03] observed that BLS signatures [BLS01] can be aggregated provided that the users are signing distinct messages. If the messages are not distinct (e.g., multisignatures) then care has to be taken to avoid rogue key attacks and such as by providing a proof-of-knowledge of the public key [RY07] or applying recent results by Boneh et al. [BDN18]. While Boneh et al. demonstrated how to generate multisignatures, they fall short of reducing the number of pairings required to verify an aggregate signature with distinct messages. This work discusses how to reduce the number of pairings required to verify aggregate signature and we only discuss the scenario where messages are distinct.

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 utilizes their commitment scheme. Where Lai et al. mention that it is possible to use a inner pairing product argument to reduce the size of Abe et al.'s structure preserving signatures [LMR19], we instead directly use BLS signatures. These signatures are more efficient–they consist of a single group element per message–and thus the proofs generated using them will also be more efficient. Further, where the inner pairing product argument is in the random oracle model anyway, many of the advantages in using a structure preserving scheme have already been lost.

2 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 that used to describe the generalized Pedersen inner product introduced in [BBBPWM18]. We write all 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 $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$ we let $\boldsymbol{A}^{\boldsymbol{x}} = \prod_{i=1}^n A_i^{x_i} \in \mathbb{G}$. 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 $\boldsymbol{A} \circ \boldsymbol{A}' = \prod_{i=1}^n A_i \cdot A_i'$ denote the entrywise group operation product of two vectors.

Let $\boldsymbol{A} \| \boldsymbol{A}' = (A_1, \ldots, A_n, A'_1, \ldots, A'_m)$ be the concatenation of two vectors $\boldsymbol{A} \in \mathbb{G}$ and $\boldsymbol{A}' \in \mathbb{G}$. To denote slices of vectors given $\boldsymbol{A} \in \mathbb{G}^n_\cdot$ and $1 \le \ell < n$ we write

$$\boldsymbol{A}_{[:\ell]} = (A_1, \ldots, A_\ell) \in \mathbb{G}^\ell \quad \text{and} \quad \boldsymbol{A}_{[\ell:]} = (A_{\ell+1}, \ldots, A_n) \in \mathbb{G}^{n-\ell} \ .$$

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 $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$.

3 Outsourcing inner pairing products

In this section we introduce an interactive, statistically-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.

3.1 Construction

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

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

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 statistical soundness (Theorem 3.1), the probability

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

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. 1.

3.2 Efficiency

We provide a detailed breakdown of the communication and computation complexity of the protocol in Table 1. 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. In the first round the prover and verifier require $n/2$ exponentiations in $\mathbb{G}_1$ and $n/2$ exponentiations in $\mathbb{G}_2$ to compute $\boldsymbol{A}'$ and $\boldsymbol{B}'$, respectively. This halves to $n/4$ in the second and $n/2^i$ in the i-th. Over $\log(n)$ rounds the prover and verifier each compute a total of n exponentiations in each of the source groups. To compute Z' in each round the verifier computes two target group exponentiations. Over $\log(n)$ rounds the verifier then computes a total of $2\log(n)$ exponentiations in $\mathbb{G}_T$.

We are currently working on an implementation of the SIPP. Preliminary results and estimates show that the majority of the speedup comes from using multi-exponentiation to verify the final SIPP exponentiation (see Appendix B for more details on optimizations). Multi-exponentiation algorithms[Pip80] roughly use $\frac{\lambda n}{\log(n)}$ group operations to perform a multi-exponentiation with n bases and λ-bit exponents. Verifying the pairing product can be sped up by first computing the n Miller loops and then applying a single final exponentiation to their product [Sco05; Sco07]. However, the computation still scales linearly in n using this technique and techniques to batch Miller loops achieve a speedup that is just barely sublinear [Sco19]. Preliminary results suggest for $4,096$ pairings the SIPP verifier is 3.8 times faster than directly checking the pairing product by first computing the Miller loops and then doing a single final exponentiation. For 1 million pairings this advantage increases to a factor of 7.

3.3 Security

We prove the following theorem showing SIPP satisfies statistical soundness in Appendix C.1.

Theorem 3.1 (Statistical soundness of SIPP)**.** *The protocol defined by Fig. 1 for the language $\mathcal{L}_{\mathsf{SIPP}}$ has statistical soundness (Definition A.3).*

Prove$(\langle \mathsf{group} \rangle, \boldsymbol{A}, \boldsymbol{B}, Z)$ | Verify$(\langle \mathsf{group} \rangle, \boldsymbol{A}, \boldsymbol{B}, Z)$

$m' = m/2$ | $m' = m/2$

$Z_L = \boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']}$

$Z_R = \boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]}$

$$\xrightarrow{Z_L, Z_R \in \mathbb{G}_T}$$

$x \xleftarrow{\$} \mathbb{F}$

If $m' \geq 2$

$$\xleftarrow{x \in \mathbb{F}}$$

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

$\boldsymbol{B}' = \boldsymbol{B}^{x^{-1}}_{[m':]} \circ \boldsymbol{B}_{[:m']}$ | $\boldsymbol{B}' = \boldsymbol{B}^{x^{-1}}_{[m':]} \circ \boldsymbol{B}_{[:m']}$

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

Recurse on $(\langle \mathsf{group} \rangle, \boldsymbol{A}', \boldsymbol{B}', Z')$

Else $m' = 1$

$A' = A_1^{x} \cdot A_0$

$B' = B_1^{x^{-1}} \cdot B_0$

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

Return 1 if $e(A', B') = Z'$, else 0

Figure 1: A statistically sound protocol for outsourcing pairings.

4 An inner pairing product argument

In this section we introduce an interactive argument of knowledge that allows a prover to prove knowledge of two committed vectors of source group elements such that their pairing inner product is equal to some public value. This argument has logarithmic-sized proofs and wide ranging applications to proving. We note that while it is possible to achieve significantly smaller proof sizes and verifier time using SNARKs, the prover costs can pose as a significant barrier to adopting such protocols in practice. The pairing product equations must be reduced to a set of NP constraints, and the prover time will scale quasi-linearly with respect to this (much larger) set of constraints.

4.1 Construction

We use the non-hiding variant of the AFGHO[AFGHO16] commitment scheme to commit to a vector of group elements in $\mathbb{G}_1$ and one in $\mathbb{G}_2$. The commitment is simply the pairing product of the group elements with a vector of random 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 CRS.

We present our inner pairing product argument IPP which proves that given T and U that $Z = \boldsymbol{A} * \boldsymbol{B}$, i.e. Z is the inner pairing product between $\boldsymbol{A}$ and $\boldsymbol{B}$.

More formally the relation $\mathcal{R}_{\mathsf{pair}}$ is defined by

$$\left\{ \begin{array}{c} (\langle \mathsf{group} \rangle, \boldsymbol{w} \in \mathbb{G}_1^m,\ \ \boldsymbol{v} \in \mathbb{G}_2^m,\ \ T, U, Z \in \mathbb{G}_T\ ; \quad \boldsymbol{A} \in \mathbb{G}_1^m,\ \ \boldsymbol{B} \in \mathbb{G}_2^m) : \\ T = \boldsymbol{A} * \boldsymbol{v} \quad \wedge \quad U = \boldsymbol{w} * \boldsymbol{B} \quad \wedge \quad Z = \boldsymbol{A} * \boldsymbol{B} \end{array} \right\} .$$

Without loss of generality assume m is a power of two. Our argument is defined by a recursive protocol that in each loop "scales" vectors $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}, \boldsymbol{w}$ into new vectors $\boldsymbol{A}', \boldsymbol{B}', \boldsymbol{v}', \boldsymbol{w}'$ of length $m' = m/2$. In each loop the prover also commits a set of values that the verifier uses to update target group elements T, U, Z from the last round to new $T', U', Z' \in \mathbb{G}_T$. The recursive protocol proceeds until the final round of recursion where $m = 1$ and the inputs are source group elements $A, w \in \mathbb{G}_1,\ B, v \in \mathbb{G}_2$ sent to the verifier from the prover and the T, U, Z the verifier derived the penultimate round. In this final round the prover reveals A, B and the verifier (who has computed v, w) outputs "accept" iff

$$e(A, v) = T \quad \text{and} \quad e(w, B) = U \quad \text{and} \quad e(A, B) = Z.$$

Our key insight is that the inner product argument of Bootle et al. [BCCGP16] can be adapted to work over a pairing-based language. While one might be tempted to use the improved inner product argument of Bünz et al. [BBBPWM18], we warn the reader that in our pairing-based setting such a scheme would be insecure (we would require a version of DBP that works assuming the adversary outputs elements in both source groups–an assumption which is provably false).

We now describe the protocol that runs in each round until $m = 1$. The prover sets $m' = m/2$ and then computes the 6 target group elements

$$\begin{array}{ll} T_L = \boldsymbol{A}_{[:m']} * \boldsymbol{v}_{[m':]} & \qquad T_R = \boldsymbol{A}_{[m':]} * \boldsymbol{v}_{[:m']} \\ U_L = \boldsymbol{w}_{[m':]} * \boldsymbol{B}_{[:m']} & \qquad U_R = \boldsymbol{w}_{[:m']} * \boldsymbol{B}_{[m':]} \\ Z_L = \boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]} & \qquad Z_R = \boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']} \end{array}$$

and then sends them all to the verifier. Next, the verifier samples a fresh challenge $x \xleftarrow{\$} \mathbb{F}$ and sends x to the prover. The prover and the verifier each independently compute

$$\boldsymbol{v}' = \boldsymbol{v}_{[:m']}^{x^{-1}} \circ \boldsymbol{v}_{[m':]} \in \mathbb{G}_2^{m'} \qquad \boldsymbol{w}' = \boldsymbol{w}_{[:m']}^{x} \circ \boldsymbol{w}_{[m':]} \in \mathbb{G}_1^{m'}$$

The verifier alone computes the target group elements

$$T' = T_L^x \cdot T \cdot T_R^{x^{-1}} \qquad U' = U_L^x \cdot U \cdot U_R^{x^{-1}} \qquad Z' = Z_L^x \cdot Z \cdot Z_R^{x^{-1}}$$

The prover alone computes$\boldsymbol{A}' = \boldsymbol{A}_{[:m']}^{x} \circ \boldsymbol{A}_{[m':]} \in \mathbb{G}_1^{m'} \qquad \boldsymbol{B}' = \boldsymbol{B}_{[:m']}^{x^{-1}} \circ \boldsymbol{B}_{[m':]} \in \mathbb{G}_2^{m'}$ and the protocol recurses on $\left(\langle \mathsf{group} \rangle, \boldsymbol{w}' \in \mathbb{G}_1^{m'},\ \ \boldsymbol{v}' \in \mathbb{G}_2^{m'},\ \ T', U', Z' \in \mathbb{G}_T; \quad \boldsymbol{A}' \in \mathbb{G}_1^{m'},\ \ \boldsymbol{B}' \in \mathbb{G}_2^{m'}\right)$.

Pseudocode. In Fig. 2 we present the interactive protocols $(\mathsf{Prove}, \mathsf{Verify})$.

```
Prove($\langle\text{group}\rangle, \boldsymbol{w}, \boldsymbol{v}, \boldsymbol{A}, \boldsymbol{B}$)
    $m' = m/2$
    $T_L = \boldsymbol{A}_{[:m']} * \boldsymbol{v}_{[m':]}$
    $T_R = \boldsymbol{A}_{[m':]} * \boldsymbol{v}_{[:m']}$
    $U_L = \boldsymbol{w}_{[m':]} * \boldsymbol{B}_{[:m']}$
    $U_R = \boldsymbol{w}_{[:m']} * \boldsymbol{B}_{[m':]}$
    $Z_L = \boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]}$
    $Z_R = \boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']}$

Verify($\langle\text{group}\rangle, \boldsymbol{w}, \boldsymbol{v}, T, U, Z$)
    $m' = m/2$

Prover → Verifier: $T_L, T_R, U_L, U_R, Z_L, Z_R \in \mathbb{G}_T$

Verifier: $x \overset{\$}{\leftarrow} \mathbb{F}$

Verifier → Prover: $x \in \mathbb{F}$

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

Verifier:
    $\boldsymbol{v}' = \boldsymbol{v}_{[:m']}^{x^{-1}} \circ \boldsymbol{v}_{[:m']}$
    $\boldsymbol{w}' = \boldsymbol{w}_{[:m']}^{x} \circ \boldsymbol{w}_{[m':]}$
    $T' = T_L^x \cdot T \cdot T_R^{x^{-1}}$
    $U' = U_L^x \cdot U \cdot U_R^{x^{-1}}$
    $Z' = Z_L^x \cdot Z \cdot Z_R^{x^{-1}}$

........................................ If $m' \geq 2$ ........................................
Prover: Recurse on ($\langle\text{group}\rangle, \boldsymbol{w}', \boldsymbol{v}', \boldsymbol{A}', \boldsymbol{B}'$)
Verifier: Recurse on ($\langle\text{group}\rangle, \boldsymbol{w}', \boldsymbol{v}', \boldsymbol{A}', \boldsymbol{B}'$,
                 $T', U', Z'$)

........................................ Else $m' = 1$ ........................................
Prover → Verifier: $A' \in \mathbb{G}_1, B' \in \mathbb{G}_2$

Verifier:
    Return 1 if $e(A', v') = T'$
             $\wedge\, e(w', B') = U'$
             $\wedge\, e(A', B') = Z'$,
         else 0
```

Figure 2: Protocol for inner pairing product argument.

Transparent setup. Note that generating the public parameters $\mathsf{pp} = (\langle\text{group}\rangle, \boldsymbol{v}, \boldsymbol{w})$ does not require a trusted setup. The Setup algorithm can use a hash function $H_n : \{0,1\}^* \to \mathbb{G}_1^n \times \mathbb{G}_2^n$ that, given a small seed (public randomness), outputs the group elements needed to run IPP. Where support for memory-constrained devices that cannot store the entire CRS is needed, it is possible to define this hash function to allow individual

CRS elements to be computed on-the-fly (at a cost to efficiency).

4.2 Efficiency

We provide a detailed breakdown of the communication and computation complexity of the protocol in Table 1. Our protocol requires $\log(n)$ rounds. In each round of Fig. 2 the prover and verifier independently compute new generators $\boldsymbol{v}', \boldsymbol{w}'$ requiring $4n$ exponentiations: $2n$ in the first round, n in the second round, and $\frac{2n}{2^{j-1}}$ in the j-th. ($2n$ of these exponentiations are in $\mathbb{G}_1$ and $2n$ in $\mathbb{G}_2$.) The verifier alone computes the T', U', Z' values, requiring an additional $6\log(n)$ exponentiations in $\mathbb{G}_T$. The prover alone computes the new $\boldsymbol{A}', \boldsymbol{B}'$ values–an additional $2n$ exponentiations in each source group.

The $T_L, T_R, U_L, U_R, Z_L, Z_R$ values the prover computes in each round require $6n$ pairings to compute: $3n$ in the first round, $\frac{3n}{2}$ in the second round, and $\frac{3n}{2^{j-1}}$ in the j-th. These elements, $6\log(n)$, are sent to the verifier along with the final $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$. The verifier needs compute just 3 pairings to check the 3 equations in the final round.

4.3 Security

We prove the following theorem showing IPP satisfies computational witness-extended emulation in Appendix C.2.

Theorem 4.1 (Computational witness-extended emulation of IPP)**.** *The protocol defined by Fig. 2 for the NP relation $\mathcal{R}_{\mathsf{pair}}$ has computational witness-extended emulation (Definition A.5) under* SXDH.

5 An aggregate signature scheme based on BLS

Boneh, Lynn, and Shacham introduced the BLS signature scheme [BLS01] which supports offline aggregation [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 aggregate signature scheme where the verifier is required to compute just one pairing for any number of signatures. Previous aggregate signature schemes based on BLS [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 n-sized multi-exponentiations in the source groups at the cost of a logarithmic-sized signature.

The basic BLS signature scheme is given in Fig. 3. 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. Note that the source groups that the public keys, signatures, and hashed messages are in can be swapped.

```
$\underline{\mathsf{Setup}(1^\lambda)}$                                $\underline{\mathsf{KeyGen}(\langle\mathsf{group}\rangle)}$
$\langle\mathsf{group}\rangle \leftarrow \mathsf{SampleGrp}_3(1^\lambda)$       $\mathsf{sk} \xleftarrow{\$} \mathbb{F}$
Return $\langle\mathsf{group}\rangle$                      $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
                                           Return $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Sign}(\langle\mathsf{group}\rangle, \mathsf{sk}, m)}$ :                 $\underline{\mathsf{Verify}(\langle\mathsf{group}\rangle, \mathsf{pk}, m, \sigma)}$
$h \in \mathbb{G}_2 \leftarrow \mathsf{RO}_2(m)$                     $h \in \mathbb{G}_2 \leftarrow \mathsf{RO}_2(m)$
$\sigma \leftarrow h^{\mathsf{sk}}$                              Check $e(g, \sigma) = e(\mathsf{pk}, h)$
Return $\sigma$                               Return 1 if check passes
                                           Else return 0
```

Figure 3: The BLS signature scheme where $\mathsf{RO}_2 : \{0,1\}^* \to \mathbb{G}_2$ is a random oracle.

5.1 Construction

We introduce a pair of algorithms $(\mathsf{AggSign}, \mathsf{VerifyAgg})$ that extends the BLS signature scheme into a aggregate signature scheme.

Recall from Appendix A.5 that AggSign is given some public parameters (here 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 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 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 SIPP.Verify, and outputs the result of checking the inner pairing product proof.

Pseudocode. We present pseudocode for our BLS aggregation protocol in Fig. 4.

```
$\underline{\mathsf{AggSign}(\langle\mathsf{group}\rangle, [\mathsf{pk}_i]_{i=1}^n, \{m_i\}_{i=1}^n, [\sigma_i]_{i=1}^n)}$ :        $\underline{\mathsf{VerifyAgg}(\langle\mathsf{group}\rangle, [\mathsf{pk}_i]_{i=1}^n, \{m_i\}_{i=1}^n, \Sigma)}$ :
$(h_1, \ldots, h_n) \leftarrow (\mathsf{RO}_2(m_i), \ldots, \mathsf{RO}_2(m_n))$             $(\sigma_A, \pi) \leftarrow \Sigma$
$\sigma_A \leftarrow \prod_{i=1}^n \sigma_i$                                   $(h_1, \ldots, h_n) \leftarrow (\mathsf{RO}_2(m_i), \ldots, \mathsf{RO}_2(m_n))$
$\pi \leftarrow \mathsf{SIPP.Prove}(\langle\mathsf{group}\rangle, g^{-1} \| \mathbf{apk}, \sigma_A \| \boldsymbol{h}, 1)$      Check $\mathsf{SIPP.Verify}(\langle\mathsf{group}\rangle, g^{-1} \| \mathbf{apk}, \sigma_A \| \boldsymbol{h}, 1, \pi)$
$\Sigma \leftarrow (\sigma_A, \pi)$                                   Return 1 if both checks pass
Return $\Sigma$                                         Else return 0
```

Figure 4: Aggregation and verification algorithms for an aggregate signature scheme.

5.1.1 Discussion

We make a few remarks on the construction. First, to call SIPP in a non-interactive protocol such as this one, we must first apply the Fiat-Shamir transformation, where in each round the messages the prover sends (including the public parameters and instance in the first round) are input to a random oracle, which outputs the next verifier challenge.

Second, we observe that σ_A is exactly the aggregate signature in the aggregate signature schemes [BGLS03] and [RY07]. Combining the pairing equations in this way results in a verification equation that only achieves aggregate unforgeability and does not achieve the stronger notion of batch verification (see Remark A.8). The standard way of doing batch verification is using the small exponent test (see Appendix B) to combine the pairing equations, but making this test publicly-verifiable requires hashing all inputs to obtain the exponents, which is clearly incompatible with aggregation. We remark that IPP can be used to create an aggregate signature that achieves batch verification at a cost to signature size and verifier time, but we leave the formulation of this construction as an exercise to the reader.

Finally, we discuss *multisignature schemes*. Ristenpart et al. [RY07] and Boneh et al. [BDN18] introduced multisignature schemes, which allow multiple signatures on the *same* message to be aggregated into a single group element (the multisignature), and additionally for the corresponding public keys to be aggregated into a single group element (the aggregate public key). The multisignature and aggregate public key can then be verified using the original BLS verification equation. Ristenpart et al. require all signers output valid proof-of-possession for each public key which they output. Boneh et al. present an alternative that involves hashing the public keys to create a random challenge that's used both in computing the multisignature and aggregate public key. We refer the reader to the respective works for construction details and proofs, and simply observe that their techniques could be used in conjunction with ours to produce an *aggregate multisignature scheme* as defined in [BDN18].

5.2 Efficiency

Verifying n signatures on n different messages using VerifyAgg requires computing n hashes and then running SIPP.Verify. Running AggSign requires n hash evaluations and a call to SIPP.Prove. Ignoring small constants, the prover computes a total of $2n$ pairings, n exponentiations in each source group, and n hashes. The verifier computes 1 pairing, n exponentiations in each source group, and n hashes.

The aggregate signatures consists of $\log_2(n)$ elements in $\mathbb{G}_T$ and 1 element in $\mathbb{G}_2$.

5.3 Security

We prove our aggregate signature scheme is computationally unforgeable in the random oracle model under ψ-co-CDH in Appendix C.3.

Theorem 5.1. *The scheme in described in Section 5.1 and Fig. 4 is an unforgeable aggregate signature scheme (Definition A.7) in the random oracle model under* ψ-co-CDH *(Assumption 3).*

6 Aggregating Groth16 proofs

We now quickly sketch how the inner pairing product can be used to verify that n independently generated Groth16 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 IPP protocol run by an untrusted aggregator can be used to aggregate these proofs into a small logarithmic sized batch proof. The verifiers only need to check the aggregated batch proof to be convinced of the existence of the underlying pairing-based SNARKs.

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

$$e(A, B) = e(g^\alpha, h^\beta) \cdot e(X, h^\gamma) \cdot e(C, h^\delta)$$

where $X \in \mathbb{G}_1$ depends only on the instance $\mathbb{x}$. x, α, β, and δ form the SRS trapdoor and $e(g^\alpha, h^\beta)$, $[g^{(\beta u_i(x)+\alpha v_i(x)-w_i(x))/\gamma}]_{i=0}^{\ell}, h^\gamma$, and h^δ form part of public SRS.

Given n instances $[[a_{i,j}]_{i=0}^{\ell}]_{j=1}^{n}$ and proofs $[(A_j, B_j, C_j)]_{j=1}^{n}$, checking each equation separately requires performing $3n$ pairings and exponentations (factoring in that $e(g^\alpha, h^\beta)$ can be precomputed). To reduce this computation to a single verification, the verifier can take a random linear combination between all equations. That is, the verifier samples $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{F}$ and then checks whether

$$\boldsymbol{A}^{-\boldsymbol{r}} * \boldsymbol{B} \cdot e(\boldsymbol{C}^{\boldsymbol{r}}, h^\delta) \cdot e(\boldsymbol{X}^{\boldsymbol{r}}, h^\gamma) \cdot e(g^{\alpha R}, h^\beta) = 1 \ , \tag{1}$$

where $R = \sum_{j=1}^{n} r_j$.

This reduction is known as the small exponent test (SET), and if this equation holds, then with overwhelming probability each individual verification holds [BGR98]. It therefore suffices to check this one pairing product instead of checking all SNARKs individually.

Note that the verifier, who has access to the instances can compute $e(\boldsymbol{X}^{\boldsymbol{r}}, h^\gamma) \cdot e(g^{\alpha R}, h^\beta)$ on its own. The challenge, thus, becomes to convince the verifier that there exists vectors of group elements $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, such that $\boldsymbol{A}^{-\boldsymbol{r}} \| \boldsymbol{C}^{\boldsymbol{r}} * \boldsymbol{B} \| \boldsymbol{h}^{\boldsymbol{\delta}}$ is equal to some constant. $\boldsymbol{h}^{\boldsymbol{\delta}}$ is the concatenation of n copies of h^δ.

To do this the verifier commits to $\boldsymbol{A} \| \boldsymbol{C}$ and $\boldsymbol{C} \| \boldsymbol{h}^{\boldsymbol{\delta}}$ using the Abe et al. [AFGHO16] commitment scheme. That is, given a CRS $\boldsymbol{v}_L, \boldsymbol{v}_R \in \mathbb{G}_2^n$, $\boldsymbol{w}_L$ and $\boldsymbol{w}_R \in \mathbb{G}_1^n$ they set $T \leftarrow \boldsymbol{A} \| \boldsymbol{C} * \boldsymbol{v}_L \| \boldsymbol{v}_R$ and $U \leftarrow \boldsymbol{w}_L \| \boldsymbol{w}_R * \boldsymbol{B} \| \boldsymbol{h}^{\boldsymbol{\delta}}$.

Note that just like the Pedersen vector commitment [Ped92] used in the inner product argument of Bootle et al. [BCCGP16] this commitment is key-homomorphic. That is given a commitment T to a vector $\boldsymbol{A}$ using a commitment key $\boldsymbol{v}$, T is also a commitment to a vector $\boldsymbol{A}^{\boldsymbol{r}}$ using the commitment key $\boldsymbol{v}^{\boldsymbol{r}^{-1}}$.

In the SNARK aggregation protocol the prover sends T and U. The verifier then generates a random challenge vector $\boldsymbol{r}$ used to create the combined SNARK verification equation and sends it to the prover. Both the prover and verifier update the CRS such that T is a commitment to $\boldsymbol{A}^{-\boldsymbol{r}} \| \boldsymbol{C}^{\boldsymbol{r}}$. The verifier then computes $Z \leftarrow e(\boldsymbol{X}^{-\boldsymbol{r}}, h^\gamma) \cdot e(g^{-\alpha R}, h^\beta)$ using the CRS and the instances $\boldsymbol{X}$. Note that if the batch verification equation (Eq. (1)) holds then Z should be equal to $e(\boldsymbol{A}^{-\boldsymbol{r}}, \boldsymbol{B}) \cdot e(\boldsymbol{C}^{\boldsymbol{r}}, h^\delta)$. The prover and verifier then run IPP to show that this inner pairing product relation holds.

The protocol can be made non-interactive and publicly verifiable using the Fiat-Shamir transform. This results in a logarithmic sized aggregation of n Groth16 snarks without the need for expensive pairing-friendly cycles of elliptic curves.

Acknowledgements

Thank you to Russell Lai, Giulio Malavolta, Pratyush Mishra, and Alessandro Chiesa for helpful discussions. This work was partially supported by NSF, SGF, ONR, and the Simons Foundation and Institute for the Theory of Computation. Mary Maller was supported in part by EPSRC Grant EP/N028104/1.

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.

[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.

[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.

[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.

[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: *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*. 2010, pp. 218–235.

[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.

[BGR98] M. Bellare, J. A. Garay, and T. Rabin. “Fast Batch Verification for Modular Exponentiation and Digital Signatures”. In: *Proceedings of the 17th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’98. 1998, pp. 236–250.

[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: *CCS ’93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.* 1993, pp. 62–73.

[CHP07] J. Camenisch, S. Hohenberger, and M. Ø. Pedersen. “Batch Verification of Short Signatures”. In: *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*. 2007, pp. 246–263.

[CL06] J. H. Cheon and D. H. Lee. “Use of Sparse and/or Complex Exponents in Batch Verification of Exponentiations”. In: *IEEE Transactions on Computers* 55.12 (2006), pp. 1536–1542.

[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*. 2014, pp. 630–649.

[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.

[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.

[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.

[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*. 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.

[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. `https://eprint.iacr.org/2019/969`. 2019.

[Ped92] T. P. Pedersen. “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”. In: *Proceedings of the 11th Annual International Cryptology Conference*. CRYPTO ’91. 1992, pp. 129–140.

[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). Preliminary version appeared in STOC ’76., 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*. 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.

[Sco05] M. Scott. “Computing the Tate Pairing”. In: *Proceedings of the The Cryptographers’ Track at the RSA Conference 2005*. CT-RSA ’05. 2005, pp. 293–304.

[Sco07] M. Scott. “Implementing Cryptographic Pairings”. In: *Proceedings of the 1st First International Conference on Pairing-Based Cryptography*. Pairing ’07. 2007, pp. 177–196.

[Sco19] M. Scott. *Pairing Implementation Revisited*. 2019. URL: https://eprint.iacr.org/2019/077.

A Preliminaries

In this section we present formalizations of a number of primitives and cryptographic assumptions we construct and rely on in this paper. In Appendix A.1 we formalize the notion of a bilinear group sampler, in Appendix A.2 we present the cryptographic assumptions our constructions rely on, in Appendix A.3 we define proof and argument systems, in Appendix A.4 we introduce a forking lemma we use in proving security of IPP, and in Appendix A.5 we define aggregate signature schemes.

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 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.2 Cryptographic assumptions

IPP relies on the *symmetric external Diffie-Hellman* (SXDH) assumption, which states that the *decisional Diffie-Hellman* problem is hard in both source groups of a bilinear group. As remarked in Appendix A.1, the SXDH is only believed to hold in Type III groups. We define our assumptions relative to a Type III group generator such that the soundness of our argument relies on the existence of a group generator for which SXDH holds.

Assumption 1 (Double pairing assumption (DBP))**.** *We say the double pairing assumption holds relative to* $\mathsf{SampleGrp}_3$ *if for any efficient algorithm* $\mathcal{A}$

$$\Pr\left[\begin{array}{c|r} (a_1, a_2) \neq (1,1) & \langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}_3(1^\lambda) \\ \wedge & (h_1, h_2) \xleftarrow{\$} \mathbb{G}_2 \\ 1 = e(a_1, h_1)e(a_2, h_2) & (a_1, a_2) \leftarrow \mathcal{A}(\langle \mathsf{group} \rangle, h_1, h_2) \end{array}\right] \leq \mathrm{negl}(\lambda) \ .$$

More specifically, we refer to this as the $\mathsf{DBP}_{\mathbb{G}_2}$ assumption and also define its dual, the $\mathsf{DBP}_{\mathbb{G}_1}$ assumption, by swapping $\mathbb{G}_1$ and $\mathbb{G}_2$ in the definition above. In [AFGHO16, Lemma 2] Abe et al. prove that if DDH holds in a source group relative to $\mathsf{SampleGrp}_3$, then DBP also holds in that source group relative to $\mathsf{SampleGrp}_3$. Thus if SXDH holds, then the DBP holds in both source groups.

The q-DBP Assumption is a generalization of the DBP:

Assumption 2 (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* $q > 2 \in \mathbb{N}$*:*

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

Ultimately, we rely on the q-DBP holding in both source groups in the security proof of IPP. By the following lemma we can state the result in terms of the simpler DBP. Finally, using the aforementioned lemma from [AFGHO16] we can tie the security of our argument to the existence of a bilinear group generator for which SXDH holds.

Lemma A.1. *If the* DBP *holds relative to an asymmetric bilinear group generator* $\mathsf{SampleGrp}_3$*, then the* q-DBP *also holds relative to* $\mathsf{SampleGrp}_3$.

Proof. For the case $q = 2$ it is obvious the q-DBP coincides with the DBP. Assume the existence of an efficient algorithm $\mathcal{A}$ that for any $q > 2$ breaks the q-DBP. We construct an efficient algorithm $\mathcal{A}'$ that breaks the DBP as follows.

On input of a challenge $(\langle\mathsf{group}\rangle, h_1, h_2)$ the adversary $\mathcal{A}'$ uniformly samples $\alpha_3,\ldots,\alpha_q \xleftarrow{\$} \mathbb{F}$ for and computes $h_i = h_i^{\alpha_i}$ for $i = 3,\ldots,q$. $\mathcal{A}'$ then runs $\mathcal{A}$ on $(\langle\mathsf{group}\rangle, h_1,\ldots,h_q)$. Since the distribution of the $h_1,\ldots,h_q$ that $\mathcal{A}'$ gives as input to $\mathcal{A}$ is identical to in the q-DBP experiment, with more than negligible probability $\mathcal{A}$ outputs a non-trivial q-double pairing, i.e., values $a_1,\ldots,a_q$ such that $\prod_{i=1}^{q} e(a_i,h_i) = 1$.

In the case $\mathcal{A}$ outputs a non-trivial q-double pairing, $\mathcal{A}'$ sets $a_1' = a_1$ and computes $a_2' = a_2 \cdot \prod_{i=3}^{q} a_i^{\alpha_i}$, and outputs a double pairing a_1', a_2'. Since $\mathcal{A}$ is efficient and succeeds with more than negligible probabilty, the same can be said of $\mathcal{A}'$. □

Our construction of aggregatable signatures relies on the *ψ-co-Diffie-Hellman* (ψ-co-CDH) assumption [BDN18].

Assumption 3 (ψ-co-Diffie Hellman assumption)**.** *We say the* ψ-co-CDH *assumption holds relative to* $\mathsf{SampleGrp}_3$ *if for all efficient adversaries* $\mathcal{A}$ *it holds that*

$$\Pr\left[C = h^{\alpha\beta} \middle| \begin{array}{r} \langle\mathsf{group}\rangle \leftarrow \mathsf{SampleGrp}_3(1^\lambda) \\ (\alpha,\beta) \xleftarrow{\$} \mathbb{F} \\ C \leftarrow \mathcal{A}^{\psi}(\langle\mathsf{group}\rangle, g^\alpha, g^\beta, h^\beta) \end{array}\right] \leq \operatorname{negl}(\lambda) \ ,$$

where the adversary $\mathcal{A}^\psi$ *has oracle access to an isomorphism function* $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ *defined* $\psi(x) = g^{\log_h(x)}$.

A.3 Proof systems

Both of the inner pairing product protocols we introduce in this work are proof systems. SIPP is a proof system that allows a prover to make publicly verifiable proofs attesting to the correct computation of a pairing equation with respect to public inputs. SIPP is then a statistically sound proof of membership for a polynomial-time language. While the verifier could compute the pairing equation themselves in polynomial-time, SIPP is practically useful because verifying a proof of a claimed result is computationally less expensive.

IPP is proof system that allows a prover to prove knowledge of some private witness inputs whose pairing inner product is given as part of the public instance. IPP has computational soundness (specifically computational witness-extended emulation), and thus we refer to it as an argument of knowledge for a NP relation. IPP is useful in applications where the verifier doesn't need all the inputs to the pairing equation, but rather just needs to be convinced such inputs exist such an equation is satisfied.

Public-coin protocols. A protocol is *public-coin* if each verifier message to the prover is an independently and uniformly sampled random string of some prescribed length (or an empty string). Interactive public-coin protocols can be made non-interactive by applying the Fiat-Shamir transformation [FS86]. All the protocols introduced in this work are public-coin, and in Section 5 we indeed apply the Fiat-Shamir transformation to SIPP in order to support non-interactive aggregation of BLS signatures.

Avoiding trusted setup. The most efficient proof systems use a *trusted setup*, where the security guarantees rely on some honest party correctly executing a private-coin Setup algorithm. SIPP avoids the need for setup altogether, while IPP supports a public-coin setup algorithm that can be carried out with public randomness.

A.3.1 Statistically 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. Statistical soundness means that even an unbounded adversary has a negligible chance of convincing the honest verifier to accept an instance $\mathbb{x} \notin \mathcal{L}$. We define these properties formally below.

Definition A.2 (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*

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

Definition A.3 (Statistical soundness)**.** *The proof system* $(\mathsf{Prove}, \mathsf{Verify})$ *for a language* $\mathcal{L}$ *has statistical soundness if for all instances* $\mathbb{x}$ *and adversaries it holds that*

$$\Pr\left[\langle \mathsf{Verify}, \mathcal{A}\rangle(1^\lambda, \mathbb{x}) = 1 \ \wedge \ \mathbb{x} \notin \mathcal{L}\right] \leq \mathrm{negl}(\lambda) \ .$$

A.3.2 Arguments of knowledge

We now define interactive arguments of knowledge $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for NP relations, following Bootle et al. [BCCGP16]. We note that while not made explicit in our definitions below, the relation $\mathcal{R}$ is parametrized by the public parameters pp generated during the proof system setup.

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 $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$.

Definition A.4 (Perfect completeness)**.** *The argument system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for a relation* $\mathcal{R}$ *has perfect completeness if for all instance-witness pairs* $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ *it holds that*

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

Informally, computational witness-extended emulation means 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.5 (Computational witness-extended emulation). *The argument system* (Setup, Prove, Verify) *for a relation $\mathcal{R}$ 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*

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

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

where tr *is the transcript of communication between $\mathcal{P}^*$ and $\mathcal{V}$, the transcript oracle is given by $\mathcal{O} = \langle \mathcal{P}^*(\mathsf{st}),\ \mathcal{V}\rangle(\mathsf{pp}, \mathbb{x})$, 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.6.

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 ree 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.6 (Forking lemma). *Let* (Setup, Prove, 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* (Setup, Prove, Verify) *has computational witness-extended emulation.*

Observe that it is possible to take any NP relation $\mathcal{R}$ and define a relation $\mathcal{R}' = \mathcal{R} \cup \mathcal{R}_{\mathsf{SXDH}}$, where $\mathcal{R}_{\mathsf{SXDH}}$ encompasses instance-witness pairs assumed computationally hard to find under the SXDH. Such a transformation of the relation is implicitly considered in [BBBPWM18] when proving their inner-product argument satisfies *statistical* witness-extended emulation.

A.5 Aggregate signatures

Our definitions of an aggregate scheme follow [BDN18]. An aggregate signature scheme consists of a 5-tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{AggSign}, \mathsf{VerifyAgg})$ that behave as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: a public-coin setup algorithm that, given a security parameter λ (represented in unary), outputs a set of public system parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \overset{\$}{\to} (\mathsf{pk}, \mathsf{sk})$: a key generation algorithm that, given public parameters pp, outputs a public-secret key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m) \to \sigma$: a signing algorithm that, given a secret key sk and message $m \in \{0,1\}^*$, returns a signature σ.
- $\mathsf{AggSign}(\mathsf{pp}, [\mathsf{pk}_i]_{i=1}^n, \{m_i\}_{i=1}^n, [\sigma_i]_{i=1}^n) \to \Sigma$: an offline signature aggregation algorithm that, given a set of n (aggregate public key, message, signature)-triplets $\{(\mathsf{apk}_i, m_i, \sigma_i)\}_{i=1}^n$, outputs aggregate signature Σ.
- $\mathsf{VerifyAgg}(\mathsf{pp}, [\mathsf{pk}_i]_{i=1}^n, \{m_i\}_{i=1}^n, \Sigma) \to 0/1$: an aggregate signature verification algorithm that, given a set of n (aggregate public key, message)-pairs $\{(\mathsf{apk}_i, m_i)\}_{i=1}^n$ and an aggregate signature Σ, returns 1 or 0 to indicate the signature is valid or invalid, respectively.

We require that an aggregate signature scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{AggSign}, \mathsf{VerifyAgg})$ satisfy unforgeability. Informally, in the unforgeability game an adversary is given a challenge public key and signing oracle access to the corresponding secret key. Their goal is to output a valid aggregate signature for a set of distinct messages and a list of corresponding public keys, where for some index j the challenge public key is lined up with a message they did not previously query the oracle on.

Definition A.7 (Computationally unforgeable aggregate signature)**.** *For an aggregate signature scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{AggSign}, \mathsf{VerifyAgg})$ *we define the advantage of an adversary against unforgeability to be defined by* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{forge}}(1^\lambda) = \Pr\left[\mathsf{Game}_{\mathcal{A}}^{\mathsf{forge}}(1^\lambda) = 1\right]$ *where the game* $\mathsf{Game}_{\mathcal{A}}^{\mathsf{forge}}$ *is defined as follows.*

```
$\underline{\mathsf{Game}_{\mathcal{A}}^{\mathsf{forge}}(1^\lambda)}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}^*, \mathsf{sk}^*) \overset{\$}{\leftarrow} \mathsf{KeyGen}(\mathsf{pp})$
$Q \leftarrow \emptyset$
$([\mathsf{pk}_i]_{i=1}^n, \{m_i\}_{i=1}^n, \Sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}}(\mathsf{pp}, \mathsf{pk}^*)$
If $\exists j \ : \ \mathsf{pk}^* = \mathsf{pk}_j \ \wedge \ m_j \notin Q \ \wedge \ \mathsf{VerifyAgg}(\mathsf{pp}, [\mathsf{pk}_i]_{i=1}^n, \{m_i\}_{i=1}^n, \Sigma)$ :
    Return 1
Else return 0

$\underline{\mathsf{Sign}(m)}$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}^*, m)$
$Q \leftarrow Q \cup \{m\}$
Return $\sigma$
```

We say an aggregate signature scheme is computationally unforgeable if for all efficient adversaries $\mathcal{A}$ *it holds that* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{forge}}(1^\lambda) \leq \mathrm{negl}(\lambda)$.

Note that an unforgeable aggregate signature scheme implies an unforgeable signature scheme as defined in [BDN18].

Remark A.8. (Batch verification) While often used interchangably, we note that there is a distinction to be made between our notions of *batch verification* and *aggregate signatures*. In particular, a batch verification algorithm does not require any third party advice in order to verify multiple signatures. An adversary against batch verification succeeds if they can output a set of signatures such that at least one signature would fail verification if checked individually, but the set still passes batch verification. This is a different notion that aggregate unforgeability defined above. Our schemes require third party advice (the aggregated signature and proof) to enable verifiers to check multiple signatures at the same time, and thus we merely satisfy aggregatability and not batch verification. For further discussion on batch verification versus aggregate signatures see [CHP07].

B Optimizations and trade-offs

In this section we present a small survey optimizations we are aware of that can be applied to the protocols we introduce.

Delayed Multi-exponentiations. In inner product arguments the verifier can delay their exponentiations until the final round. After a small precomputation in the field (see section 3.1 of Bünz et al. [BBBPWM18] for details), verification can be reduced to five multi-exponentiations: one of size n in each of the source groups and one (SIPP) or three (IPP) of size $2\log(n)$ in the target group. Using Pippenger's multi-exponentiation algorithm [Pip80], one obtains a $\log(n)$ speedup over computing n exponentiations individually.

The prover can also perform its scaling work in each round using multi-exponentiations.

Batching pairing final exponentiations. Bilinear pairings can generally be broken down into two parts: an initial Miller loop computation and then a target group exponentiation (sometimes called the final exponentiation). When computing a pairing product it is possible to run the Miller loop of each pairing, compute the product of the values, and the compute just a single final exponentiation on that product [Sco19]. Prover time can generally be halved or better by applying this technique.

An additional small speedup can be gained by batching some of the computation of the Miller loops themselves as described in [Sco19].

Sparse Exponents. Pairing-friendly curves at the 128-bit security level generally have a prime-order subgroup of at least 256 bits. For simplicity in our constructions, we have the verifier draw their challenges from a scalar field of this size, when they really only need to draw challenges from a space of size 2^{128}. Using sparse and/or complex exponents in particular can result in a very significant speedup for the prover [CL06], and possibly for the verifier. Indeed, the verifier should expect negligible benefit for large n with this technique if they combine it with previous technique of delaying their multi-exponentiations, since the final exponent values derived in the pre-computation step will no longer be sparse. Sparse exponents provide a constant speedup, versus the logarithmicly increasing speedup of Pippenger's algorithm; the size n at which delaying the multi-exponentiation becomes faster will depend on the curve parameters.

Minimizing proof size and verifier time in IPP. Assume that compressed $\mathbb{G}_2$ elements are twice the size of compressed $\mathbb{G}_1$ elements, and compressed $\mathbb{G}_T$ are six times the size of compressed $\mathbb{G}_1$ elements. In IPP, if one wishes to minimize proof size, then it is optimal to end the recusion early at $m = 16$ and send the vectors $\boldsymbol{A} \in \mathbb{G}_1^{16}$, $\boldsymbol{B} \in \mathbb{G}_2^{16}$. If one wishes to minimize verifier time, then sending $\boldsymbol{A}, \boldsymbol{B}$ as above and then running SIPP on the those vectors is optimal.

C Deferred proofs

In this section we provide proofs for Theorems 3.1, 4.1 and 5.1.

C.1 Proof of Theorem 3.1

Here we prove Theorem 3.1.

Proof. We prove soundness inductively by showing that in each round if $Z' = \boldsymbol{A}' * \boldsymbol{B}'$ then with overwhelming probability $Z = \boldsymbol{A} * \boldsymbol{B}$. To see this, fix any target group generator and 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 $2/|\mathbb{F}|$. This probability follows from the observation that $f(x) \in \mathbb{F}[x, x^{-1}] \cong \mathbb{F}[x, y]/(xy - 1)$. Using the isomorphism that maps $f(x)$ to $f(x, y)$ our claim then follows from the DeMillo-Lipton-Schwartz–Zippel lemma. This is exactly the check the verifier performs in the exponent when they check that $Z_L^x \cdot Z \cdot Z_R^{x^{-1}} = Z' = \boldsymbol{A}' * \boldsymbol{B}'$, where as observed in the completeness proof it holds that

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

Security thus inductively follows from the fact that in the final round the verifier checks that Z' is equal to $e(A', B')$. More precisely, the soundness error ϵ is at most $2\log(m)/|\mathbb{F}|$. □

C.2 Proof of Theorem 4.1

Here we prove Theorem 4.1.

Proof. To prove witness-extended emulation we construct an efficient extractor $\mathcal{E}$ that uses n^2 transcripts, as need by Theorem A.6. On input $(\mathsf{pair}, \boldsymbol{v}, \boldsymbol{w}, T, U, Z)$ our extractor either extracts a witness $(\boldsymbol{A}, \boldsymbol{B})$ such that relation $\mathcal{R}_{\mathsf{pair}}$ holds, or a non-trivial q-double pairing in one of the source groups. Note that the hardness of computing a m'-double pairing that breaks q-DBP in $\mathbb{G}_1$ given $\boldsymbol{w}' \in \mathbb{G}_1^{m'}$ implies the hardness of computing a m-double pairing given $\boldsymbol{w} \in \mathbb{G}_1^m$ (and likewise with respect to $\mathbb{G}_2$, $\boldsymbol{v}'$, and $\boldsymbol{v}$). We proceed by an inductive argument showing in each loop of the protocol we either extract a witness or a non-trivial q-double pairing.

In the base case where $m = 1$ the prover reveals the witness (A, B) in the protocol and the relations $T = e(A, v)$, $U = e(w, B)$, and $Z = e(A, B)$ can be checked directly.

For $m > 1$ the extractor runs the prover to get T_L, T_R, U_L, U_R, Z_L, and Z_R. The extractor then rewinds the prover, giving it three distinct challenges $\{x_1, x_2, x_3\} \xleftarrow{\$} \mathbb{F}$, to obtain three pairs $(\boldsymbol{A}_i', \boldsymbol{B}_i') \in \mathbb{G}_1^{m'} \times \mathbb{G}_2^{m'}$ such that for $i \in [3]$ it holds that

$$\begin{aligned} T_L^{x_i} \cdot T \cdot T_R^{x_i^{-1}} &= \boldsymbol{A}_i' * \left(\boldsymbol{v}_{[:m']}^{x_i^{-1}} \circ \boldsymbol{v}_{[m':]} \right) \\ U_L^{x_i} \cdot U \cdot U_R^{x_i^{-1}} &= \left(\boldsymbol{w}_{[:m']}^{x_i} \circ \boldsymbol{w}_{[m':]} \right) * \boldsymbol{B}_i' \\ Z_L^{x_i} \cdot Z \cdot Z_R^{x_i^{-1}} &= \boldsymbol{A}_i' * \boldsymbol{B}_i' \end{aligned} \tag{2}$$

We use the first three challenges to compute $\nu_1, \nu_2, \nu_3 \in \mathbb{F}_q^\star$ such that

$$\sum_{i=1}^{3} \nu_i x_i = 1, \quad \sum_{i=1}^{3} \nu_i = 0, \quad \sum_{i=1}^{3} \nu_i x_i^{-1} = 0.$$

Then we can write T_L as

$$T_L = \prod_{i=1}^{3} \left(T_L^{x_i} \cdot T \cdot T_R^{x_i^{-1}} \right)^{\nu_i} = \prod_{i=1}^{3} \left(\boldsymbol{A}_i' * \left(\boldsymbol{v}_{[:m']}^{x_i^{-1}} \circ \boldsymbol{v}_{[m':]}^{x_i} \right) \right)^{\nu_i} = \left(\bigcirc_{i=1}^{3} (\boldsymbol{A}_i')^{x_i^{-1} v_i} \| (\boldsymbol{A}_i')^{x_i v_i} \right) * \boldsymbol{v}$$

We define $\boldsymbol{A}_L$ such that $\boldsymbol{A}_L * \boldsymbol{v} = T_L$. By using different systems of equations to define ν_1, ν_2, ν_3, we can use the same technique to derive expressions

$$\begin{array}{lll} T_L = \boldsymbol{A}_L * \boldsymbol{v} & T = \boldsymbol{A}_C * \boldsymbol{v} & T_R = \boldsymbol{A}_R * \boldsymbol{v} \\ U_L = \boldsymbol{w} * \boldsymbol{B}_L & U = \boldsymbol{w} * \boldsymbol{B}_C & U_R = \boldsymbol{w} * \boldsymbol{B}_R \\ Z_L = Y_L & Z = Y_C & Z_R = Y_R \end{array}$$

Now for each $x \in \{x_1, x_2, x_3, \}$ and the corresponding $\boldsymbol{A}', \boldsymbol{B}'$ we can rewrite 2 as

$$\begin{aligned} \boldsymbol{A}' * \left(\boldsymbol{v}_{[:m']}^{x^{-1}} \circ \boldsymbol{v}_{[m':]} \right) &= T_L^x \cdot T \cdot T_R^{x^{-1}} &&= \left(\boldsymbol{A}_L^x \circ \boldsymbol{A}_C \circ \boldsymbol{A}_R^{x^{-1}} \right) * \boldsymbol{v} \\ \left(\boldsymbol{w}_{[:m']}^{x} \circ \boldsymbol{w}_{[m':]} \right) * \boldsymbol{B}' &= U_L^x \cdot U \cdot U_R^{x^{-1}} &&= \boldsymbol{w} * \left(\boldsymbol{B}_L^x \circ \boldsymbol{B}_C \circ \boldsymbol{B}_R^{x^{-1}} \right) \\ \boldsymbol{A}_i' * \boldsymbol{B}_i' &= Z_L^{x_i} \cdot Z \cdot Z_R^{x_i^{-1}} &&= Y_L^x \cdot Y_C \cdot Y_R^{x^{-1}} \end{aligned}$$

Then assuming the q-DBP holds in both source groups it is implied that with all but negligible probability that it holds for each $x \in \{x_1, x_2, x_3\}$ and corresponding $\boldsymbol{A}', \boldsymbol{B}'$ that

$$\begin{aligned} (\boldsymbol{A}')^{x^{-1}} &= \boldsymbol{A}_{L,[:m']}^x \circ \boldsymbol{A}_{C,[:m']} \circ \boldsymbol{A}_{R,[:m']}^{x^{-1}} \\ (\boldsymbol{A}')^{x} &= \boldsymbol{A}_{L,[m':]}^x \circ \boldsymbol{A}_{C,[m':]} \circ \boldsymbol{A}_{R,[m':]}^{x^{-1}} \\ (\boldsymbol{B}')^{x} &= \boldsymbol{B}_{L,[:m']}^x \circ \boldsymbol{B}_{C,[:m']} \circ \boldsymbol{B}_{R,[:m']}^{x^{-1}} \\ \\ (\boldsymbol{B}')^{x^{-1}} &= \boldsymbol{B}_{L,[m':]}^x \circ \boldsymbol{B}_{C,[m':]} \circ \boldsymbol{B}_{R,[m':]}^{x^{-1}} \end{aligned} \tag{3}$$

If any of the first four of these equalities do not hold, we directly obtain a m'-double pairing in one of the source groups. If the equalities hold, we can deduce that for each challenge $x \in \{x_1, x_2, x_3\}$

$$\mathbf{1} = \boldsymbol{A}_{L,[:m']}^{x^2} \circ \left(\boldsymbol{A}_{C,[:m']} \circ \boldsymbol{A}_{L,[m':]}^{-1} \right)^x \circ \left(\boldsymbol{A}_{R,[:m']}) \circ \boldsymbol{A}_{C,[m':]}^{-1} \right)^{x^{-1}} \circ \boldsymbol{A}_{R,[m':]}^{-x^{-2}} \tag{4}$$

$$\mathbf{1} = \boldsymbol{B}_{L,[m':]}^{x^2} \circ \left(\boldsymbol{B}_{C,[m':]} \circ \boldsymbol{B}_{L,[:m']}^{-1} \right)^x \circ \left(\boldsymbol{B}_{R,[m':]} \circ \boldsymbol{B}_{C,[:m']}^{-1} \right)^{x^{-1}} \circ \boldsymbol{B}_{R,[:m']}^{-x^{-2}} \tag{5}$$

where Eq. (4) follows from the first two equations in Eq. (3) and and Eq. (5) follows from the second two equations in Eq. (3). The only way this equality holds for all 3 challenges is if

$$\boldsymbol{A}_{L,[:m']} = \boldsymbol{A}_{R,[m':]} = \boldsymbol{B}_{L,[m':]} = \boldsymbol{B}_{R,[:m']} = \mathbf{1}$$

$$\boldsymbol{A}_{L,[m':]} = \boldsymbol{A}_{C,[:m']} \qquad \boldsymbol{A}_{R,[:m']} = \boldsymbol{A}_{C,[m':]}$$
$$\boldsymbol{B}_{L,[:m']} = \boldsymbol{B}_{C,[m':]} \qquad \boldsymbol{B}_{R,[m':]} = \boldsymbol{B}_{C,[:m']}$$

Plugging these expressions into 3 we have that

$$\boldsymbol{A}' = \boldsymbol{A}^{x}_{C,[:m']} \circ \boldsymbol{A}^{x^{-1}}_{C,[m':]} \quad \text{and} \quad \boldsymbol{B}' = \boldsymbol{B}^{x^{-1}}_{C,[:m']} \circ \boldsymbol{B}^{x}_{C,[m':]}.$$

Using these relations we can see that

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

Since this holds for each $x \in \{x_1, x_2, x_3\}$ it must be that $Z = \boldsymbol{A}_C * \boldsymbol{B}_C$. Thus, the extractor either extracts a q-double pairing or the witness $(\boldsymbol{A}_C, \boldsymbol{B}_C)$.

Using Theorem A.6 we see the extractor requires $3^{\log_2(n)} < n^2$ transcripts and thus runs in expected polynomial time in n and λ, concluding that the protocol has witness-extended emulation. □

C.3 Proof of Theorem 5.1

Here we prove Theorem 5.1.

Proof. Let $\mathcal{A}$ be an adversary against aggregate unforgeability that convinces BLS.VerifyAgg after making no more than q_H queries of the form (m). We show that if $\mathcal{A}$ succeeds then we can break the soundness of the SIPP protocol or we can break the ψ-co-CDH assumption in the random oracle model.

Set $\mathcal{B}$ to be an adversary that behaves as follows.

1. Receive ψ-co-CDH challenge $(g^{\alpha}, g^{\beta}, h^{\beta})$ with respect to the bilinear group pair. Choose $k \xleftarrow{\$} \{1, \ldots, q_H\}$.
2. Set $q = 1$ and run A on input $\mathsf{pk}^* = g^{\alpha}$. When $\mathcal{A}$ queries RO on (m) then program RO such that:
   - if (m) is already assigned, return assigned response.
   - if $\mathsf{pk} \neq \mathsf{pk}^*$ choose $r \xleftarrow{\$} \mathbb{F}$ and assign h^r to (m). Return h^r.
   - if $\mathsf{pk} = \mathsf{pk}^*$ and $q \neq k$, choose $r \xleftarrow{\$} \mathbb{F}$ and assign h^r to (m). Return h^r.
   - if $\mathsf{pk} = \mathsf{pk}^*$ and $q = k$, assign h^{β} to (m) and increment $q = q + 1$. Return h^{β}.
3. When $\mathcal{A}$ queries Sign on $\{m_i, \mathsf{pk}_i\}_{i=0}^{n-1}$ then for $\mathsf{pk}_i = \mathsf{pk}^*$:
   - if (m) is already assigned to h^{β}, return $\perp$.
   - if $q = k$ return $\perp$

   Else, for all $\mathsf{pk}_0, \ldots, \mathsf{pk}_{n-1}$, program RO such that:

   - if (m_i) is already assigned, return assigned response h^{r_i}.

- else choose $r_i \xleftarrow{\$} \mathbb{F}$ and assign h^{r_i} to (m_i). Return h^{r_i}.

   Set $\sigma_A = \prod_{i=0}^{n-1} \psi(\mathsf{pk}_i)^{r_i}$ by querying $\psi(\cdot)$ on each $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ to obtain h^{sk_i}. Return σ_A.

4. When $\mathcal{A}$ returns $\{pk_i, m_i\}, \sigma_A, \pi$, let j be the index such that $\mathsf{pk}_j = \mathsf{pk}^*$ (abort if there is no such index). Abort if $\mathsf{RO}(m_j) \neq h^{\beta}$.

5. If $e(g, \sigma_A) \neq \prod_{i=0}^{n-1} e(\mathsf{pk}_i, \mathsf{RO}(m_i))$ return $g^{-1} \| \boldsymbol{pk}, \sigma_A \| \boldsymbol{h}, 1, \pi$.

6. Compute $B = \sigma_A \prod_{i=0, i\neq j}^{n-1} \psi(\mathsf{pk}_i)^{-r_i}$. Return B.

We will now argue that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{aforge}}(1^{\lambda}) \leq q_H \mathsf{Adv}_{B}^{\mathsf{sound}}(1^{\lambda}) + q_H \mathsf{Adv}_{B}^{\psi-\mathsf{co}-\mathsf{CDH}}(1^{\lambda}).$$

First observe that the probability that $\mathcal{B}$ correctly guesses the $(\mathsf{pk}^* || m)$ which is output by $\mathcal{A}$ in Step 4 is $\frac{1}{q_H}$. This is because $\mathcal{A}$ cannot distinguish between the different oracle queries, and $\mathcal{A}$ queries RO on a maximum of q_H values. Further, at least of these query values is not called by the Sign simulation, because otherwise $\mathcal{A}$ would be forced to return (pk^*, m) which it has queried Sign on (recall messages are distinct).

Assuming that $\mathcal{B}$ correctly guesses k, then either $e(g, \sigma_A) = \prod_{i=0}^{n-1} e(\mathsf{pk}_i, \mathsf{RO}(m_i))$ or $\mathcal{B}$ succeeds against the soundness of the SIPP protocol.

Assuming that $\mathcal{B}$ correctly guesses k and that $e(g, \sigma_A) = \prod_{i=0}^{n-1} e(\mathsf{pk}_i, \mathsf{RO}(m_i))$, then with $\mathsf{pk}_j = \mathsf{pk}^*$

$$\sigma_A = h^{\alpha\beta} \prod_{i=0, i\neq j}^{n-1} h^{\mathsf{sk}_i r_i}.$$

Thus $B = \sigma_A \prod_{i=0, i\neq j}^{n-1} \psi(\mathsf{pk}_i)^{r_i}$ satisfies

$$B = h^{\alpha\beta}.$$

Hence $\mathcal{B}$ succeeds against ψ-co-CDH. □