Aggregatable Signatures from an Inner Pairing Product Argument

Mary Maller
mary.maller.15@ucl.ac.uk
University College London

Noah Vesely
noah.vesely.18@ucl.ac.uk
University College London

October 9, 2019

Abstract

We present a new public-coin setup protocol for aggregating BLS signatures on distinct messages. For n messages the verifier computes just 6 pairings and $6(n + \log(n))$ exponentiations–an improvement on previous aggregate schemes in which the verifier computes $n + 1$ pairings. Our aggregate signature is logarithmic in size. This result uses an *inner pairing product* argument of knowledge that can be used to prove membership in pairing-based languages.

Keywords: Inner Pairing Product Argument, Aggregatable Signatures, Bilinear Pairings

Contents

1 Introduction

In this paper we look into argument systems for proving membership in a relation specific language - namely pairing-based languages. As first observed in the influential paper of Groth and Sahai [GS08], bilinear pairings are a powerful tool. Essentially, they allow verifiers to directly check that given g^a and g^b, a group element C is equal to g^{ab}. This check can be carried out without any additional information from the prover. Thus, using a bilinear pairing, one can check that a *quadratic* equation in unknown variables is satisfied. This puts cryptographers in a position of power. With pairings, we can 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 standardisation efforts to help the efforts of developers [Lan08]. Nonetheless, pairing operations are computationally intensive and protocols usually prioritise minimising pairings above other operations such as exponentiations. We thus believe that it is a worthy goal to design proving systems that can reduce the number of pairing operations that verifiers must perform.

We observe two advantages in designing computation specific provers: (1) there is no requirement to carry out a reduction to NP (which typically adds orders of magnitude onto the prover complexity), and (2) cryptographers can find interesting tricks that use the underlying structure of the computation. In this work, using similar techniques to concurrent work Lai et al. [LMR19], we are able to prove to a verifier that a pairing-based language is satisfied and our proof size is logarithmic. Our scheme is built from a transparent setup. Our verifier complexity, while still linear, trades a linear number of pairings for three pairings and a linear number of group exponentiations. Using this technique, we demonstrate that one can build aggregate signature schemes in which verifiers only need to compute six pairings for any number of distinct messages. Our techniques for reducing the number of pairing equations required to verify BLS signatures also work Pixel [DGNW19] as demonstrated by Vesely in concurrent work [Ves19], and we believe may be extended to batch verification of SNARKs. Our protocols are secure under a public-coin (transparent) setup.

This paper introduces an aggregatable signature scheme which was inspired by Boneh, Gentry, Lynn and Shacham [BGLS03]. Where Bohen et al. observed that BLS signatures can be aggregated, they still require the verifier to compute one pairing per distinct message. We aggregate further by first providing the output of the pairings, and then using our inner pairing product argument to prove the result correct. As a result our verifier needs to evaluate just six pairings. That is, no matter how many messages have been signed, provided one party has performed the aggregation process, all the other verifiers merely need to evaluate the exponentiations (plus six pairings). The aggregation process is approximately six times the price of verifying BLS signatures and we require no special structure on the format of our field.

A promising alternative to our approach of proving pairing-based languages is to represent the pairings in an NP language and prove with a SNARK that the computation is correct [BCTV14]. Bowe et al. achieved this goal using approximately 20,000 constraints per pairing [BCG$^+$18]. We suggest a means by which our techniques might be complimentary, as opposed to competing, to these techniques. If one applied a SNARK to the verifier of our algorithm, it seems likely that the size of the constraint system might be smaller, and thus affecting the concrete performance of the algorithm. The reason we believe this might follow is because representing exponentiations inside a SNARK is cheaper than representing pairings inside a SNARK. Further, our logarithmic sized proof could be hidden inside the witness, and would not affect the size of the resulting argument. The formal and practical analysis of such an approach is outside the scope of this work and we leave it as an open research problem.

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.

	sizes		time complexity	
	$\|\text{CRS}\|$	$\|\pi\|$	prover	verifier
$\mathbb{G}_1$	n	1	$4n$	$2n$
$\mathbb{G}_2$	n	1	$4n$	$2n$
$\mathbb{G}_T$	—	$6\log(n)$	$6\log(n)$	$6\log(n)$
e	—	—	$6n$	3

Table 1: Computational and communication complexity of IPP.

In our aggregate signature scheme, aggregating n signatures requires $6n$ exponentiations in $\mathbb{G}_1$, $4n$ exponentiations in $\mathbb{G}_2$, $6\log(n)$ exponentiations in $\mathbb{G}_T$, and $6n + 2$ pairings. The aggregated signature consists of 1 element in $\mathbb{G}_1$, 2 elements in $\mathbb{G}_2$, and $6\log(n) + 1$ elements in $\mathbb{G}_T$. The verifier computes $4n$ exponentiations in $\mathbb{G}_1$, $2n$ exponentiations in G_2, $6\log(n)$ exponentiations in $\mathbb{G}_T$, and 6 pairings. 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 foundation of this paper is built upon a inner pairing product argument. Using a CRS $(\boldsymbol{w}, \boldsymbol{v}) \in \mathbb{G}_1^n \times \mathbb{G}_2^n$, the inner pairing product argument (IPP) is used to show that for in instance $\mathbb{x} = (T, U, Z) \in \mathbb{G}_T$ there exists a witness $\mathbb{w} = (\boldsymbol{A}, \boldsymbol{B}) \in \mathbb{G}_1^n \times \mathbb{G}_2^n$ such that

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

To obtain this argument, we started from Bootle et al.'s inner product argument [BCC$^+$16] 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. We prove our scheme secure under the Double-Pairing 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 the 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. One complication is that the verifier's equation is applied to hashed messages and not to predetermined random group elements. To combat this, we use the hashed messages as part of the CRS when we apply the inner product argument.

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 verifier time, and a CRS half the size. Both systems achieve identical proof sizes and prove the same security notion

of witness-extended emulation. 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. [BFI$^+$10] noted that it is possible to batch verify pairing equations that share a source group element. However, their results do not extend to the setting where both source group elements are different. Our work can be used to aggregate pairing equations where the source group elements differ. GS proofs are secure in the standard model under standard assumptions. Thus their linear sized proofs and verifier time are optimal [GW11], whereas our smaller proofs can only be obtained because we are in the random oracle model. Nonetheless, this means that GS proofs can be used for applications that require straight-line extractors whereas ours cannot. We also note that, unlike GS proofs [BCC$^+$09], our proofs are not rerandomisable.

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 - an improvement upon our approach. 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 [AFG$^+$16] 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 utilises 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 - indeed they consist of a single group element per message - and thus

the proofs generated using them will also be more efficient. Furthermore, 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 tuple $(a_1, \ldots, a_n)$, and $[\boldsymbol{a}_i]_{i=1}^n = [[a_{i,j}]_{j=1}^m]_{i=1}^n$ as a short-hand for the tuple $(a_{1,1}, \ldots, a_{1,m}, \ldots, a_{n,1}, \ldots, a_{n,m})$; $|\boldsymbol{a}|$ denotes the number of entries in $\boldsymbol{a}$. 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 at random from S.

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 $\mathcal{A}$ is an *efficient adversary* we mean that $\mathcal{A}$ is a family $\{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of non-uniform polynomial-size circuits. If the adversary consists of multiple circuit families $\mathcal{A}_1, \mathcal{A}_2, \ldots$ then we write $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots)$.

We write $\{(\mathbb{x}; \mathbb{w}) : \mathcal{R}\}$ to describe a NP relation $\mathcal{R}$ between public input $\mathbb{x}$ and witness $\mathbb{w}$.

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 a probabilistic polynomial-time algorithm $\mathsf{SampleGrp}$ that, on input 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., DBP 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.

We write the group operation multiplicatively for all three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$.

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 [BBB+18]. For a scalar $x \in \mathbb{F}_q$ 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 $\boldsymbol{x} \in \mathbb{F}_q^n$ we have $\boldsymbol{A}^{\boldsymbol{x}} = \prod_{i=1}^n A_i^{x_i} \in \mathbb{G}$. For two 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. Where we believe it clear to do so we write $\boldsymbol{A} \circ \boldsymbol{A'}$ as $\boldsymbol{A}\boldsymbol{A'}$ and likewise $a_i \cdot a_i'$ as $a_i a_i'$.

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

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

For $x \in \mathbb{F}_q$ we use $\boldsymbol{x}^n$ to denote the vector containing the first n powers of x:

$$\boldsymbol{x}^n = (1, x, x^2, \ldots, x^{n-1}) \in \mathbb{F}_q^n.$$

3 Background

3.1 Definitions

In this section we define what it means for an argument scheme to be sound and an aggregate signature scheme to be unforgeable. Our definitions are defined in the interactive setting, however since the verifier responses in all our arguments simply sample random coins, we imagine the constructions would most generally be realised in the non-interactive setting using random oracles.

3.1.1 Arguments of knowledge

We will later prove that our inner pairing product argument satisfies witness-extended emulation in the sense that any adversary that convinces the verifier must also know a valid witness. Our definition was inspired by Bootle et al. [BCC$^+$16]

Definition 3.1 (Witness-extended emulation)**.** *The argument system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *has witness-extended emulation if for all deterministic polynomial time provers* $\mathcal{P}^*$ *there exists a polynomial time emulator* $\mathcal{E}$ *such that for all pairs of interactive adversaries* $\mathcal{A}_1, \mathcal{A}_2$ *it holds that*

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

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

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

3.1.2 Aggregatable signatures

An aggregate signature scheme consists of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Agg}, \mathsf{VerifyAgg})$ that behave as follows.

- $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$: an efficient, public-coin setup algorithm that on input a security parameter and an integer $n = 2^\ell$ indicating the maximum number of signatures that can be aggregated, outputs public parameters pp.
- $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$: The PPT key generation algorithm takes as input the public parametersa bilinear group description $\langle \mathsf{group} \rangle \in \Lambda$, and outputs a secret key sk and the corresponding public key.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: The DPT signing algorithm takes as input the public parameters, a secret key and a message $m \in \{0,1\}^*$. It returns a signature σ.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: The DPT verification algorithm takes as input the public parameters, a public key, a message and a signature. It outputs 1 if it considers the message to be valid and 0 otherwise.
- $(\sigma_A, \pi) \leftarrow \mathsf{Agg}(\mathsf{pp}, \{\mathsf{pk}_i, m_i, \sigma_i\}_{i=0}^{n-1})$: The DPT signature aggregation algorithm takes as input the public parameters and a list of public keys, messages, and signatures. It outputs an aggregate signature and a proof.

- $\{0,1\} \leftarrow \mathsf{VerifyAgg}(\mathsf{pp}, \{\mathsf{pk}_i, m_i\}_{i=0}^{n-1}, \sigma_A, \pi)$: On input of the public parameters, a list of public keys and messages, a signature and a proof, the DPT aggregate verification algorithm outputs 1 if it considers the signature to be valid and 0 otherwise.

We require that (Setup, KeyGen, Sign, Verify) is unforgeable and that (Setup, KeyGen, Sign, Verify, Agg, Verify) satisfies signature-extended emulation.

We define a signature scheme to be unforgeable if an adversary cannot output a verifying signature under a public key which it does not own, even if it can see signed messages. Observe that this notion is less strong than saying that any adversary that outputs a verifying signature can compute a corresponding secret key — intuitively this is why aggregation algorithms must be careful to prevent rogue key attacks.

Definition 3.2 (Unforgeable Signature)**.** *For a signature scheme* (Setup, KeyGen, Sign, Verify) *we define the advantage of an adversary against unforgeability to be defined by* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{forge}}(1^\lambda) = \Pr[\mathsf{Game}_{\mathcal{A}}^{\mathsf{forge}}(1^\lambda)]$ *where the game* $\mathsf{Game}_{\mathcal{A}}^{\mathsf{forge}}$ *is defined as follows.*

$\underline{\textit{Main}\ \mathsf{Game}_{\mathcal{A}}^{\mathsf{forge}}(1^\lambda)}$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
$Q \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$(m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}}(\mathsf{pk})$
Return 1 *if* $(m, \sigma) \notin Q \ \wedge\ \mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma) = 1$
Else return 0

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

We say a signature scheme is unforgeable if all PPT adversaries $\mathcal{A}$ *have no more than negligible advantage against* $\mathsf{Game}^{\mathsf{forge}}$.

An aggregate signature satisfies signature-extended emulation in the sense that any adversary that outputs a verifying aggregate signature and proof for a set of public keys and messages must also know a verifying signature for each public key and message pair. When combined with the notion of unforgeability, this suffices to demonstrate that an adversary cannot forge aggregate signatures.

Definition 3.3 (Aggregate Unforgeable Signature)**.** *For an aggregate signature scheme*

$$(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Agg}, \mathsf{VerifyAgg})$$

we define the advantage of an adversary against aggregate unforgeability to be defined by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{aforge}}(1^\lambda) = \Pr[\mathsf{Game}_{\mathcal{A}}^{\mathsf{aforge}}(1^\lambda)]$ *where the game* $\mathsf{Game}_{\mathcal{A}}^{\mathsf{aforge}}$ *is defined as follows.*

$\underline{\textit{Main}\ \mathsf{Game}_{\mathcal{A}}^{\mathsf{forge}}(1^\lambda)}$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
$Q \leftarrow \emptyset$
$(\mathsf{pk}_0, \mathsf{sk}_0) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$(\{\mathsf{pk}_i, m_i\}_{i=0}^{n-1}, \sigma_A) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}}(\mathsf{pk}_0)$
Return 1 *if* $(m_0) \notin Q \ \wedge\ \mathsf{Verify}(\mathsf{pp}, \{\mathsf{pk}_i, m_i\}_{i=0}^{n-1}, \sigma_A) = 1$
Else return 0

$\underline{\mathsf{Sign}(m)}$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}_0, m)$
$Q \leftarrow Q \cup \{m\}$
Return σ

We say a signature scheme is unforgeable if all PPT adversaries $\mathcal{A}$ *have no more than negligible advantage against* $\mathsf{Game}^{\mathsf{aforge}}$.

3.2 Assumptions

Our inner pairing product argument relies on the double pairing assumption. We define our assumptions relative to the group generator such that the soundness of our argument relies on the existence of a group generator for which the relevant assumptions hold.

Assumption 1 (Double pairing assumption (DBP))**.** *We say the double pairing assumption holds relative to* $\mathsf{SampleGrp_3}$ *if for any probabilistic polynomial-time algorithm* $\mathcal{A}$

$$\Pr\left[\begin{array}{c|r} (a_1,a_2)\neq(1,1) & \langle\mathsf{group}\rangle\leftarrow\mathsf{SampleGrp_3}(1^\lambda) \\ \wedge & (h_1,h_2)\xleftarrow{\$}\mathbb{G}_2 \\ 1=e(a_1,h_1)e(a_2,h_2) & (a_1,a_2)\leftarrow\mathcal{A}(\langle\mathsf{group}\rangle,h_1,h_2) \end{array}\right]$$

is negligible.

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 [AFG+16, Lemma 2] Abe et al. prove that if $\mathsf{DDH}_{\mathbb{G}}$ holds relative to $\mathsf{SampleGrp_3}$ then so does $\mathsf{DBP}_{\mathbb{G}}$ relative to $\mathsf{SampleGrp_3}$.

The q-DBP Assumption is a generalisation 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* $q\geq 2$*:*

$$\Pr\left[\begin{array}{c|r} (a_1,\ldots,a_q)\neq\mathbf{1} & \langle\mathsf{group}\rangle\leftarrow\mathsf{SampleGrp_3}(1^\lambda) \\ \wedge & (h_1,\ldots,h_q)\xleftarrow{\$}\mathbb{G}_2 \\ 1=\prod_{i=1}^{q}e(a_i,h_i) & (a_1,\ldots,a_q)\leftarrow\mathcal{A}(\langle\mathsf{group}\rangle,h_1,\ldots,h_q) \end{array}\right]$$

is negligible.

While we use the q-DBP in the security proof for our PC scheme, by the following lemma we can state the result in terms of the simpler DBP.

Lemma 3.4. *If the* DBP *holds relative to an asymmetric bilinear group generator* $\mathsf{SampleGrp_3}$*, then the* q-DBP *also holds relative to that* $\mathsf{SampleGrp_3}$*.*

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 $\alpha_3,\ldots,\alpha_q\xleftarrow{\$}\mathbb{F}_q$ 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 $\mathcal{A}$ runs in probabilitic polynomial-time in λ and succeeds with more than negligible probabilty, so does $\mathcal{A}'$. Clearly, for q super-polynomial in λ, no efficient adversary $\mathcal{A}$ can exist. □

3.3 Forking lemma

We will use the following forking lemma from [BCC$^+$16, Lemma 1] in our proof of witness-extended emulation:

Theorem 3.5 (Forking lemma)**.** *Let* $(\mathsf{Prove}, \mathsf{Verify})$ *be a* $(2\mu+1)$*-move, public-coin interactive protocol. Let* $\mathcal{E}$ *be a witness extraction algorithm that always succeeds in extracting a witness from an* $(n_1, \ldots, n_\mu)$*-tree of accepting transcripts in probabilistic polynomial time. Assume that* $\prod_{i=1}^{\mu} n_i$ *is bounded above by a polynomial in the security parameter* λ*. Then* $(\mathsf{Prove}, \mathsf{Verify})$ *has witness-extended emulation.*

4 Inner pairing product argument

It has long been known that pairing-based languagues are expressive enough to capture many relations of interest: indeed Groth-Sahai proofs are often used to build applications such as voting schemes and anonymous credentials. However, many constructions for pairing-based languages have had one significant drawback being that the size of proofs and the cost of the verifier are linearly linked to the pairing product equation being solved. Considering the proofs are interactive, in the standard model it is not possible to achieve better than this. In this section we discuss how in the interactive setting (or the non-interactive setting with random oracles) one can achieve logarithmic sized proofs for pairing-based languages. The prover costs are only a small constant factor more expensive then verifying the statement directly. The verifier costs decrease from a constant number of pairings to a constant number of group exponentiations; further, in the universal SRS model we design a logarithmic verifier.

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

In this section we present our inner pairing product argument IPP for the relation:

$$\mathcal{R}_{\mathsf{pair}} = \left\{ \begin{array}{c} (\langle \mathsf{group} \rangle, \boldsymbol{w} \in \mathbb{G}_1^m,\ \boldsymbol{v} \in \mathbb{G}_2^m,\ T, U, Z \in \mathbb{G}_T; \quad \boldsymbol{A} \in \mathbb{G}_1^m,\ \boldsymbol{B} \in \mathbb{G}_2^m) : \\ T = \boldsymbol{A} * \boldsymbol{v} \quad \wedge \quad U = \boldsymbol{w} * \boldsymbol{B} \quad \wedge \quad Z = \boldsymbol{A} * \boldsymbol{B} \end{array} \right\}. \tag{1}$$

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 $\boldsymbol{A}', \boldsymbol{B}', \boldsymbol{A}', \boldsymbol{B}'$, repectively, 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. [BCC$^+$16] can be adapted to work over a pairing-based language. While one might be tempted to use the improved inner product argument of Bunz et al. [BBB$^+$18], 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

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

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 independently compute

$$\begin{aligned} T' &= T_L^{x^2} \cdot T \cdot T_R^{x^{-2}} & \boldsymbol{v}' &= \boldsymbol{v}_{[:m']}^{x^{-1}} \circ \boldsymbol{v}_{[m':]}^{x} & \in \mathbb{G}_2^{m'} \\ U' &= U_L^{x^2} \cdot U \cdot U_R^{x^{-2}} & \boldsymbol{w}' &= \boldsymbol{w}_{[:m']}^{x} \circ \boldsymbol{w}_{[m':]}^{x^{-1}} & \in \mathbb{G}_1^{m'} \\ Z' &= Z_L^{x^2} \cdot Z \cdot Z_R^{x^{-2}} \end{aligned}$$

The prover finally computes

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

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. 1 we present the interactive protocols $(\mathsf{Prove}, \mathsf{Verify})$.

Transparent setup. Note that generating the public parameters $\mathsf{pp} = (\langle \mathsf{group} \rangle, \boldsymbol{v}, \boldsymbol{w})$ does not require a trusted setup. Indeed the Setup algorithm can use a hash function $H_n : \{0,1\}^* \to \mathbb{G}_1^n \times \mathbb{G}_2^n$ that on input of 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

Our protocol requires $\log(n)$ rounds. In each round of Fig. 1 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 $\mathbb{G}_2$.) Both parties must also compute 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.

4.2.1 Optimizations

Following Section 3.1 of Bünz et al. [BBB$^+$18], the verifier can delay their exponentiations until the final round. After a pre-computation requiring n multi-exponentiations in $\mathbb{F}$ of size at most $\log(n)$, the verification can be reduced to five multi-exponentiations: one of size n in each of the source groups and three of size $2\log(n)$ in $\mathbb{G}_T$. This technique halves the number of group exponentiations that the verifier computes. Additionally, multi-exponentiations can be computed faster than separate exponentiations (see Chapter 6 of [BBB$^+$18] for more details).

$\mathsf{Prove}(\langle \mathsf{group} \rangle,\ \boldsymbol{w}, \boldsymbol{A} \in \mathbb{G}_1^m,\ \boldsymbol{v}, \boldsymbol{B} \in \mathbb{G}_2^m,\ T, U, Z \in \mathbb{G}_T)$ $\mathsf{Verify}(\langle \mathsf{group} \rangle, \boldsymbol{g}, \boldsymbol{h}, T, U, Z)$

if $m = 1$**:**

$$A \in \mathbb{G}_1, B \in \mathbb{G}_2 \longrightarrow$$

if $e(A, v) = T \wedge e(w, B) = U \wedge e(A, B) = Z$**:**
return "accept"

else:

$m' = m/2$ $\quad m' = m/2$
$T_L = \boldsymbol{A}_{[:m']} * \boldsymbol{v}_{[m':]}$
$T_R = \boldsymbol{A}_{[m':]} * \boldsymbol{v}_{[:m']}$
$U_L = \boldsymbol{w}_{[m':]} * \boldsymbol{B}_{[:m']}$
$U_R = \boldsymbol{w}_{[:m']} * \boldsymbol{B}_{[m':]}$
$Z_L = \boldsymbol{A}_{[:m']} * \boldsymbol{B}_{[m':]}$
$Z_R = \boldsymbol{A}_{[m':]} * \boldsymbol{B}_{[:m']}$

$$T_L, T_R, U_L, U_R, P_L, P_R \in \mathbb{G}_T \longrightarrow$$

$x \xleftarrow{\$} \mathbb{F}_q^\star$

$$\longleftarrow x$$

$T' = T_L^{x^2} \cdot T \cdot T_R^{x^{-2}} \in \mathbb{G}_T$ $\quad T' = T_L^{x^2} \cdot T \cdot T_R^{x^{-2}}$
$\boldsymbol{v}' = \boldsymbol{v}_{[:m']}^{x^{-1}} \circ \boldsymbol{v}_{[m':]}^{x} \in \mathbb{G}_2^{m'}$ $\quad \boldsymbol{v}' = \boldsymbol{v}_{[:m']}^{x^{-1}} \circ \boldsymbol{v}_{[m':]}^{x}$
$U' = U_L^{x^2} \cdot U \cdot U_R^{x^{-2}} \in \mathbb{G}_T$ $\quad U' = U_L^{x^2} \cdot U \cdot U_R^{x^{-2}}$
$\boldsymbol{w}' = \boldsymbol{w}_{[:m']}^{x} \circ \boldsymbol{w}_{[m':]}^{x^{-1}} \in \mathbb{G}_1^{m'}$ $\quad \boldsymbol{w}' = \boldsymbol{w}_{[:m']}^{x} \circ \boldsymbol{w}_{[m':]}^{x^{-1}}$
$Z' = Z_L^{x^2} \cdot Z \cdot Z_R^{x^{-2}} \in \mathbb{G}_T$ $\quad Z' = Z_L^{x^2} \cdot Z \cdot Z_R^{x^{-2}}$
$\boldsymbol{A}' = \boldsymbol{A}_{[:m']}^{x} \circ \boldsymbol{A}_{[m':]}^{x^{-1}} \in \mathbb{G}_1^{m'}$
$\boldsymbol{B}' = \boldsymbol{B}_{[:m']}^{x^{-1}} \circ \boldsymbol{B}_{[m':]}^{x} \in \mathbb{G}_2^{m'}$
continue

Figure 1: Protocol for inner pairing product argument.

4.3 Security

In this section we prove that, under the DBP assumption, IPP satisfies witness-extended emulation. Now we prove the following theorem:

Theorem 4.1 (Inner pairing product argument)**.** *The argument defined by Protocol 1 for $\mathcal{R}_{\mathsf{pair}}$ has perfect completeness and witness-extended emulation for either extracting a non-trivial q-double pairing in one of the source groups, or extracting a valid witness $\boldsymbol{A}, \boldsymbol{B}$.*

Proof. To prove witness-extended emulation we construct an efficient extractor $\mathcal{E}$ that uses n^2 transcripts, as need by Theorem 3.5. 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 4 times to giving it four challenges $\{x_0, x_1, x_2, x_3\} \xleftarrow{\$} (\mathbb{F}_q^\star)^4$ to obtain four pairs $(\boldsymbol{A}'_i,\ \boldsymbol{B}'_i) \in \mathbb{G}_1^{m'} \times \mathbb{G}_2^{m'}$ such that for $i \in [4]$ it holds that

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

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

$$\sum_{i=1}^{3} \nu_i x_i^2 = 1, \quad \sum_{i=1}^{3} \nu_i = 0, \quad \sum_{i=1}^{3} \nu_i x_i^{-2} = 0.$$

Then we can write T_L as

$$T_L = \prod_{i=1}^{3} \left(T_L^{x_i^2} \cdot T \cdot T_R^{x_i^{-2}} \right)^{\nu_i} = \prod_{i=1}^{3} \left(\boldsymbol{A}'_i * \left(\boldsymbol{v}_{[:m']}^{x_i^{-1}} \circ \boldsymbol{v}_{[m':]}^{x_i} \right) \right)^{\nu_i} := \boldsymbol{A}_L * \boldsymbol{v}.$$

Using the same technique we can 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 $x \in \{x_1, x_2, x_3, x_4\}$ and the corresponding $\boldsymbol{A}', \boldsymbol{B}'$ we can rewrite 2 as

$$\begin{aligned} \left(\boldsymbol{A}_{L,[:m']}^{x^2} \circ \boldsymbol{A}_{C,[:m']} \circ \boldsymbol{A}_{R,[:m']}^{x^{-2}} \right) * \boldsymbol{v} &= T_L^{x_i^2} \cdot T \cdot T_R^{x_i^{-2}} &&= \boldsymbol{A}' * \left(\boldsymbol{v}_{[:m']}^{x^{-1}} \circ \boldsymbol{v}_{[m':]}^{x} \right) \\ \boldsymbol{w} * \left(\boldsymbol{B}_{L,[:m']}^{x^2} \circ \boldsymbol{B}_{C,[:m']} \circ \boldsymbol{B}_{R,[:m']}^{x^{-2}} \right) &= U_L^{x_i^2} \cdot U \cdot U_R^{x_i^{-2}} &&= \left(\boldsymbol{w}_{[:m']}^{x} \circ \boldsymbol{w}_{[m':]}^{x^{-1}} \right) * \boldsymbol{B}'. \end{aligned}$$

This implies that

$$\begin{aligned}
(\boldsymbol{A}')^{x^{-1}} &= \boldsymbol{A}_{L,[:m']}^{x^2} \circ \boldsymbol{A}_{T,[:m']} \circ \boldsymbol{A}_{R,[:m']}^{x^{-2}} \\
(\boldsymbol{A}')^{x} &= \boldsymbol{A}_{L,[m':]}^{x^2} \circ \boldsymbol{A}_{T,[m':]} \circ \boldsymbol{A}_{1,[m':]}^{x^{-2}} \\
(\boldsymbol{B}')^{x} &= \boldsymbol{B}_{L,[:m']}^{x^2} \circ \boldsymbol{B}_{C,[:m']} \circ \boldsymbol{B}_{R,[:m']}^{x^{-2}} \\
(\boldsymbol{B}')^{x^{-1}} &= \boldsymbol{B}_{L,[m':]}^{x^2} \circ \boldsymbol{B}_{C,[m':]} \circ \boldsymbol{B}_{R,[m':]}^{x^{-2}} \\
\boldsymbol{A}' * \boldsymbol{B}' &= Y_L^{x^2} \cdot Y_C \cdot Y_R^{x^{-2}}
\end{aligned} \tag{3}$$

If any of the first four of these equalities do not hold, we directly obtain a q-double pairing in one of the source groups. If the equalities hold, we can deduce that for each challenge $x \in \{x_1, x_2, x_3, x_4\}$

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

The only way this equality holds for all 4 challenges is if

$$\begin{aligned}
\boldsymbol{A}_{L,[:m']} = \boldsymbol{A}_{R,[m':]} = \boldsymbol{B}_{L,[m':]} = \boldsymbol{B}_{R,[:m']}^{x^{-3}} = \mathbf{1} \\
\boldsymbol{A}_{T,[m':]} = \boldsymbol{A}_{R,[:m']} \qquad \boldsymbol{A}_{T,[:m']} = \boldsymbol{A}_{L,[m':]} \\
\boldsymbol{B}_{U,[m':]} = \boldsymbol{B}_{L,[:m']} \qquad \boldsymbol{B}_{U,[:m']} = \boldsymbol{B}_{R,[m':]}
\end{aligned}$$

Plugging these expressions into 3 we have that

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

Using these relations we can see that

$$\begin{aligned}
Y_L^{x^2} \cdot Y_C \cdot Y_R^{x^{-1}} &= \boldsymbol{A}' * \boldsymbol{B}' \\
&= \boldsymbol{A}_{C,[:m']}^{x} \boldsymbol{A}_{C,[m':]}^{x^{-1}} * \boldsymbol{B}_{C,[:m']}^{x^{-1}} \boldsymbol{B}_{C,[m':]}^{x} \\
&= \left(\boldsymbol{A}_{C,[:m']} * \boldsymbol{B}_{C,[m':]}\right)^{x^2} \cdot \boldsymbol{A}_C * \boldsymbol{B}_C \cdot \left(\boldsymbol{A}_{C,[m':]} * \boldsymbol{B}_{C,[:m']}\right)^{x^{-2}}
\end{aligned}$$

Since this holds for each $x \in \{x_1, x_2, x_3, x_4\}$ 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 3.5 we see the extractor requires $4^{\log_2(n)} = n^2$ transcripts and thus runs in expected polynomial time in n and λ, concluding that the protocol has witness-extended emulation. □

5 Aggregating BLS signatures with distinct messages

Boneh, Lynn, and Shacham introduced the BLS signature scheme [BLS01] which supports off-line 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 three pairings for any number of signatures, even with distinct messages. This is at the expense of a logarithmic aggregation proof in the target group and the verifier must still compute a linear number of hashes and exponentiations.

The basic BLS signature scheme is given in Figure 2. 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. Boneh et al. [BDN18] remarked that the original security proof still applies in Type III groups under the stonger ψ-co-CDH assumption.

$\underline{\mathsf{KeyGen}(\langle \mathsf{group} \rangle)}$:
$\mathsf{sk} \xleftarrow{\$} \mathbb{F}$
$\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
return $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Sign}(\langle \mathsf{group} \rangle, \mathsf{sk}, m)}$:
$h_1 \in \mathbb{G}_2 \leftarrow \mathsf{Hash}_2(m)$
$\sigma_1 \leftarrow h_1^{\mathsf{sk}}$
return σ_1

$\underline{\mathsf{Verify}(\langle \mathsf{group} \rangle, \mathsf{pk}, m, \sigma)}$:
$h_1 \in \mathbb{G}_2 \leftarrow \mathsf{Hash}_2(m)$
Check $e(g, \sigma) = e(\mathsf{pk}, h_1)$
Return 1 if check passes
Else return 0

Figure 2: The BLS signature scheme where Hash_2 is a hash function that maps the message space to the second source group $\mathsf{Hash}_2 : \{0,1\}^* \to \mathbb{G}_2$. We have not described the setup algorithm, which is simply used to determine $\langle \mathsf{group} \rangle$.

Remark 5.1. Boneh et al. [BDN18] designed an aggregate *multisignature* scheme based on BLS signatures where the signature consists of a single group element and verification consists of two pairing equations and a linear number of group exponentiations. However, we observe that their techniques can only be applied when all users that are signing the same message, and whereas our techniques apply to the more general setting where messages may be different. If one wanted to aggregate multi-signatures, we suspect a combination of these schemes would yield promising results.

Using IPP as a subroutine, we introduce a new pair of efficient algorithms $(\mathsf{BLS.Agg}, \mathsf{BLS.VerifyAgg})$. The algorithm $\mathsf{BLS.Agg}$ is a non-interactive signature aggregation protocol, where the prover performing the aggregation does not need to know any secrets. Informally, the prover computes the right side of the original verification equation, and they additionally compute T and a proof of its correct computation using $\mathsf{IPP.Prove}$. The verifier then checks this proof using $\mathsf{IPP.Verify}$ and if it accepts, it then outputs the result of checking the equality $e(g, \sigma) = T$.

5.1 Construction

We present pseudocode for the $\mathsf{BLS.}\{\mathsf{Agg}, \mathsf{VerifyAgg}\}$ protocols in Figure 3. The protocols are directly given with respect to their random oracles because this allows us to define non-interactive protocols for signature schemes.

First, a transparent setup phase is needed to produce a CRS with n elements in $\mathbb{G}_1$ and one element in $\mathbb{G}_2$. This setup also determines the bilinear group.

The aggregator and verifier begin by ensuring that these messages are distinct, and then query a random oracle to obtain values $h_0, \ldots, h_{n-1} \in \mathbb{G}_2$. The aggregator then computes the aggregated signature simply by multiplying all the signatures together. However they also need to prove that

$$e(g, \sigma_A) = \prod_{i=0}^{n-1} e(\mathsf{pk}_i, h_i).$$

Now we are going to use our IPP protocol, and we want to argue that the inner product of T with an easily computable value is equal to a second easily computable value if and only if T is correct. The aggregate sends $T = \prod_{i=0}^{n-1} e(\mathsf{pk}_i, h_i)$ to a random oracle. Observe that T is one of the input values to the inner product and *not* the output value. The random oracle returns $r \in \mathbb{F}$. It then holds that

$$\prod_{i=0}^{n-1} e(\mathsf{pk}_i, v^{r^i}) = T * \prod_{i=0}^{n-1} e(w_i, v_i^{r^i})$$

with high probability if and only if

$$T = \prod_{i=0}^{n-1} e(\mathsf{pk}_i, h_i).$$

Note that

$$U = \prod_{i=0}^{n-1} e(w_i, v^{r_i}) \ \wedge \ Z = \prod_{i=0}^{n-1} e(\mathsf{pk}_i, v_i^{r_i})$$

each only require one pairing to evaluate. We can see this by rearranging the pairing equations

$$U = \prod_{i=0}^{n-1} e(w_i, v^{r^i}) = \prod_{i=0}^{n-1} e(w_i^{r^i}, v) = e(\prod_{i=0}^{n-1} w_i^{r^i}, v)$$

and

$$Z = \prod_{i=0}^{n-1} e(\mathsf{pk}_i, v^{r^i}) = \prod_{i=0}^{n-1} e(\mathsf{pk}_i^{r^i}, v) = e(\prod_{i=0}^{n-1} \mathsf{pk}_i^{r^i}, v).$$

Having decided on the values T, U, Z, the prover and verifier are ready to run the IPP protocol to show that

$$\prod_{i=0}^{n-1} e(\mathsf{pk}_i, v_i) = T * \prod_{i=0}^{n-1} e(w_i, v_i)$$

and thus that

$$T = \prod_{i=0}^{n-1} e(\mathsf{pk}_i, h_i).$$

If the verifier is convinced that T was computed correctly, then all that is left to do is check the original aggregate signature verification equation, namely that

$$e(\mathbb{G}_1, \sigma_A) = T.$$

It returns 1 if this holds.

$\underline{\mathsf{Setup}(1^\lambda, n)}$:
$\langle \mathsf{group} \rangle \xleftarrow{\$} \mathsf{SampleGrp}_3(1^\lambda)$
$w_0, \ldots, w_{n-1} \xleftarrow{\$} \mathbb{G}_1$
$v \leftarrow \mathbb{G}_2$
$\mathsf{pp} \leftarrow (\langle \mathsf{group} \rangle, \boldsymbol{w}, v)$
Return pp

$\underline{\mathsf{BLS.Agg}(\mathsf{pp}, \{\mathsf{pk}_i, m_i, \sigma_i\}_{i=0}^{n-1})}$:
if $i \neq j \ \wedge \ m_i = m_j$ return $\perp$
$h_1, \ldots, h_{n-1} \leftarrow \mathsf{RO}_2(m_0), \ldots, \mathsf{RO}_2(m_{n-1})$
$\sigma_A \leftarrow \prod_{i=0}^{n-1} \sigma_i$
$T \leftarrow \mathbf{pk} * \boldsymbol{h}$
$r \leftarrow \mathsf{RO}_{\mathbb{F}}(T)$
$U \leftarrow \prod_{i=0}^{n-1} e(w_i, v^{r^i})$
$Z \leftarrow \prod_{i=0}^{n-1} e(\mathsf{pk}_i^{r^i}, v)$
$\pi \xleftarrow{\$} \mathsf{IPPProve}\left(\begin{array}{l} \langle \mathsf{group} \rangle, \boldsymbol{w}, \boldsymbol{pk} \in \mathbb{G}_1^n, \\ \boldsymbol{h}, v^{\boldsymbol{r}} \in \mathbb{G}_2^n, \\ T, U, Z \in \mathbb{G}_T \end{array} \right)$
Return (σ_A, T, π)

$\underline{\mathsf{BLS.Verify}(\mathsf{pp}, \{\mathsf{pk}_i, m_i\}_{i=0}^{n-1})}$:
if $i \neq j \ \wedge \ m_i = m_j$ return $\perp$
$h_1, \ldots, h_{n-1} \leftarrow \mathsf{RO}_2(m_0), \ldots, \mathsf{RO}_2(m_{n-1})$
$r \leftarrow \mathsf{RO}_{\mathbb{F}}(T)$
$U \leftarrow \prod_{i=0}^{n-1} e(w_i, v^{r^i})$
$Z \leftarrow \prod_{i=0}^{n-1} e(\mathsf{pk}_i^{r^i}, v)$
Check $\mathsf{IPPVerify}(\langle \mathsf{group} \rangle, \boldsymbol{w}, \boldsymbol{h}, T, U, Z)$
Check $e(g, \sigma_A) \stackrel{?}{=} T$
Return 1 if checks pass

Figure 3: An aggregated BLS signature scheme where $\mathsf{RO}_{\mathbb{F}}$ is a random oracle that returns field elements and RO_2 is a random oracle that returns elements in $\mathbb{G}_2$.

5.2 Efficiency

Verifying n signatures on n different messages using BLS.VerifyAgg requires running IPP.Verify and an additional $2n$ exponentiations in $\mathbb{G}_1$ and 2 pairings to compute U and Z. They also compute a final pairing to check if $e(g, \sigma_A) = T$. In total, this makes $4n$ exponentiations in $\mathbb{G}_1$, $2n$ exponentiations in G_2, $6\log(n)$ exponentiations in $\mathbb{G}_T$, and 6 pairings. Running BLS.Agg requires running IPP.Prove and an additional $2n$ exponentiations in $\mathbb{G}_1$ and 2 pairings to compute U and Z. In total, this makes $6n$ exponentiations in $\mathbb{G}_1$, $4n$ exponentiations in $\mathbb{G}_2$, $6\log(n)$ exponentiations in $\mathbb{G}_T$, and $6n + 2$ pairings. The output (σ_A, T, π) of the aggregator consists of 1 element in $\mathbb{G}_1$, 2 elements in $\mathbb{G}_2$, and $6\log(n) + 1$ elements in $\mathbb{G}_T$. The CRS required to support these computations consists of n elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$.

5.3 Security

Theorem 5.2. *The aggregate signature scheme in Figure 3 has aggregate unforgeability if the inner pairing product argument* IPP *has witness-extended emulation and the ψ-co-CDH assumption holds.*

Proof. Let $\mathcal{A}$ be an adversary against aggregate unforgeability that convinces BLS.VerifyAgg. We show that if $\mathcal{A}$ succeeds then we can break the witness-extended emulation of the IPP protocol ψ-co-CDH assumption.

We have that $\mathcal{A}$ convinces the IPP verifier. Thus there exists an emulator that outputs

$$A_0, \ldots, A_{n-1}$$

such that $T = \prod_{i=0}^{n-1} e(A_i, h_i)$ and

$$\prod_{i=0}^{n-1} e(A_i, v^{r_i}) = \prod_{i=0}^{n-1} e(\mathsf{pk}_i^{r^i}, v) \tag{4}$$

Setting $s_i = \log_g(\mathsf{pk}_i)$ and $a_i = \log_g(A_i)$, we observe that (4) holds if and only if

$$\sum_{i=0}^{n-1} (a_i - s_i) r^i = 0.$$

By the Schartz-Zippel Lemma this holds with negligible probability unless $(a_i - s_i) = 0$ for all i, implying that $A_i = \mathsf{pk}_i$. Hence either σ_A satisfies

$$e(g, \sigma_A) = \prod_{i=0}^{n-1} e(A_i, h_i)$$

or $\mathcal{A}$ breaks the witness-extended emulation of IPP with overwhelming probability.

The rest of the proof that given an adversary that can compute σ_A there exists an adversary that breaks the ψ-co-CDH assumption now follows from a modification of Theorem 3.2 in [BGLS03]. □

References

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

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the 39th IEEE Symposium on Security and Privacy*, S&P '18, pages 315–334, 2018.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '09, pages 108–125, 2009.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Proceedings of the 35th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '16, pages 327–357, 2016.

[BCG+18] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. *IACR Cryptology ePrint Archive*, 2018:962, 2018.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *Proceedings of the 34th Annual International Cryptology Conference*, CRYPTO '14, pages 276–294, 2014. Extended version at `http://eprint.iacr.org/2014/595`.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory 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, pages 435–464, 2018.

[BFI+10] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch groth-sahai. In *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*, pages 218–235, 2010.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav 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*, pages 416–432, 2003.

[BLS01] Dan Boneh, Ben Lynn, and Hovav 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, pages 514–532, 2001.

[BR93] Mihir Bellare and Phillip 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.*, pages 62–73, 1993.

[DGNW19] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. Cryptology ePrint Archive, Report 2019/514, 2019. `https://eprint.iacr.org/2019/514`.

[EG14] Alex Escala and Jens 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*, pages 630–649, 2014.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, EUROCRYPT '06, pages 339–358, 2006.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[GS08] Jens Groth and Amit 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, pages 415–432, 2008.

[GSW10] Essam Ghadafi, Nigel P. Smart, and Bogdan 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*, pages 177–192, 2010.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, STOC '11, pages 99–108, 2011.

[Lan08] Hoes Lane. Draft standard for identity-based publickey cryptography using pairings. *IEEE P1636*, 3:D1, 2008.

[LMR19] Russel W. F. Lai, Giulio Malavolta, and Viktoria Ronge. Succinct arguments for bilinear group arithmetic: Practical structure-preserving cryptography. Cryptology ePrint Archive, Report 2019/969, 2019. `https://eprint.iacr.org/2019/969`.

[PGHR13] Brian Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, S&P '13, pages 238–252, 2013.

[PS16] David Pointcheval and Olivier 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*, pages 111–126, 2016.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. 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*, pages 228–245, 2007.

[Ves19] Noah Vesely. On the design of polynomial commitment schemes. Master's thesis, University College London, 2019.