SnarkPack: Practical SNARK Aggregation

Nicolas Gailly[1], Mary Maller[2], and Anca Nitulescu[1]

[1] Protocol Labs.
[2] Ethereum Fondation.
`nikkolasg@protocol.ai, mary.maller@ethereum.org, anca@protocol.ai`

Abstract. We present and implement SnarkPack, an argument for aggregating n Groth16 zkSNARKs with a $O(\log n)$ proof size and verifier time. Our techniques are inspired from the inner pairing product argument introduced by Bünz et al. with the difference that our final scheme does not require a different trusted setup, but it reuses the one from the pairing-based SNARK that it aggregates.
The key tool for our SnarkPack construction is a new commitment scheme that allows us to instantiate the inner product pairing argument of Bünz et al. by using existing powers of tau ceremony transcripts. We also describe a scheme that merge together a multi-exponentiation argument and an inner pairing product argument for some common randomness vector with minimal overhead. We further apply some optimisations to our protocol and illustrate it's efficiency by implementing it.
SnarkPack can aggregate 8192 proofs in 8.7s and verify them in 33ms, including un-serialization time, yielding a verification mechanism that is exponentially faster than batching and previous solutions in the field.

Keywords: public-key cryptography, SNARKs, proof aggregation, bilinear pairings

Table of Contents

1 Introduction

Arguments of Knowledge. In decentralised systems, there is need for protocols that enable a prover to post a statement together with a *short* proof, such that any verifier can publicly check that the statement (e.g., correctness of a computation, claims of storage etc.) is true while expending fewer resources, e.g. less time than would be required to re-execute the function. The two key properties in these practical settings are the size of the proof and the verification time. A popular application is a blockchain, in which nodes need to verify all proofs posted in each (periodic) block. A low verification time is, therefore, critical.

A proof system with *succinct verification* allows a verifier to check a nondeterministic polynomial-time computation in time that is much shorter than the time required to run the computation given the NP witness. Succinct Non-interactive ARguments of Knowledge (SNARKs) are proof systems that fulfill these requirements and they are increasingly popular in real-world applications. There has been a series of works on constructing SNARKs [Gro10, Lip12, BCI+13, GGPR13, PHGR13, Lip13, BCTV14, Gro16] with constant-size proofs.

Due to its shortest proof size, Groth16 SNARK have become a de facto standard in blockchain projects. In this usage, all nodes verify proofs posted on each block individually, setting an upper bound on the number of proofs allowed per block and on system scalability.

This work looks into reducing proof size and verifier time even further by exploring techniques to aggregate Groth16 SNARKs without the requirement for additional trusted setups.

Motivation. While there are many SNARKs with constant-sized proofs and succinct verifiers, all nodes in a blockchain network need to verify many proofs coming from different provers individually. This creates a bottleneck where the blockchain bandwitdh is limited by the number of proofs that the nodes can verify in an epoch, i.e. the maximum number of proofs that can be included in a block. One extreme example is the Filecoin [Lab18] proof-of-space blockchain. Filecoin miners must post a Groth16 proof that they correctly computed a Proof-of-Space [Fis19] to onboard storage in the network. Each proof guarantees that the miner correctly "reserves" 32GB of space to store specific files. The chain currently processes a large number of proofs each day: approximately 500,000 Groth16 proofs, representing 15 PiB of storage. This paper presents a way to aggregate these proofs, leading to a reduction in gas spent in those transactions and in verification time. Combined, these effects lead to Filecoin having a larger onboarding rate. Bünz et al. [BMM+19] present a framework for aggregating Groth16 proofs into a $O(\log n)$ sized final proof with verification requiring $O(\log n)$ group exponentiations (in addition to $O(n)$ field operations from the public inputs). This construction can be applied to aggregating Groth16 proofs but requires a specific trusted setup to construct the structured reference string necessary to verify such aggregated proofs. We explore alternative ways to aggregate Groth16 proofs by limiting ourselves to the already available Groth16 trusted setup transcript, therefore avoiding an additional trust assumptions for existing systems.

Contribution. Our construction is based on techniques from [BMM+19] and follows the same framework. However, it aims to make this realizable in practice without the need of further trusted setup ceremonies and to take full advantage of existing building blocks for further optimisations. We focus specifically on Groth16 proofs [Gro16] generated using the same SRS, as opposed to the generic result in [BMM$^+$19] that allow aggregation of proofs generated using different SRS. Our techniques can apply to other pairing-based SNARKs scheme, but this possible extension is out of scope for the present work.

We propose an Argument for Aggregation that has logarithmic-sized proof and a verifier that runs in logarithmic time in the number of proofs to be aggregated. The reference string needed to compute and verify this aggregated proof can be constructed from any pairs of powers of tau ceremonies (for example those of Zcash [zca18] and Filecoin [Fil20]). Our protocol is exponentially more efficient than aggregating these SNARKs via batching: it takes 33ms to verify an aggregated proof for 8192 proofs (including unserialization) versus 621ms when doing batch verification. The former is of 40kB in size. The aggregator can aggregate 8192 proofs in 8.7s.

2 Preliminaries

2.1 Notations and General Background

Bilinear Groups. A bilinear group is given by a description $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ such that

- p is prime, so $\mathbb{Z}_p^* = \mathbb{F}$ is a field.
- $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$ are multiplicative cyclic groups of prime order p.
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear asymmetric map (pairing), which means that $\forall a, b \in \mathbb{Z}_p$: $e(g^a, h^b) = e(g, h)^{ab}$.
- Then we implicitly have that $e(g, h)$ generates $\mathbb{G}_T$.
- Membership in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ can be efficiently decided, group operations and the pairing $e(\cdot,\ \cdot)$ are efficiently computable, generators can be sampled efficiently, and the descriptions of the groups and group elements each have linear size.

Vectors. For n-dimensional vectors $\mathbf{a} \in \mathbb{Z}_p^n, \mathbf{A} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$, we denote their i-th entry by $a_i \in \mathbb{Z}_p, A_i \in \mathbb{G}_1, B_i \in \mathbb{G}_2$ respectively.

Let $\mathbf{A} \| \mathbf{A}' = (A_0, \ldots, A_{n-1}, A'_0, \ldots, A'_{n-1})$ be the concatenation of 2 vectors $\mathbf{A}, \mathbf{A}' \in \mathbb{G}_1^n$.

We write $\mathbf{A}_{[:\ell]} = (A_0, \ldots, A_{\ell-1}) \in \mathbb{G}_1^\ell$ and $\mathbf{A}_{[\ell:]} = (A_\ell, \ldots, A_{n-1}) \in \mathbb{G}_1^{n-\ell}$ to denote slices of vectors $\mathbf{A} \in \mathbb{G}_1^n$ for $0 \le \ell < n-1$.

Same notation holds for vectors $\mathbf{B} \in \mathbb{G}_2^n$ in the second source group.

Inner pairing product. We write group operations as multiplications.

We define $\mathbf{A}^x = (A_0^x, \ldots, A_{n-1}^x) \in \mathbb{G}_1^n$ for a scalar $x \in \mathbb{Z}_p$ and a vector $\mathbf{A} \in \mathbb{G}_1^n$.

We define $\mathbf{A}^{\mathbf{x}} = (A_0^{x_0}, \ldots, A_{n-1}^{x_{n-1}}) \in \mathbb{G}_1^n$ for vectors $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{Z}_p^n, \mathbf{A} \in \mathbb{G}_1^n$.

We define $\mathbf{A} * \mathbf{x} = \prod_{i=0}^{n-1} A_i^{x_i}$ for vectors $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{Z}_p^n, \mathbf{A} \in \mathbb{G}_1^n$.

We define $\mathbf{A} * \mathbf{B} \coloneqq \prod_{i=0}^{n-1} e(A_i, B_i)$ for a pair of source group vectors $\mathbf{A} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$.

We define $\mathbf{A} \circ \mathbf{A}' \coloneqq (A_0 A'_0, \ldots, A_{n-1} A'_{n-1})$ for two vectors in the same group $\mathbf{A}, \mathbf{A}' \in \mathbb{G}_1^n$.

Relations. We use the notation $\mathcal{R}$ to denote an efficiently decidable binary relation.

For pairs $(u, w) \in \mathcal{R}$ we call u the statement and w the witness. We write $\mathcal{R} = \{(u; w) : p(u, w)\}$ to describe a NP relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ between instances u and witnesses w decided by the polynomial-time predicate $p(\cdot,\ \cdot)$. Let $L_\mathcal{R}$ be the language consisting of statements u for which there exist matching witnesses in $\mathcal{R}$.

Polynomial-Time Algorithms. Unless otherwise specified, all the algorithms defined throughout this work are assumed to be probabilistic Turing machines with running time bounded by a polynomial in his input size, where the expectation is taken over the random coins of the algorithm - i.e., PPT.

If $\mathcal{A}$ is a randomized algorithm, we use $y \leftarrow\$ \mathcal{A}(x)$ to denote that y is the output of $\mathcal{A}$ on x. We write $x \leftarrow\$ X$ to mean sampling a value x uniformly from the set X.

By writing $\mathcal{A}\|\chi_{\mathcal{A}}(\sigma)$ we denote the execution of $\mathcal{A}$ followed by the execution of $\chi_{\mathcal{A}}$ on the same input σ and with the same random coins. The output of the two are separated by a semicolon.

Security Parameter. We denote the computational security parameter with $\lambda \in \mathbb{N}$: A cryptosystem provides λ bits of security if it requires 2^λ elementary operations to be broken.

We say that a function is *negligible* in λ, and we denote it by $\mathsf{negl}(\lambda)$, if it is a $f(\lambda) = \mathcal{O}(\lambda^{-c})$ for any fixed constant c.

Adversaries. Adversaries are PPT algorithms denoted with calligraphic letters (*e.g.*$\mathcal{A}, \mathcal{B}$). They will be usually be modeled as efficient algorithms taking 1^λ as input.
We define the adversary's advantage as a function of parameters to be $\Pr[\mathcal{A} \text{ wins}]$. For a system to be secure, we require that for any efficient adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in the security parameter.

Common and Structured Reference String. The common reference string (CRS) model, introduced by Damgård [Dam00], captures the assumption that a trusted setup in which all involved parties get access to the same string crs taken from some distribution D exists. Schemes proven secure in the CRS model are secure given that the setup was performed correctly. The common reference string model is a generalization of the common random string model, in which D is the uniform distribution of bit strings. We will use the recommended terminology "Structured Reference String" (SRS) since all our crs are structured.

Generic Group Model. The generic group model [Sho97, Mau05] is an idealised cryptographic model, where algorithms do not exploit any special structure of the representation of the group elements and can thus be applied in any cyclic group.

In this model, the adversary is only given access to a randomly chosen encoding of a group, instead of efficient encodings, such as those used by the finite field or elliptic curve groups used in practice.

One of the primary uses of the generic group model is to analyse computational hardness assumptions. An analysis in the generic group model can answer the question: "What is the fastest generic algorithm for breaking a cryptographic hardness assumption". A generic algorithm is an algorithm that only makes use of the group operation, and does not consider the encoding of the group.

2.2 Cryptographic Primitives

SNARKs Let $\mathcal{R}$ be an efficiently computable binary relation which consists of pairs of the form (u, w) and let $L_{\mathcal{R}}$ be the language associated with $\mathcal{R}$.

A Proof or Argument System for $\mathcal{R}$ consists in a triple of PPT algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ defined as follows:

$\mathsf{Setup}(1^\lambda, \mathcal{R}) \rightarrow \mathsf{crs}$**:** takes a security parameter λ and a binary relation $\mathcal{R}$ and outputs a common (structured) reference string crs.

$\mathsf{Prove}(\mathsf{crs}, u, w) \rightarrow \pi$**:** on input crs, a statement u and the witness w, outputs an argument π.

$\mathsf{Verify}(\mathsf{crs}, u, \pi) \rightarrow 1/0$**:** on input crs, a statement u, and a proof π, it outputs either 1 indicating accepting the argument or 0 for rejecting it.

We call Π a Succinct Non-interactive ARgument of Knowledge (SNARK) if further it is complete, succinct and satisfies *Knowledge Soundness* (also called *Proof of Knowledge*).

Non-black-box Extraction. The notion of *Knowledge Soundness* requires the existence of an extractor that can compute a witness whenever the prover $\mathcal{A}$ produces a valid argument. The extractor we defined bellow is non-black-box and gets full access to the prover's state, including any random coins. More formally, a SNARK satisfies the following definition:

Definition 1 (SNARK). *$\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a SNARK for an NP language $L_{\mathcal{R}}$ with corresponding relation $\mathcal{R}$, if the following properties are satisfied.*

Completeness. *For all $(x, w) \in \mathcal{R}$, the following holds:*

$$\Pr\left(\mathsf{Verify}(\mathsf{crs}, u, \pi) = 1 \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, u, w) \end{array}\right) = 1$$

Knowledge Soundness. *For any* PPT *adversary $\mathcal{A}$, there exists a* PPT *extractor* $\mathsf{Ext}_{\mathcal{A}}$ *such that the following probability is negligible in λ:*

$$\Pr\left(\begin{array}{c} \mathsf{Verify}(\mathsf{crs}, u, \pi) = 1 \\ \wedge\, \mathcal{R}(u, w) = 0 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ ((u, \pi); w) \leftarrow \mathcal{A} \| \chi_{\mathcal{A}}(\mathsf{crs}) \end{array}\right) = \mathsf{negl}(\lambda).$$

Succinctness. *For any u and w, the length of the proof π is given by $|\pi| = \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(|u| + |w|)$.*

Zero-Knowledge. A SNARK is zero-knowledge if it does not leak any information besides the truth of the statement. More formally:

Definition 2 (zk-SNARK). *A SNARK for a relation $\mathcal{R}$ is a zk-SNARK if there exists a* PPT *simulator $(\mathcal{S}_1, \mathcal{S}_2)$ such that $\mathcal{S}_1$ outputs a simulated common reference string* crs *and trapdoor* td*; $\mathcal{S}_2$ takes as input* crs*, a statement u and* td*, and outputs a simulated proof π; and, for all* PPT *(stateful) adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, for a state* st*, the following is negligible in λ:*

$$\left| \Pr\left(\begin{array}{c} (u, w) \in \mathcal{R}\, \wedge \\ \mathcal{A}_2(\pi, \mathsf{st}) = 1 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (u, w, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, u, w) \end{array}\right) - \Pr\left(\begin{array}{c} (u, w) \in \mathcal{R}\, \wedge \\ \mathcal{A}_2(\pi, \mathsf{st}) = 1 \end{array} \middle| \begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{S}_1(1^\lambda) \\ (u, w, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs}) \\ \pi \leftarrow \mathcal{S}_2(\mathsf{crs}, \mathsf{td}, u) \end{array}\right) \right| = \mathsf{negl}(\lambda).$$

Commitment Schemes A non-interactive commitment scheme allows a sender to create a commitment to a secret value. It may later open the commitment and reveal the value or some information about the value in a verifiable manner. More formally:

Definition 3 (Non-Interactive Commitment). *A non-interactive commitment scheme is a pair of algorithms* $\mathsf{Com} = (\mathsf{KG}, \mathsf{CM})$*:*

$\mathsf{KG}(1^\lambda) \to \mathsf{ck}$**:** *given a security parameter* λ*, it generates a commitment public key* ck*. This* ck *implicitly specifies a message space* M_{ck}*, a commitment space* C_{ck} *and (optionally) a randomness space* R_{ck}*,. This algorithm is run by a trusted or distributed authority.*

$\mathsf{CM}(\mathsf{ck}; m) \to C$**:** *given* ck *and a message* m*, outputs a commitment* C*. This algorithm specifies a function* $\mathsf{Com}_{\mathsf{ck}} : M_{\mathsf{ck}} \times \mathcal{R}_{\mathsf{ck}} \to C_{\mathsf{ck}}$*. Given a message* $m \in M_{\mathsf{ck}}$*, the sender (optionally) picks a randomness* $\rho \in R_{\mathsf{ck}}$ *and computes the commitment* $C = \mathsf{Com}_{\mathsf{ck}}(m, \rho)$

For deterministic commitments we simply use the notation $C = \mathsf{CM}(\mathsf{ck}; m) := \mathsf{Com}_{\mathsf{ck}}(m, 0)$*, while for randomised ones we write* $C \leftarrow\!\$\, \mathsf{CM}(\mathsf{ck}; m) := \mathsf{Com}_{\mathsf{ck}}(m, \rho)$*.*

A commitment scheme is asked to satisfy one or more of the following properties:

Binding Definition. It is computationally hard, for any PPT adversary $\mathcal{A}$, to come up with two different openings $m \neq m^* \in M_{\mathsf{ck}}$ for the same commitment C. More formally:

Definition 4 (Computationally Binding Commitment). *A commitment scheme* $\mathsf{Com} = (\mathsf{KG}, \mathsf{CM})$ *is computationally binding if for any* PPT *adversary* $\mathcal{A}$*, the following probability is negligible*

$$\Pr\left[\begin{array}{c} m \neq m^* \\ \wedge\ \mathsf{CM}(\mathsf{ck}; m) = \mathsf{CM}(\mathsf{ck}; m^*) = C \end{array} \middle| \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{KG}(1^\lambda) \\ (C; m, m^*) \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Hiding Definition. A commitment can be hiding in the sense that it does not reveal the secret value that was committed.

Definition 5 (Statistically Hiding Commitment). *A commitment scheme* $\mathsf{Com} = (\mathsf{KG}, \mathsf{CM})$ *is statistically hiding if it is statistically hard, for any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, to first generate two messages* $\mathcal{A}_0(\mathsf{ck}) \to m_0, m_1 \in M_{\mathsf{ck}}$ *such that* $\mathcal{A}_1$ *can distinguish between their corresponding commitments* C_0 *and* C_1 *where* $C_0 \leftarrow\!\$\, \mathsf{CM}(\mathsf{ck}; m_0)$ *and* $C_1 \leftarrow\!\$\, \mathsf{CM}(\mathsf{ck}; m_1)$*.*

$$\Pr\left[b = b' \middle| \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{KG}(1^\lambda) \\ (m_0, m_1) \leftarrow \mathcal{A}_0(\mathsf{ck}) \\ b \leftarrow \{0,1\},\ C_b \leftarrow\!\$\, \mathsf{CM}(\mathsf{ck}; m_b) \\ b' \leftarrow \mathcal{A}_1(\mathsf{ck}, C_b) \end{array}\right] = \mathsf{negl}(\lambda).$$

Homomorphic Commitment Scheme. A commitment scheme can also be homomorphic, either in the space of messages or in the space of keys or in both. We call the later *doubly-homomorphic* commitments.

- *Message Homomorphism.* For a group law $+$ on the message space M_{ck} and $\oplus$ on the commitment space C_{ck}, we have that from $C_0 = \mathsf{CM}(\mathsf{ck}; m_0)$ and $C_1 = \mathsf{CM}(\mathsf{ck}; m_1)$, one can efficiently generate $C = \mathsf{CM}(\mathsf{ck}; m_0 + m_1)$ by computing $C = C_0 \oplus C_1 = \mathsf{CM}(\mathsf{ck}; m_0 + m_1)$.

– *Key Homomorphism.* For a group law $\star$ on the key space K_{ck}, and $\oplus$ on the commitment space C_{ck}, we have that from $C_0 = \mathsf{CM}(\mathsf{ck}_0; m)$ and $C_1 = \mathsf{CM}(\mathsf{ck}_1; m)$, one can efficiently generate C so that $C = C_0 \oplus C_1 = \mathsf{CM}(\mathsf{ck}_0 \star \mathsf{ck}_1; m)$.

Polynomial Commitments Polynomial commitments (PCs) first introduced by [KZG10] are commitments for the message space $\mathbb{F}^{\leq d}[X]$, the ring of polynomials in X with maximum degree $d \in \mathbb{N}$ and coefficients in the field $\mathbb{F} = \mathbb{Z}_p$, that support an interactive argument of knowledge $(\mathsf{KG}, \mathsf{Open}, \mathsf{Check})$ for proving the correct evaluation of a committed polynomial at a given point without revealing any other information about the committed polynomial.

A polynomial commitment scheme over a field family $\mathcal{F}$ consists in 4 algorithms $\mathsf{PC} = (\mathsf{KG}, \mathsf{CM}, \mathsf{Open}, \mathsf{Check})$ defined as follows:

$\mathsf{KG}(1^\lambda, d) \to (\mathsf{ck}, \mathsf{vk})$**:** given a security parameter λ fixing a field $\mathcal{F}_\lambda$ family and a maximal degree d samples a group description gk containing a description of a field $\mathbb{F} \in \mathcal{F}_\lambda$, and commitment and verification keys $(\mathsf{ck}, \mathsf{vk})$. We implicitly assume ck and vk each contain gk.

$\mathsf{CM}(\mathsf{ck}; f(X)) \to C$**:** given ck and a polynomial $f(X) \in \mathbb{F}^{\leq d}[X]$ outputs a commitment C.

$\mathsf{Open}(\mathsf{ck}; C, x, y; f(X)) \to \pi$**:** given a commitment C, an evaluation point x, a value y and the polynomial $f(X) \in \mathbb{F}[X]$, it output a prove π for the relation:

$$\mathcal{R}_{\mathsf{kzg}} := \left\{ (\mathsf{ck}, C, x, y; f(X)) : \begin{array}{r} C = \mathsf{CM}\left(\mathsf{ck}; f(X)\right) \\ \wedge \ \deg(f(X)) \leq d \\ \wedge \ y = f(x) \end{array} \right\}$$

$\mathsf{Check}(\mathsf{vk}, C, x, y, \pi) \to 1/0$: Outputs 1 if the proof π verifies and 0 if π is not a valid proof for the opening (C, x, y).

A polynomial commitment satisfy hiding property and an extractable version of binding stated as follows:

Definition 6 (Computational Knowledge Binding). *For every* PPT *adversary $\mathcal{A}$ that produces a valid proof π for statement C, x, y, i.e. such that* $\mathsf{Check}(\mathsf{vk}, C, x, y, \pi) = 1$*, there is an extractor* $\mathsf{Ext}_{\mathcal{A}}$ *that is able to output a pre-image polynomial $f(X)$ with overwhelming probability:*

$$\Pr\left[\begin{array}{c} \mathsf{Check}(\mathsf{vk}, C, x, y, \pi) = 1 \\ \wedge \ C = \mathsf{CM}(\mathsf{ck}; f(X)) \end{array} \middle| \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{KG}(1^\lambda, d) \\ (C, x, y, \pi; f(X)) \leftarrow (\mathcal{A} \| \mathsf{Ext}_{\mathcal{A}})(\mathsf{ck}) \end{array} \right] = 1 - \mathsf{negl}(\lambda).$$

2.3 Assumptions

ASSGP - Auxiliary Structured Single Group Pairing Informally, we assume that a PPT adversary cannot find a vector of group elements $A_0, \ldots, A_{q-1} \in \mathbb{G}_1^q$ such as:

1. $\exists A_i \neq 1_{\mathbb{G}_1}$
2. $e(A_0, h)e(A_1, h^a) \ldots e(A_{q-1}, h^{a^{q-1}}) = 1_{\mathbb{G}_T}$
3. $e(A_0, h)e(A_1, h^b) \ldots e(A_{q-1}, h^{b^{q-1}}) = 1_{\mathbb{G}_T}$

Formally, q-Auxiliary Structured Single Group Pairing (q-ASSGP) assumption can be stated as:

Assumption 1 (ASSGP) *The q-Auxiliary Structured Single Group Pairing (q-ASSGP) assumption holds for the bilinear group generator $\mathcal{G}$ if for all* PPT *adversaries $\mathcal{A}$ we have, on the probability space* $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T) \leftarrow \mathcal{G}(1^\lambda)$, $g \leftarrow\!\$\, \mathbb{G}_1, h \leftarrow\!\$\, \mathbb{G}_2$ *and* $a, b \leftarrow\!\$\, \mathbb{Z}_p$ *the following holds:*

$$\Pr\left[\begin{array}{c|c} (A_0, \ldots, A_{q-1}) \neq \mathbf{1}_{\mathbb{G}_1} & g \leftarrow\!\$\, \mathbb{G}_1, h \leftarrow\!\$\, \mathbb{G}_2, a, b \leftarrow\!\$\, \mathbb{Z}_p \\ \wedge\ \prod_{i=0}^{q-1} e(A_i, h^{a^i}) = 1_{\mathbb{G}_T} & \sigma \leftarrow ([g^{b^i}]_{i=0}^{2q-1}, [g^{a^i}]_{i=0}^{2q-1}, [h^{b^i}]_{i=0}^{2q-1}, [h^{a^i}]_{i=0}^{2q-1}) \\ \wedge\ \prod_{i=0}^{q-1} e(A_i, h^{b^i}) = 1_{\mathbb{G}_T} & (A_1, \ldots, A_q) \leftarrow \mathcal{A}(\mathsf{gk}, \sigma) \end{array}\right] = \mathsf{negl}(\lambda)$$

We refer to this assumption defined for single group paring with elements from second group $\mathbb{G}_2$ as the q-ASSGP2 assumption and also define its dual, the q-ASSGP1 assumption, by swapping $\mathbb{G}_1$ and $\mathbb{G}_2$ in the definition above.

Lemma 1. *The q-ASSGP assumption holds in the generic group model.*

The proof of the lemma can be find in Appendix A.1.

ASDGP - Auxiliary Structured Double Group Pairing Formally, q-Auxiliary Structured Double Group Pairing (q-ASDGP) assumption can be stated as:

Assumption 2 (ASDGP) *The q-ASDGP assumption holds for the bilinear group generator $\mathcal{G}$ if for all* PPT *adversaries $\mathcal{A}$ we have, on the probability space* $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T) \leftarrow \mathcal{G}(1^\lambda)$, $g \leftarrow\!\$\, \mathbb{G}_1, h \leftarrow\!\$\, \mathbb{G}_2$ *and* $a, b \leftarrow\!\$\, \mathbb{Z}_p$ *the following holds:*

$$\Pr\left[\begin{array}{c|c} (\mathbf{A} \neq \mathbf{1}_{\mathbb{G}_1} \vee\ \mathbf{B} \neq \mathbf{1}_{\mathbb{G}_2}) & g \leftarrow\!\$\, \mathbb{G}_1, h \leftarrow\!\$\, \mathbb{G}_2, a, b \leftarrow\!\$\, \mathbb{Z}_p \\ \wedge\ \prod_{i=0}^{q-1} e(A_i, h^{a^i}) \prod_{i=q}^{2q-1} e(g^{a^i}, B_i) = 1_{\mathbb{G}_T} & \sigma \leftarrow ([g^{a^i}, g^{b^i}, h^{a^i}, h^{b^i}]_{i=0}^{2q-1}) \\ \wedge\ \prod_{i=0}^{q-1} e(A_i, h^{b^i}) \prod_{i=q}^{2q-1} e(g^{b^i}, B_i) = 1_{\mathbb{G}_T} & (\mathbf{A}, \mathbf{B}) \leftarrow \mathcal{A}(\mathsf{gk}, \sigma) \end{array}\right] = \mathsf{negl}(\lambda)$$

Lemma 2. *The q-ASDGP assumption holds in the generic group model.*

The proof of the lemma can be found in Appendix A.2.

3 Overview of our Techniques

In this section we present the necessary background and building blocks for aggregating multiple Groth16 proofs for the same SRS (same verification key). This was first introduced by the work [BMM+19].

3.1 Background on Groth16

Before presenting the aggregation argument scheme, we recall here [Gro16] SNARK scheme construction.

Let C be an arithmetic circuit over $\mathbb{Z}_p$, with m wires and d multiplication gates. Let $Q = (t(x), \{v_k(x), w_k(x), y_k(x)\}_{k=0}^{m})$ be a QAP which computes C.

We denote by $I_{io} = \{1, 2, \ldots \ell\}$ the indices corresponding to the public input and public output values of the circuit wires and by $I_{mid} = \{\ell + 1, \ldots m\}$, the wire indices corresponding to the private input and non-input, non-output intermediate values (for the witness).

```
$\underline{\text{Groth.}\mathsf{Setup}(1^\lambda, \mathcal{R})}$
$\alpha, \beta, \gamma, \delta \leftarrow\!\$\, \mathbb{Z}_p^*, \quad s \leftarrow\!\$\, \mathbb{Z}_p^*,$
$\mathsf{crs} = \Big(\mathsf{QAP}, g^\alpha, g^\beta, g^\delta, \{g^{s^i}\}_{i=0}^{d-1}, \Big\{g^{\frac{\beta v_k(s)+\alpha w_k(s)+y_k(s)}{\gamma}}\Big\}_{k=0}^{\ell}, \Big\{g^{\frac{\beta v_k(s)+\alpha w_k(s)+y_k(s)}{\delta}}\Big\}_{k>\ell}, \Big\{g^{\frac{s^i t(s)}{\delta}}\Big\}_{i=0}^{d-2},$
        $h^\beta, h^\gamma, h^\delta, \{h^{s^i}\}_{i=0}^{d-1}\Big)$
$\mathsf{vk} := \big(P = g^\alpha, Q = h^\beta, \Big\{S_k = g^{\frac{\beta v_k(s)+\alpha w_k(s)+y_k(s)}{\gamma}}\Big\}_{k=0}^{\ell}, H = h^\gamma, D = h^\delta\big)$
$\mathsf{td} = (s, \alpha, \beta, \gamma, \delta)$
return $(\mathsf{crs}, \mathsf{td})$
```

```
$\underline{\text{Groth.}\mathsf{Prove}(\mathsf{crs}, u, w)}$
$u = (a_1, \dots, a_\ell),\ a_0 = 1$
$w = (a_{\ell+1}, \dots, a_m)$
$v(x) = \sum_{k=0}^{m} a_k v_k(x)$
$v_{mid}(x) = \sum_{k \in I_{mid}} a_k v_k(x)$
$w(x) = \sum_{k=0}^{m} a_k w_k(x)$
$w_{mid}(x) = \sum_{k \in I_{mid}} a_k w_k(x)$
$y(x) = \sum_{k=0}^{m} a_k y_k(x)$
$y_{mid}(x) = \sum_{k \in I_{mid}} a_k y_k(x)$
$h(x) = \frac{(v(x)w(x) - y(x))}{t(x)}$
$r, u \leftarrow\!\$\, \mathbb{Z}_p^*$
$f_{mid} = \frac{\beta v_{mid}(s) + \alpha w_{mid}(s) + y_{mid}(s)}{\delta}$
$a = \alpha + v(s) + r\delta$
$b = \beta + w(s) + u\delta$
$c = f_{mid} + \frac{t(s)h(s)}{\delta} + ua + rb - ur\delta$
return $\pi = (A = g^a, B = h^b, C = g^c)$
```

```
$\underline{\text{Groth.}\mathsf{Verify}(\mathsf{vk}, u, \pi)}$
$\pi = (A, B, C)$
$v_{io}(x) = \sum_{i=0}^{\ell} a_i v_i(x)$
$w_{io}(x) = \sum_{i=0}^{\ell} a_i w_i(x)$
$y_{io}(x) = \sum_{i=0}^{\ell} a_i y_i(x)$
$f_{io} = \frac{\beta v_{io}(s) + \alpha w_{io}(s) + y_{io}(s)}{\gamma}$
Check
$e(A, B) = e(g^\alpha, h^\beta) \cdot e(g^{f_{io}}, h^\gamma) \cdot e(C, h^\delta)$
```

```
$\underline{\text{Groth.}\mathsf{Sim}(\mathsf{td}, u)}$
$a, b \leftarrow\!\$\, \mathbb{Z}_p^*$
$A = g^a,\ B = g^b$
$c = \frac{ab - \alpha\beta - \beta v_{io}(s) + \alpha w_{io}(s) + y_{io}(s)}{\delta}$
return $\pi = (A = g^a, B = h^b, C = g^c)$
```

Fig. 1. Groth16 Construction from QAP.

We describe SNARK= $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ scheme in [Gro16] that consists in 3 algorithms as per Figure 1.

Note that the Groth16 SRS consist in consecutive powers of some random evaluation point s in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$:

$$\{g^{s^i}\}_{i=0}^{d-1} \in \mathbb{G}_1^d, \quad \{h^{s^i}\}_{i=0}^{d-1} \in \mathbb{G}_2^d.$$

and some additional polynomials evaluated in this random point s.

Remark that for the verification algorithm, we do not use the entire structured reference string crs, but just part of it. For the sake of presentation, we will call the verifier key vk and set it using the necessary elements from the crs:

$$\mathsf{vk} := \Big(P = g^\alpha, Q = h^\beta, \Big\{S_j = g^{\frac{\beta v_j(s)+\alpha w_j(s)+y_j(s)}{\gamma}}\Big\}_{j=0}^{\ell}, H = h^\gamma, D = h^\delta\Big)$$

3.2 Building Blocks

SRS. We need elements from two independent compatible Groth16 SRS:

- Common Bilinear group description for both SRS: $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$

– Common group generators for both SRS: $g \in \mathbb{G}_1, h \in \mathbb{G}_2$
– First SRS with random evaluation point $a \in \mathbb{Z}_p$ for :

$$\mathbf{v}_1 = (h, h^a, \ldots, h^{a^{n-1}}) \text{ and } \mathbf{w}_1 = (g^{a^n}, \ldots, g^{a^{2n-1}})$$

– Second SRS with random evaluation point $b \in \mathbb{Z}_p$ for :

$$\mathbf{v}_2 = (h, h^b, \ldots, h^{b^{n-1}}) \text{ and } \mathbf{w}_2 = (g^{b^n}, \ldots, g^{b^{2n-1}})$$

Inner Product Commitments. To instantiate our commitments, we use pairing commitment schemes inspired by the ones of [LMR19]. These schemes need to satisfy special properties (as discussed in Section 5.1) and they require structured commitment keys $\mathsf{ck}_s, \mathsf{ck}_d$ of the form $\mathsf{ck}_s = (\mathbf{v}_1, \mathbf{v}_2), \mathsf{ck}_d = (\mathbf{v}_1, \mathbf{w}_1, \mathbf{v}_2, \mathbf{w}_2)$. We then commit to vectors $\mathbf{A} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$ as follows:

1. Single group version $\mathsf{CM}_s(\mathbf{A}) \coloneqq \mathsf{CM}_s((\mathbf{v}_1, \mathbf{v}_2); \mathbf{A}) = (T_A, U_A)$ where

$$T_A = \mathbf{A} * \mathbf{v}_1 = e(A_0, h)e(A_1, h^a) \ldots . e(A_{n-1}, h^{a^n})$$
$$U_A = \mathbf{A} * \mathbf{v}_2 = e(A_0, h)e(A_1, h^b) \ldots . e(A_{n-1}, h^{b^n})$$

2. Double group version $\mathsf{CM}_d(\mathbf{A}, \mathbf{B}) \coloneqq \mathsf{CM}_d((\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}_1, \mathbf{w}_2); \mathbf{A}, \mathbf{B}) = (T_{AB}, U_{AB})$ where

$$T_{AB} = (\mathbf{A} * \mathbf{v_1})(\mathbf{w_1} * \mathbf{B}) = e(A_0, h) \cdot \ldots e(A_{n-1}, h^{a^{n-1}}) \cdot e(g^{a^n}, B_0) \cdot \ldots e(g^{a^{2n-1}}, B_{n-1})$$
$$U_{AB} = (\mathbf{A} * \mathbf{v_2})(\mathbf{w_2} * \mathbf{B}) = e(A_0, h) \cdot \ldots e(A_{n-1}, h^{b^{n-1}}) \cdot e(g^{b^n}, B_0) \cdot \ldots e(g^{b^{2n-1}}, B_{n-1})$$

GIPA Protocols. One of the key building blocks for our aggregation protocol are *generalized inner product arguments*, called GIPA protocols.

These protocols, as designed in [BMM$^+$19], enable proving the correctness a large class of inner products between vectors and/or field elements committed using (possibly distinct) doubly-homomorphic commitment schemes.

The schemes we will need here are particular cases of GIPA and require structured references string as commitment keys. Their construction is based on Inner Product Commitment schemes and the KZG Polynomial Commitment [KZG10] (see 2.2). We restate the relations for the two specialized GIPA constructions from [BMM$^+$19] below:

Multiexponentiation Inner Product Proof (MIPP). The relation for MIPP for a known vector $\mathbf{r} \in \mathbb{Z}_p^n$ and a commitment (T_A, U_A) to a vector $\mathbf{A} \in \mathbb{G}_1^n$ is defined by:

$$\mathcal{R}_{\mathsf{mipp}} \coloneqq \{((T_A, U_A), Z, \mathbf{r}; \mathbf{A}) : Z = \mathbf{A} * \mathbf{r} \ \wedge \ (T_A, U_A) = \mathsf{CM}_s(\mathbf{A})\}.$$

Target Inner Pairing Product Proof (TIPP). A TIPP allows a prover to demonstrate that certain pairing relations hold between committed group elements.

More precisely, the relation for the TIPP we need in Groth16 aggregation is defined by:

$$\mathcal{R}_{\mathsf{tipp}} \coloneqq \left\{((T_{AB}, U_{AB}), Z, r; \mathbf{A}, \mathbf{B}) : Z = \mathbf{A} * \mathbf{B}^{\mathbf{r}} \wedge (T_{AB}, U_{AB}) = \mathsf{CM}_d(\mathbf{A}, \mathbf{B}) \wedge \mathbf{r} = (r^i)_{i=0}^{n-1}\right\}.$$

4 Framework for Aggregation

An Argument for Aggregation is a proof system that takes as input multiple proofs and computes a new smaller proof, in this case for n initial proofs we end up with a final aggregated proof of size $\mathcal{O}(\log n)$.

Overview of the protocol. The high-level idea of Groth16 aggregation is quite simple: since Groth16 verification consists in checking a pairing equation between the proof elements $\pi = (A, B, C)$, instead of checking that n pairing equations are simultaneously satisfied it is sufficient to prove that only one inner pairing product of a random linear combination of these initial equations defined by a verifier's random challenge $r \in \mathbb{Z}_p$ holds. In a bit more detail, Groth16 verification asks to check an equation of the type $e(A_i, B_i) = Y_i \cdot e(C_i, D)$ for $Y_i \in \mathbb{G}_T, D \in \mathbb{G}_2$ where Y_i is a value computed from each statement $u_i = \mathbf{a}_i$ and $\pi_i = (A_i, B_i, C_i)_{i=0}^{n-1}$ are proof triples.

The aggregation will instead check a single randomized equation:

$$\prod_{i=0}^{n-1} e(A_i, B_i)^{r^i} = \prod_{i=0}^{n-1} Y_i^{r^i} \cdot e\big(\prod_{i=0}^{n-1} C_i^{r^i}, D\big).$$

This can be rewritten using an inner product notation as :

$$Z_{AB} = Y'_{prod} \cdot e(Z_C, D), \quad \text{and} \quad Z_{AB} \coloneqq \mathbf{A} * \mathbf{B}^{\mathbf{r}} \quad \text{and} \quad Z_C \coloneqq \mathbf{C} * \mathbf{r}$$

where we denoted by $Y'_{prod} \coloneqq \prod_{i=0}^{n-1} Y_i^{r^i}$.

What is left after checking that this unified equation holds is to verify that the elements Z_{AB}, Z_C are consistent with the initial proof triples in the sense that they compute the required inner product. This is done by combining pairing commitments schemes with TIPP and MIPP arguments: the TIPP argument shows that $Z_{AB} = \mathbf{A} * \mathbf{B}^{\mathbf{r}}$ for some initial vectors $\mathbf{A} \in \mathbb{G}_1, \mathbf{B} \in \mathbb{G}_2$ committed using CM_d; the MIPP argument shows that $Z_C = \mathbf{C} * \mathbf{r}$ for some vector $\mathbf{C} \in \mathbb{G}_1$ committed under CM_s.

Our scheme is non-interactive and uses a hash function Hash_0 modeled as a random oracle. We will consider the description of this hash function publicly available for prover and verifier and part of their keys.

Relation for Aggregation. More formally, we introduce the relation for aggregation of n triplets of Groth16 proofs $\mathbf{A}, \mathbf{C} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$:

$$\mathcal{R}_{\mathsf{AGG}} \coloneqq \big\{(\mathsf{vk}, \mathsf{pk}_{\mathsf{agg}}, \mathbf{u} = \{\mathbf{a}_i\}_{i=0}^{n-1}; \pi = \{(\mathbf{A}, \mathbf{B}, \mathbf{C})\}) : \text{ Groth.}\mathsf{Verify}(\mathsf{vk}, u_i, \pi_i) = 1, \ \forall i\big\}$$

where $u_i = \mathbf{a}_i = \{a_{i,j}\}_{j=0}^{\ell}, \pi_i = (A_i, B_i, C_i) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$ for $i = 0, \dots n-1$.

Setup Algorithm
Inputs: $(1^\lambda, \mathcal{R}_{\mathsf{AGG}})$

1. Set commitment keys for both single and double commitment schemes using crs:

$$\mathsf{ck}_s = (\mathbf{v}_1, \mathbf{v}_2), \ \mathsf{ck}_d = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}_1, \mathbf{w}_2).$$

Recall the structure of vectors $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{G}_2$ and $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{G}_1$:

$$\mathbf{v}_1 = (h, h^a, \ldots, h^{a^{n-1}}), \qquad \mathbf{w}_1 = (g^{a^n}, \ldots, g^{a^{2n-1}})$$
$$\mathbf{v}_2 = (h, h^b, \ldots, h^{b^{n-1}}), \qquad \mathbf{w}_2 = (g^{b^n}, \ldots, g^{b^{2n-1}})$$

2. Construct MIPP and TIPP CRS from these keys: $\mathsf{crs_{mipp}}, \mathsf{crs_{tipp}}$
3. Choose a hash function $\mathsf{Hash}_0 : \mathbb{G}_T^4 \to \mathbb{Z}_p$ given by its description hk_0.

Output: Keys: $\mathsf{pk_{agg}} = (\mathsf{vk}, \mathsf{crs_{mipp}}, \mathsf{crs_{tipp}}, \mathsf{hk}_0)$, $\mathsf{vk_{agg}} = (\mathsf{vk}, \mathsf{crs_{mipp}}, \mathsf{crs_{tipp}}, \mathsf{hk}_0)$

Prove Algorithm

Inputs: $\mathsf{pk_{agg}}, \mathbf{u} = \{a_{i,j}\}_{i=0,\ldots n-1; j=0,\ldots \ell}$, $\pi = \{\pi_i\}_{i=0}^{n-1} = (\mathbf{A}, \mathbf{B}, \mathbf{C})$

1. Parse proving key $\mathsf{pk_{agg}} := (\mathsf{vk}, \mathsf{crs_{mipp}}, \mathsf{crs_{tipp}}, \mathsf{hk}_0)$
2. Commit to $\mathbf{A}$ and $\mathbf{B}$: $\mathsf{CM}_d(\mathsf{ck}_d; \mathbf{A}, \mathbf{B}) = (T_{AB}, U_{AB})$
3. Commit to $\mathbf{C}$: $\mathsf{CM}_s((\mathbf{v_1}, \mathbf{v_2}); \mathbf{C}) = (T_C, U_C)$
4. Derive random challenge $r = \mathsf{Hash}_0(T_{AB}, U_{AB}, T_C, U_C) \in \mathbb{Z}_p$ and set $\mathbf{r} = \{r^i\}_{i=0}^{n-1}$
5. Compute $Z_{AB} = \mathbf{A}^{\mathbf{r}} * \mathbf{B}$
6. Compute $Z_C = \mathbf{C} * \mathbf{r} = \prod_{i=0}^{n-1} C_i^{r_i}$.
7. Run TIPP proof:

$$\pi_{\mathsf{tipp}} = \mathsf{TIPP.Prove}(\mathsf{crs_{tipp}}, (T_{AB}, U_{AB}), Z_{AB}, r; \mathbf{A}, \mathbf{B}) \tag{1}$$

8. Run MIPP proof:

$$\pi_{\mathsf{mipp}} = \mathsf{MIPP.Prove}(\mathsf{crs_{mipp}}, (T_C, U_C), Z_C, r; \mathbf{C}) \tag{2}$$

Output: Aggregated proof $\pi_{\mathsf{agg}} = ((T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, Z_C, \pi_{\mathsf{tipp}}, \pi_{\mathsf{mipp}})$

Verification Algorithm

Inputs: $\mathsf{vk_{agg}}, \mathbf{u} = \{a_{i,j}\}_{i=0,\ldots n-1; j=0,\ldots \ell}, \pi_{\mathsf{agg}}$

1. Parse verification key $\mathsf{vk_{agg}} := (\mathsf{vk}, \mathsf{crs_{mipp}}, \mathsf{crs_{tipp}}, \mathsf{hk}_0)$
2. Parse $\mathsf{vk} := \big(P = g^\alpha, Q = h^\beta, \{S_j = g^{\frac{\beta v_j(s) + \alpha w_j(s) + y_j(s)}{\gamma}}\}_{j=0}^{\ell}, H = h^\gamma, D = h^\delta\big)$
3. Compute $Z_{S_j} = S_j^{\sum_{i=0}^{n-1} a_{ij} r_i}$ for all $j = 0 \ldots \ell$.
4. Derive random challenge $r = \mathsf{Hash}_0(T_{AB}, U_{AB}, T_C, U_C)$
5. Check TIPP proof $b_1 \leftarrow \mathsf{TIPP.Verify}(\mathsf{crs_{tipp}}, (T_{AB}, U_{AB}), Z_{AB}, r, \pi_{\mathsf{tipp}})$.
6. Check MIPP proof $b_2 \leftarrow \mathsf{MIPP.Verify}(\mathsf{crs_{mipp}}, (T_C, U_C), Z_C, r, \pi_{\mathsf{mipp}})$.
7. Check Groth16 aggregated equations to the decision bit b_3:

$$Z_{AB} \stackrel{?}{=} e(P^{\sum_{i=0}^{n-1} r^i}, Q) e(\prod_{j=0}^{\ell} Z_{S_j}, H) e(Z_C, D)$$

Output: Decision bit $b = b_1 \wedge b_2 \wedge b_3$

5 Building Blocks Instantiation

5.1 Inner Product Pair Commitments

In this section we are looking for a commitment scheme to group elements in a bilinear group that can be compatible with the GIPA protocol described in [BMM$^+$19]. Our goal is to find such a commitment scheme that uses a structured reference string similar to the one used in many popular SNARK implementations, e.g. Groth16.

In order to use them in specialized GIPA protocols, we require the following properties from our commitment schemes:

- *Computationally Binding Commitment:* as per Definition 4
- *Constant Size Commitment:* the commitment value is independent of the length of the committed vector (two target group elements in our case)
- *Doubly-Homomorphic:* homomorphic both in the message space and in the key space

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

- *Collapsing Property:* double-homomorphism implies a distributive property between keys and messages that allow to collapse multiple messages via a deterministic function $\mathsf{Collapse}$ defined as follows:

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

There are a few candidates for such schemes, but none of them are adapted for fulfilling our goals.

The commitment scheme proposed by [BMM$^+$19] works under some new assumption that asks for the commitment keys to be structured in a specific way. In order to use this commitment, we need to be careful to not give out certain elements which are present in most SRS from available SNARK setup ceremonies (that we would like to reuse).

The commitment scheme proposed by Lai, Malavolta and Ronge [LMR19] is likely to satisfy the properties, but it is shown to be binding only for unstructured random public parameters, while in order to obtain a log-time verification GIPA scheme, we would need some structure for the commitment keys.

We adapt these commitments from [LMR19] to work with structured keys and we introduce a new assumption that holds in the generic group model and prove the commitments binding for an adversary that has access to these structured public parameters.

To better adapt to the application in the two specialized GIPA protocols, we define two different variants of the commitment scheme, one that takes a vector of elements of a single group $\mathbb{G}_1$, and one that takes two vectors of points in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.

We describe our two schemes and the SRS generation ceremonies required for the generation of the public commitment keys in the following:

SRS. The two commitment schemes have the advantage that they can reuse two compatible (independent) SNARK setup ceremonies for their structured keys generation and therefore can be easily deployed without requiring a new trusted setup.

We ask from the two ceremonies to be using the same basis in the same bilinear group, but two different randomnesses:

1. Generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$
2. Elements related to a random $a \in \mathbb{Z}_p : \mathbf{v}_1 = (h, h^a, \ldots, h^{a^{n-1}})$ and $\mathbf{w}_1 = (g^{a^n}, \ldots, g^{a^{2n-1}})$
3. Elements related to a random $b \in \mathbb{Z}_p : \mathbf{v}_2 = (h, h^b, \ldots, h^{b^{n-1}})$ and $\mathbf{w}_2 = (g^{b^n}, \ldots, g^{b^{2n-1}})$

Construction from 2 powers of tau. The construction comes naturally from 2 power of tau that used the same generators g and h, they will both have different powers $a = \tau_1$ and $b = \tau_2$: $g, h, g^{\tau_1}, \ldots, g^{\tau_1^n}, h^{\tau_1}, \ldots, h^{\tau_1^n}, g^{\tau_2}, \ldots, g^{\tau_2^n}, h^{\tau_2}, \ldots, h^{\tau_2^n}$

Our assumptions rely on the fact that cross powers (e.g. $g^{\tau_1\tau_2}$) are not known to the prover. Since the two SRS we use are the result of two independent ceremonies, it is unlikely that such terms can be learned since τ_1 and τ_2 were destroyed after the SRS generation.

Single group version CM_s. This version is useful for the MIPP argument used during Groth16 aggregation. It takes one vector $\mathbf{A} \in \mathbb{G}_1^n$ and outputs two target group elements $(T_A, U_A) \in \mathbb{G}_T^2$ as a commitment.

$\mathsf{KG}(1^\lambda) \rightarrow \mathsf{ck}_s = (\mathbf{v}_1, \mathbf{v}_2)$
$\mathsf{Com}(\mathsf{ck}_s = (\mathbf{v}_1, \mathbf{v}_2), \mathbf{A} = (A_0, \ldots, A_{n-1})) \rightarrow (T_A, U_A)$:

1. $T_A = \mathbf{A} * \mathbf{v}_1 = e(A_0, h) \cdot e(A_1, h^a) \ldots e(A_{n-1}, h^{a^{n-1}})$
2. $U_A = \mathbf{A} * \mathbf{v}_2 = e(A_0, h) \cdot e(A_1, h^b) \ldots e(A_{n-1}, h^{b^{n-1}})$
3. For the sake of presentation, we will use the notation $\mathsf{CM}_s((\mathbf{v}_1, \mathbf{v}_2); \mathbf{A}) = (T_A, U_A)$

Lemma 3. *Under the hardness of q-ASSGP assumption for $q = n$, this commitment scheme is computationally binding as per Definition 4.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks the binding property of the commitment scheme. Then, given the output $((T_A, U_A); \mathbf{A}, \mathbf{A}^*)$ of the adversary $\mathcal{A}$ we have that $(T_A, U_A) = (T_{A^*}, U_{A^*})$:

$$e(A_0, h)e(A_1, h^a) \ldots e(A_{n-1}, h^{a^{n-1}}) = e(A_0^*, h)e(A_1^*, h^a) \ldots e(A_{n-1}^*, h^{a^{n-1}}) \tag{3}$$
$$e(A_0, h)e(A_1, h^b) \ldots e(A_{n-1}, h^{b^{n-1}}) = e(A_0^*, h)e(A_1^* h^b) \ldots e(A_{n-1}^*, h^{b^{n-1}}) \tag{4}$$

By applying the homomorphic properties of the commitment scheme to these equations we get:

$$e(A_0/A_0^*, h)e(A_1/A_1^*, h^a) \ldots e(A_{n-1}/A_{n-1}^*, h^{a^{n-1}}) = 1 \tag{5}$$
$$e(A_0/A_0^*, h)e(A_1/A_1^*, h^b) \ldots e(A_{n-1}/A_{n-1}^*, h^{b^{n-1}}) = 1 \tag{6}$$

where the vector $(A_0/A_0^*, A_1/A_1^*, \ldots A_{n-1}/A_{n-1}^*) \neq \mathbf{1}_{\mathbb{G}_1}$. This breaks the n-ASSGP assumption.

Double group version CM_d. This version is useful for the TIPP argument used during Groth16 aggregation. It takes two vectors $\mathbf{A} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$ and outputs two target group elements $(T_{AB}, U_{AB}) \in \mathbb{G}_T^2$ as a commitment.

$\mathsf{KG}(1^\lambda) \rightarrow \mathsf{ck}_d = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}_1, \mathbf{w}_2)$
$\mathsf{Com}(\mathsf{ck}_d, \mathbf{A}, \mathbf{B}) \rightarrow (T_{AB}, U_{AB})$:

1. $T_{AB} = (\mathbf{A} * \mathbf{v_1})(\mathbf{w_1} * \mathbf{B}) = e(A_0, h) \dots e(A_{n-1}, h^{a^{n-1}}) \cdot e(g^{a^n}, B_0) \dots e(g^{a^{2n-1}}, B_{n-1})$
2. $U_{AB} = (\mathbf{A} * \mathbf{v_2})(\mathbf{w_2} * \mathbf{B}) = e(A_0, h) \dots e(A_{n-1}, h^{b^{n-1}}) \cdot e(g^{b^n}, B_0) \dots e(g^{b^{2n-1}}, B_{n-1})$
3. We write $\mathsf{CM}_d((\mathbf{v_1}, \mathbf{v_2}, \mathbf{w_1}, \mathbf{w_2}); \mathbf{A}, \mathbf{B}) = (T_{AB}, U_{AB})$

Lemma 4. *Under the hardness of q-ASDGP assumption for $q = n$, this commitment scheme is computationally binding.*

Proof. The proof is analogous to the one of Lemma 3. Since the commitment is homomorphic breaking the binding is equivalent to finding a non-trivial opening to 1. Thus it breaks the assumption.

It is straightforward to check that the two version of pairing commitment schemes CM_s and CM_d are inner product commitments, in the sense that they satisfy the other necessary properties: constant size, doubly-homomorphic and identity is a Collapse function defined $\mathsf{Collapse}_{id}(C) = C$.

5.2 Generalized Inner Product Arguments

Generalized Inner Product Arguments (GIPA) are designed to prove generalizations of the inner product argument. For completeness, we describe the framework used by [BMM+19] to build GIPA schemes for two different types of inner product. A main observation is that both TIPP and MIPP protocols are described with respect to doubly-homomorphic inner product commitment schemes such that the inner product map is well-defined over their message space. For our instantiations, we will use the inner product commitment schemes introduced in Section 5.1 which satisfy the desired properties for usage in the two inner pairing product protocols.

We present the two protocols TIPP and MIPP each for a different generalization of the inner product. The two generalized inner product maps for bilinear group $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ that we consider are defined by the following operations:

1. Multiexponentiation inner product map $\mathbb{G}_1^m \times \mathbb{F}^m \to \mathbb{G}_1$:
$$\mathbf{A} * \mathbf{b} = \prod A_i^{b_i}$$
2. Inner pairing product map $\mathbb{G}_1^m \times \mathbb{G}_2^m \to \mathbb{G}_T$:
$$\mathbf{A} * \mathbf{B} := \prod e(A_i, B_i) = \prod e(g^{a_i}, g^{b_i}) = e(g, g)^{\sum a_i b_i}$$
where e is the standard pairing map from $\mathsf{gk} : e : \mathbb{G}_1^m \times \mathbb{G}_2^m \to \mathbb{G}_T$

Commitment Schemes: For the sake of simplicity, we use a "merged" commitment that can be applied to the vectors $\mathbf{A}$ and $\mathbf{B}$ and the result of the inner product $Z = \mathbf{A} * \mathbf{B}^{\mathbf{r}}$ at once with a composed commitment key defined as $\mathsf{ck} = (\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3)$.

We recall the doubly-homomorphic property of the commitment scheme in multiplicative notations:

1. $\mathsf{CM}(\mathsf{ck}, M) \cdot \mathsf{CM}(\mathsf{ck}, M') = \mathsf{CM}(\mathsf{ck}, M \cdot M')$
2. $\mathsf{CM}(\mathsf{ck}, M) \cdot \mathsf{CM}(\mathsf{ck}', M) = \mathsf{CM}(\mathsf{ck} \cdot \mathsf{ck}', M)$
3. (Follows from 1&2) $\mathsf{CM}(\mathsf{ck}^x, M) = \mathsf{CM}(\mathsf{ck}, M^x)$, where $x \in \mathbb{Z}_p$.

KZG Polynomial Commitment. We will need a polynomial commitment scheme (Definition 2.2) that allows for openings of evaluations on a point and proving correctness of these openings. Specifically we need the polynomial commitments to prove *succinctly* the correctness of the final commitment keys v_1, v_2 and w_1, w_2 at the end of the protocol. We can use a polynomial commitment scheme here because these final commitment keys have a well defined structure as shown in [BMM⁺19]. The candidate for the PC is KZG scheme in [KZG10] which allows us to have a constant time verifier for this check.

$\mathsf{KZG.PC} = (\mathsf{KZG.KG}, \mathsf{KZG.CM}, \mathsf{KZG.Open}, \mathsf{KZG.Check})$ defined over bilinear groups $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$ as follows:

$\mathsf{KZG.KG}(1^\lambda, n) \to (\mathsf{ck_{kzg}}, \mathsf{vk_{kzg}})$**:** Set keys $\mathsf{ck_{kzg}} = \{g^{\alpha^i}\}_{i=0}^{n-1}, \mathsf{vk_{kzg}} = h^\alpha$.

$\mathsf{KZG.CM}(\mathsf{ck_{kzg}}; f(X)) \to C_f$**:** For $f(X) = \sum_{i=0}^{n-1} f_i X^i$, computes $C_f = \prod_{i=0}^{n-1} g^{f_i \alpha^i} = g^{f(\alpha)}$.

$\mathsf{KZG.Open}(\mathsf{ck_{kzg}}; C_f, x, y; f(X)) \to \pi$**:** For an evaluation point x, a value y, compute the quotient polynomial $q(X) = \dfrac{f(X) - y}{X - x}$ and output prove $\pi = C_q = \mathsf{KZG.CM}(\mathsf{ck_{kzg}}; q(X))$.

$\mathsf{KZG.Check}(\mathsf{vk_{kzg}} = h^\alpha, C_f, x, y, \pi) \to 1/0$**:** Check if $e(C_f \cdot g^{-y}, h) = e(C_q \cdot g^{-x}, h^\alpha)$.

The $\mathsf{KZG.PC}$ scheme works in a similar fashion for a pair of keys of the form $\mathsf{ck_{kzg}} = \{h^{\alpha^i}\}_{i=0}^{n-1}, \mathsf{vk_{kzg}} = g^\alpha$, by just swapping the values in the final pairing equation check to match the correct basis.

5.3 MIPP with Pair Group Commitment

This is a multiexponentiation inner product argument for the relation

$$\mathcal{R}_{\mathsf{mipp}} := \{((T_A, U_A), A, \mathbf{r}; \mathbf{A}) : Z = \mathbf{A} * \mathbf{r} \ \wedge \ (T_A, U_A) = \mathsf{CM}_s(\mathsf{ck}_s; \mathbf{A})\}.$$

The following protocol is a variation of the original MIPP argument presented in [BMM⁺19]. The main difference is the pairing commitment scheme used, in our version we employ CM_s which relies on a Groth16-friendly setup ceremony as discussed in the introduction.

For the proving strategy, the idea is the same as in the MIPP from [BMM⁺19]: In a nutshell, the prover runs a loop and in each iteration it is first splitting the initial vector $\mathbf{A}$ in half and then using collapsing property of the commitment (see definition for $\mathsf{Collapse}$ function in Section 5.1) to recommit to both halves together. Since all components of the commitment are compact, the identity collapsing function is sufficient for us.

After reducing the size of the commitment keys to 1, the prover has to show well-formedness of the such-obtained final structured commitment keys. The final keys v_1, v_2 are interpreted as a KZG polynomial commitment that the prover must open at a random point z. For a more detailed discussion see Section 5.5.

Construction. Our MIPP argument consists in 3 algorithms $\mathsf{MIPP} = (\mathsf{MIPP.Setup}, \mathsf{MIPP.Prove}, \mathsf{MIPP.Verify})$ that work as follows:

$\mathsf{MIPP.Setup}(1^\lambda, \mathcal{R}_{\mathsf{mipp}}) \to \mathsf{crs_{mipp}}$**:**

1. Set commitment keys for KZG scheme from ck_s (CM_s is specified in $\mathcal{R}_{\mathsf{mipp}}$):

$$\begin{aligned} \mathsf{ck}_{1v} &:= \{h^{a^i}\}_{i=0}^{n-1}, & \mathsf{vk}_{1v} &:= g^a \\ \mathsf{ck}_{2v} &:= \{h^{b^i}\}_{i=0}^{n-1}, & \mathsf{vk}_{2v} &:= g^b \\ \mathsf{ck_{kzg}} &:= (\mathsf{ck}_{1v}, \mathsf{ck}_{2v}), & \mathsf{vk_{kzg}} &:= (\mathsf{vk}_{1v}, \mathsf{vk}_{2v}) \end{aligned}$$

2. Fix a hash function $\mathsf{Hash}_1 : \mathbb{Z}_p \times \mathbb{G}_T^6 \to \mathbb{Z}_p$ and its description hk_1.
3. Fix a hash function $\mathsf{Hash}_2 : \mathbb{Z}_p \times \mathbb{G}_2^2 \to \mathbb{Z}_p$ and its description hk_2.
4. Set $\mathsf{crs}_{\mathsf{mipp}} \coloneqq (\mathsf{hk}_1, \mathsf{hk}_2, \mathsf{ck}_s, \mathsf{ck}_{\mathsf{kzg}}, \mathsf{vk}_{\mathsf{kzg}})$

$\mathsf{MIPP.Prove}(\mathsf{crs}_{\mathsf{mipp}}, (T_A, U_A), Z, r; \mathbf{A}, \mathbf{r}) \to \pi_{\mathsf{mipp}}$:

– Loop "split & collapse" for step i
  1. $n_i = n_{i-1}/2$ where $n_0 = n$
  2. If $n_i == 1$: *break*
  3. Compute left and right inner products

$$Z_L = \mathbf{A}_{[n':]}^{\mathbf{r}_{[:n']}} = \prod_{i=0}^{n'} A_{i+n'}^{r_i}$$
$$Z_R = \mathbf{A}_{[:n']}^{\mathbf{r}_{[n':]}} = \prod_{i=0}^{n'} A_{i}^{r_{i+n'}}$$

  4. Compute right cross commitments:

$$\begin{aligned} T_L, U_L &= \mathsf{CM}_s((\mathbf{v}_1, \mathbf{v}_2), \mathbf{A}_{[n':]} || \mathbf{0}) \\ &= ((\mathbf{A}_{[n':]} * \mathbf{v_1}_{[:n']}), (\mathbf{A}_{[n':]} * \mathbf{v_2}_{[:n']})) \end{aligned}$$

  5. Compute left cross commitments:

$$\begin{aligned} T_R, U_R &= \mathsf{CM}_s((\mathbf{v_1}, \mathbf{v_2}), \mathbf{0} || \mathbf{A}_{[:n']}) \\ &= ((\mathbf{A}_{[:n']} * \mathbf{v_1}_{[n':]}), (\mathbf{A}_{[:n']} * \mathbf{v_2}_{[n':]})) \end{aligned}$$

  6. Compute challenge $x_i = \mathsf{Hash}_1(x_{i-1}, Z_L, Z_R, T_R, T_L, U_R, U_L)$ (with $x_0 = 0$)
  7. Compute Hadamard products

$$\mathbf{A} = \mathbf{A}_{[:n']} \circ \mathbf{A}_{[n':]}^{x_i} = (A_0 A_{n'}^{x_i}, \dots, A_{n'-1} A_{n-1}^{x_i})$$
$$\mathbf{r} = \mathbf{r}_{[:n']} \circ \mathbf{r}_{[n':]}^{x_i^{-1}} = (r_0 r_{n'}^{x_i^{-1}}, \dots, r_{n'-1} r_{n-1}^{x_i^{-1}})$$

  8. Set new rescaled key

$$(\mathbf{v}_1, \mathbf{v}_2) \coloneqq (\mathbf{v_1}_{[:n']} \circ \mathbf{v_1}_{[n':]}^{x_i^{-1}}, \mathbf{v_2}_{[:n']} \circ \mathbf{v_2}_{[n':]}^{x_i^{-1}}) \tag{7}$$

– Prove correctness of final commitment key $(v_1, v_2) \in \mathbb{G}_2^2$:
  1. Define $f_v(X) = \prod_{i=0}^{\ell-1}(1 + x_{\ell-j}^{-1} X^{2^j})$ for $n = 2^\ell$
  2. Draw challenge $z = \mathsf{Hash}_2(x_\ell, v_1, v_2)$
  3. Prove that $v_1 = h^{f_v(a)}$ and $v_2 = h^{f_v(b)}$ are KZG commitments of $f_v(X)$ by evaluation in z

$$\pi_{v_1} \leftarrow \mathsf{KZG.Open}(\mathsf{ck}_{1v}; v_1, z, f_v(z); f_v(X))$$
$$\pi_{v_2} \leftarrow \mathsf{KZG.Open}(\mathsf{ck}_{2v}; v_2, z, f_v(z); f_v(X))$$

  4. Set $\pi_v = (\pi_{v_1}, \pi_{v_2})$

– For A and r' (elements from the last step of the loop with respect $\mathbf{A}$ and to $\mathbf{r}$) set

$$\pi_{\mathsf{mipp}} = (A, r', \mathbf{Z_L}, \mathbf{Z_R}, \mathbf{T_L}, \mathbf{T_R}, \mathbf{U_L}, \mathbf{U_R}, (v_1, v_2), \pi_v)$$

$\mathsf{MIPP.Verify}(\mathsf{crs}_{\mathsf{mipp}}, (T_A, U_A), Z, r; \pi_{\mathsf{mipp}}) \to b$**:**

```
– Loop iterator $i : 1 \to \ell$:
    1. Reconstruct challenges $\{x_i = H(x_{i-1}, \mathbf{Z_L}[i], \mathbf{Z_R}[i], \mathbf{T_L}[i], \mathbf{T_R}[i], \mathbf{U_L}[i], \mathbf{U_R}[i])\}_{i=1}^{\ell}$
       with $x_0 = 0$
    2. Construct final commitment values:
        • $Z = \mathbf{Z_L}[i]^{x_i} \cdot Z \cdot \mathbf{Z_R}[i]^{x_i^{-1}}$ (representing $\mathsf{CM}_{id}(\mathbf{1}_{\mathbb{G}_2}; \mathbf{A}^{\mathbf{r}})$)
        • $T = \mathbf{T_L}[i]^{x_i} \cdot T_A \cdot \mathbf{T_R}[i]^{x^{-1}}$,
        • $U = \mathbf{U_L}[i]^{x_i} \cdot U_A \cdot \mathbf{U_R}[i]^{x^{-1}}$ (representing $\mathsf{CM}_s(\mathsf{ck}_s; \mathbf{A})$)
– Verify commitments into decision bit $b_0$:
    1. $Z == A^{r'}$
    2. Check if $T == e(A, v_1)$,    $U == e(A, v_2)$
– Verify final commitment keys $v_1, v_2$ via KZG
    1. Reconstruct KZG challenge point: $z = H(x_\ell, v_1, v_2)$
    2. Reconstruct commitment polynomial $f_v(X) = \prod_{i=0}^{l-1}(1 + x_{\ell-j}^{-1} X^{2^j})$
    3. Run verification for openings of evaluations in $z$

          $b_1 \leftarrow \mathsf{KZG.Check}(\mathsf{vk}_{1v}; v_1, z, f_v(z); \pi_{v_1})$
          $b_2 \leftarrow \mathsf{KZG.Check}(\mathsf{vk}_{2v}; v_2, z, f_v(z); \pi_{v_2})$

    4. Set $b = b_0 \wedge b_1 \wedge b_2$
```

The security result for the MIPP protocol is following the same arguments as the one in [BMM+19]:

Theorem 3. *If* CM_s *is a binding inner product commitment,* $\mathsf{KZG.PC}$ *is a polynomial commitment with Computational Knowledge Binding as per Definition 6, then the protocol MIPP has computational knowledge soundness (Definition 1).*

Remark that both CM_s and $\mathsf{KZG.PC}$ schemes are secure in the Generic Group Model (or under specific assumptions such as q-ASSGP and q-SDH).

5.4 TIPP with Pair Group Commitment

This is an inner pairing argument for the relation:

$$\mathcal{R}_{\mathsf{tipp}} \coloneqq \big\{((T_{AB}, U_{AB}), Z, r; \mathbf{A}, \mathbf{B}) : Z = \mathbf{A} * \mathbf{B}^{\mathbf{r}} \wedge \\ (T_{AB}, U_{AB}) = \mathsf{CM}_d(\mathsf{ck}_d; \mathbf{A}, \mathbf{B}) \ \wedge \ \mathbf{r} = (r^i)_{i=0}^{n-1}\big\}.$$

where $(T_{AB}, U_{AB}) \in \mathbb{G}_T \times \mathbb{G}_T,\ Z = \mathbf{A} * \mathbf{B}^{\mathbf{r}} \in \mathbb{G}_T,\ \mathbf{A} \in \mathbb{G}_1^n,\ \mathbf{B} \in \mathbb{G}_2^n,\ r \in \mathbb{Z}_p$.

It works similarly to MIPP argument, with the difference that vectors of group elements $\mathbf{A}, \mathbf{B}$ are committed together using the double version of the pairing commitment scheme CM_d.

Construction. Our TIPP argument consists in 3 algorithms $\mathsf{TIPP} = (\mathsf{TIPP.Setup}, \mathsf{TIPP.Prove}, \mathsf{TIPP.Verify})$ that work as follows:

$\mathsf{TIPP.Setup}(1^\lambda, \mathcal{R}_{\mathsf{tipp}}) \to \mathsf{crs}_{\mathsf{tipp}}$**:**

1. Set commitment keys for KZG scheme:

$$\mathsf{ck}_{1v} := \{h^{a^i}\}_{i=0}^{n-1},\ \mathsf{vk}_{1v} := g^a \qquad \mathsf{ck}_{1w} := \{g^{a^i}\}_{i=0}^{2n-1},\ \mathsf{vk}_{1w} := h^a$$
$$\mathsf{ck}_{2v} := \{h^{b^i}\}_{i=0}^{n-1},\ \mathsf{vk}_{2v} := g^b \qquad \mathsf{ck}_{2w} := \{g^{b^i}\}_{i=0}^{2n-1},\ \mathsf{vk}_{2w} := h^b$$

2. Define $\mathsf{ck}_{\mathsf{kzg}} := (\mathsf{ck}_{j\sigma}),\ \mathsf{vk}_{\mathsf{kzg}} := (\mathsf{vk}_{j\sigma})$ for $j = 1, 2;\ \sigma = v, w$.
3. Fix a hash function $\mathsf{Hash}_1 : \mathbb{Z}_p \times \mathbb{G}_T^6 \to \mathbb{Z}_p$ and its description hk_1.
4. Fix a hash function $\mathsf{Hash}_2 : \mathbb{Z}_p \times \mathbb{G}_2^2 \times \mathbb{G}_1^2 \to \mathbb{Z}_p$ and its description hk_2.
5. Set $\mathsf{crs}_{\mathsf{tipp}} := (\mathsf{hk}_1, \mathsf{hk}_2, \mathsf{ck}_d, \mathsf{ck}_{\mathsf{kzg}}, \mathsf{vk}_{\mathsf{kzg}})$.

$\mathsf{TIPP.Prove}(\mathsf{crs}_{\mathsf{tipp}}, (T_{AB}, U_{AB}), Z, r; \mathbf{A}, \mathbf{B}, \mathbf{r}) \to \pi_{\mathsf{tipp}}$**:**

For ease of exposition, set $\mathbf{B'} := \mathbf{B}^{\mathbf{r}}$ and $\mathbf{w}_1' := \mathbf{w}_1^{\mathbf{r}^{-1}}$ and $\mathbf{w_2'} := \mathbf{w}_2^{\mathbf{r}^{-1}}$.

– Loop "split & collapse" for step i

1. $n_i = n_{i-1}/2$ where $n_0 = n$
2. If $n_i == 1$: *break*
3. Compute left and right inner products

$$Z_L = \mathbf{A}_{[n':]} * \mathbf{B}'_{[:n']} \ \text{ and } \ Z_R = \mathbf{A}_{[:n']} * \mathbf{B}'_{[n':]}$$

4. Compute right cross commitments:

$$\begin{aligned} T_L, U_L &= \mathsf{CM}_d((\mathbf{v}_1, \mathbf{w}_1'; \mathbf{v}_2, \mathbf{w}_2'); \mathbf{A}_{[n':]}||\mathbf{0}, \mathbf{0}||\mathbf{B}'_{[:n']})) \\ &= ((\mathbf{A}_{[n':]} * \mathbf{v_1}_{[:n']})(\mathbf{w_1'}_{[n:']} * \mathbf{B}'_{[:n']}), (\mathbf{A}_{[n':]} * \mathbf{v_2}_{[:n']})(\mathbf{w_2'}_{[n':]} * \mathbf{B}'_{[:n']})) \end{aligned}$$

5. Compute left cross commitments:

$$\begin{aligned} T_R, U_R &= \mathsf{CM}_d((\mathbf{v}_1, \mathbf{w}_1'; \mathbf{v}_2, \mathbf{w}_2'); \mathbf{0}||\mathbf{A}_{[:n']}, \mathbf{B}'_{[n':]}||\mathbf{0}) \\ &= ((\mathbf{A}_{[:n']} * \mathbf{v_1}_{[n':]})(\mathbf{w}'_{1_{[:n']}} * \mathbf{B}'_{[n':]}), (\mathbf{A}_{[:n']} * \mathbf{v_2}_{[n':]})(\mathbf{w}'_{2_{[:n']}} * \mathbf{B}'_{[n':]}) \end{aligned}$$

6. Compute challenge $x_i = \mathsf{Hash}_1(x_{i-1}, Z_L, Z_R, T_R, T_L, U_R, U_L)$ (with $x_0 = 0$)
7. Compute Hadamard products on vectors

$$\mathbf{A} = \mathbf{A}_{[:n']} \circ \mathbf{A}_{[n':]}^{x_i} = (A_{0A_{n'}}^{x_i}, \ldots, A_{n'-1}A_{n-1}^{x_i}) \ \text{ and } \ \mathbf{B'} = \mathbf{B}'_{[:n']} \circ \mathbf{B'}_{[n':]}^{x_i^{-1}}$$

8. Compute Hadamard products on keys

$$(\mathbf{v_1}, \mathbf{v_2}) = (\mathbf{v_1}_{[:n']} \circ \mathbf{v_1}_{[n':]}^{x_i^{-1}}, \mathbf{v_2}_{[:n']} \circ \mathbf{v_2}_{[n':]}^{x_i^{-1}}) \tag{8}$$
$$(\mathbf{w}_1', \mathbf{w}_2') = (\mathbf{w}'_{1_{[:n']}} \circ \mathbf{w}'^{x_i}_{1_{[n':]}}, \mathbf{w}'_{2_{[:n']}} \circ \mathbf{w}'^{x_i}_{2'_{[n':]}}) \tag{9}$$

9. Set $n = n'$

– Prove correctness of final commitment keys $(v_1, v_2) \in \mathbb{G}_2^2;\ (w_1, w_2) \in \mathbb{G}_1^2$ after ℓ rounds $(n = 2^\ell)$ using KZG:

1. Define $f_v(X) = \prod_{j=0}^{\ell-1}(1 + x_{\ell-j}^{-1} X^{2^j})$ and $f_w(X) = X^n \prod_{j=0}^{\ell-1}\left(1 + x_{\ell-j} r^{-2^j} X^{2^j}\right)$
2. Draw challenge $z = \mathsf{Hash}_2(x_\ell, v_1, v_2, w_1, w_2)$ where $n = 2^\ell$
3. Prove that $v_1 = g^{f_v(a)},\ v_2 = h^{f_v(a)}$ and $w_1 = g^{f_w(a)},\ w_2 = h^{f_w(b)}$ are KZG commitments of $f_v(X)$ by evaluating in z

$$\pi_{v_j} \leftarrow \mathsf{KZG.Open}(\mathsf{ck}_{jv}; v_j, z, f_v(z); f_v(X)) \text{ for j=1,2}$$
$$\pi_{w_j} \leftarrow \mathsf{KZG.Open}(\mathsf{ck}_{jw}; w_j, z, f_w(z); f_w(X)) \text{ for j=1,2}$$

– Set

$$\pi_{\mathsf{tipp}} = (A, B', \mathbf{Z_L}, \mathbf{Z_R}, \mathbf{T_L}, \mathbf{T_R}, \mathbf{U_L}, \mathbf{U_R}, (v_1, v_2), (w'_1, w'_2), (\pi_{v_j}, \pi_{w_j})_{j=1,2})$$

where A and B' are the final elements from the loop after collapsing $\mathbf{A}$ and $\mathbf{B'}$.

$\mathsf{TIPP.Verify}(\mathsf{crs}_{\mathsf{tipp}}, (T_{AB}, U_{AB}), Z, r; \pi_{\mathsf{tipp}}) \to b$**:**
- Loop iterator $i : 1 \to \ell = \log(n)$:
  1. Reconstruct challenges $\{x_i = H(x_{i-1}, \mathbf{Z_L}[i], \mathbf{Z_R}[i], \mathbf{T_L}[i], \mathbf{T_R}[i], \mathbf{U_L}[i], \mathbf{U_R}[i])\}_{i=1}^{\ell}$ with $x_0 = 0$
  2. Construct final commitment values recursively, $i = 1 \to \ell$:
     - $Z_i = \mathbf{Z_L}[i]^{x_i} \cdot Z_{i-1} \cdot \mathbf{Z_R}[i]^{x_i^{-1}}$
     - $T_i = \mathbf{T_L}[i]^{x_i} \cdot T_{i-1} \cdot \mathbf{T_R}[i]^{x^{-1}}$
     - $U_i = \mathbf{U_L}[i]^{x_i} \cdot U_{i-1} \cdot \mathbf{U_R}[i]^{x^{-1}}$

     where $Z_0 = Z, T_0 = T_{AB}, U_0 = U_{AB}$
- Verify commitments into decisional bit b_0:
  1. $Z_\ell \stackrel{?}{=} e(A, B')$
  2. Check if $e(A, v_1)e(w'_1, B') \stackrel{?}{=} T_\ell$ and $e(A, v_2)e(w'_2, B') \stackrel{?}{=} U_\ell$
- Verify final commitment keys v_j, w'_j, for $j = 1, 2$ via KZG
  1. Reconstruct KZG challenge point: $z = H(x_\ell, v_1, v_2, w'_1, w'_2)$ for $n = 2^\ell$
  2. Reconstruct commitment polynomials:

$$f_v(X) = \prod_{j=0}^{\ell-1}\left(1 + x_{\ell-j}^{-1} X^{2^j}\right) \tag{10}$$
$$f_w(X) = X^n \prod_{j=0}^{\ell-1}\left(1 + x_{\ell-j} r^{-2^j} X^{2^j}\right) \tag{11}$$

  3. Run verification for openings of evaluations in z for $j = 1, 2$:

$$b_{1j} \leftarrow \mathsf{KZG.Check}(\mathsf{vk}_{jv}; v_j, z, f_v(z); \pi_{v_j})$$
$$b_{2j} \leftarrow \mathsf{KZG.Check}(\mathsf{vk}_{jw}; w_j, z, f_w(z); \pi_{w_j})$$

- Set $b = b_0 \wedge b_{11} \wedge b_{12} \wedge b_{21} \wedge b_{22}$

The security result for the TIPP protocol is following the same proving strategy as the one in [BMM+19]:

Theorem 4. *If* CM_d *is a binding inner product commitment,* $\mathsf{KZG.PC}$ *is a polynomial commitment with Computational Knowledge Binding as per Definition 6, then the protocol TIPP has computational knowledge soundness (Definition 1).*

Remark that both CM_d and $\mathsf{KZG.PC}$ schemes are secure in the Generic Group Model (or under specific assumptions such as q-ASDGP for the CM_d commitment scheme and q-SDH for the validity of the final commitment keys done using $\mathsf{KZG.PC}$ scheme).

Proof. The proof follows the same ideas as [BMM+19] proof for their TIPP scheme.

5.5 Formula for Final Commitment Keys

There is one step in showing that such protocol satisfies computational knowledge soundness that is slightly different in our case: Defining the correct polynomials to be committed under $\mathsf{kzg.PC}$ scheme in order to show that the structure of the honestly generated final commitment keys is correct.

Recall that the two schemes MIPP and TIPP achieve log-time verification using a specially structured commitment scheme that allows the prover to use one new challenge x_j in each round of recursion to transform the commitments homomorphically. Because of this, the verifier must also perform a linear amount of work in rescaling the commitment keys ($\mathsf{ck}_s/\mathsf{ck}_d$ for MIPP/TIPP). To avoid having the verifier rescale the commitment keys, our schemes apply the same trick as [BMM+19]: we do this by outsourcing the work of rescaling the commitment keys to the prover.

Then what is left is to convince a verifier that this rescaling was done correctly just by checking the final commitment keys and a succinct proof (a KZG polynomial opening). This is verified via a log-time evaluation of the polynomial and two/four (for MIPP/TIPP) pairings.

Recall the structure of the 4 vectors $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{G}_2$ and $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{G}_1$ used for the commitment keys $\mathsf{ck}_s, \mathsf{ck}_d$:

$$\mathbf{v}_1 = (h, h^a, \dots, h^{a^{n-1}}), \qquad \mathbf{w}_1 = (g^{a^n}, \dots, g^{a^{2n-1}}), \qquad \mathbf{w_1'} := \mathbf{w}_1^{\mathbf{r}^{-1}}$$
$$\mathbf{v}_2 = (h, h^b, \dots, h^{b^{n-1}}), \qquad \mathbf{w}_2 = (g^{b^n}, \dots, g^{b^{2n-1}}), \qquad \mathbf{w_2'} := \mathbf{w}_2^{\mathbf{r}^{-1}}$$

We will show the formulae for the final commitment keys v_1, v_2, w_1', w_2' (the result of many rounds of rescaling $\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}_1', \mathbf{w}_2'$ until the end of the loop) used in our schemes MIPP and TIPP are correct. (The way we define the two polynomials $f_v(X)$ for v_1, v_2 and $f_w(X)$ for w_1', w_2'.)

For ease of presentation, we state and prove the formula for a generic vector $\mathbf{v} = (v_1, v_2, \dots, v_{2^\ell}) = (g, g^\alpha, g^{\alpha^2}, \dots g^{\alpha^{2^\ell - 1}})$ of length $n = 2^\ell$ to which we apply the same rescaling as for the commitment keys $\mathsf{ck}_s, \mathsf{ck}_d$. The specific formulae for $\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}_1', \mathbf{w}_2'$ are easy to deduce once we have a formula for $\mathbf{v}$.

Consider a challenge x_j for round j, where the total number of rounds is ℓ and $x_0 = 0$.

Note that at each round j we split the sequence $v_1, v_2, \dots, v_n$ in half and we use x_j to rescale first half and the second half of the vector recursively until we end up with a single value v.

We claim that the formula for some initial key $\mathbf{v} = (v_1 = g, v_2 = g^\alpha, \dots, v_n = g^{\alpha^{n-1}})$ is:

$$v = g^{\prod_{j=0}^{\ell-1}(1 + x_{\ell-j}\alpha^{2^j})}$$

for a vector of challenges $x_0 \coloneqq 0, x_1 \dots x_{\ell-1}, x_\ell$. We will prove the general formula by induction:

First step, check the formula for $\ell = 1$ (initial commitment key $\mathbf{v}$ has two elements v_1, v_2):

$$v = v_1 v_2^{x_1} = g^{1+x_1\alpha} = g^{\prod_{j=0}^{0}(1+x_{\ell-j}\alpha^{2^j})}.$$

Secondly, suppose the statement is true for $\ell - 1$. We prove it for ℓ.

On the first round, we have a challenge x_1 and we rescale the commitment key $\mathbf{v}$ which has length $n = 2^\ell$ as follows:
$\mathbf{v}' = \mathbf{v}_{[:2^{\ell-1}]} \circ \mathbf{v}_{[2^{\ell-1}:]}^{x_1}$,
$\mathbf{v}' = (g \cdot g^{x_1\alpha^{2^{\ell-1}}}, g^\alpha \cdot g^{x_1\alpha^{2^{\ell-1}+1}}, g^{\alpha^2} \cdot g^{x_1\alpha^{2^{\ell-1}+2}}, \dots)$.
We can write this differently as: $\mathbf{v}' = (v_1 v_1^{x_1\alpha^{2^{\ell-1}}}, \dots v_{2^{\ell-1}} v_{2^{\ell-1}}^{x_1\alpha^{2^{\ell-1}}})$.
This gives us a nicely written commitment key after first round

$$\mathbf{v}' = (v_1^{1+x_1\alpha^{2^{\ell-1}}}, v_2^{1+x_1\alpha^{2^{\ell-1}}}, \dots v_{2^{\ell-1}}^{1+x_1\alpha^{2^{\ell-1}}}) = \mathbf{v}_{[:2^{\ell-1}]}^{1+x_1\alpha^{2^{\ell-1}}}.$$

We can apply the induction assumption for step $\ell - 1$ to $\mathbf{v}_{[:2^{\ell-1}]}$ which is a commitment key of length $2^{\ell-1}$. This means the final key for $\mathbf{v}$ is:

$$v = \left(g^{\prod_{j=0}^{\ell-2}\left(1+x_{\ell-j}\alpha^{2^j}\right)}\right)^{(1+x_1\alpha^{2^{\ell-1}})} = g^{\prod_{j=0}^{\ell-1}(1+x_{\ell-j}\alpha^{2^j})}.$$

Remark than in more generality, this can be written as:

$$v = v_1^{\prod_{j=0}^{\ell-1}(1+x_{\ell-j}\alpha^{2^j})}.$$

Therefore, if we start with an initial key $\mathbf{w} = (w_1 = g^{\alpha^n}, w_2^{\alpha n+1} \dots, v_n = g^{\alpha^{2n-1}})$, the final key w can be written as:

$$w = w_1^{\prod_{j=0}^{\ell-1}(1+x_{\ell-j}\alpha^{2^j})} = g^{\alpha^n \prod_{j=0}^{\ell-1}(1+x_{\ell-j}\alpha^{2^j})}.$$

5.6 Merged Protocol MT-IPP for Optimized Aggregation.

In order to optimize the aggregation contruction based on MIPP and TIPP schemes presented in Section 4, we will "fuse" together the two schemes MIPP and TIPP. More precisely MIPP and TIPP are at the origin interactive protocols, that are turned into non-interactive arguments using Fiat-Shamir transformation. This means that at each round the challenges are generated by a hash function that is modeled by a random oracle (see Item 6 in MIPP, Item 6 in TIPP). We will define the fusion protocol MT-IPP that simultaneously generate the challenges for both MIPP and TIPP, by running a common hash function on both MIPP and TIPP inputs for each round.

This new protocol will be used to replace the steps Item 8, Item 7 in the proving algorithm Section 4 of the Groth16 Aggregation argument and Item 6, Item 5 in the verification algorithm Section 4 by a unique prove and verification for the fusioned MT-IPP scheme.

MT-IPP Scheme.

Relation. First we define the relation proven using the merged MT-IPP argument:

$$\mathcal{R}_{\mathsf{mt}} := \left\{ \begin{array}{c} ((T_{AB}, U_{AB}), (T_C, U_C), \\ \quad Z_{AB}, Z_C, r; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{r}) \end{array} : \begin{array}{c} (\mathsf{CM}_d(\mathbf{A}, \mathbf{B}), Z_{AB}, r; \mathbf{A}, \mathbf{B}, \mathbf{r}) \in \mathcal{R}_{\mathsf{mipp}} \\ \wedge \\ (\mathsf{CM}_s(\mathbf{C}), Z_C, r; \mathbf{C}, \mathbf{r}) \in \mathcal{R}_{\mathsf{tipp}} \end{array} \right\}$$

for vectors $\mathbf{A}, \mathbf{C} \in \mathbb{G}_1$ and $\mathbf{B} \in \mathbb{G}_2$.

Construction. MT-IPP argument works similarly to TIPP and MIPP arguments, by merging together the operations related to the vectors $\mathbf{A}, \mathbf{C}$ by using the same commitment keys and challenges for them. It consists of 3 algorithms $\mathsf{MT\text{-}IPP} = (\mathsf{MT.Setup}, \mathsf{MT.Prove}, \mathsf{MT.Verify})$ described in the following, where we highlighted in grey the main changes needed to merge MIPP and TIPP.

$\mathsf{MT.Setup}(1^\lambda, \mathcal{R}_{\mathsf{tipp}}) \rightarrow \mathsf{crs}_{\mathsf{mt}}$**:** 1. Define $\mathsf{ck}_{\mathsf{kzg}}$, $\mathsf{vk}_{\mathsf{kzg}}$ as in Item 2 from TIPP.

2. Fix a hash function $\mathsf{Hash} : \mathbb{Z}_p \times \mathbb{G}_T^{12} \rightarrow \mathbb{Z}_p$ and its description hk.
3. Fix a hash function $\mathsf{Hash}_2 : \mathbb{Z}_p \times \mathbb{G}_2^2 \times \mathbb{G}_1^2 \rightarrow \mathbb{Z}_p$ and its description hk_2.
4. Set $\mathsf{crs}_{\mathsf{mt}} := (\mathsf{hk}, \mathsf{hk}_2, \mathsf{ck}_d, \mathsf{ck}_{\mathsf{kzg}}, \mathsf{vk}_{\mathsf{kzg}})$.

$\mathsf{MT.Prove}(\mathsf{crs}_{\mathsf{mt}}, (T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, Z_C, r; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{r}) \rightarrow \pi_{\mathsf{mt}}$**:**

Loop "split & collapse" for step i

1. $n_i = n_{i-1}/2$ where $n_0 = n$
2. If $n_i \overset{?}{=} 1$: *break*
3. Compute L/R inner products as for MIPP & TIPP: $(Z_L, Z_R)_{AB}, (Z_L, Z_R)_C$
4. Compute L/R cross commitments: $(T_L, U_L; T_R, U_R)_{AB}$, $(T_L, U_L; T_R, U_R)_C$
5. Compute challenge
$$x_i = \mathsf{Hash}\,(x_{i-1}; (Z_L, Z_R)_{AB}, (Z_L, Z_R)_C, (T_L, U_L; T_R, U_R)_{AB}, (T_L, U_L; T_R, U_R)_C)$$
6. Compute Hadamard products on vectors
$$\mathbf{A} = \mathbf{A}_{[:n']} \circ \mathbf{A}_{[n':]}^{x_i}, \ \mathbf{B'} = \mathbf{B'}_{[:n']} \circ \mathbf{B'}_{[n':]}^{x_i^{-1}} \ \text{ and } \ \mathbf{C} = \mathbf{C}_{[:n']} \circ \mathbf{C}_{[n':]}^{x_i}$$
7. Compute Hadamard products on keys $\mathbf{v}_1, \mathbf{v}_2$ and $\mathbf{w}'_1 := \mathbf{w}_1^{\mathbf{r}^{-1}}, \mathbf{w}'_2 := \mathbf{w}_2^{\mathbf{r}^{-1}}$
$$\begin{aligned} (\mathbf{v_1}, \mathbf{v_2}) &= (\mathbf{v_1}_{[:n']} \circ \mathbf{v_1}_{[n':]}^{x^{-1}}, \mathbf{v_2}_{[:n']} \circ \mathbf{v_2}_{[n':]}^{x^{-1}}) \\ (\mathbf{w}'_1, \mathbf{w}'_2) &= (\mathbf{w}'_{1_{[:n']}} \circ \mathbf{w}'^{x}_{1_{[n':]}}, \mathbf{w}'_{2_{[:n']}} \circ \mathbf{w}'^{x}_{2_{[n':]}}) \end{aligned} \tag{12}$$
8. Set $n = n'$

Compute proofs $(\pi_{v_j}, \pi_{w_j})_{j=1,2}$ of correctness of final commitment keys exactly as in Section 5.4 in TIPP.
Set

$$\begin{aligned} \pi_{\mathsf{mt}} = \big(&A, B, C, (\mathbf{Z_L}, \mathbf{Z_R})_{AB}, (\mathbf{Z_L}, \mathbf{Z_R})_C, (\mathbf{T_L}, \mathbf{U_L})_{AB}, (\mathbf{T_R}, \mathbf{U_R})_{AB}, \\ &(\mathbf{T_L}, \mathbf{U_L})_C, (\mathbf{T_R}, \mathbf{U_R})_C, (v_1, v_2), (w'_1, w'_2), (\pi_{v_j}, \pi_{w_j})_{j=1,2} \big) \end{aligned}$$

where A, B', C and $(v_1, v_2), (w'_1, w'_2)$ are the final elements from the loop after collapsing $\mathbf{A}, \mathbf{B}' = \mathbf{B}^{\mathbf{r}}, \mathbf{C}$ and $\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}'_1, \mathbf{w}'_2$.

$\mathsf{MT.Verify}(\mathsf{crs}_{\mathsf{mt}}, \mathsf{statement}; \pi_{\mathsf{mt}}) \rightarrow b$**:**

1. Parse $\mathsf{statement} = ((T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, Z_C, r)$
2. Reconstruct challenges x_i for $i = 1, \ldots \log(n)$ common to MIPP and TIPP:

$$\big\{x_i = \mathsf{Hash}\big(x_{i-1}, (\mathbf{Z_L}[i], \mathbf{Z_R}[i])_{AB}, (\mathbf{Z_L}[i], \mathbf{Z_R}[i])_C, (\mathbf{T_L}[i], \mathbf{T_R}[i])_{AB}, (\mathbf{U_L}[i], \mathbf{U_R}[i])_{AB}, (\mathbf{T_L}[i], \mathbf{T_R}[i])_C, (\mathbf{U_L}[i], \mathbf{U_R}[i])_C\big)\big\}_{i=1}^{\log(n)}$$

3. Construct final commitments recursively as in TIPP for $\mathbf{A}, \mathbf{B}$ and as in MIPP for $\mathbf{C}$:

$$(Z_{AB}, T_{AB}, U_{AB}), (Z_C, T_C, U_C)$$

4. Verify final commitment keys v_1, v_2, w'_1, w'_2 via KZG as for TIPP.
   Final keys v_1, v_2 are checked once and they are common to MIPP and TIPP.

The security of the MT-IPP protocol follows from the security of MIPP and TIPP assuming the random oracle model and the commitment algebraic model (see [BMM+19] for details). This is a standard AND-composition technique for proofs of two relations (in our case $\mathcal{R}_{\mathsf{tipp}} \wedge \mathcal{R}_{\mathsf{mipp}}$).

6 SnarkPack: Practical Aggregation Argument

The resulting argument for aggregation using the MT is described by 3 algorithms $\mathsf{SnarkPack} = (\mathsf{SP.Setup}, \mathsf{SP.Prove}, \mathsf{SP.Verify})$ as follows:

$\mathsf{SP.Setup}(1^\lambda, \mathcal{R}_{\mathsf{AGG}}) \rightarrow (\mathsf{pk}_{\mathsf{agg}}, \mathsf{vk}_{\mathsf{agg}})$

1. Generate commitment key for $\mathsf{CM}_d : \mathsf{ck}_d = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}_1, \mathbf{w}_2) \leftarrow \mathsf{CM}_d.\mathsf{KG}(1^\lambda)$
2. Set commitment key for $\mathsf{CM}_s : \mathsf{ck}_s = (\mathbf{v}_1, \mathbf{v}_2)$
3. Call $\mathsf{crs}_{\mathsf{mt}} \leftarrow \mathsf{MT.Setup}(1^\lambda, \mathcal{R}_{\mathsf{tipp}})$
4. Choose a hash function $\mathsf{Hash}_0 : \mathbb{G}_T^4 \rightarrow \mathbb{Z}_p$ given by its description hk_0.
5. Set aggregation keys $\mathsf{pk}_{\mathsf{agg}} = (\mathsf{vk}, \mathsf{crs}_{\mathsf{mt}}, \mathsf{ck}_s, \mathsf{ck}_d, \mathsf{hk}_0), \mathsf{vk}_{\mathsf{agg}} = (\mathsf{vk}, \mathsf{crs}_{\mathsf{mt}}, \mathsf{hk}_0)$

$\mathsf{SP.Prove}(\mathsf{pk}_{\mathsf{agg}}, \mathbf{u}, \pi = (\mathbf{A}, \mathbf{B}, \mathbf{C})) \rightarrow \pi_{\mathsf{agg}}$

1. Parse proving key $\mathsf{pk}_{\mathsf{agg}} := (\mathsf{vk}, \mathsf{crs}_{\mathsf{mt}}, \mathsf{ck}_s, \mathsf{ck}_d, \mathsf{hk})$
2. Parse $\mathsf{ck}_s = (\mathbf{v}_1, \mathbf{v}_2),\ \mathsf{ck}_d = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}_1, \mathbf{w}_2)$
3. Commit to $\mathbf{A}$ and $\mathbf{B}$: $\mathsf{CM}_d((\mathbf{v_1}, \mathbf{v_2}, \mathbf{w_1}, \mathbf{w_2}); \mathbf{A}, \mathbf{B}) = (T_{AB}, U_{AB})$
4. Commit to $\mathbf{C}$: $\mathsf{CM}_s((\mathbf{v_1}, \mathbf{v_2}); \mathbf{C}) = (T_C, U_C)$
5. Derive random challenge $r = \mathsf{Hash}_0(T_{AB}, U_{AB}, T_C, U_C) \in \mathbb{Z}_p$ and set $\mathbf{r} = \{r^i\}_{i=0}^{n-1}$
6. Compute $Z_{AB} = \mathbf{A}^{\mathbf{r}} * \mathbf{B}$
7. Compute $Z_C = \mathbf{C}^{\mathbf{r}} = \prod_{i=0}^{n-1} C_i^{r_i}$.
8. Run MT proof:

$$\pi_{\mathsf{mt}} = \mathsf{MT.Prove}(\mathsf{crs}_{\mathsf{mt}}, (T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, r; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{r}) \quad (13)$$

9. Set $\pi_{\mathsf{agg}} = ((T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, Z_C, r, \pi_{\mathsf{mt}})$

$\mathsf{SP.Verify}(\mathsf{vk}_{\mathsf{agg}}, \mathbf{u}, \pi_{\mathsf{agg}}) \to b$

1. Parse SNARK statements $\mathbf{u} = \{a_{i,j}\}_{i=1,\dots n; j=0,\dots \ell}$
2. Parse verification key $\mathsf{vk}_{\mathsf{agg}} := (\mathsf{vk}, \mathsf{crs}_{\mathsf{mt}}, \mathsf{hk})$
3. Parse $\mathsf{vk} := \left(P = g^{\alpha}, Q = h^{\beta},\ \{S_j\}_{j=0}^{\ell}, H = h^{\gamma}, D = h^{\delta}\right)$
4. Compute $Z_{S_j} = S_j^{\sum_{i=1}^{n} a_{ij} r_i}$ for all $j = 0 \dots \ell$
5. Derive random challenge $r = \mathsf{Hash}_0(T_{AB}, U_{AB}, T_C, U_C)$
6. Check MT proof $b_1 \leftarrow \mathsf{MT.Verify}(\mathsf{crs}_{\mathsf{mt}}, \mathsf{statement}, \pi_{\mathsf{mt}})$
7. Check Groth16 aggregated equations to the decision bit b_2:

$$Z_{AB} \stackrel{?}{=} e(P^{\sum_{i=1}^{n} r^i}, Q) e(\prod_{j=0}^{\ell} Z_{S_j}, H) e(Z_C, D)$$

8. Set decision bit $b = b_1 \wedge b_2$

7 Implementation

7.1 Setup

We have implemented the scheme in Rust, using the paired [Fil18b] library on the BLS12-381 curve. The code can be found on the feat-ipp2 branch [Fil21] of the bellperson repository [Fil18a]. We have taken the original code of the arkwork library [ark19] and modified it both for fitting the scheme presented in this paper and for performance. All proofs are Groth16 proofs with 350 public inputs, which is similar to the proofs posted by Filecoin miners. All benchmarks are done on a 32 cores / 64 threads machine with AMD Raizen Threadripper CPUs.

Parallelism: It is important to note that the protocol allows for some parallel operations and our implementation makes use of that. Therefore, all benchmarks presented here can change depending on the degree of parallelism of the machine.

7.2 Trusted Setup

We created a condensed version of the SRS required for our protocol from the powers of tau transcript of both Zcash [zca18] and Filecoin [Lab18]. The code to assemble the SRS from two powers of tau can be found at [nik21]. The srs created allows to aggregate up to 2^{19} proofs.

7.3 Optimizations

Merging TIPP and MIPP: We have implemented the optimized version of the scheme SnarkPack that enabled us to achieve a 20-30% improvement in verification time as well as a slighter reduction in proof size. The optimization leads to twice less calls to the random oracle and it saves one KZG proof to verify, more precisely 4 pairings and a logarithmic number of group multiplications.

Field elements compression: The proof requires many pairing operations and multiplications in the target group which employ arithmetic over the finite field $\mathbb{F}_{p^{12}}$. We implemented compression of these field elements that still allow some computations without decompression using algorithms derived from Diego F. Aranha's RELIC library [AGM$^+$]. You can find the specific implementation in this branch [dig21]. This led to a 40% reduction in proof size.

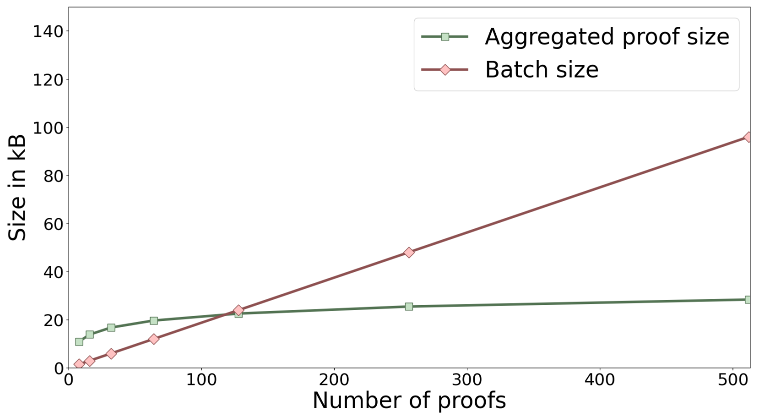

Fig. 2. Proof size: Aggregation vs Batching.

Compressing pairing checks: We randomize each pairing checks of the form

$$e(A, B)e(C, D)... = T$$

with a random exponent when verifying so we can compress multiple such checks into one. This randomized checking technique is borrowed from the Zcash specs [HBHW21]. Specifically, we have a list P of length n of pairing checks of the form $e(A, B)e(C, D)... = T$. The verifier performs the following step to verify all checks in a compressed manner:

1. Choose n randoms scalars r_i with $r_0 = 1$
2. Randomize each pairing check P_i for $i > 1$: $e(r_iA_i, B_i)e(r_iC_i, D_i) \cdots = T_i^r$
3. Compute the miller loop on the left side of each pairing check: $m_i = Miller((r_iA_i, B_i), (r_iC_i, D_i), \dots)$
4. Multiply all results together and apply the final exponentiation (FE) at the end:

$$FE(\prod_i m_i) == \prod_i T_i^{r_i}$$

The final verification equation looks like this:

$$FE(\prod_i Miller((r_iA_i, B_i), (rC_i, D_i) \dots)) == \prod_i T_i^{r_i}$$

Note that doing the random linear combination using the $\mathbb{G}_1$ component of the check is much faster than simply doing the exponentiation on the result (i.e. $e(A_i, B_i)_i^r$) as the exponentiation is then in $\mathbb{G}_\mathbb{T}$.

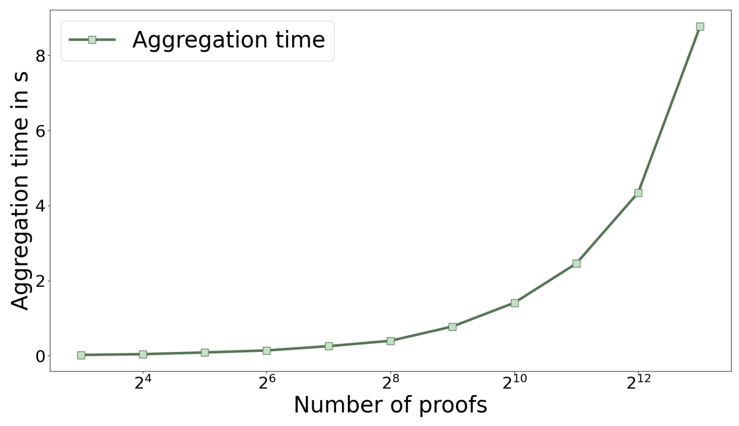

Fig. 3. Aggregation Time

7.4 Proof size

The proof size in Fig. 2 compares the size of n proofs versus the size of one aggregated proof. The figure shows the break even point around 150 proofs where aggregation takes less space than batching. At 128 proofs, the size of aggregated proof is of 23kB versus 24kB for individual proofs.

7.5 Aggregation time

Figure 3 shows the time taken by the aggregator to create an aggregated proof. We can see for example that it can aggregate 1024 proofs in 1.4s. The prover is required to compute a logarithmic number of multi-exponentiations and expensive pairing products. Our implementation perform these in parallel and in batches (batching miller loop operations).

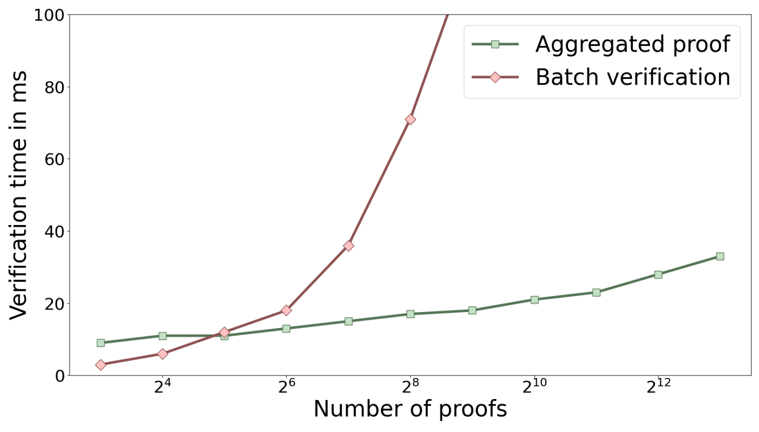

Fig. 4. Verifcation time: Aggregation vs Batching.

7.6 Verification time

The major point of interest in our application to Filecoin is the verification time of Groth16 proofs. Figure 4 shows the comparison between the verification of an aggregated proof and using batching techniques as described in the zcash protocol [HBHW21]. Verifying Groth16 proofs in batches is what is commonly used in zcash as well as Filecoin to get a sublinear verification time. The graph shows that batching is more efficient when verifying less 32 Groth16 proofs but aggregation becomes exponentially faster after that point. Our protocol can verify a 8192 proof in 33ms, including unserialization and it scales logarithmically. Note the verification algorithm is *linear* in terms of the public inputs. In our case, 350 public inputs is small enough to barely count for the total verification time.

Acknowledgements

We would like to thank Benedikt Bunz, Pratyush Mishra, and Psi Vesely for valuable discussions on this work, as well as Ben Fisch and Nicola Greco for the initial intuition of using IPP for aggregating Filecoin SNARK-based proofs. We are also grateful to dignifiedquire for his contributions to the Rust codebase.

References

AGM^{+}. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

ark19. arkwork. Rust inner pairing product, 2019. `https://github.com/arkworks-rs/ripp`.

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

BCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. pages 781–796, 2014.

BMM^{+}19. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. Cryptology ePrint Archive, Report 2019/1177, 2019. `https://eprint.iacr.org/2019/1177`.

Dam00. Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, Heidelberg, May 2000.

dig21. dignifiedquire. Compression on fp12 elements, 2021. `https://github.com/filecoin-project/blstrs/compare/feat-compression`.

Fil18a. Filecoin. bellperson, groth16 library, 2018. `https://github.com/filecoin-project/bellperson`.

Fil18b. Filecoin. paired: high performance bls12-381 library, 2018. `https://github.com/filecoin-project/paired`.

Fil20. Filecoin. Filecoin powers of tau ceremony attestations, 2020. `https://github.com/arielgabizon/perpetualpowersoftau`.

Fil21. Filecoin. Groth16 aggregation library, 2021. `https://github.com/filecoin-project/bellperson/tree/feat-ipp2`.

Fis19. Ben Fisch. Tight proofs of space and replication, 2019. `https://web.stanford.edu/~bfisch/tight_pos.pdf`.

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

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

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

HBHW21. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification, 2021. `https://zips.z.cash/protocol/protocol.pdf`.

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

Lab18. Protocol Labs. Filecoin, 2018. `https://filecoin.io/filecoin.pdf`.

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

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

LMR19. Russell W. F. Lai, Giulio Malavolta, and Viktoria Ronge. Succinct arguments for bilinear group arithmetic: Practical structure-preserving cryptography. In *ACM CCS 19*, pages 2057–2074. ACM Press, 2019.

Mau05. Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005.

nik21. nikkolasg. Tau aggregation for ipp, 2021. `https://github.com/nikkolasg/taupipp`.

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

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

zca18. zcash. Zcash powers of taus ceremony attestation, 2018. `https://github.com/ZcashFoundation/powersoftau-attestations`.

A Assumptions in Generic Group Model

A.1 ASSGP Assumption in GGM

Assumption 5 (ASSGP) *The q-ASSGP assumption holds for the bilinear group generator $\mathcal{G}$ if for all* PPT *adversaries $\mathcal{A}$ we have, on the probability space* $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T) \leftarrow \mathcal{G}(1^\lambda)$, $g \leftarrow\!\$\, \mathbb{G}_1, h \leftarrow\!\$\, \mathbb{G}_2$ *and* $a, b \leftarrow\!\$\, \mathbb{Z}_p$ *the following holds:*

$$\Pr\left[\begin{array}{c|c} (A_0,\ldots,A_{q-1}) \neq \mathbf{1}_{\mathbb{G}_1} & g \leftarrow\!\$\, \mathbb{G}_1, h \leftarrow\!\$\, \mathbb{G}_2, a, b \leftarrow\!\$\, \mathbb{Z}_p \\ \wedge \prod_{i=0}^{q-1} e(A_i, h^{a^i}) = 1_{\mathbb{G}_T} & \sigma \leftarrow ([g^{b^i}]_{i=0}^{2q-1}, [g^{a^i}]_{i=0}^{2q-1}, [h^{b^i}]_{i=0}^{2q-1}, [h^{a^i}]_{i=0}^{2q-1}) \\ \wedge \prod_{i=0}^{q-1} e(A_i, h^{b^i}) = 1_{\mathbb{G}_T} & (A_1,\ldots,A_q) \leftarrow \mathcal{A}(\mathsf{gk}, \sigma) \end{array}\right] = \mathsf{negl}(\lambda)$$

Lemma 5. *The q-ASSGP assumption holds in the generic group model.*

Proof. Suppose $\mathcal{A}$ is an adversary that on input (gk, σ), outputs $(A_0,\ldots,A_{q-1}) \in \mathbb{G}_1^n$ such that $\prod_{i=0}^{q-1} e(A_i, h^{a^i}) = 1_{\mathbb{G}_T}$ and $\prod_{i=0}^{q-1} e(A_i, h^{b^i}) = 1_{\mathbb{G}_T}$. Then its GGM extractor outputs $\alpha_i(X,Y) = \sum_{j=0}^{2q-1}(x_j X^j + y_j Y^j + c_j)$ then we have:

$$\alpha_0(X,Y) + X\alpha_1(X,Y) + X^2\alpha_2(X,Y) + \cdots + X^{q-1}\alpha_{q-1}(X,Y) = 0 \quad (14)$$
$$\alpha_0(X,Y) + Y\alpha_1(X,Y) + Y^2\alpha_2(X,Y) + \cdots + Y^{q-1}\alpha_{q-1}(X,Y) = 0 \quad (15)$$

Then we have:

$$\alpha_0(X,Y) = -X\alpha_1(X,Y) - X^2\alpha_2(X,Y) - \cdots - X^{q-1}\alpha_{q-1}(X,Y) \quad (16)$$
$$\alpha_0(X,Y) = -Y\alpha_1(X,Y) - Y^2\alpha_2(X,Y) - \cdots - Y^{q-1}\alpha_{q-1}(X,Y) \quad (17)$$

If we substract (17) and (16) we got

$$0 =(X-Y)\alpha_1(X,Y) + \cdots + (X^{q-1} - Y^{q-1})\alpha_{q-1}(X,Y) \quad (18)$$
$$-(X-Y)\alpha_1(X,Y) =(X^2 - Y^2)\alpha_2(X,Y) + \cdots + (X^{q-1} - Y^{q-1})\alpha_{q-1}(X,Y) \quad (19)$$

Now we can divide by $(X - Y)$ and obtain:

$$\begin{aligned} -\alpha_1(X,Y) =&(X+Y)\alpha_2(X,Y) + (X^2 + XY + Y^2)\alpha_3(X,Y) + \cdots + \\ &+ (X^{q-2} + YX^{q-3} + \cdots + Y^{q-3}X + Y^{q-2})\alpha_{q-1}(X,Y) \end{aligned} \quad (20)$$

Substitute the expression of $-\alpha_1(X,Y)$ in equation (16) and remark that all $X^i\alpha_i(X,Y)$ terms are vanishing:

$$\begin{aligned} \alpha_0(X,Y) &= X[(X+Y)\alpha_2(X,Y) + (X^2 + XY + Y^2)\alpha_3(X,Y) + \cdots + (X^{q-2} + X^{q-3}Y + \cdots + \\ &\quad + XY^{q-3} + Y^{q-2})\alpha_{q-1}(X,Y)] - X^2\alpha_2(X,Y) - \cdots - X^{q-1}\alpha_{q-1}(X,Y) \\ \alpha_0(X,Y) &= XY\alpha_2(X,Y) + (X^2Y + XY^2)\alpha_3(X,Y) + \cdots + (X^{q-2}Y + \cdots + XY^{q-2})\alpha_{q-1}(X,Y) \\ \alpha_0(X,Y) &= XY[\alpha_2(X,Y) + (X+Y)\alpha_3(X,Y) + \cdots + (X^{q-3} + X^{q-4}Y + \cdots + Y^{q-3})\alpha_{q-1}(X,Y)] \end{aligned} \quad (21)$$

This implies that either $\alpha_0(X,Y)$ is a multiple of XY or $\alpha_0(X,Y)=0$.

By the GGM assumption, we have that $\alpha_0(X,Y)=0$.

We continue by replacing $\alpha_0(X,Y)=0$ in equation (21):

$$0=\alpha_2(X,Y)+(X+Y)\alpha_3(X,Y)+\cdots+(X^{q-3}+X^{q-4}Y+\cdots+Y^{q-3})\alpha_{q-1}(X,Y) \tag{22}$$

$$-\alpha_2(X,Y)=(X+Y)\alpha_3(X,Y)+\cdots+(X^{q-3}+X^{q-4}Y+\cdots+Y^{q-3})\alpha_{q-1}(X,Y) \tag{23}$$

Substitute the expression of $-\alpha_2(X,Y)$ in equation (17) and remark that all $Y^i\alpha_i(X,Y)$ terms are vanishing:

$$\begin{aligned}0=&-Y\alpha_1(X,Y)-Y^2[(X+Y)\alpha_3(X,Y)+\cdots+(X^{q-3}+X^{q-4}Y+\cdots+Y^{q-3})\alpha_{q-1}(X,Y)]-\\&-Y^3\alpha_3(X,Y)-\cdots-Y^{q-1}\alpha_{q-1}(X,Y)\\Y\alpha_1(X,Y)=&Y^2X\alpha_3(X,Y)+\cdots+(X^{q-3}Y^2+X^{q-4}Y^3+\cdots+XY^{q-2})\alpha_{q-1}(X,Y)]\\Y\alpha_1(X,Y)=&Y^2X[\alpha_3(X,Y)+\cdots+(X^{q-4}+X^{q-5}Y+\cdots+Y^{q-4})\alpha_{q-1}(X,Y)]\end{aligned} \tag{24}$$

This implies that either $\alpha_1(X,Y)$ is a multiple of XY or $\alpha_1(X,Y)=0$.

By the GGM assumption, we have that $\alpha_1(X,Y)=0$.

We continue by replacing $\alpha_1(X,Y)=0$ in equation (24):

$$0=\alpha_3(X,Y)+\cdots+(X^{q-4}+X^{q-5}Y+\cdots+Y^{q-4})\alpha_{q-1}(X,Y)$$

$$-\alpha_3(X,Y)=(X^2+XY+Y^2)\alpha_4(X,Y)+\cdots+(X^{q-4}+X^{q-5}Y+\cdots+Y^{q-4})\alpha_{q-1}(X,Y) \tag{25}$$

And so on... till we show that $\alpha_i(X,Y)=0\ \ \forall i=0\ldots q-1$.

A.2 ASDGP Assumption in GGM

Assumption 6 (ASDGP) *The q-ASDGP assumption holds for the bilinear group generator $\mathcal{G}$ if for all* PPT *adversaries $\mathcal{A}$ we have, on the probability space* $\mathsf{gk}=(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T)\leftarrow\mathcal{G}(1^\lambda)$, $g\leftarrow\$\,\mathbb{G}_1, h\leftarrow\$\,\mathbb{G}_2$ *and* $a,b\leftarrow\$\,\mathbb{Z}_p$ *the following holds:*

$$\Pr\left[\begin{array}{c|c}(\mathbf{A}\neq\mathbf{1}_{\mathbb{G}_1}\vee\ \mathbf{B}\neq\mathbf{1}_{\mathbb{G}_2}) & g\leftarrow\$\,\mathbb{G}_1,h\leftarrow\$\,\mathbb{G}_2,a,b\leftarrow\$\,\mathbb{Z}_p\\ \wedge\ \prod_{i=0}^{q-1}e(A_i,h^{a^i})\prod_{i=q}^{2q-1}e(g^{a^i},B_i)=1_{\mathbb{G}_T} & \sigma\leftarrow([g^{a^i},g^{b^i},h^{a^i},h^{b^i}]_{i=0}^{2q-1})\\ \wedge\ \prod_{i=0}^{q-1}e(A_i,h^{b^i})\prod_{i=q}^{2q-1}e(g^{b^i},B_i)=1_{\mathbb{G}_T} & (\mathbf{A},\mathbf{B})\leftarrow\mathcal{A}(\mathsf{gk},\sigma)\end{array}\right]=\mathsf{negl}(\lambda)$$

Lemma 6. *The q-ASDGP assumption holds in the generic group model.*

Proof. Suppose $\mathcal{A}$ is an adversary that on input (gk,σ), outputs $(A_0,\ldots,A_{q-1}),(B_0,\ldots,B_{q-1})$ such that $\prod_{i=0}^{q-1}e(A_i,h^{a^i})\prod_{i=q}^{2q-1}e(g^{a^i},B_i)=1_{\mathbb{G}_T}$ and $\prod_{i=0}^{q-1}e(A_i,h^{b^i})\prod_{i=q}^{2q-1}e(g^{b^i},B_i)=1_{\mathbb{G}_T}$. Then its GGM extractor outputs $\alpha_i(X,Y)=\sum_{j=0}^{2q-1}(x_jX^j+y_jY^j+c_j)$ and $\beta_i(X,Y)=\sum_{j=0}^{2q-1}(x_jX^j+y_jY^j+c_j)$ then we have:

$$\alpha_0(X,Y)+X\alpha_1(X,Y)+\cdots+X^{q-1}\alpha_{q-1}(X,Y)+X^q\beta_0(X,Y)+\cdots+X^{2q-1}\beta_{q-1}(X,Y)=0 \tag{26}$$

$$\alpha_0(X,Y)+Y\alpha_1(X,Y)+Y^2\alpha_2(X,Y)+Y^q\beta_0(X,Y)+\cdots+Y^{2q-1}\beta_{q-1}(X,Y)=0 \tag{27}$$

By substracting (27) and (26) we got

$$0 = (X - Y)\alpha_1(X,Y) + \cdots + (X^{q-1} - Y^{q-1})\alpha_{q-1}(X,Y) + (X^q - Y^q)\beta_q(X,Y) + \ldots \tag{28}$$

Now we can factor $(X - Y)$ and then divide by it and obtain:

$$\begin{aligned} -\alpha_1(X,Y) =& (X+Y)\alpha_2(X,Y) + (X^2 + XY + Y^2)\alpha_3(X,Y) + \cdots + \\ &+ (X^{2q-2} + YX^{2q-3} + \cdots + Y^{2q-3}X + Y^{2q-2})\beta_{2q-1}(X,Y) \end{aligned} \tag{29}$$

Substitute $-\alpha_1(X,Y)$ in equation (26) and remark that all $X^i\alpha_i(X,Y), X^{q+i}\beta_{q+i}(X,Y)$ terms are vanishing:

$$\begin{aligned} \alpha_0(X,Y) &= X\left[\sum_{i=2}^{q-1}\left(\sum_{j=0}^{i-1} X^{i-j-1}Y^j\right)\alpha_i(X,Y) + \sum_{i=q}^{2q-1}\left(\sum_{j=0}^{i-1} X^{i-j-1}Y^j\right)\beta_i(X,Y)\right] - \\ &\quad - \sum_{i=2}^{q-1} X^i\alpha_i(X,Y) - \sum_{i=q}^{2q-1} X^i\beta_i(X,Y) \\ \alpha_0(X,Y) &= X\left[\sum_{i=2}^{q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^j\right)\alpha_i(X,Y) + \sum_{i=q}^{2q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^j\right)\beta_i(X,Y)\right] \\ \alpha_0(X,Y) &= XY\left[\sum_{i=2}^{q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^{j-1}\right)\alpha_i(X,Y) + \sum_{i=q}^{2q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^{j-1}\right)\beta_i(X,Y)\right] \end{aligned} \tag{30}$$

This implies that either $\alpha_0(X,Y)$ is a multiple of XY or $\alpha_0(X,Y) = 0$.

By the GGM assumption, we have that $\alpha_0(X,Y) = 0$.

We continue by replacing $\alpha_0(X,Y) = 0$ in equation (30):

$$-\alpha_2(X,Y) = \sum_{i=3}^{q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^{j-1}\right)\alpha_i(X,Y) + \sum_{i=q}^{2q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^{j-1}\right)\beta_i(X,Y) \tag{31}$$

Substitute the expression of $-\alpha_2(X,Y)$ in equation (26) or (27) and remark that all terms $X^i\alpha_i(X,Y), X^i\beta_i(X,Y)$ (respectively $Y^i\alpha_i(X,Y), Y^i\beta_i(X,Y)$) terms are vanishing

And so on... till we show that $\alpha_i(X,Y) = 0 \quad \forall i = 0 \ldots q-1$ and $\beta_i(X,Y) = 0\ \forall i = q \ldots 2q-$.