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. Zero-knowledge SNARKs (zk-SNARKs) are non-interactive proof systems with short and efficiently verifiable proofs that do not reveal anything more than the correctness of the statement. zk-SNARKs are widely used in decentralised systems to address privacy and scalability concerns. One of the main applications is the blockchain, where SNARKs are used to prove computations with private inputs and reduce on-chain footprint verification and transaction sizes.

We design and implement SnarkPack, a new argument that further reduces the size of SNARK proofs by means of aggregation. Our goal is to provide an off-the-shelf solution that is practical in the following sense: (1) it is compatible with existing deployed systems, (2) it does not require any extra setup.

SnarkPack is designed to work with Groth16 scheme and has logarithmic size proofs and a verifier that runs in logarithmic time in the number of proofs to be aggregated. Most importantly, SnarkPack reuses the public parameters from Groth16 system, so it does not require a separate trusted setup ceremony.

The key tool for our construction is a new commitment scheme that uses as public parameters two existing "powers of tau" ceremony transcripts. The commitment scheme allows us to instantiate a new optimised version of the inner product pairing arguments (IPP) of Bünz et al. without additional trusted setup.

SnarkPack can aggregate 8192 proofs in 8.7s and verify them in 163ms, 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 non-deterministic polynomial-time computation in time that is much shorter than the time required to run the computation given the witness. 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 [BCI$^+$13, GGPR13, PHGR13, BCTV14, Gro16] with constant-size proofs.

Trusted Setup Ceremony. All these constant-size zkSNARK protocol have a common major disadvantage in practice: they rely on some public parameters SRS that are generated by a trusted setup. In theory, this setup is run by a trusted-third party, while in practice, such a bit string can be generated by a so called "ceremony", a multi-party computation between participants who are believed not to collude. Generating such trusted setup is a cumbersome task in practice. These ceremonies are expensive in terms of resources, they must follow specific rules and are generally hard to organise: hundreds of participants with powerful machines need to join efforts to perform a multi-party computation over multiple months.

Groth16. The result of Groth [Gro16] is the state-of-the-art for SNARKs with trusted setup and it achieves the shortest proof size. Groth16 scheme is a SNARK for circuits, requiring to express the computation as an arithmetic circuit and allowing to prove circuit satisfiability, using a Quadratic Arithmetic Program (QAP) characterisation. Briefly, a QAP as introduced by [GGPR13] is translating a circuit into an equivalent arithmetic relation that holds only if the circuit has a solution.

Due to its short proof size and verifier's efficiency, Groth16 SNARK [Gro16] have become a de facto standard in blockchain projects. This results in a great number of available implementations, code auditing and multiple trusted setup ceremonies run by independent institutions.

Motivation. Importantly, the trusted setup in SNARK schemes sets an upper bound on the size of computations (number of constrains in the circuit description) that can be proven. Because modern applications have an increased demand for the size of circuits to be proven, Groth16 proofs start to face scalability problems. A simple solution is to split the computation in different pieces, and prove them independently, but this increases the number of proofs to be added to a single statement.

We address this problem by showing a method to reduce the overhead in communication and verification time for multiple proofs without the need of further trusted setup ceremonies and allowing to take full advantage of existing building blocks for further optimisations.

Being able to rely on the security of well-known trusted setups for which the ceremonies have been largely publicly advertised by participants is a stronger and simpler solution than doing it from scratch.

Filecoin System. 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.

Contribution. We look into reducing proof size and verifier time for SNARKs even further by exploring techniques to aggregate proofs without the requirement for additional trusted setups.

We design SnarkPack, an argument that allows to aggregate n Groth16 zkSNARKs with a $O(\log n)$ proof size and verifier time. Our scheme is based on a trusted setup that can be constructed from two different ceremonies (e.g. the so-called "powers of tau" for Zcash [zca18] and Filecoin [Fil20]).

Our techniques are generic and can apply to other pairing-based SNARKs. However, we chose to focus on Groth16 proofs and tailor optimisations for this case, since it is the most popular scheme among practitioners. Therefore, SnarkPack is the first practical system that can be used in blockchains applications to reduce the on-chain work by employing verifiable outsourcing to process a large number of proofs off-chain. This applies broadly to any system that needs to delegate batches of state updates to an untrusted server.

Related Work. Prior works have built similar schemes for recursion or aggregation of proofs, but they all have critical shortcomings when it comes to implementing them in real-world systems.

Bünz et al. [BMM+19] presented a scheme for aggregating Groth16 proofs that requires a specific trusted setup to construct the structured reference string (SRS) necessary to verify such aggregated proofs. Our result is conceptually similar with the Bünz et al. construction. However, because we avoid the need of a new trusted setup ceremony, we believe that our approach is preferable to [BMM+19] in practical use cases.

Our scheme benefits from many further optimizations and has the advantage of avoiding additional trust assumptions for existing systems, such as an extra ceremony for public parameters generation.

We focus specifically on aggregating proofs generated using the same Groth16 SRS which is the common use case, as opposed to the generic result in [BMM+19] that allow aggregation of proofs from different SRSes. Our result can be extended to support this later case as well.

Other approach to aggregation rely on recursive composition. In more detail, [BCG+20] propose a new SNARK for the circuit that contains n copies of the Groth16 verifier's circuit. However, constructing arithmetic circuits for pairings is expensive (e.g., computing a pairing on the BLS12-377 curve requires ≈ 15000 constraints as shown in [BCG+20]). The advantage of using such expensive schemes for aggregation is their transparent setup.

However, the costs are significant compared with our scheme: they compute FFTs, which require time $O(n \log n)$, the verifier performs $O(n)$ cryptographic operations as opposed to $O(n)$ field operations in our scheme and they require special cycles of curves.

SnarkPack has the advantages of both worlds: it benefits from the power of structured public parameters to avoid expensive computations, while it does not have to run a new setup since it relies on already available trusted setup transcripts for the underlying Groth16 scheme.

Our Techniques and Roadmap. Our main contributions are organised as follows:
Framework for Aggregation: In Section 3 we describe the general framework for aggregating Groth16 proofs for the same verification key crs. The framework is based on inner pairing products arguments from Bünz et al. [BMM+19] for two specialised computations: multi-exponentiation inner product (MIPP) and an target inner pairing product (TIPP).
New Commitment Schemes: In Section 4 we introduce our new commitment scheme with doubly-homomorphic property that rely on existing public setups for Groth16 to generate their structured commitment keys. Both schemes have constant size commitments and are proved to be binding based on assumptions that hold in the generic group model. We think these schemes can be of independent interest in protocols that need to commit to source group elements. Our second scheme has the advantage that allows to commit to two vectors from two different groups with no size overhead.
MT-Inner Pairing Product Argument: In Section 5.2 we design an efficient argument that proves together a multi-exponentiation inner product (MIPP) and a target inner pairing product (TIPP) with minimal overhead. The key ingredients in the $\mathsf{MT\text{-}IPP}$ scheme are our new pair group commitments. Therefore, this makes our $\mathsf{MT\text{-}IPP}$ argument compatible with existing Groth16 setup ceremonies. We further add some important optimisations that make our scheme more efficient than the two different schemes proposed by [BMM+19].
SnarkPack Scheme: In Section 6 we introduce SnarkPack, our efficient aggregation argument based on $\mathsf{MT\text{-}IPP}$ and the new commitments for group vectors.
Implementation: In Section 7 we provide benchmarks, optimisation details for our implementation in Rust, and evaluate its efficiency against batching. SnarkPack is exponentially more efficient than aggregating SNARKs via batching: it takes 163ms 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 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 $0 \leq \ell < n-1$.

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 $x \in \mathbb{Z}_p$ and a vector $\mathbf{A} \in \mathbb{G}_1^n$.
- $\mathbf{A}^{\mathbf{x}} = (A_0^{x_0}, \ldots, A_{n-1}^{x_{n-1}}) \in \mathbb{G}_1^n$ for vectors $\mathbf{x} \in \mathbb{Z}_p^n, \mathbf{A} \in \mathbb{G}_1^n$.
- $\mathbf{A} * \mathbf{x} = \prod_{i=0}^{n-1} A_i^{x_i}$ for vectors $\mathbf{x} \in \mathbb{Z}_p^n, \mathbf{A} \in \mathbb{G}_1^n$.
- $\mathbf{A} * \mathbf{B} \coloneqq \prod_{i=0}^{n-1} e(A_i, B_i)$ for group vectors $\mathbf{A} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$.
- $\mathbf{A} \circ \mathbf{A'} \coloneqq (A_0 A'_0, \ldots, A_{n-1} A'_{n-1})$ for vectors $\mathbf{A}, \mathbf{A'} \in \mathbb{G}_1^n$.

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 an NP relation. 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 their 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. 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:

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

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

$\mathsf{Verify}(\mathsf{crs}, u, \pi) \to 1/0$**:** 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:*

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

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 $C = \mathsf{CM}(\mathsf{ck}; m) \coloneqq \mathsf{Com}_{\mathsf{ck}}(m, 0)$, while for randomised ones we write $C \leftarrow\!\!\$\, \mathsf{CM}(\mathsf{ck}; m) \coloneqq \mathsf{Com}_{\mathsf{ck}}(m, \rho)$.

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:*

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

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:

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

$\mathsf{Check}(\mathsf{vk}, C, x, y, \pi) \to 1/0$: 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 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:*

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

2.3 Assumptions

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

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

Assumption 1 (ASSGP) *The* (q, m)*-Auxiliary Structured Single Group Pairing 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:*

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

We can similarly define the dual assumption, by swapping $\mathbb{G}_1$ and $\mathbb{G}_2$ in the definition above.

Lemma 1. *The* (q, m)*-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, m)-Auxiliary Structured Double Group Pairing ((q, m)-ASDGP) assumption can be stated as:

Assumption 2 (ASDGP) *The* (q, m)*-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:*

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

Lemma 2. *The* (q, m)*-ASDGP assumption holds in the generic group model.*

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

Groth.$\mathsf{Setup}(1^\lambda, \mathcal{R})$

$\alpha, \beta, \gamma, \delta \leftarrow\$ \mathbb{Z}_p^*, \quad s \leftarrow\$ \mathbb{Z}_p^*,$

$\mathsf{crs} = \Big(\mathsf{QAP}, g^\alpha, g^\beta, g^\delta, \{g^{s^i}\}_{i=0}^{d-1}, \ \Big\{g^{\frac{\beta v_j(s)+\alpha w_j(s)+y_j(s)}{\gamma}}\Big\}_{j=0}^{t}, \Big\{g^{\frac{\beta v_j(s)+\alpha w_j(s)+y_j(s)}{\delta}}\Big\}_{j>t}, \Big\{g^{\frac{s^i t(s)}{\delta}}\Big\}_{i=0}^{d-2},$
$h^\beta, h^\gamma, h^\delta, \{h^{s^i}\}_{i=0}^{d-1}\Big)$

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

$\mathsf{td} = (s, \alpha, \beta, \gamma, \delta)$

return $(\mathsf{crs}, \ \mathsf{td})$

Groth.$\mathsf{Prove}(\mathsf{crs}, u, w)$

$u = (a_1, \ldots, a_t), \ a_0 = 1$
$w = (a_{t+1}, \ldots, a_m)$
$v(x) = \sum_{j=0}^{m} a_j v_j(x)$
$v_{mid}(x) = \sum_{j \in I_{mid}} a_j v_j(x)$
$w(x) = \sum_{j=0}^{m} a_j w_j(x)$
$w_{mid}(x) = \sum_{j \in I_{mid}} a_j w_j(x)$
$y(x) = \sum_{j=0}^{m} a_j y_j(x)$
$y_{mid}(x) = \sum_{j \in I_{mid}} a_j y_j(x)$
$h(x) = \frac{(v(x)w(x)-y(x))}{t(x)}$
$f_{mid} = \dfrac{\beta v_{mid}(s) + \alpha w_{mid}(s) + y_{mid}(s)}{\delta}$
$r, u \leftarrow\$ \mathbb{Z}_p^*$
$a = \alpha + v(s) + r\delta, \qquad b = \beta + w(s) + u\delta$
$c = f_{mid} + \frac{t(s)h(s)}{\delta} + ua + rb - ur\delta$
return $(\pi = (A = g^a, B = h^b, C = g^c))$

Groth.$\mathsf{Verify}(\mathsf{vk}, u, \pi)$

$\pi = (A, B, C)$
$v_{io}(x) = \sum_{i=0}^{t} a_i v_i(x)$
$w_{io}(x) = \sum_{i=0}^{t} a_i w_i(x)$
$y_{io}(x) = \sum_{i=0}^{t} a_i y_i(x)$
$f_{io} = \dfrac{\beta v_{io}(s) + \alpha w_{io}(s) + y_{io}(s)}{\gamma}$
Check
$e(A, B) = e(g^\alpha, h^\beta) \cdot e(g^{f_{io}}, h^\gamma) \cdot e(C, h^\delta)$

Groth.$\mathsf{Sim}(\mathsf{td}, u)$

$a, b \leftarrow\$ \mathbb{Z}_p^*$
$c = \dfrac{ab - \alpha\beta - \beta v_{io}(s) + \alpha w_{io}(s) + y_{io}(s)}{\delta}$
return $(\pi = (A = g^a, B = h^b, C = g^c))$

Fig. 1. Groth16 Construction from QAP.

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).

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

We describe Groth = (Setup, Prove, 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:

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

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:

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

Pair Group Commitments. To instantiate our aggregated scheme, we use two new pairing commitment schemes. These schemes need to satisfy special properties (as discussed in Section 4) 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 $\mathsf{CM}_s(\mathbf{A}) \coloneqq \mathsf{CM}_s(\mathsf{ck}_s; \mathbf{A}) = (T_A, U_A)$ where

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

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

$$T_{AB} = (\mathbf{A} * \mathbf{v_1})(\mathbf{w_1} * \mathbf{B}), \quad U_{AB} = (\mathbf{A} * \mathbf{v_2})(\mathbf{w_2} * \mathbf{B})$$

IPP Protocols. One of the key building blocks for our aggregation protocol are *generalized inner product arguments*, called GIPA or IPP protocols. These protocols, as designed in [BMM+19], enable proving the correctness of a large class of inner products between vectors of group and/or field elements committed using (possibly distinct) doubly-homomorphic commitment schemes.

While GIPA schemes from [BMM+19] achieve logarithmic communication, they only have linear verifier time. For our aggregation protocol, we can instantiate two specialised cases of GIPA – multi-exponentiation inner product (MIPP) and an target inner pairing product (TIPP) – using our new commitment schemes under structured references string, and thus, we obtain logarithmic verifier time.

We restate the relations for the two specialized IPP constructions below:
Multi-exponentiation Inner Product (MIPP). The relation for MIPP proofs 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:

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

Target Inner Pairing Product (TIPP). A TIPP proof allows a prover to demonstrate that certain pairing relation hold between committed group elements $\mathbf{A} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$.

The relation for the TIPP we need in Groth16 aggregation is defined by:

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

3.3 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 :

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

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

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.

Relation for Aggregation. More formally, we introduce the relation for aggregating n Groth16 proof vectors $\mathbf{A}, \mathbf{C} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n$ with respect to a fixed verification key vk:

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

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

Algorithms for Aggregation. The aggregation protocol is non-interactive and uses a hash function Hash_0 modeled as a random oracle. The three algorithms for the scheme are described in the following:

Setup Algorithm

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

1. Set commitment keys for both single and double commitment schemes using Groth16 crs:

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

2. Construct MIPP and TIPP SRS from these keys: $\mathsf{crs}_{\mathsf{mipp}}, \mathsf{crs}_{\mathsf{tipp}}$
3. Choose a hash function $\mathsf{Hash}_r : \mathbb{Z}_p^{t \cdot n} \times \mathbb{G}_T^4 \to \mathbb{Z}_p$ given by its description hk_r.
   Output: Keys: $\mathsf{crs}_{\mathsf{agg}} = (\mathsf{vk}, \mathsf{crs}_{\mathsf{mipp}}, \mathsf{crs}_{\mathsf{tipp}}, \mathsf{hk}_r)$

Prove Algorithm

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

1. Parse proving key $\mathsf{crs}_{\mathsf{agg}} \coloneqq (\mathsf{vk}, \mathsf{crs}_{\mathsf{mipp}}, \mathsf{crs}_{\mathsf{tipp}}, \mathsf{hk}_0)$
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 $r = \mathsf{Hash}_r(\mathbf{u}, T_{AB}, U_{AB}, T_C, U_C) \in \mathbb{Z}_p$ 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:

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

8. Run MIPP proof:

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

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

1. Parse verification key $\mathsf{vk}_{\mathsf{agg}} := (\mathsf{vk}, \mathsf{crs}_{\mathsf{mipp}}, \mathsf{crs}_{\mathsf{tipp}}, \mathsf{hk}_0)$
2. Parse $\mathsf{vk} := \left(P = g^{\alpha}, Q = h^{\beta},\ \{S_j\}_{j=0}^{t}, H = h^{\gamma}, D = h^{\delta}\right)$
3. Derive random challenge $r = \mathsf{Hash}_r(T_{AB}, U_{AB}, T_C, U_C)$
4. Compute $Z_{S_j} = S_j^{\sum_{i=0}^{n-1} a_{ij} r^i}$ for all $j = 0 \ldots t$.
5. Check TIPP proof π_{tipp}

$$b_1 \leftarrow \mathsf{TIPP.Verify}(\mathsf{crs}_{\mathsf{tipp}}, (T_{AB}, U_{AB}), Z_{AB}, r, \pi_{\mathsf{tipp}}) \tag{3}$$

6. Check MIPP proof π_{mipp}

$$b_2 \leftarrow \mathsf{MIPP.Verify}(\mathsf{crs}_{\mathsf{mipp}}, (T_C, U_C), Z_C, r, \pi_{\mathsf{mipp}}) \tag{4}$$

7. Check Groth16 aggregated equations to the decision bit b_3:

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

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

4 Pair Group Commitment Schemes

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 optimise the commitment sizes 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.

4.1 Reusing Groth16 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.

The SRSes required for the generation of the public commitment keys should satisfy some properties. 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 two "powers of tau". The construction comes naturally from two ceremonies for Groth16 setup (also known as "powers of tau") for the same generators g and h and different powers $a = \tau_1$ and $b = \tau_2$ up to n and up to $m \geq n$: $g, h, g^{\tau_1}, \ldots, g^{\tau_1^n}, h^{\tau_1}, \ldots, h^{\tau_1^n}, g^{\tau_2}, \ldots, g^{\tau_2^m}, h^{\tau_2}, \ldots, h^{\tau_2^m}$.

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.

In practice, we fortunately have at least two ceremonies that satisfy the requirements for same group generators and different powers: Such values can be obtained from the powers of tau transcript of Zcash [zca18] and Filecoin [Lab18]. The SRS created goes up to $n = 2^{19}$ for τ_1 and $m = 2^{127}$ for τ_2.

4.2 Single group version $\mathbf{CM}_s$.

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.

$\mathsf{KG}(1^\lambda) \to \mathsf{ck}_s = (\mathbf{v}_1, \mathbf{v}_2)$

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

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, m)-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 $(T_A, U_A) = (T_{A^*}, U_{A^*})$:

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

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

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

$$e(A_0/A_0^*, h)e(A_1/A_1^*, h^a) \ldots e(A_{n-1}/A_{n-1}^*, h^{a^{n-1}}) = 1 \tag{7}$$

$$e(A_0/A_0^*, h)e(A_1/A_1^*, h^b) \ldots e(A_{n-1}/A_{n-1}^*, h^{b^{n-1}}) = 1 \tag{8}$$

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, m)-ASSGP assumption.

4.3 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.

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

1. $T_{AB} = (\mathbf{A} * \mathbf{v_1})(\mathbf{w_1} * \mathbf{B}) = e(A_0, h) \ldots e(A_{n-1}, h^{a^{n-1}}) \cdot e(g^{a^n}, B_0) \ldots e(g^{a^{2n-1}}, B_{n-1})$
2. $U_{AB} = (\mathbf{A} * \mathbf{v_2})(\mathbf{w_2} * \mathbf{B}) = e(A_0, h) \ldots e(A_{n-1}, h^{b^{n-1}}) \cdot e(g^{b^n}, B_0) \ldots e(g^{b^{2n-1}}, B_{n-1})$
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, m)-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 MT-IPP: Inner Pairing Product Argument for Aggregation

In this section we will present a Generalized Inner Product Argument MT-IPP which is designed to prove two precise generalizations of the inner product computations for a bilinear group $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$:

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 $\mathbb{G}_1^m \times \mathbb{G}_2^m \to \mathbb{G}_T$:
$$\mathbf{A} * \mathbf{B} := \prod e(A_i, B_i) = \prod e(g^{a_i}, h^{b_i}) = e(g,h)^{\sum a_i b_i}$$

5.1 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. The candidate for the PC is KZG scheme in [KZG10] which allows us to have a constant time verifier for this check.

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

$\mathsf{KZG.KG}(1^\lambda, n) \to (\mathsf{ck}, \mathsf{vk}_h)$**:** Set keys $\mathsf{ck}_g = \{g^{\alpha^i}\}_{i=0}^{n-1}, \mathsf{vk}_h = h^\alpha$.
$\mathsf{KZG.CM}(\mathsf{ck}_g; f(X)) \to C_f$**:** For $f(X) = \sum_{i=0}^{n-1} f_i X^i$, computes $C_f = \prod_{i=0}^{n-1} g^{f_i \alpha^i} = g^{f(\alpha)}$.
$\mathsf{KZG.Open}(\mathsf{ck}_g; C_f, x, y; f(X)) \to \pi$**:** For an evaluation point x, a value y, compute the quotient polynomial
$$q(X) = \frac{f(X) - y}{X - x}$$
and output prove $\pi := C_q = \mathsf{KZG.CM}(\mathsf{ck}_g; q(X))$.
$\mathsf{KZG.Check}(\mathsf{vk}_h = h^\alpha, C_f, x, y, \pi) \to 1/0$**:** Check if
$$e(C_f \cdot g^{-y}, h) = e(C_q, \mathsf{vk}_h \cdot h^{-x}).$$

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

5.2 MT-IPP Scheme

In order to optimize the aggregation contruction based on MIPP and TIPP schemes presented in Section 3.3, 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. We will design a new protocol MT-IPP that simultaneously generate these challenges 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 Equation (2), Equation (1) in the proving algorithm Section 3.3 of the Groth16 Aggregation argument and Equation (4), Equation (3) in the verification algorithm Section 3.3 by a unique prove and verification for the fusioned MT-IPP scheme.

Relation. First we define the relation proven using the merged MT-IPP argument. This is a conjunction of two relations:

MIPP Relation. The multiexponentiation product relation:

$$\mathcal{R}_{\mathsf{mipp}} := \{((T_A, U_A), Z, r; \mathbf{A}, \mathbf{r}) : Z = \mathbf{A} * \mathbf{r} \ \wedge \ (T_A, U_A) = \mathsf{CM}_s(\mathsf{ck}_s; \mathbf{A}) \ \wedge \ \mathbf{r} = (r^i)_{i=0}^{n-1}\}.$$

TIPP Relation. The target inner pairing relation:

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

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

MT-IPP Relation. The merged MT-IPP relation:

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

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

Construction. MT-IPP argument is instantiated using the pair group commitment schemes introduced in Section 4 which satisfy the desired properties for usage in inner pairing product protocols. Our scheme consists of 3 algorithms MT-IPP = (MT.Setup, MT.Prove, MT.Verify) described in the following:

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

1. Set commitment keys for CM_s and CM_d : $\mathsf{ck}_s = (\mathbf{v}_1, \mathbf{v}_2), \quad \mathsf{ck}_d = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{w}_1, \mathbf{w}_2)$
2. Set commitment keys for KZG scheme:

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

3. Define $\mathsf{ck}_{\mathsf{kzg}} := (\mathsf{ck}_{j\sigma}),\ \mathsf{vk}_{\mathsf{kzg}} := (\mathsf{vk}_{j\sigma})$ for $j = 1, 2;\ \sigma = v, w$.
4. Fix Hash_{com}: $\mathbb{G}_T^4 \to \mathbb{Z}_p$ and its description hk_{com}.
5. Fix Hash_{x_0}: $\mathbb{Z}_p^2 \times \mathbb{G}_T \times \mathbb{G}_1 \to \mathbb{Z}_p$ and its description hk_{x_0}.
6. Fix $\mathsf{Hash} : \mathbb{Z}_p \times \mathbb{G}_T^{12} \to \mathbb{Z}_p$ and its description hk.
7. Fix a hash function $\mathsf{Hash}_z : \mathbb{Z}_p \times \mathbb{G}_2^2 \times \mathbb{G}_1^2 \to \mathbb{Z}_p$ and its description hk_z.
8. Set $\mathsf{crs}_{\mathsf{mt}} := (\mathsf{hk}_{com}, \mathsf{hk}_{x_0}, \mathsf{hk}, \mathsf{hk}_z, \mathsf{ck}_s, \mathsf{ck}_d, \mathsf{ck}_{\mathsf{kzg}}, \mathsf{vk}_{\mathsf{kzg}})$.

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

– Loop "split & collapse" for step i

1. $n' = n_{i-1}/2$ where $n_0 = n$
2. If $n' < 1$: *break*
3. Set $\mathbf{B}' := \mathbf{B}^{\mathbf{r}}, \mathbf{w}_1' := \mathbf{w}_1^{\mathbf{r}^{-1}}, \mathbf{w}_2' := \mathbf{w}_2^{\mathbf{r}^{-1}}$.

4. Compute L/R inner products:

$$(Z_L)_{AB} = \mathbf{A}_{[n':]} * \mathbf{B}'_{[:n']} \text{ and } (Z_R)_{AB} = \mathbf{A}_{[:n']} * \mathbf{B}'_{[n':]}$$
$$(Z_L)_C = \mathbf{C}^{\mathbf{r}_{[:n']}}_{[n':]} \text{ and } (Z_R)_C = \mathbf{C}^{\mathbf{r}_{[n':]}}_{[:n']}$$

5. Compute left cross commitments:

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

6. Compute right cross commitments:

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

7. Compute hash to the vector commitments

$$h_{com} = \mathsf{Hash}_{com}((T_{AB}, U_{AB}), (T_C, U_C)).$$

8. Compute challenge x_i, where $x_0 = \mathsf{Hash}_{x_0}(r, h_{com}, Z_{AB}, Z_C)$:

$$x_i = \mathsf{Hash}\left(x_{i-1}; (Z_L, Z_R)_{AB}, (Z_L, Z_R)_C, (T_L, U_L; T_R, U_R)_{AB}, (T_L, U_L; T_R, U_R)_C\right)$$

9. Compute Hadamard products on vectors

$$\mathbf{A} \coloneqq \mathbf{A}_{[:n']} \circ \mathbf{A}^{x_i}_{[n':]},\ \mathbf{B}' \coloneqq \mathbf{B}'_{[:n']} \circ \mathbf{B}'^{x_i^{-1}}_{[n':]},\ \mathbf{C} \coloneqq \mathbf{C}_{[:n']} \circ \mathbf{C}^{x_i}_{[n':]}$$

10. Compute Hadamard products on keys $\mathbf{v}_1, \mathbf{v}_2$ and $\mathbf{w}'_1 \coloneqq \mathbf{w}_1^{\mathbf{r}^{-1}}, \mathbf{w}'_2 \coloneqq \mathbf{w}_2^{\mathbf{r}^{-1}}$

$$\begin{aligned}(\mathbf{v_1}, \mathbf{v_2}) &\coloneqq (\mathbf{v_1}_{[:n']} \circ \mathbf{v_1}^{x^{-1}}_{[n':]}, \mathbf{v_2}_{[:n']} \circ \mathbf{v_2}^{x^{-1}}_{[n':]}) \\ (\mathbf{w}'_1, \mathbf{w}'_2) &\coloneqq (\mathbf{w}'_{1_{[:n']}} \circ \mathbf{w}'^{x}_{1_{[n':]}}, \mathbf{w}'_{2_{[:n']}} \circ \mathbf{w}'^{x}_{2_{[n':]}})\end{aligned} \tag{9}$$

11. Set $n_i = n'$

– Compute proofs $(\pi_{v_j}, \pi_{w_j})_{j=1,2}$ of 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 $z = \mathsf{Hash}_z(x_\ell, v_1, v_2, w_1, w_2)$ for $n = 2^\ell$
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 opening evaluations 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

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

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

$\mathsf{MT.Verify}(\mathsf{crs_{mt}}, \mathsf{statement}; \pi_{\mathsf{mt}}) \to b$**:**

1. Parse $\mathsf{statement} = ((T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, Z_C, r)$
2. Compute hash to the commitments $h_{com} = \mathsf{Hash}_{com}((T_{AB}, U_{AB}), (T_C, U_C))$
3. Reconstruct challenges $\{x_i\}_{i=1}^{\ell}$:

$$x_0 = \mathsf{Hash}_{x_0}(r, h_{com}, Z_{AB}, Z_C)$$

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

4. Construct products and commitments recursively, $i = 1 \to \ell$:
   – $(Z_i)_{AB} = \mathbf{Z_L}[i]_{AB}^{x_i} \cdot (Z_{i-1})_{AB} \cdot \mathbf{Z_R}[i]_{AB}^{x_i^{-1}}$
   – $(T_i)_{AB} = \mathbf{T_L}[i]_{AB}^{x_i} \cdot (T_{i-1})_{AB} \cdot \mathbf{T_R}[i]_{AB}^{x_i^{-1}}$
   – $(U_i)_{AB} = \mathbf{U_L}[i]_{AB}^{x_i} \cdot (U_{i-1})_{AB} \cdot \mathbf{U_R}[i]_{AB}^{x_i^{-1}}$
   where $(Z_0)_{AB} = Z_{AB}, (T_0))_{AB} = T_{AB}, (U_0))_{AB} = U_{AB}$
   – $(Z_i)_C = \mathbf{Z_L}[i]_C^{x_i} \cdot (Z_{i-1})_C \cdot \mathbf{Z_R}[i]_C^{x_i^{-1}}$
   – $(T_i)_C = \mathbf{T_L}[i]_C^{x_i} \cdot (T_{i-1})_C \cdot \mathbf{T_R}[i]_C^{x_i^{-1}}$,
   – $(U_i)_C = \mathbf{U_L}[i]_C^{x_i} \cdot (U_{i-1})_C \cdot \mathbf{U_R}[i]_C^{x_i^{-1}}$
   where $(Z_0)_C = Z_C, (T_0)_C = T_C, (U_0)_C = U_C$
5. Compute final vector value from r:

$$r' = \prod_{i=0}^{\ell-1}(1 + x_{\ell-i}^{-1} r^{2^i})$$

6. Verify final values $(T_\ell, U_\ell, Z_\ell)_{AB}, (T_\ell, U_\ell, Z_\ell)_C$:
   (a) $(Z_\ell)_{AB} \stackrel{?}{=} e(A, B')$
   (b) $(Z_\ell)_C \stackrel{?}{=} C^{r'}$
   (c) Check if $(T_\ell)_{AB} \stackrel{?}{=} e(A, v_1)e(w'_1, B')$ and $(U_\ell)_{AB} \stackrel{?}{=} e(A, v_2)e(w'_2, B')$
   (d) Check if $(T_\ell)_C \stackrel{?}{=} e(C, v_1), \quad (U_\ell)_C \stackrel{?}{=} e(C, v_2)$
7. Verify final commitment keys v_1, v_2, w'_1, w'_2 via KZG
   (a) Reconstruct KZG challenge point: $z = \mathsf{Hash}_z(x_\ell, v_1, v_2, w'_1, w'_2)$ for $n = 2^\ell$
   (b) Reconstruct commitment polynomials:

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

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

(c) Run verification for openings of evaluations in z for $j = 1, 2$:

$$b_{11} \leftarrow \mathsf{KZG.Check}(\mathsf{vk}_{1v}; v_1, z, f_v(z); \pi_{v_1}), \qquad b_{12} \leftarrow \mathsf{KZG.Check}(\mathsf{vk}_{2v}; v_2, z, f_v(z); \pi_{v_2})$$
$$b_{21} \leftarrow \mathsf{KZG.Check}(\mathsf{vk}_{1w}; w_1, z, f_w(z); \pi_{w_1}), \qquad b_{22} \leftarrow \mathsf{KZG.Check}(\mathsf{vk}_{2w}; w_2, z, f_w(z); \pi_{w_2})$$

All KZG.Checks are batched into a single pairing check.

The security of the MT-IPP protocol follows from the same proving strategy as for the security of MIPP and TIPP in [BMM+19]. This is assuming the random oracle model and the commitment algebraic model (see [BMM+19] for details).

Theorem 3. *If* $\mathsf{CM}_s, \mathsf{CM}_d$ *are binding inner product commitments,* KZG.PC *is a polynomial commitment with Computational Knowledge Binding as per Definition 6, then the protocol* MT-IPP *has computational knowledge soundness (Definition 1).*

Proof. This is a standard AND-composition technique for proofs of two relations (in our case $\mathcal{R}_{\mathsf{tipp}} \wedge \mathcal{R}_{\mathsf{mipp}}$). The proof for each individual relation follows the same ideas as [BMM+19] proof for their MIPP and TIPP schemes.

Remark that both CM_d and 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 KZG.PC scheme).

5.3 Formula for Final Commitment Keys

In this section, we will detail one step in showing that such protocol satisfies computational knowledge soundness: Defining the correct polynomials to be committed under kzg.PC scheme in order to show that the structure of the honestly generated final commitment keys is correct.

Recall that our scheme MT-IPP achieves 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 $(\mathsf{ck}_s, \mathsf{ck}_d)$. 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 four pairing checks.

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 $\mathsf{ck}_s, \mathsf{ck}_d$:

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

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 scheme MT-IPP 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 $\mathbf{v} = (v_1, v_2, \ldots, v_{2^\ell}) = (g, g^\alpha, g^{\alpha^2}, \ldots g^{\alpha^{2^\ell - 1}})$ 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. Note that at each round j we split the sequence $v_1, v_2, \ldots, v_n$ 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 $\mathbf{v} = (v_1 = g, v_2 = g^\alpha, \ldots, v_n = g^{\alpha^{n-1}})$ and for a vector of challenges $x_1 \ldots x_{\ell-1}, x_\ell$ is:

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

We will prove the general formula by induction:

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

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

Step 2. 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:
$\mathbf{v}' = \mathbf{v}_{[:2^{\ell-1}]} \circ \mathbf{v}^{x_1}_{[2^{\ell-1}:]}$,
$\mathbf{v}' = (g \cdot g^{x_1\alpha^{2^{\ell-1}}}, g^\alpha \cdot g^{x_1\alpha^{2^{\ell-1}+1}}, g^{\alpha^2} \cdot g^{x_1\alpha^{2^{\ell-1}+2}}, \ldots)$.
We can write this differently as $\mathbf{v}'$=$(v_1 v_1^{x_1\alpha^{2^{\ell-1}}}, \ldots v_{2^{\ell-1}} v_{2^{\ell-1}}^{x_1\alpha^{2^{\ell-1}}})$.
This gives us a nicely written commitment key after first round

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

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 $\mathbf{w} = (w_1 = g^{\alpha^n}, w_2^{\alpha^{n+1}} \ldots, w_n = g^{\alpha^{2n-1}})$, the final key w can be written as:

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

6 SnarkPack: Aggregation Scheme

In this section we show how to use the framework presented in Section 3.3 together with all the pieces described previously in order to build the most efficient scheme for aggregation: SnarkPack. The resulting argument for aggregation is described by 3 algorithms $\mathsf{SnarkPack} = (\mathsf{SP.Setup}, \mathsf{SP.Prove}, \mathsf{SP.Verify})$ as follows:

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

1. Generate commitment key for 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 $\mathsf{Hash}_r : \mathbb{Z}_p^{t \cdot n} \times \mathbb{G}_T^4 \to \mathbb{Z}_p$ given by its description hk_r
5. Set aggregation public parameters: $\mathsf{crs}_{\mathsf{agg}} = (\mathsf{vk}, \mathsf{crs}_{\mathsf{mt}}, \mathsf{hk}_r)$

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

1. Parse proving key $\mathsf{crs}_{\mathsf{agg}} := (\mathsf{vk}, \mathsf{crs}_{\mathsf{mt}}, \mathsf{ck}_s, \mathsf{ck}_d, \mathsf{hk})$
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. Hash these commitments $h_{com} = \mathsf{Hash}_{com}((T_{AB}, U_{AB}), (T_C, U_C))$
6. Derive random challenge $r = \mathsf{Hash}_r(\mathbf{u}, h_{com})$ and set $\mathbf{r} = \{r^i\}_{i=0}^{n-1}$
7. Compute $Z_{AB} = \mathbf{A}^{\mathbf{r}} * \mathbf{B}$
8. Compute $Z_C = \mathbf{C}^{\mathbf{r}} = \prod_{i=0}^{n-1} C_i^{r_i}$.
9. Run MT proof:
$$\pi_{\mathsf{mt}} = \mathsf{MT.Prove}(\mathsf{crs}_{\mathsf{mt}}, (T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, Z_C, r; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{r})$$
10. Set $\pi_{\mathsf{agg}} = ((T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, Z_C, \pi_{\mathsf{mt}})$

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

1. Parse SNARK instances $\mathbf{u} = \{a_{i,j}\}_{i=0,\dots n-1; j=0,\dots t}$
2. Parse verification key $\mathsf{vk}_{\mathsf{agg}} := (\mathsf{vk}, \mathsf{crs}_{\mathsf{mt}}, \mathsf{hk})$
3. Hash the commitments $h_{com} = \mathsf{Hash}_{com}((T_{AB}, U_{AB}), (T_C, U_C))$
4. Parse $\mathsf{vk} := \left(P = g^\alpha, Q = h^\beta, \ \{S_j\}_{j=0}^t, H = h^\gamma, D = h^\delta\right)$
5. Derive random challenge $r = \mathsf{Hash}_r(\mathbf{u}, h_{com})$
6. Set $\mathsf{statement} = (\mathbf{u}, (T_{AB}, U_{AB}), (T_C, U_C), Z_{AB}, Z_C, r)$
7. Check MT proof $b_1 \leftarrow \mathsf{MT.Verify}(\mathsf{crs}_{\mathsf{mt}}, \mathsf{statement}, \pi_{\mathsf{mt}})$
8. Compute $Z_{S_j} = S_j^{\sum_{i=0}^{n-1} a_{ij} r^i}$ for all $j = 0 \dots t$
9. Check Groth16 final equation to the decision bit b_2:
$$Z_{AB} \stackrel{?}{=} e(P^{\sum_{i=0}^{n-1} r^i}, Q) e(\prod_{j=0}^{t} Z_{S_j}, H) e(Z_C, D)$$
10. 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.

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.

7.2 Optimizations

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 compared with the generic framework in Section 3.3. 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.

Compressing pairing checks: A further performance gain in our SnarkPack with respect to the initial framework in Section 3 is given by the verification batch which applies to the pairing checks from MT-IPP verification: 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 $i > 1$:

$$e(r_i A_i, B_i)e(r_i C_i, D_i) \cdots = T_i^r$$

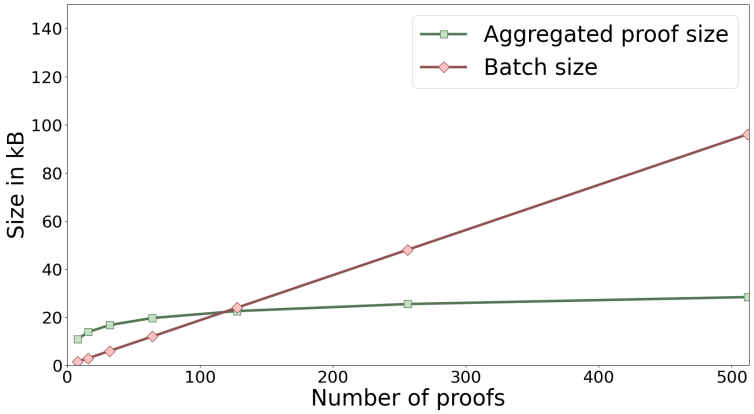

Fig. 2. Proof size: Aggregation vs Batching.

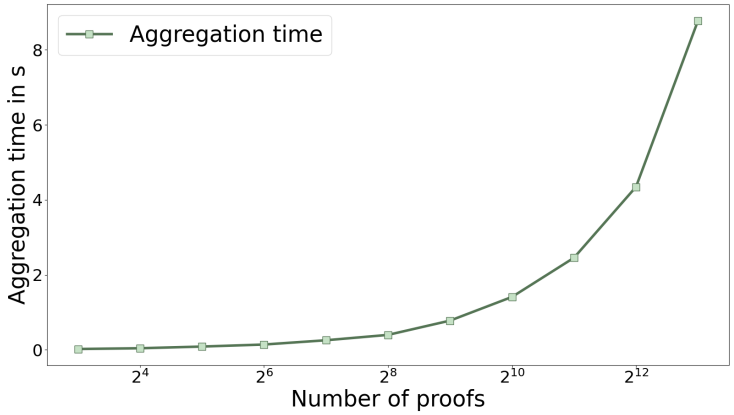

Fig. 3. Aggregation Time

3. Compute the miller loop on the left side of each pairing check:

$$m_i = \text{Miller}((r_i A_i, B_i), (r_i C_i, D_i), \dots)$$

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: $FE(\prod_i \text{Miller}((r_i A_i, B_i), (r_i C_i, D_i) \dots)) = \prod_i T_i^{r_i}$. 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 $\mathbb{G}_\mathbb{T}$.

7.3 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.

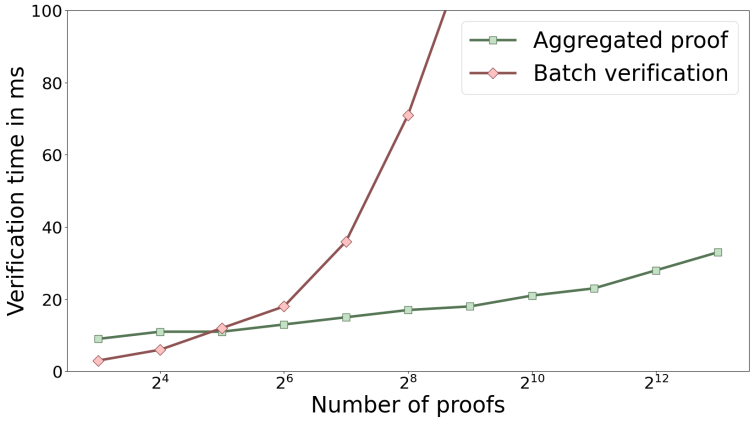

Fig. 4. Verifcation time: Aggregation vs Batching.

7.4 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).

7.5 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.

BCG+20. Sean Bowe, A. Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and H. Wu. Zexe: Enabling decentralized private computation. *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964, 2020.

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.

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.

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 GGM

A.1 ASSGP Assumption in GGM

Assumption 4 (ASSGP) *The (q,m)-Auxiliary Structured Single Group Pairing 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:*

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

We can similarly define the dual assumption, by swapping $\mathbb{G}_1$ and $\mathbb{G}_2$ in the definition above.

Lemma 5. *The (q,m)-ASSGP assumption holds in the generic group model.*

Proof. Suppose $\mathcal{A}$ is an adversary that on input $(\mathsf{gk}, \sigma, \mathsf{aux})$, outputs $(A_0, \ldots, A_{q-1}) \in \mathbb{G}_1^q$ 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 $\alpha_i(X,Y) = \sum_{j=0}^{m}(x_j X^j + y_j Y^j + c_j)$ then we have:

$$\alpha_0(X,Y) + X\alpha_1(X,Y) + X^2\alpha_2(X,Y) + \cdots + X^{q-1}\alpha_{q-1}(X,Y) = 0 \tag{12}$$
$$\alpha_0(X,Y) + Y\alpha_1(X,Y) + Y^2\alpha_2(X,Y) + \cdots + Y^{q-1}\alpha_{q-1}(X,Y) = 0 \tag{13}$$

Then we have:

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

If we substract (15) and (14) we got

$$0 = (X-Y)\alpha_1(X,Y) + \cdots + (X^{q-1} - Y^{q-1})\alpha_{q-1}(X,Y) \tag{16}$$
$$-(X-Y)\alpha_1(X,Y) = (X^2 - Y^2)\alpha_2(X,Y) + \cdots + (X^{q-1} - Y^{q-1})\alpha_{q-1}(X,Y) \tag{17}$$

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

$$\begin{aligned} -\alpha_1(X,Y) =& (X+Y)\alpha_2(X,Y) + (X^2 + XY + Y^2)\alpha_3(X,Y) + \cdots + \\ &+ (X^{q-2} + YX^{q-3} + \cdots + Y^{q-3}X + Y^{q-2})\alpha_{q-1}(X,Y) \end{aligned} \tag{18}$$

Substitute the expression of $-\alpha_1(X,Y)$ in equation (14) and remark that all $X^i\alpha_i(X,Y)$ terms are vanishing:

$$\begin{aligned} \alpha_0(X,Y) &= X[(X+Y)\alpha_2(X,Y) + (X^2 + XY + Y^2)\alpha_3(X,Y) + \cdots + (X^{q-2} + X^{q-3}Y + \cdots + \\ &\quad + XY^{q-3} + Y^{q-2})\alpha_{q-1}(X,Y)] - X^2\alpha_2(X,Y) - \cdots - X^{q-1}\alpha_{q-1}(X,Y) \\ \alpha_0(X,Y) &= XY\alpha_2(X,Y) + (X^2Y + XY^2)\alpha_3(X,Y) + \cdots + (X^{q-2}Y + \cdots + XY^{q-2})\alpha_{q-1}(X,Y) \\ \alpha_0(X,Y) &= XY[\alpha_2(X,Y) + (X+Y)\alpha_3(X,Y) + \cdots + (X^{q-3} + X^{q-4}Y + \cdots + Y^{q-3})\alpha_{q-1}(X,Y)] \end{aligned} \tag{19}$$

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 (19):

$$0=\alpha_2(X,Y)+(X+Y)\alpha_3(X,Y)+\cdots+(X^{q-3}+X^{q-4}Y+\cdots+Y^{q-3})\alpha_{q-1}(X,Y) \tag{20}$$

$$-\alpha_2(X,Y)=(X+Y)\alpha_3(X,Y)+\cdots+(X^{q-3}+X^{q-4}Y+\cdots+Y^{q-3})\alpha_{q-1}(X,Y) \tag{21}$$

Substitute the expression of $-\alpha_2(X,Y)$ in equation (15) and remark that all $Y^i\alpha_i(X,Y)$ terms are vanishing:

$$\begin{aligned}0=&-Y\alpha_1(X,Y)-Y^2[(X+Y)\alpha_3(X,Y)+\cdots+(X^{q-3}+X^{q-4}Y+\ldots\\&+Y^{q-3})\alpha_{q-1}(X,Y)]-Y^3\alpha_3(X,Y)-\cdots-Y^{q-1}\alpha_{q-1}(X,Y)\\Y\alpha_1(X,Y)=&Y^2X\alpha_3(X,Y)+\cdots+(X^{q-3}Y^2+X^{q-4}Y^3+\cdots+XY^{q-2})\alpha_{q-1}(X,Y)\\Y\alpha_1(X,Y)=&Y^2X[\alpha_3(X,Y)+\cdots+(X^{q-4}+X^{q-5}Y+\cdots+Y^{q-4})\alpha_{q-1}(X,Y)]\end{aligned} \tag{22}$$

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 (22):

$$\begin{aligned}0=&\alpha_3(X,Y)+\cdots+(X^{q-4}+X^{q-5}Y+\cdots+Y^{q-4})\alpha_{q-1}(X,Y)\\-\alpha_3(X,Y)=&(X^2+XY+Y^2)\alpha_4(X,Y)+\cdots+(X^{q-4}+X^{q-5}Y+\cdots+Y^{q-4})\alpha_{q-1}(X,Y)\end{aligned} \tag{23}$$

And so on... till we show that $\alpha_i(X,Y)=0\ \ \forall i=0\ldots q-1$. We conclude that the adversarly produced vector $(A_0,\ldots,A_{q-1})=\mathbf{1}_{\mathbb{G}_1}$.

A.2 ASDGP Assumption in GGM

Assumption 5 (ASDGP) *The (q,m)-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:*

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

Lemma 6. *The (q,m)-ASDGP assumption holds in the generic group model.*

Proof. Suppose $\mathcal{A}$ is an adversary that on input $(\mathsf{gk},\sigma,\mathsf{aux})$, outputs $\mathbf{A}=(A_0,\ldots,A_{q-1})$ and $\mathbf{B}=(B_0,\ldots,B_{q-1})$ 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 $\alpha_i(X,Y)=\sum_{j=0}^{m}(x_jX^j+y_jY^j+c_j)$ and $\beta_i(X,Y)=\sum_{j=0}^{m}(x_jX^j+y_jY^j+c_j)$ such that:

$$\alpha_0(X,Y) + X\alpha_1(X,Y) + \cdots + X^{q-1}\alpha_{q-1}(X,Y) + X^q\beta_0(X,Y) + \cdots + X^{2q-1}\beta_{q-1}(X,Y) = 0 \tag{24}$$

$$\alpha_0(X,Y) + Y\alpha_1(X,Y) + \cdots + Y^{q-1}\alpha_{q-1}(X,Y) + Y^q\beta_0(X,Y) + \cdots + Y^{2q-1}\beta_{q-1}(X,Y) = 0 \tag{25}$$

By substracting (25) and (24) we got

$$0 = (X-Y)\alpha_1(X,Y) + \cdots + (X^{q-1} - Y^{q-1})\alpha_{q-1}(X,Y) + (X^q - Y^q)\beta_q(X,Y) + \ldots \tag{26}$$

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

$$\begin{aligned} -\alpha_1(X,Y) =& (X+Y)\alpha_2(X,Y) + (X^2+XY+Y^2)\alpha_3(X,Y) + \cdots + \\ &+ (X^{2q-2} + YX^{2q-3} + \cdots + Y^{2q-3}X + Y^{2q-2})\beta_{2q-1}(X,Y) \end{aligned} \tag{27}$$

Substitute $-\alpha_1(X,Y)$ in equation (24) and remark that all $X^i\alpha_i(X,Y), X^{q+i}\beta_{q+i}(X,Y)$ terms are vanishing:

$$\begin{aligned} \alpha_0(X,Y) &= X\left[\sum_{i=2}^{q-1}\left(\sum_{j=0}^{i-1} X^{i-j-1}Y^j\right)\alpha_i(X,Y) + \sum_{i=q}^{2q-1}\left(\sum_{j=0}^{i-1} X^{i-j-1}Y^j\right)\beta_i(X,Y)\right] - \\ &\quad - \sum_{i=2}^{q-1} X^i\alpha_i(X,Y) - \sum_{i=q}^{2q-1} X^i\beta_i(X,Y) \\ \alpha_0(X,Y) &= X\left[\sum_{i=2}^{q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^j\right)\alpha_i(X,Y) + \sum_{i=q}^{2q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^j\right)\beta_i(X,Y)\right] \\ \alpha_0(X,Y) &= XY\left[\sum_{i=2}^{q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^{j-1}\right)\alpha_i(X,Y) + \sum_{i=q}^{2q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^{j-1}\right)\beta_i(X,Y)\right] \end{aligned} \tag{28}$$

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 (28):

$$-\alpha_2(X,Y) = \sum_{i=3}^{q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^{j-1}\right)\alpha_i(X,Y) + \sum_{i=q}^{2q-1}\left(\sum_{j=1}^{i-1} X^{i-j-1}Y^{j-1}\right)\beta_i(X,Y) \tag{29}$$

Substitute the expression of $-\alpha_2(X,Y)$ in equation (24) or (25) 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 $\alpha_i(X,Y) = 0 \ \forall i = 0 \ldots q-1$ and $\beta_i(X,Y) = 0 \, \forall i = q \ldots 2q-1$. We conclude that the adversarly produced vectors $(A_0, \ldots, A_{q-1}) = \mathbf{1}_{\mathbb{G}_1}, (B_0, \ldots, B_{q-1}) = \mathbf{1}_{\mathbb{G}_2}$.