Multi-party Setup Ceremony for Generating Multivariate zk-SNARK Parameters

Muhammed Ali Bingol

Tokamak Network

muhammed@tokamak.network

Abstract

The development of succinct non-interactive arguments of knowledge (SNARKs), which maintain a constant proof size regardless of computational complexity, has led to substantial improvements in both the scalability and privacy of verifiable computations. By enabling efficient verification of complex computations without revealing sensitive data, SNARK systems underpin modern applications ranging from privacy-focused protocols to high-performance blockchain infrastructures.

This work presents a generalised specification for a Multi-Party Computation (MPC) setup ceremony designed to generate structured reference strings for non-transparent, pairing-based zk-SNARKs. Building on the MMORPG framework of Bowe, Gabizon, and Miers, we design a unified MPC protocol that accommodates multi-dimensional parameter structures required by contemporary SNARK designs. While our methodology is applicable to a broad class of structured CRS constructions, the Tokamak zk-SNARK serves as a motivating example illustrating the need for such a generalisation. The proposed ceremony supports universal and circuit-specific parameter generation, enabling reuse of CRS components across multiple instantiations while maintaining strong soundness guarantees under the presence of at least one honest participant. The protocol operates in two phases: (i) a universal "Powers of τ"-style phase generating parameter sets independent of circuit topology; and (ii) a circuit-specialised phase derived from a subcircuit-based universal circuit paradigm. For both phases, we formalise complete computation and verification procedures, including proof-of-knowledge mechanisms, cross-parameter consistency checks, and structured multivariate parameter updates. Optional integration of a random beacon—as in the MMORPG construction—is supported to enhance unpredictability and public auditability without altering the core security assumptions.

Overall, this work provides a modular, extensible, and publicly verifiable MPC framework suitable for SNARK systems requiring structured, multi-parameter CRS generation, while Tokamak serves as a practical motivating instance of the broader methodology.

Keywords: Multi-party computation, zk-SNARKs, setup ceremony, cryptographic protocol

1 Introduction

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) have become a cornerstone in modern cryptographic systems due to their ability to efficiently and succinctly verify the correctness of computations without revealing any additional information beyond the fact that the computation was performed correctly. zk-SNARKs are highly valued for their conciseness, efficiency, and public verifiability, making them indispensable in fields such as privacy-preserving protocols, decentralized systems, and, particularly, blockchain technologies. Recent advancements in zk-SNARKs have further driven their adoption in practical applications because of their small proof size and fast verification times.

zk-SNARK protocols can be broadly categorized into transparent and non-transparent constructions based on their setup requirements. Transparent zk-SNARKs eliminate the need for a setup by deriving their public parameters through deterministic processes, often leveraging publicly verifiable randomness or cryptographic hash functions. In contrast, non-transparent zk-SNARKs rely on a common or structured reference string generated through a setup ceremony, where the secrecy of certain parameters (commonly referred to as "toxic waste") must be securely discarded to maintain soundness. While

transparent schemes do not require a setup for their parameters, non-transparent schemes necessitate a trustless setup ceremony due to potential trust issues. While transparent constructions offer stronger trust assumptions, non-transparent zk-SNARKs achieve significantly smaller proof sizes and faster verification, making them widely adopted in practical applications despite the setup complexity.

Among non-transparent schemes, Groth16, introduced at EUROCRYPT 2016 [1], is one of the most popular zk-SNARK constructions, primarily due to its small proof size. Despite its efficiency, Groth16 still requires a setup process, in which both proving and verification keys are generated.

The deployment of non-transparent zk-SNARKs require a crucial setup phase known as the generation of a Common Reference String (CRS), or public parameters, which are essential for both the proof construction and verification processes. This setup phase introduces a significant vulnerability: the entity responsible for generating the CRS gains access to secret trapdoor information, commonly referred to as "toxic waste." This toxic waste can be exploited to forge fraudulent proofs, an issue that poses a critical security risk in sensitive applications such as cryptocurrencies, where such an attack could result in undetected financial losses worth billions of dollars.

To mitigate this risk, Multi-Party Computation (MPC) protocols are often employed during the setup phase to distribute the responsibility of generating the CRS among multiple participants. If at least one participant in the setup ceremony is honest and securely deletes their portion of the toxic waste, the integrity of the system is guaranteed. A prominent example of this approach is the MMORPG (Multi-Party Mathematical Operations with Random Public Generation) protocol, proposed by Bowe, Gabizon, and Miers in [2]. MMORPG operates in random beacon mode, providing a decentralized and publicly verifiable setup, which has been widely adopted in zk-SNARK setups like Zcash, Semaphore, and others.

The use of a random beacon in the MMORPG protocol helps ensure the randomness and integrity of the setup. However, obtaining a secure random beacon is a non-trivial problem, as it introduces its own set of challenges. Alternatives, such as using blockchain block headers or verifiable delay functions (VDFs) [3, 4, 5, 6], have been explored, but these come with their own limitations, such as susceptibility to bias or the need for specialized hardware.

In this paper, we present a comprehensive protocol design for a secure, decentralized, and scalable MPC setup scheme. We begin by revisiting the Groth16 setup ceremonies based on BGM17 [2], and then introduce our MPC setup ceremony specifically designed for the recent Tokamak Network protocol [7]. Detailed pseudocode is provided for the Tokamak zk-SNARK MPC setup, outlining the key steps involved in parameter generation and verification. Furthermore, we present a step-by-step description of the parameter generation and verification processes. The application of BGM17's MMORPG MPC scheme to Groth16 is presented in the section A.

1.1 Comparison of Circuit-Specific and Universal Setups

Classical zk-SNARKs such as Groth16 [1] rely on a *circuit-specific* setup, requiring a fresh CRS for each new circuit. This CRS is non-updatable and cannot be reused, which limits scalability in applications requiring continual deployment of new constraint systems.

Recent SNARK designs have explored *universal* or *field-programmable* setups, where a single CRS supports a family of circuits. Such designs often rely on modular circuit libraries, subcircuit instantiation, or universal circuits that can be specialized via wiring or copying operations. These approaches reduce the verifier's preprocessing burden by separating universal components from circuit-specific structure, thereby lowering communication and storage costs.

Some works additionally demonstrate that verifier preprocessing can be minimized or even eliminated through richer subcircuit representations or mixed R1CS/Plonkish encodings. While these constructions sacrifice CRS updatability, they offer improved scalability for applications such as verifiable RAM, distributed computation, and blockchain-based state transitions. Our MPC framework is compatible with such universal or multivariate SNARK architectures, enabling secure and decentralized generation of the associated CRS.

In [7], the authors introduce the Tokamak zk-SNARK mechanism that efficiently manages verifier preprocessing through field-programmable circuit derivation, which starts with a universal circuit composed of subcircuits and derives program-specific circuits by replicating and connecting these subcircuits. While the setup does not support updatability, it significantly reduces the data dimensionality for verifier preprocessing, addressing high communication complexity in verifiable RAM computation within distributed networks of untrusted nodes. By integrating a permutation argument, they transform a SNARK with a common reference string, like Groth16, into one with a universal setup, separating circuit configuration into two algorithms—setup and verifier—allowing adjustable security dependencies.

their SNARK achieves state-of-the-art communication and computation efficiency with verifier preprocessing and surpasses others when preprocessing is eliminated. It combines R1CS and Plonkish circuit representations, using a permutation map for wiring, which reduces data dimensionality compared to PlonK or Marlin but sacrifices setup updatability. This SNARK is highly efficient, suitable for verifiable machine computation, and particularly effective in distributed networks like blockchains, where it reduces the burden of verifier preprocessing data on network resources.

1.2 Related Work

zk-SNARKs often require a common reference string (CRS) generated via a trusted setup. This process, known as the setup ceremony, is critical because any leakage of the secret randomness—commonly referred to as "toxic waste"—can compromise the entire proof system by allowing adversaries to forge proofs. Unlike STARKs or Bulletproofs, which are transparent and do not require trusted setup, zk-SNARKs (particularly non-universal schemes like Groth16) continue to dominate in terms of succinctness and verifier efficiency, making them the preferred choice in many blockchain applications. Therefore, the integrity and decentralization of the CRS generation remain a fundamental security consideration.

To reduce the need for trust, multi-party computation (MPC) has become the common method for setting up zk-SNARKs. One of the first and most well-known MPC ceremonies was run by Zcash in 2016, where six participants worked together to generate the public parameters for the Sprout system. In their influential work, Ben-Sasson et al. proposed an MPC protocol that remains secure as long as at least one participant is honest. However, early implementations such as the BCGTV [8] protocol used in Zcash's "Sprout" setup faced criticism due to limited and pre-identified participation, which conflicted with decentralization ideals. Later, Bowe et al. proposed the MMORPG protocol [2], introducing a two-phase MPC process: the generic "Powers of Tau" phase and a circuit-specific phase, with coordination managed by a central actor. These advancements have shaped modern MPC-based ceremonies by improving participation scalability, modularity, and trust assumptions.

Another important contribution is Snarky Ceremonies by Kohlweiss et al. [9], which analyzes the security, transparency, and usability of setup ceremonies used for SNARK systems. The authors study how users perceive trust in such ceremonies and explore different ceremony models, including those with public audits and decentralized participation. They emphasize the importance of user-friendly tools, strong communication, and transparency in encouraging honest contributions. Their work also proposes design principles for building future ceremonies that are secure, understandable, and scalable. This research helps bridge the gap between cryptographic soundness and real-world usability in trusted setup processes.

A notable example of a trusted setup is the ongoing"powers-of-tau" ceremony, originally created for Semaphore [1]—a privacy-focused protocol enabling anonymous signaling on Ethereum. This setup, based on the BN254 elliptic curve, has attracted 71 participants to date. Several major projects have since built their own setup ceremonies on top of it, such as Tornado.Cash, as well as the Hermez network and Loopring.The Aleo setup ceremony [10] currently has more contributions than the previous record, Tornado Cash's ceremony's 1114 contributions Filecoin, a decentralized protocol for data storage, conducted setup ceremonies [11] involving groups of 19 and 33 participants, respectively [2] .

As part of the Aztec protocol, one of the largest MPC setup ceremonies in history was successfully completed [12]. Conducted over the BN254 curve, the ceremony involved 202 active participants selected from over 600 applicants, spanning 105 cities across 41 countries. Designed with a strong emphasis on transparency and decentralization, the ceremony generated over 100 million elliptic curve points, demonstrating the feasibility of large-scale, global collaboration in secure parameter generation[3].

Recent work by Nikolaenko et al. [13] focuses on making trusted setup ceremonies for SNARKs more decentralized and open. Traditional ceremonies rely on a central coordinator, which limits transparency and participation. Their proposed system removes the need for a coordinator by using blockchain smart contracts to manage contributions in a permissionless way. The setup remains secure as long as at least one participant is honest. They implement their protocol on Ethereum and explore both on-chain and off-chain data storage options, using efficient proof techniques to verify updates. This work helps make setup ceremonies more scalable, transparent, and secure.

[1] `https://github.com/worldcoin/semaphore-mtb-setup`
[2] `https://github.com/filecoin-project/phase2-attestations/tree/master`
[3] `https://github.com/AztecProtocol/ignition-verification`

Our Contributions

This work introduces a general-purpose Multi-Party Computation (MPC) framework for generating structured reference strings in non-transparent, pairing-based zk-SNARKs. Although some aspects are motivated by multivariate SNARK architectures—such as the Tokamak-style design [7]—the framework itself is protocol-agnostic and applies to any proving system requiring multi-dimensional CRS elements. Building upon the MMORPG paradigm of BGM17 [2], we develop new cryptographic components, verification logic, and engineering techniques to support structured monomials beyond the univariate setting. Our main contributions are summarised below:

- **Tri-variant MPC powers-of-τ construction.** We present the first MPC ceremony capable of generating CRS elements involving *tri-variant* powers-of-τ, i.e., structured monomials of the form $\{\alpha^h x^i y^k\}$. This generalises the univariate CRS used in systems such as Groth16 and requires new correctness constraints, update rules, and pairing-based consistency checks.
- **Generalised multivariate MPC framework.** We design a modular MPC methodology that supports any multivariate CRS involving multiple toxic-waste parameters and higher-degree cross-terms. The framework formalises reusable building blocks—randomness contributions, proof-of-knowledge mechanisms, and multivariate consistency checks—that can serve as a baseline for future structured SNARK setups.
- **Performance optimisations.** The proposed ceremony significantly reduces computational and verification overhead for participants. All heavy computations are isolated to a one-time intermediate preparation step, which can be accelerated using GPU implementations and whose correctness can be verified probabilistically. An optimised reference implementation is available in [14].
- **Two-phase modular architecture.** Our framework naturally separates the setup into a *universal* phase and a *circuit-specialised* phase. Universal parameters can be reused across circuits as long as predefined bounds are respected, while circuit-specific parameters need only be regenerated when the circuit template or subcircuit library changes. This modularity supports scalable deployments and reduces the frequency of full ceremonies.
- **Asynchronous and decentralised participation.** The protocol accommodates arbitrary participant ordering and does not require a fixed number of contributors. Participants may join independently in either phase, enabling decentralised coordination and robustness in permissionless or large-scale ceremonies.
- **Public verifiability and auditability.** Every contribution includes a proof of knowledge of the randomness added, and all updates undergo rigorous multivariate consistency checks. This ensures full public auditability without relying on trusted intermediaries. Additionally, batch verification supports efficient validation of many contributions at once.
- **Optional random beacon integration.** Inspired by prior ceremonies [2, 9], the framework optionally incorporates random-beacon entropy to further decentralise trust. Beacon-derived randomness may be injected at designated points without modifying the core soundness assumptions, mitigating adversarial bias in parameter generation.

1.3 Organization

The rest of the paper is organised as follows. Section 2 introduces the necessary cryptographic preliminaries, notation, and definitions used throughout our generalised multivariate MPC framework. Section 3 presents the pseudocode for the proposed MPC setup, describing the computation and verification routines for multi-variant parameter generation. Section 4 provides the full two-phase MPC setup ceremony, including the universal "Powers of τ" phase and the circuit-specialised phase, together with participant roles, message flow, and optional random-beacon integration. We illustrate our ceremony instantiation using the Tokamak zk-SNARK as a use case, since its three-variant parameter structure aligns naturally with our framework. Section 5 concludes the paper by summarising contributions and outlining future directions. Finally, Section A revisits the MMORPG MPC protocol of BGM17 [2] and demonstrates its application to Groth16 [1] as a classical baseline.

2 Preliminaries

2.1 Definitions and notation

Our operations will be conducted within bilinear groups $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_T$ each of prime order p, together with respective generators G_1, G_2 and G_T. These groups are equipped with a non-degenerate bilinear pairing $e : G_1 \times G_2 \to G_T$ with $e(G_1, G_2) = G_T$. We write $\mathbb{G}_1$ and $\mathbb{G}_2$ additively, and $\mathbb{G}_T$ multiplicatively. For $a \in \mathbb{F}_p$, we denote $[A]_1 := a \cdot G_1$, $[A]_2 := a \cdot G_2$. We use the notation $\mathbb{G} := \mathbb{G}_1 \times \mathbb{G}_2$. We denote $\mathbb{G}_1^*$, $\mathbb{G}_2^*$ the non-zero elements of $\mathbb{G}_1$, $\mathbb{G}_2$ and denote $\mathbb{G}^* := G_1^* \times G_2^*$.

Table 1 summarizes the key symbols and notation used throughout this paper. These definitions provide the necessary background for understanding the algebraic structure, group operations, and protocol-specific variables employed in our MPC setup ceremony.

Symbols & Notation	Description
$\mathbb{G}$	$\mathbb{G}_1 \times \mathbb{G}_2$
$\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_T$	bilinear groups
e	a non-degenerate bilinear pairing s.t. $e(G_1, G_2) = G_T$
$\mathbb{G}_1^*$	the non-zero elements of $\mathbb{G}_1$
$\mathbf{G}$	defines both group elements (G_1, G_2) s.t. $G_1 \in \mathbb{G}_1, G_2 \in \mathbb{G}_2$
$[A]_1$ or $[A]_2$ or $[A]$	$a \cdot G_1$ or $a \cdot G_2$, or $a \cdot \mathbf{G}$ where $a \in \mathbb{F}_p$
$[\delta_j]_1$	the parameter δ of jth participant in $\mathbb{G}_1$
x_j^i	the parameter x^i of the jth participant
$\left[x^i\right]_1$	$x_1^i \cdot x_2^i \cdots x_j^i \cdots x_N^i \cdot G_1 \in \mathbb{G}_1$
$[\alpha x^i]^{\mathbf{j}}$	$(\alpha_0 x_0^i)(\alpha_1 x_1^i) \cdots (\alpha_j x_j^i) \cdot G_1$
$\text{transcript}_{i,j}$	the record of the generated values in the protocol up to the point when participant j transmits their message during phase i
$\text{RandomBeacon}(J, k)$	takes a time slot J and a positive integer k as input, and produces k elements $a_1, \ldots, a_k \in \mathbb{F}_p^*$.
$\mathcal{RO}$	It is an oracle that takes as input strings of arbitrary length and output is a uniform independent elements of $\mathbb{G}_2^*$.
$\mathbf{n}$	the maximum number of constraints
s_D	The number of subcircuits in a subcircuit library.
$l_{pub}^{(in)}$	The number of input wires of the public input buffer subcircuit.
$l_{pub}^{(out)}$	The number of output wires of the public output buffer subcircuit.
$l_{prv}^{(in)}$	The number of input wires of the private input buffer subcircuit.
$l_{prv}^{(out)}$	The number of output wires of the private output buffer subcircuit.
m_D	The total number of wires of all subcircuits in a subcircuit library ($l < m_D$).
l_D	The total number of input and output wires of all subcircuits in a subcircuit library ($l_D < m_D$).
s_{max}	The maximum number of subcircuit placements that a circuit can be composed of.
l_{pub}	$l_{pub}^{(in)} + l_{pub}^{(out)}$.
l_{prv}	$l_{prv}^{(in)} + l_{prv}^{(out)}$.
l	$l_{pub} + l_{prv}$.
m_I	$l_D - l$: The number of interface wires of all subcircuits in a subcircuit library.

Table 1: Symbols & Notation Table

2.2 Verifiable Delay Functions and Random Beacons

Reliable sources of public randomness are critical for ensuring fairness and security in cryptographic protocols, particularly in multi-party computation (MPC) setup ceremonies . Two foundational primitives in this context are *Random Beacons* and *Verifiable Delay Functions (VDFs)* [3, 5, 15].

Verifiable Delay Functions (VDFs) are cryptographic primitives that require a prescribed, inherently sequential amount of real time to evaluate, regardless of available computational resources or parallelism. However, once computed, the output can be verified efficiently. A VDF is defined by a tuple of three algorithms: Setup, Eval, and Verify [3].

- $\mathsf{Setup}(\lambda, t) \to \mathsf{pp}$: Takes a security parameter λ and a delay parameter t to generate public parameters pp.
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y, \pi)$: Computes the delayed output y and a (possibly optional) proof π from input x.
- $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) \to \{\mathsf{true}, \mathsf{false}\}$: Efficiently verifies the correctness of y on input x using the proof π.

A secure VDF should satisfy the following properties:

- **Sequentiality**: The evaluation must inherently require t sequential steps, making it infeasible for a parallel adversary to significantly accelerate the process.
- **Efficient Verifiability**: The verification process should be fast, ideally in $\mathcal{O}(\text{polylog}(t))$ time.
- **Uniqueness**: For every input x, there must exist a unique output y such that $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = \mathsf{true}$.

The core security assumption behind VDFs is that no adversary can compute the output significantly faster than the prescribed delay, even with massive parallelism. This makes VDFs especially suitable for scenarios requiring time-delayed but publicly verifiable computation—most notably, in generating trusted randomness where precomputation or grinding attacks must be prevented. Common constructions include repeated squaring in hidden-order groups or sequential walks over isogeny-based elliptic curves [3]. These features make VDFs suitable for secure randomness beacons, time-stamping, and consensus mechanisms in blockchain protocols.

Random Beacons are public randomness services that periodically emit unpredictable and unbiased values. To be secure, a random beacon must satisfy three key properties: *unpredictability* (outputs cannot be known before release), *unbiasability* (no participant can influence the output), and *public verifiability* (any observer can confirm the validity of the output). However, naively implemented beacons may be vulnerable to adversaries who can evaluate multiple inputs and selectively reveal favorable outputs.

To mitigate such grinding attacks and ensure unpredictability until a predetermined time, VDFs are often employed within random beacon constructions. By enforcing a delay between the randomness input and its publication, VDF-backed beacons make it computationally infeasible for adversaries to bias the outcome. In our MPC protocol, we optionally use a beacon function $\text{RandomBeacon}(J, k)$ that takes a time slot J and returns k uniformly random elements from $\mathbb{F}_p^*$. This enhances entropy in the setup process and mitigates trust assumptions. Notable implementations of such beacons include Ethereum 2.0's beacon chain [16] and Chia's blockchain protocol [17], both of which leverage VDFs to ensure cryptographic fairness and temporal integrity.

In the context of this work, we consider random beacon functions as optional but powerful tools to enhance the security of MPC-based parameter generation. By injecting VDF-backed entropy at critical stages, we reduce reliance on participant honesty and improve resistance to late-stage manipulation.

2.3 Bilinear Pairings

A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ between three cyclic groups of prime order p, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are groups equipped with efficient group operations. The pairing e satisfies the following properties:

- **Bilinearity:** For all $a, b \in \mathbb{Z}_p$ and for all $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$, we have
$$e(aP, bQ) = e(P, Q)^{ab}.$$
- **Non-degeneracy:** $e(P, Q) \neq 1$ for some $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$.
- **Efficiency:** The map e can be computed efficiently.

Pairings are central to pairing-based zk-SNARKs like Groth16, where they are used to construct succinct proofs and enable efficient verification. In such protocols, the pairing function allows the verifier to check polynomial relationships in encoded form via a single pairing equation, thereby compressing complex arithmetic checks into short, constant-size proofs.

2.4 Langrange Interpolation

Lagrange interpolation is a method for reconstructing a univariate polynomial $f(x)$ of degree at most d from its evaluations at $d+1$ distinct points. Given evaluation points $\{x_i\}_{i=0}^{d}$ and values $\{y_i\}_{i=0}^{d}$ such that $f(x_i) = y_i$, the interpolating polynomial is defined as:

$$f(x) = \sum_{i=0}^{d} y_i \cdot \ell_i(x), \quad \text{where} \quad \ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{d} \frac{x - x_j}{x_i - x_j}.$$

In zk-SNARK schemes, Lagrange interpolation is used to construct polynomials that encode the witness and the constraint system of an arithmetic circuit. Specifically, when transforming the Rank-1 Constraint System (R1CS) into a Quadratic Arithmetic Program (QAP), interpolation allows the prover to build target polynomials from known evaluations at fixed domain points. These interpolated polynomials are then committed using elliptic curve encodings and verified through pairing equations.

In the second phase of zk-SNARK setup ceremonies, especially in schemes like Groth16, Lagrange interpolation plays a central role in transforming universal CRS elements into circuit-specific parameters. This transformation involves evaluating Lagrange basis polynomials over a finite field domain, typically defined over a multiplicative subgroup of $\mathbb{F}_p$ of size $n = 2^t$ with a primitive root of unity ω.

For each $i \in [1..n]$, the i-th Lagrange basis polynomial $L_i(x)$ satisfies $L_i(\omega^i) = 1$ and $L_i(\omega^j) = 0$ for all $j \neq i$. These polynomials form a basis for all degree-$(n-1)$ polynomials over $\mathbb{F}_p$, enabling structured polynomial construction via evaluation at domain points.

Given a polynomial $f(x)$ evaluated at $\{\omega^i\}$, the vector of values $\big(f(\omega^i)\big)_{i=1}^{n}$ can be compactly represented as:

$$\mathrm{LAG}_x := ([L_i(x)])_{i=1}^{n},$$

which can be efficiently computed using a Fast Fourier Transform (FFT) in $O(n \log n)$ operations. In SNARK systems, these vectors are used to linearly combine CRS elements

Moreover, since the QAP polynomials $u_j(x), v_j(x), w_j(x)$ are typically sparse combinations of a small number of Lagrange basis polynomials, one can efficiently compute expressions like:

$$o_j(X) = \alpha u_j(X) + \alpha^2 v_j(X) + \alpha^3 w_j(X)$$

using linear operations over the group $\mathbb{G}_1$. These operations preserve zero-knowledge and correctness while allowing scalable circuit-specific CRS instantiations.

This Lagrange-based technique ensures that the transformation from universal to specialized CRS remains efficient, algebraically sound, and easily verifiable through bilinear pairings in the final SNARK protocol.

3 Pseudocode for multi-variant MPC Setup

In this section, we present the pseudocode for the MPC setup ceremony of the proposed multi-variant scheme. Our framework focuses on the generation of structured CRS elements involving multiple toxic-waste parameters—most notably the tri-variant case involving monomials of the form $(\alpha^h x^i y^k)$. Throughout the protocol, it is assumed that all participants have synchronized access to a common random oracle $\mathcal{RO}$. This oracle $\mathcal{RO}$ takes as input an arbitrary-length bitstring together with a group element from $\mathbb{G}_1^*$, and deterministically maps them to uniformly distributed elements of $\mathbb{G}_2$.

Internally, the functionality of $\mathcal{RO}$ relies on a secure hash-to-curve procedure that converts hash digests into valid points on $\mathbb{G}_2$. We first present the pseudocode for the random oracle itself, followed by the detailed subroutine responsible for performing the hash-to-$\mathbb{G}_2$ mapping.

The `HashToG2` function is responsible for securely mapping the intermediate digest to a valid $\mathbb{G}_2$ group element. This process incorporates randomness extraction from the digest and uses a seeded random number generator to generate the target group element.

Algorithm 1 Random Oracle $\mathcal{RO}$

function $\mathcal{RO}(A \in \mathbb{G}_1^*,\ v \in \{0,1\}^*)$ ▷ Maps inputs deterministically to $\mathbb{G}_2$
    $digest \leftarrow H(v \,||\, \text{SerializeCompress}(A))$
    $y \leftarrow \text{HashToG2}(digest)$
    return $y \in \mathbb{G}_2$
end function

Algorithm 2 Hash to $\mathbb{G}_2$ (Compressed)

function HashToG2($digest \in \{0,1\}^*$)
    seed $\leftarrow$ First 32 bytes of $digest$
    rng $\leftarrow$ RNG(seed)
    $P \leftarrow \text{RandomPoint}_{\mathbb{G}_2}(\text{rng})$
    $y \leftarrow \text{SerializeCompress}(P)$
    return $y \in \mathbb{G}_2$
end function

We denote by $\text{transcript}_{\ell,i}$ the transcript of the protocol up to the point where player i has sent his message in phase ℓ.

In the following, we introduce a set of core algorithms, namely Algorithm 3, Algorithm 4, Algorithm 5, and Algorithm 6, which define the main computation and verification procedures utilized throughout the protocol. The definitions of these algorithms are based on the framework presented in [2].

Algorithm 3 Construct a proof of knowledge of α

function POK(α, v) ▷ Where α is the input, v is a string
    $y \leftarrow \mathcal{RO}([\alpha]_1, v) \in \mathbb{G}_2^*$ ▷ $[\alpha]_1 := \alpha \cdot G_1$
    return $(\alpha \cdot y)$
end function

Algorithm 4 verifies the validity of the proof of knowledge by checking whether the provided tuple (A, B) satisfies the same ratio relation with respect to the random oracle output y.

Algorithm 4 Verify a proof of knowledge of α

function CheckPOK(A, v, B) ▷ Where $A \in \mathbb{G}_1^*$, $B \in \mathbb{G}_2^*$
    $y \leftarrow \mathcal{RO}(A, v) \in \mathbb{G}_2^*$
    return SameRatio$((G_1, A), (y, B))$
end function

Moreover, the same ratio between two sets of points can be verified using Algorithm-5, while the consistency of the ratio between two different sets of points is controlled by Algorithm-6, which also utilizes Algorithm-5.

Algorithm 5 Determine if $x \in \mathbb{F}_p^*$ exists such that $B = x \cdot A$, and $D = x \cdot C$.

function SameRatio($(A, B), (C, D)$) ▷ Where $A, B \in \mathbb{G}_1$ and $C, D \in \mathbb{G}_2$
    if $e(A, D) = e(B, C)$ **then** ▷ A, B, C, D are not the identity elements.
        return $True$
    else
        return $False$
    end if
end function

We use the notation $Consistent(A - B; C)$ for the below function with inputs A, B, C, where $A, B \in \mathbb{G}_1^2$ or $A, B \in \mathbb{G}^2$ and $C \in \mathbb{G}_2^*$ or $C \in (\mathbb{G}_2^*)^2$

The algorithm **Consistent**$((A - B; C))$ is designed to verify whether the ratio between A and B is encoded in an element $s \in \mathbb{F}_p^*$ that is represented within C. A and B are group elements, either belonging to $\mathbb{G}_1^2$ or $\mathbb{G}^2$ depending on the context. C represents some form of commitment or encoding in either $\mathbb{G}_2^2$ or $(\mathbb{G}_2^*)^2$, indicating that it holds information about the ratio between A and B The algorithm aims

to validate whether the relationship between A and B matches what is encoded in C. The **Consistent** algorithm uses the notation $(A - B; C)$ to emphasize that it is examining the relationship between A and B in light of the information contained in C. It employs a combination of pairing checks and ratio consistency conditions, facilitated by the SameRatio algorithm. The detailed checks ensure that A and B maintain the expected ratio as encoded or committed in C

Algorithm 6 Check whether the ratio between A and B is the $s \in \mathbb{F}_p^*$ that is encoded in C.

function Consistent($(A - B; C)$) ▷ Where $A, B \in \mathbb{G}_1^2$ or $A, B \in \mathbb{G}^2$. And $C \in \mathbb{G}_2^*$ or $C \in (\mathbb{G}_2^*)^2$
  if $C \in (\mathbb{G}_2^*)^2$ **then**
    $r \leftarrow$ **SameRatio**$((A_1, B_1), (C_1, C_2))$
  else
    $r \leftarrow$ **SameRatio**$((A_1, B_1), (G_2, C))$
  end if
  if $A, B \in \mathbb{G}_1$ **then**
    return r
  else
    return r AND **SameRatio**$((A_1, B_1), (A_2, B_2))$
  end if
end function

Using the building blocks described in the previous section (i.e., the Random Oracle $\mathcal{RO}$, proof-of-knowledge algorithms, and consistency checks), we now define a series of parameter generation algorithms that together constitute the full multi-party setup procedure for the new scheme.

In this setup, each participant P_j contributes random secret values at their turn, incrementally updating the public parameters while simultaneously producing proofs that their contributions have been applied correctly. For each parameter type, we define two sub-algorithms: a *compute* algorithm executed by the participant to generate new parameter values, and a corresponding *verify* algorithm used by the protocol verifier (or by any external auditor) to validate the correctness of these contributions.

The computation and verification procedures are categorized into several types depending on the structure of the underlying parameters. Each type corresponds to specific monomial terms (or combinations thereof) that are required for the instantiation of the proving system's Common Reference String (CRS). We describe each type in detail below.

3.1 Type-1: α

The first category handles the generation of the basic α parameters. Each participant introduces their own secret scalar α_j, applies it to the running product of previous α values, and generates a proof of knowledge to ensure correctness. Let $[\alpha]^{\mathbf{0}} = G_1$ or $[\alpha]^{\mathbf{0}} = \mathbf{G}$.

Algorithm 3.1.1 The participant P_j computes all parameters for Type-1 parameter $([\alpha]_1 := \alpha \cdot G_1)$

function Compute1($[\alpha]^{\mathbf{j-1}}, \mathrm{v}_{rd,j-1}$) ▷ $\mathrm{v}_{rd,j-1} :=$ transcript$_{1,j-1}$, where $rd \in \{1, 2\}$
  $\alpha_j \in_R \mathbb{F}_p^*$ ▷ pick α random number
  $p_{\alpha_j} = \mathbf{POK}(\alpha_j, \mathrm{v}_{rd,j-1})$
  $[\alpha]^{\mathbf{j}} = \alpha_j \cdot [\alpha]^{\mathbf{j-1}}$
  return $([\alpha]^{\mathbf{j}}, [\alpha_j]_1, p_{\alpha_j})$
end function

The corresponding verification algorithm checks the submitted proof of knowledge and verifies that the updated parameter has been correctly computed via consistency checks involving the previous state.

3.2 Type-2: x^i

The second category introduces a family of parameters associated with powers of x. Here, each participant samples a fresh random scalar x_j. Again, proofs of knowledge are generated for x_j to ensure honest behavior. Let $[x^i]_1^{\mathbf{0}} = G_1$ or $[x^i]^{\mathbf{0}} = \mathbf{G}$

Verification proceeds similarly: the submitted proof of knowledge for x_j is first validated, after which the updated parameter values are checked for consistency with the prior state and the public transcript.

Algorithm 3.1.2 The protocol verifier verifies for each $j \in [N]$ for a single parameter ($[\alpha]_1 := \alpha \cdot G_1$))

function VERIFY1($[\alpha]^{\mathbf{j-1}}, [\alpha]^{\mathbf{j}}, [\alpha_j]_1, \mathrm{v}_{rd,j-1}, p_{\alpha_j}$) ▷ $\mathrm{v}_{rd,j-1} := \mathrm{transcript}_{1,j-1}$, where $rd \in \{1,2\}$
  if (**CheckPOK**($[\alpha_\mathrm{j}]_1, \mathrm{v}_{rd,j-1}, p_{\alpha_j}$)) **then**
    $r_{\alpha_j} = \mathcal{RO}([\alpha_j]_1, \mathrm{v}_{rd,j-1})$
    return Consistent$\big([\alpha]^{\mathbf{j-1}} - [\alpha]^{\mathbf{j}}; (r_{\alpha_j}, p_{\alpha_j})\big)$
  else
    return $False$
  end if
end function

Algorithm 3.2.1 The P_j computes parameters for (Type-2: $[x^i]_1 := x^i G_1$, where $i \in [1, ..., n]$)

function COMPUTE2($[x^i]_1^{\mathbf{j-1}}, [x]_2^{\mathbf{j-1}}, \mathrm{v}_{rd,j-1}$) ▷ $\mathrm{v}_{rd,j-1} := \mathrm{transcript}_{1,j-1}$, where $rd \in \{1,2\}$
  $x_j \in_R \mathbb{F}_p^*$ ▷ pick random numbers
  $p_{x_j} = \mathbf{POK}(x_j, \mathrm{v}_{rd,j-1})$
  $[x]_2^{\mathbf{j}} = x_j \cdot [x]_2^{\mathbf{j-1}}$
  $[x^i]_1^{\mathbf{j}} = x_j^i \cdot [x^i]_1^{\mathbf{j-1}}$, for each $i \in [1, ..., n]$
  return $([x^i]_1^{\mathbf{j}}, [x]_2^{\mathbf{j}}, [x_j]_1, p_{x_j})$, for each $i \in [1, ..., n]$
end function

Algorithm 3.2.2 Verification for each $j \in [N]$ for a power of parameter
(Type-2: $[x^i]_1 := x^i G_1$, where $i \in [1, ..., n]$)

function VERIFY2($[x^i]_1^{\mathbf{j-1}}, [x^i]_1^{\mathbf{j}}, [x]_2^{\mathbf{j-1}}, [x]_2^{\mathbf{j}}, [x_j]_1, \mathrm{v}_{rd,j-1}, p_{x_j}$)
  if CheckPOK($[\mathrm{x_j}]_1, \mathrm{v}_{rd,j-1}, p_{x_j}$) **then**
    $r_{x_j} = \mathcal{RO}([x_j]_1, \mathrm{v}_{rd,j-1})$
    if Consistent$\big([x]^{\mathbf{j-1}} - [x]^{\mathbf{j}}; (r_{x_j}, p_{x_j})\big)$ **then**
      return Consistent$\big([x^{i-1}]_1^{\mathbf{j}} - [x^i]_1^{\mathbf{j}}; [x]_2^{\mathbf{j}}\big)$ ▷ for each $i \in [1, ..., n]$
    else
      return False
    end if
  end if
end function

3.3 Type-3: $x^i y^k$

The third category extends parameter generation to bilinear cross-terms of the form $x^i y^k$. In this step, each participant samples fresh secrets for both x_j and y_j simultaneously, and generates separate proofs of knowledge for each. These values are then incorporated into both x-related and y-related parameters before updating the cross-term $x^i y^k$ accordingly. Let $[x^i]^{\mathbf{0}} = G_1$ and $[x^i y^k]^{\mathbf{0}} = G_1$

Algorithm 3.3.1 The P_j computes Type-3 parameter s.t. ($x^i y^k$, where $i \in [1, ..., n]$, $k \in [1, ..., m]$)

function COMPUTE3($[x^i y^k]_1^{\mathbf{j-1}}, [x^i]^{\mathbf{j-1}}, [y^k]_1^{\mathbf{j-1}}, [y]_2^{\mathbf{j-1}}, \mathrm{v}_{rd,j-1}$)
  $x_j, y_j \in_R \mathbb{F}_p^*$ ▷ pick random numbers
  $p_{x_j} = \mathbf{POK}(x_j, \mathrm{v}_{rd,j-1})$
  $p_{y_j} = \mathbf{POK}(y_j, \mathrm{v}_{rd,j-1})$
  $[x^i]^{\mathbf{j}} = x_j \cdot [x^i]^{\mathbf{j-1}}$, $i \in [1, ..., n]$
  $[y]_2^{\mathbf{j}} = y_j \cdot [y]_2^{\mathbf{j-1}}$
  $[y^k]_1^{\mathbf{j}} = y_j \cdot [y^k]_1^{\mathbf{j-1}}$, $k \in [2, ..., m]$
  $[x^i y^k]_1^{\mathbf{j}} = x_j^i y_j^k \cdot [x^i y^k]_1^{\mathbf{j-1}}$, $i \in [1, ..., n]$, $k \in [1, ..., m]$
  return $([x^i y^k]_1^{\mathbf{j}}, [x^i]^{\mathbf{j}}, [y]_2^{\mathbf{j}}, [y^k]_1^{\mathbf{j}}, [x_j]_1, [y_j]_1, p_{x_j}, p_{y_j})$, $i \in [1, ..., n]$, $k \in [1, ..., m]$
end function

The verification logic ensures correctness of both random contributions, as well as their proper multiplication into each subparameter. This includes multiple consistency checks for the powers of x, y, and their joint product $x^i y^k$.

Algorithm 3.3.2 Verification for each $j \in [N]$ for a power of parameter Type-3 s.t. $(x^i y^k$, where $i \in [1, ..., n]$, $k \in [1, ..., m]))$

Let $in_3 = [x^i y^k]_1^{\mathbf{j}}, [x^i y^k]_1^{\mathbf{j-1}}, [x^i y^k]_2^{\mathbf{j}}, [x^i]_1^{\mathbf{j}}, [x^i]_1^{\mathbf{j-1}}, [y^k]_1^{\mathbf{j}}, [y^k]_1^{\mathbf{j-1}}, [x]_2^{\mathbf{j}}, [y]_2^{\mathbf{j}}, [x_j]_1, [y_j]_1, \mathrm{v}_{rd,j-1}, p_{x_j}, p_{y_j}$

function VERIFY3(in_3)
  if not CheckPOK$([x_j]_1, \mathrm{v}_{rd,j-1}, p_{x_j})$ **then**
    return False
  end if
  if not CheckPOK$([y_j]_1, \mathrm{v}_{rd,j-1}, p_{y_j})$ **then**
    return False
  end if
  $r_{x_j} = \mathcal{RO}([x_j]_1, \mathrm{v}_{rd,j-1})$,
  if not Consistent$([x]^{\mathbf{j-1}} - [x]^{\mathbf{j}}; (r_{x_j}, p_{x_j}))$ **then**
    return False
  end if
  if not Consistent$([x^{i-1}]_1^{\mathbf{j}} - [x^i]_1^{\mathbf{j}}; [x]_2^{\mathbf{j}})$ **then** ▷ for each $i \in [1, ..., n]$
    return False
  end if
  $r_{y_j} = \mathcal{RO}([y_j]_1, \mathrm{v}_{rd,j-1})$
  if not Consistent$([y]^{\mathbf{j-1}} - [y]^{\mathbf{j}}; (r_{y_j}, p_{y_j}))$ **then**
    return False
  end if
  if not Consistent$([y^{k-1}]_1^{\mathbf{j}} - [y^k]_1^{\mathbf{j}}; [y]_2^{\mathbf{j}})$ **then** ▷ for each $k \in [1, ..., m]$
    return False
  end if
  if not Consistent$([y^k]_1^{\mathbf{j}} - [x^i y^k]_1^{\mathbf{j}}; [x^i]_2^{\mathbf{j}})$ **then** ▷ for each $i \in [1, ..., n]$, $k \in [1, ..., m]$
    return False
  else
    return True
  end if
end function

3.4 Type-4: αx^i

The fourth category considers mixed terms where α and x^i appear multiplicatively. Again, fresh randomness is introduced for both components, and corresponding proofs of knowledge are generated. Let $[x^i]^{\mathbf{0}} = G_1$ or $[x^i]^{\mathbf{0}} = \mathbf{G}$ Let $[\alpha x^i]^{\mathbf{0}} = G_1$ or $[\alpha x^i]^{\mathbf{0}} = \mathbf{G}$

Algorithm 3.4.1 The P_j computes Type-4 parameter s.t. $([\alpha x^i]_1 := \alpha x^i G_1$, where $i \in [1, ..., n])$

function COMPUTE4($[\alpha x^i]^{\mathbf{j-1}}, \mathrm{v}_{rd,j-1}$) ▷ $\mathrm{v}_{rd,j-1} := \mathrm{transcript}_{1,j-1}$, where $rd \in \{1, 2\}$
  $\alpha_j, x_j \in_R \mathbb{F}_p^*$ ▷ pick random numbers
  $p_{\alpha_j} = \mathbf{POK}(\alpha_j, \mathrm{v}_{rd,j-1})$
  $p_{x_j} = \mathbf{POK}(x_j, \mathrm{v}_{rd,j-1})$
  $[x^i]^{\mathbf{j}} = x_j^i \cdot [x^i]^{\mathbf{j-1}}$,
  $[\alpha x^i]^{\mathbf{j}} = \alpha_j x_j^i \cdot [\alpha x^i]^{\mathbf{j-1}}$, $i \in [1, ..., n]$
  return $([\alpha x^i]^{\mathbf{j}}, [x^i]^{\mathbf{j}}, [\alpha]^{\mathbf{j}}, [\alpha_j]_1, p_{\alpha_j}, p_{x_j})$, $i \in [1, ..., n]$
end function

The verification ensures that both the x^i progression and the α contribution are consistent with previous computations and the provided proofs of knowledge.

3.5 Type-5: $\alpha^h x^i y^k$

The fifth category generates higher-degree cross-terms involving powers of α, x, and y simultaneously. These terms are essential for encoding more complex constraints during the proving system's circuit instantiation. Let $[x]^{\mathbf{0}} = G_1$ or $[x]^{\mathbf{0}} = \mathbf{G}$, $[x^i]^{\mathbf{0}} = G_1$ or $[x^i]^{\mathbf{0}} = \mathbf{G}$

Let $[y^k x^i]^{\mathbf{0}} = G_1$ or $[y^k x^i]^{\mathbf{0}} = \mathbf{G}$

Let $[\alpha^h x^i y^k]^{\mathbf{0}} = G_1$ or $[\alpha^h x^i y^k]^{\mathbf{0}} = \mathbf{G}$

Algorithm 3.4.2 Verification for each $j \in [N]$ for a power of parameter Type-4 s.t. ($[\alpha x^i]_1 := \alpha x^i G_1$, where $i \in [1, ..., n]$))

function VERIFY4($[\alpha x^i]^{\mathbf{j}}, [x^i]^{\mathbf{j}}, [\alpha]^{\mathbf{j}}, [\alpha_j]_1, p_{\alpha_j}, p_{x_j}$)
  $r_\alpha = \mathcal{RO}([\alpha_j]_1, \mathrm{v}_{rd,j-1})$
  if CheckPOK$([\alpha_j]_1, \mathrm{v}_{rd,j-1}, p_{\alpha_j})$ **then**
    if Consistent$\big([x^{i-1}]^{\mathbf{j}} - [x^i]^{\mathbf{j}}; [x]^{\mathbf{j}}\big)$ **then** ▷ for each $i \in [1, ..., n]$
      return Consistent$\big([x^i]^{\mathbf{j}} - [\alpha x^i]^{\mathbf{j}}; [\alpha]^{\mathbf{j}}\big)$
    end if
  end if
end function

Algorithm 3.5.1 The P_j computes Type-5 parameter ($\alpha^h x^i y^k$)

function COMPUTE5($[\alpha^h x^i y^k]_1^{\mathbf{j-1}}, [\alpha^h]_1^{\mathbf{j-1}}, [x^i]_1^{\mathbf{j-1}}, [y^k]_1^{\mathbf{j-1}}, [x]_2^{\mathbf{j-1}}, [y]_2^{\mathbf{j-1}}, \mathrm{v}_{rd,j-1}$)
  Pick random $\alpha_j, x_j, y_j \in_R \mathbb{F}_p^*$
  $p_{\alpha_j} = \mathbf{POK}(\alpha_j, \mathrm{v}_{rd,j-1})$
  $p_{x_j} = \mathbf{POK}(x_j, \mathrm{v}_{rd,j-1})$
  $p_{y_j} = \mathbf{POK}(y_j, \mathrm{v}_{rd,j-1})$
  $[\alpha^h]_1^{\mathbf{j}} = \alpha_j \cdot [\alpha^h]_1^{\mathbf{j-1}}$, for $h \in [1, \ldots, 4]$
  $[\alpha^h]_2^{\mathbf{j}} = \alpha_j \cdot [\alpha^h]_2^{\mathbf{j-1}}$, for $h \in [1, \ldots, 4]$
  $[x^i]_1^{\mathbf{j}} = x_j \cdot [x^i]_1^{\mathbf{j-1}}$, for $i \in [1, \ldots, n]$
  $[y^k]_1^{\mathbf{j}} = y_j \cdot [y^k]_1^{\mathbf{j-1}}$, for $k \in [1, \ldots, m]$
  $[x]_2^{\mathbf{j}} = x_j \cdot [x]_2^{\mathbf{j-1}}$
  $[y]_2^{\mathbf{j}} = y_j \cdot [y]_2^{\mathbf{j-1}}$
  $[\alpha^h x^i y^k]_1^{\mathbf{j}} = (\alpha_j^h x_j^i y_j^k) \cdot [\alpha^h x^i y^k]_1^{\mathbf{j-1}}$,
   for all $h \in [1, \ldots, 4]$, $i \in [1, \ldots, n]$, $k \in [1, \ldots, m]$
  return $([\alpha^h x^i y^k]_1^{\mathbf{j}}, [\alpha^h]_1^{\mathbf{j}}, [\alpha^h]_2^{\mathbf{j}}, [x^i]_1^{\mathbf{j}}, [y^k]_1^{\mathbf{j}}, [x]_2^{\mathbf{j}}, [y]_2^{\mathbf{j}}, [\alpha_j]_1, [x_j]_1, [y_j]_1, p_{\alpha_j}, p_{x_j}, p_{y_j})$
end function

The verification stage reuses previous verification subroutines (e.g., `Verify2` and `Verify3`) in a modular fashion, before performing a final consistency check for the combined term.

Let $in_5 := ([\alpha^h x^i y^k]^{\mathbf{j}}, [\alpha^h x^i y^k]_1^{\mathbf{j-1}}, [x^i y^k]_1^{\mathbf{j}}, [y^k]_1^{\mathbf{j}}, [\alpha^h]_1^{\mathbf{j}}, [x^i]_1^{\mathbf{j}}, [x]_2^{\mathbf{j}}, [\alpha_j]_1, [y_j]_1, p_{\alpha_j}, p_{y_j}, \mathrm{v}_{rd,j-1})$, $h \in [1, ..., 4]$, $i \in [1, ..., n]$, $k \in [1, ..., m]$.

Algorithm 3.5.2 Verification for each $j \in [N]$ for a power of parameter Type-5

function VERIFY5(in_5)
Let $in_3 = [x^i y^k]_1^{\mathbf{j}}, [x^i y^k]_1^{\mathbf{j-1}}, [x^i y^k]_2^{\mathbf{j}}, [x^i]^{\mathbf{j}}, [x^i]^{\mathbf{j-1}}, [y^k]_1^{\mathbf{j}}, [y^k]_1^{\mathbf{j-1}}, [y]_2^{\mathbf{j-1}}, [y]_2^{\mathbf{j}}, [x_j]_1, [y_j]_1, \mathrm{v}_{rd,j-1}, p_{x_j}, p_{y_j}$
  if not Verify2$([\alpha^h]_1^{\mathbf{j-1}}, [\alpha^h]_1^{\mathbf{j}}, [\alpha]_2^{\mathbf{j-1}}, [\alpha]_2^{\mathbf{j}}, [\alpha_j]_1, p_{\alpha_j})$ **then** ▷ for each $h \in [1, ..., 4]$
    return False
  end if
  if not Verify3(in_3) **then** ▷ for each $i \in [1, ..., n]$, $k \in [1, ..., m]$
    return False
  end if
  if not Consistent$\big([x^i y^k]_1^{\mathbf{j}} - [\alpha^h x^i y^k]_1^{\mathbf{j}}; [\alpha^h]_2^{\mathbf{j}}\big)$ **then** ▷ for each $h \in [1, 4]$, $i \in [1, n]$, $k \in [1, m]$
    return False
  else
    return True
  end if
end function

4 The MPC Setup Ceremony for Multivariate SNARKs: The Tokamak zk-SNARK Use Case

In this section, we apply our general multivariate MPC setup ceremony to the Tokamak zk-SNARK, which itself requires the three-variant parameter structure. The Tokamak scheme, therefore, serves as a natural and fully compatible use case, and our ceremony instantiates its structured CRS requirements exactly and efficiently. We will cover the roles of participants, the steps involved in the generation of parameters, and the security guaranties provided by the ceremony. The goal of the MPC setup is to ensure that the final parameters are free from bias or manipulation, making them secure for use in zero-knowledge proofs on the Tokamak platform.

In this section, we first provide a brief overview of the scheme and then highlight key differences between its setup phase and that of Groth's (see Section A).

4.1 Overview of the Tokamak zk-SNARK Scheme

The MPC setup ceremony generates a structured Common Reference String (CRS) in two phases. In the first phase (Phase-1), the following parameters are securely generated by randomly choosing toxic parameters from the field $\mathbb{F}^*$.

The Tokamak zk-SNARK paper [7] defines a probabilistic algorithm $Setup(pp_\lambda, \mathcal{L}) \mapsto (\boldsymbol{\tau}, \boldsymbol{\sigma})$ to generate an encoded reference string $\boldsymbol{\sigma}$ of the library subcircuit polynomials in $\mathcal{L}$. A simulation trapdoor is defined in [7] as follows:

$$\boldsymbol{\tau} := (\alpha, \gamma, \delta, \eta, x, y) \in (\mathbb{F}^*)^6$$

This MPC ceremony produces an encoded reference string $\boldsymbol{\sigma}$ of the library subcircuit polynomials in $\mathcal{L}$:

$$\boldsymbol{\sigma} = ([\boldsymbol{\sigma}_{A,C}]_1, [\boldsymbol{\sigma}_B]_1, [\boldsymbol{\sigma}_V]_2)$$

Each of these components is composed of elliptic curve points where $[\sigma_{A,C}]_1, [\sigma_B]_1$ is in $\mathbb{G}_1$ and $[\sigma_V]_2$ in $\mathbb{G}_2$.

Component $[\sigma_{A,C}]_1$

This component consists of elliptic curve encodings of monomials in two variables, x and y. It provides a structured polynomial basis for encoding witness polynomials succinctly:

$$\sigma_{A,C} := \left\{x^h y^i\right\}_{h=0,i=0}^{2\max(n,\, m_I),\, 2s_{\max}}$$

Component $[\sigma_B]_1$

The component σ_B encodes polynomial relationships and permutations essential for verifying internal consistency within the proof:

$$\sigma_B := \left\{ \begin{array}{ll} \delta,\ \eta, & \\ \{\gamma^{-1}(L_1(y)o_j(x) + M_j(x))\}_{j=0}^{l_{pub}^{(out)}-1}, & \text{for public buffer output wires,} \\ \{\gamma^{-1}(L_0(y)o_j(x) + M_j(x))\}_{j=l_{pub}^{(out)}}^{l_{pub}-1}, & \text{for public buffer input wires,} \\ \{\gamma^{-1}(L_3(y)o_j(x))\}_{l_{pub}}^{l_{pub}+l_{prv}^{(out)}-1}, & \text{for private buffer output wires,} \\ \{\gamma^{-1}(L_2(y)o_j(x))\}_{j=l_{pub}+l_{prv}^{(out)}}^{l-1}, & \text{for private buffer input wires,} \\ \{\eta^{-1}L_i(y)(o_{j+l}(x) + \alpha^4 K_j(x))\}_{i=0,j=0}^{s_{\max}-1,m_I-1}, & \text{for interface wires,} \\ \{\delta^{-1}L_i(y)o_j(x)\}_{i=0,j=l+m_I}^{s_{\max}-1,m_D-1}, & \text{for private wires,} \\ \{\delta^{-1}\alpha^k x^h t_n(x)\}_{h=0,k=1}^{2,3}, & \text{for vanishing polynomial } t_n(x), \\ \{\delta^{-1}\alpha^4 x^j t_{m_I}(x)\}_{j=0}^{1}, & \text{for vanishing polynomial } t_{m_I}(x), \\ \{\delta^{-1}\alpha^k y^i t_{s_{\max}}(y)\}_{i=0,k=1}^{2,4} & \text{for vanishing polynomial } t_{s_{\max}}(y) \end{array} \right\}$$

Here:

- $\{L_i(Y)\}_{i=0}^{s_{max}-1}, \{M_j(X)\}_{j=0}^{l_{pub}-1}, \{K_j(X)\}_{j=0}^{m_I-1}$ are Lagrange bases.
- Vanishing polynomials $t_n(X) := X^n - 1, t_{m_I}(X) := X^{m_I} - 1, t_{s_{\max}}(Y) := Y^{s_{max}} - 1$ enforce polynomial constraints over predefined evaluation sets.

Component $[\sigma_V]_2$

The component σ_V directly encodes the toxic parameters and their exponents into elliptic curve points of group G_2. This allows the verifier to efficiently verify the correctness of the polynomial relationships presented by the prover without revealing or knowing the secret parameters themselves:

$$\sigma_V := \{\alpha, \alpha^2, \alpha^3, \alpha^4, \gamma, \delta, \eta, x, y\}$$

For each subcircuit indexed by $j \in \{0, \ldots, m_D - 1\}$, a polynomial $o_j(X)$ is constructed from the Rank-1 Constraint System (R1CS) polynomials $u_j(X), v_j(X), w_j(X)$:

$$o_j(X) = \alpha u_j(X) + \alpha^2 v_j(X) + \alpha^3 w_j(X)$$

These polynomials encode the arithmetic constraints defined by the subcircuits.

4.2 Overview of the MPC Ceremony Flow

The setup ceremony consists of two distinct phases, each conducted within a MPC framework to securely generate a CSR. Figure 1 illustrates the general flow of the protocol, including participant interactions, proof submissions, and optional beacon contributions.

Phase 1: Universal Setup. In the first phase, the ceremony generates universal parameters that are independent of any specific circuit. The process begins with an *initialization step*, after which each participant contributes sequentially:

- A **challenge** is derived from the previous response (i.e., the accumulator and its associated proof).
- Each participant executes the computation protocol, producing a new **accumulator** and a **proof of correct contribution**.
- These are recorded as the participant's **response**.

- Optionally, a **random beacon** may be used after all contributions to inject fresh, unbiased randomness into the final parameters.
- A final **batch verification** is conducted to verify all participant responses before advancing to the next phase.

Upon completion, Phase 1 outputs a *combined sigma*, which encodes all universal parameters necessary for the second phase.

Intermediate Preparation. Before entering Phase 2, a preparation step maps the outputs of Phase 1 into the initial form required for circuit-specific parameter generation. This step ensures compatibility between universal parameters and the structure of the target arithmetic circuit.

Phase 2: Circuit-Specific Setup. Phase 2 focuses on generating parameters that are specific to the circuit being instantiated. The same MPC pattern is followed:

- Each participant receives a **challenge**, defined as the response of the previous participant.
- New accumulator values and corresponding proofs are generated and recorded.
- A **random beacon** may again be used to enhance trust in the randomness used.
- A **batch verification** is executed to ensure correctness across all contributions.

The final output of Phase 2 is the **CRS (Common Reference String)**, which contains the latest combined parameters securely contributed by all participants. This CRS can now be used for zk-SNARK proof generation and verification within the Tokamak protocol.

4.3 MPC-Based Setup Ceremony Protocol Description

Building upon the high-level overview of the parameter generation process, we now present a detailed description of the MPC protocol designed for the Tokamak scheme. This section elaborates on the step-by-step procedures, phases, and cryptographic foundations involved in securely generating the Common Reference String (CRS) via multi-party computation.

In this section, we introduce the Multi-Party Computation (MPC) protocol tailored for the Tokamak scheme [7] using Random Beacon based on the paper [2]. We present the MPC protocol for the Tokamak zk-SNARK in two phases: the initial round, known as "Powers of τ", which generates universal setup parameters applicable to all circuits within the scheme, and the subsequent phase that generates circuit-specific parameters. This protocol ensures the secure and decentralized generation of the common reference string (CRS) elements required for the Tokamak zk-SNARK, thereby enhancing the overall security and integrity of the setup process.

4.3.1 First Round: Powers of τ

In this section, we generate the $\sigma_{A,C}$ parameters along with a subset of the σ_V parameter set. These parameters are independent of any specific circuit. In this phase, the parameters are first initialized using elliptic curve base points. After initialization, the computation and verification steps are executed together to securely generate the following parameters via MPC.

Initialization:

Let $x_{deg} = 2max(\mathbf{n}, m_I)$ and $y_{deg} = 2s_{max}$.

1. $\left[\alpha^h\right]_1^{\mathbf{0}} := G_1, \quad \left[\alpha^h\right]_2^{\mathbf{0}} := G_2,$ ▷ for each $h \in [1, ..., 4]$
2. $\left[x^i\right]_1^{\mathbf{0}} := G_1, \quad \left[y^k\right]_1^{\mathbf{0}} := G_1,$ ▷ for each $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$
3. $[x]_2^{\mathbf{0}} := G_2, \quad [y]_2^{\mathbf{0}} := G_2,$
4. $[\alpha^h x^i y^k]_1^{\mathbf{0}} := G_1$ ▷ for each $h \in [1, 4]$, $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$
5. $[\alpha x^i]_1^{\mathbf{0}} := G_1, \quad [\alpha y^k]_1^{\mathbf{0}} := G_1,$ ▷ for each $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$
6. $[x^i y^k]_1^{\mathbf{0}} := G_1, \quad [x^i y^k]_2^{\mathbf{0}} := G_2$ ▷ for each $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$

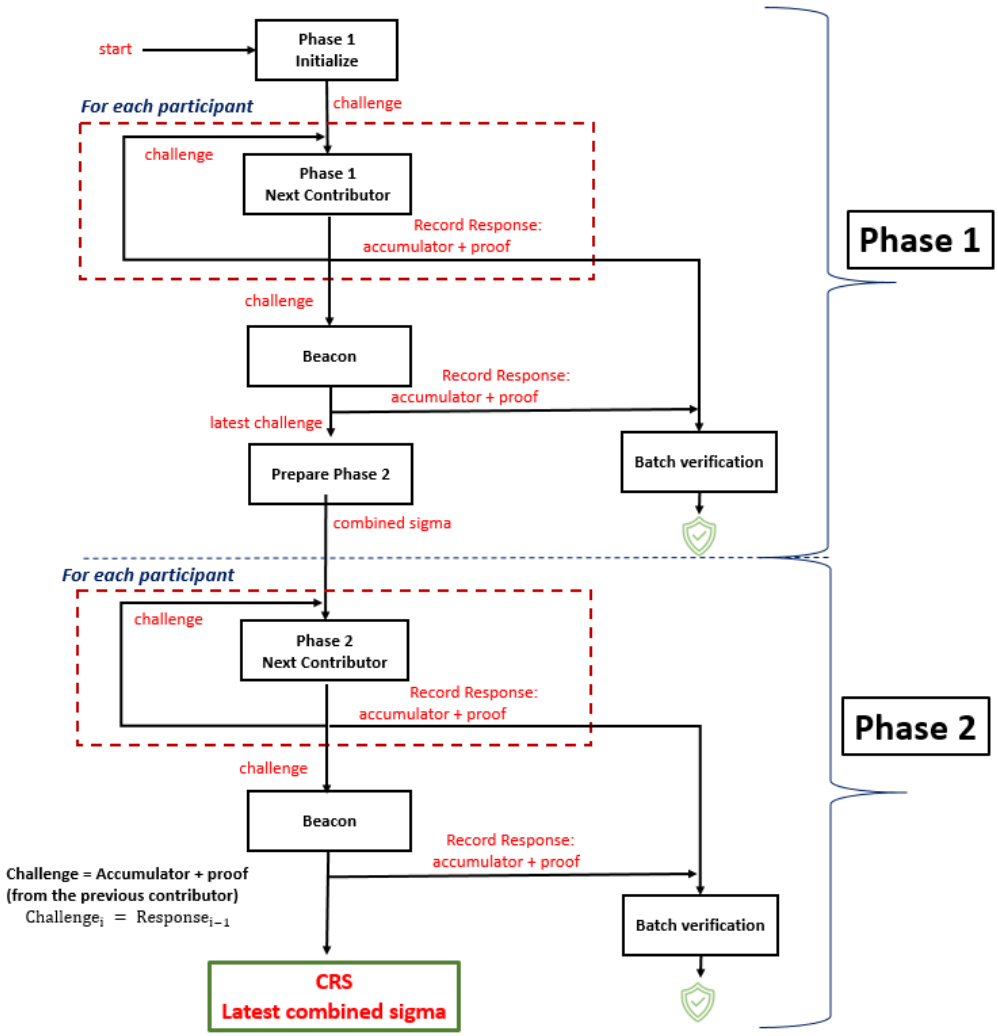

Figure 1: General flow diagram of the two-phase MPC ceremony.

Computations:

The participant P_j, performs the below computations, where $j \in [1, N]$, $x_{deg} = 2max(\mathbf{n}, m_I)$ and $y_{deg} = 2s_{max}$:

1. Pick random $\alpha_j, x_j, y_j \in_R \mathbb{F}_p^*$
2. $[\alpha_j]_1 = \alpha_j \cdot G_1$,
3. $[x_j]_1 = x_j \cdot G_1$,
4. $[y_j]_1 = y_j \cdot G_1$,
5. $p_{\alpha_j} = \mathbf{POK}(\alpha_j, \mathrm{v}_{1,j-1})$
6. $p_{x_j} = \mathbf{POK}(x_j, \mathrm{v}_{1,j-1})$
7. $p_{y_j} = \mathbf{POK}(y_j, \mathrm{v}_{1,j-1})$
8. $[\alpha^h]_1^{\mathbf{j}} = \alpha_j \cdot [\alpha^h]_1^{\mathbf{j-1}}$, ▷ for each $h \in [1, \ldots, 4]$
9. $[\alpha^h]_2^{\mathbf{j}} = \alpha_j \cdot [\alpha^h]_2^{\mathbf{j-1}}$, ▷ for each $h \in [1, \ldots, 4]$
10. $[x^i]_1^{\mathbf{j}} = x_j \cdot [x^i]_1^{\mathbf{j-1}}$, ▷ for each $i \in [1, \ldots, x_{deg}]$

11. $[y^k]_1^{\mathbf{j}} = y_j \cdot [y^k]_1^{\mathbf{j-1}}$, ▷ for each $k \in [1, \ldots, y_{deg}]$
12. $[x]_2^{\mathbf{j}} = x_j \cdot [x]_2^{\mathbf{j-1}}$
13. $[y]_2^{\mathbf{j}} = y_j \cdot [y]_2^{\mathbf{j-1}}$
14. $[\alpha^h x^i y^k]_1^{\mathbf{j}} = (\alpha_j^h x_j^i y_j^k) \cdot [\alpha^h x^i y^k]_1^{\mathbf{j-1}}$, ▷ for each $h \in [1, 4]$, $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$
15. $[\alpha x^i]_1^{\mathbf{j}} = \alpha_j x_j^i \cdot [\alpha x^i]_1^{\mathbf{j-1}}$, ▷ for each $i \in [1, x_{deg}]$
16. $[\alpha y^k]_1^{\mathbf{j}} = \alpha_j y_j^k \cdot [\alpha y^k]_1^{\mathbf{j-1}}$, ▷ for each $k \in [1, y_{deg}]$
17. $[x^i y^k]_1^{\mathbf{j}} = x_j^i y_j^k \cdot [x^i y^k]_1^{\mathbf{j-1}}$, ▷ for each $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$
18. $[x^i y^k]_2^{\mathbf{j}} = x_j^i y_j^k \cdot [x^i y^k]_2^{\mathbf{j-1}}$, ▷ for each $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$
19. This step is optional. In this phase, if desired, the randomness generated by the random beacon can be incorporated into the parameter generation process. After the final participant has submitted their contribution, this step may optionally be executed as follows:

    The procedures described from Step-8 to Step-18 (inclusive) are repeated using the newly generated randomness (via the random beacon) for the parameters x', y', α'.

    Note that $(x', y', \alpha') := \text{RandomBeacon}(J, 3)$.

Verifications:

The protocol verifier runs the following algorithms for each $j \in [N]$, $x_{deg} = 2max(\mathbf{n}, m_I)$ and $y_{deg} = 2s_{max}$:

1. **Verify2**$([\alpha^h]_1^{\mathbf{j-1}}, [\alpha^h]_1^{\mathbf{j}}, [\alpha]_2^{\mathbf{j-1}}, [\alpha]_2^{\mathbf{j}}, [\alpha_j]_1, \mathrm{v}_{1,j-1}, p_{\alpha_j})$ ▷ for each $h \in [1, ..., 4]$
2. **Verify3**$([x^i y^k]_1^{\mathbf{j}}, [x^i y^k]_1^{\mathbf{j-1}}, [x^i y^k]_2^{\mathbf{j}}, [x^i]_1^{\mathbf{j}}, [x^i]_1^{\mathbf{j-1}}, [y^k]_1^{\mathbf{j}}, [y^k]_1^{\mathbf{j-1}}, [x]_2^{\mathbf{j}}, [y]_2^{\mathbf{j}}, [x_j]_1, [y_j]_1, \mathrm{v}_{1,j-1}, p_{x_j}, p_{y_j})$ ▷ for each $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$
3. **Consistent**$\big([x^i y^k]_1^{\mathbf{j}} - [\alpha^h x^i y^k]_1^{\mathbf{j}}; [\alpha^h]_2^{\mathbf{j}}\big)$ ▷ for each $h \in [1, 4]$, $i \in [1, x_{deg}]$, $k \in [1, y_{deg}]$
4. **Consistent**$\big([x^i]_1^{\mathbf{j}} - [\alpha x^i]_1^{\mathbf{j}}; [\alpha]_2^{\mathbf{j}}\big)$ ▷ for each $i \in [1, x_{deg}]$
5. **Consistent**$\big([y^k]_1^{\mathbf{j}} - [\alpha y^k]_1^{\mathbf{j}}; [\alpha]_2^{\mathbf{j}}\big)$ ▷ for each $k \in [1, y_{deg}]$

4.3.2 Preparation: Before the Second Round

Before the second phase of the ceremony begins, it is useful to clarify a few points, especially regarding how linear combinations from the first phase relate to the second. To illustrate this connection, we present a concrete example by expanding one of the parameter constructions in the initialization of the second phase.

Let us consider the definition of the polynomial $o_j(x)$, which is a linear combination of three polynomials generated during the first phase:

$$o_j(x) = \alpha \cdot u_j(x) + \alpha^2 v_j(x) + \alpha^3 w_j(x)$$

This expression can be substituted into the following second-phase construction:

$$\gamma^{-1} \left[L_1(y) \cdot o_j(x) + M_j(x) \right]$$

Expanding this by substituting the definition of $o_j(x)$ we get:

$$\gamma^{-1} \left[L_1(y) \cdot \big(\alpha \cdot u_j(x) + \alpha^2 v_j(x) + \alpha^3 w_j(x) \big) + M_j(x) \right]$$

Distributing $L_1(y)$ over the sum gives us the fully expanded expression:

$$\gamma^{-1} \left[\alpha \cdot u_j(x) \cdot L_1(y) + \alpha^2 v_j(x) \cdot L_1(y) + \alpha^3 w_j(x) \cdot L_1(y) + M_j(x) \right]$$

Accordingly, the first parameter of the second phase, B_j^1 for $j \in \left[0,\ l_{\text{pub}}^{(\text{out})} - 1\right]$, can be explicitly defined as:

$$B_j^1 := \frac{\alpha \cdot u_j(x) \cdot L_1(y) + \alpha^2 v_j(x) \cdot L_1(y) + \alpha^3 w_j(x) \cdot L_1(y) + M_j(x)}{\gamma} \cdot\ G_1$$

Following the same approach, the remaining B-type parameters generated during the second phase can also be expressed in expanded form. This highlights how the random parameters produced in the first phase are linearly combined with public polynomials to construct new structured elements in the second phase. The full set of second-phase parameters are expressed below:

$$B_j^2 := \frac{\alpha \cdot u_j(x) \cdot L_0(y) + \alpha^2 v_j(x) \cdot L_0(y) + \alpha^3 w_j(x) \cdot L_0(y) + M_j(x)}{\gamma} \cdot\ G_1 \qquad j \in \left[l_{\text{pub}}^{(\text{out})},\ l_{\text{pub}} - 1\right]$$

$$B_j^3 := \frac{\alpha \cdot u_j(x) \cdot L_3(y) + \alpha^2 v_j(x) \cdot L_3(y) + \alpha^3 w_j(x) \cdot L_3(y)}{\gamma} \cdot\ G_1 \qquad j \in \left[l_{\text{pub}},\ l_{\text{pub}} + l_{\text{prv}}^{(\text{out})} - 1\right]$$

$$B_j^4 := \frac{\alpha \cdot u_j(x) \cdot L_2(y) + \alpha^2 v_j(x) \cdot L_2(y) + \alpha^3 w_j(x) \cdot L_2(y)}{\gamma} \cdot\ G_1 \qquad j \in \left[l_{\text{pub}} + l_{\text{prv}}^{(\text{out})},\ l - 1\right]$$

$$B_{i,j}^5 := \frac{\alpha \cdot u_{j+l}(x) \cdot L_i(y) + \alpha^2 v_{j+l}(x) \cdot L_i(y) + \alpha^3 w_{j+l}(x) \cdot L_i(y) + \alpha^4 K_j(x)}{\eta} \cdot G_1 \quad i \in [0, s_{\max} - 1]\,, j \in [0, m_I - 1]$$

$$B_{i,j}^6 := \frac{\alpha \cdot u_j(x) \cdot L_i(y) + \alpha^2 v_j(x) \cdot L_i(y) + \alpha^3 w_j(x) \cdot L_i(y)}{\delta} \cdot G_1 \qquad i \in [0, s_{\max} - 1]\,,\ j \in [l + m_I,\ m_D - 1]$$

$$B_{h,k}^7 := \frac{\alpha^k x^h t_n(x)}{\delta} \cdot G_1 \qquad h \in [0, 2]\,,\ k \in [1, 3]$$

$$B_j^8 := \frac{\alpha^4 x^j t_{m_I}(x)}{\delta} \cdot G_1 \qquad j \in [0, 1]$$

$$B_{i,k}^9 := \frac{\alpha^k y^i t_{s_{\max}}(y)}{\delta} \cdot G_1 \qquad i \in [0, 2]\,,\ k \in [1, 4]$$

For initialization purposes, we also define simplified versions of these parameters by setting $\gamma = 1, \delta = 1$ and $\eta = 1$. For instance, we define:

$$B_j^{1'} := \left(\alpha \cdot u_j(x) \cdot L_0(y) + \alpha^2 v_j(x) \cdot L_0(y) + \alpha^3 w_j(x) \cdot L_0(y) + M_j(x)\right) \cdot\ G_1$$

Likewise, the auxiliary values, $B_j^{2'}, B_j^{3'}, B_j^{4'}, B_{i,j}^{5'}, B_{i,j}^{6'}, B_{h,k}^{7'}, B_j^{8'}$ and $B_{i,k}^{9'}$ are computed at $\gamma = 1, \delta = 1$ and $\eta = 1$ for the initialization.

4.3.3 Second Round

In the first phase of the ceremony, all parameters in $[\boldsymbol{\sigma}_{A,C}]_1$ and a subset of the parameters in $[\boldsymbol{\sigma}_V]_2$ were generated.

In this phase, the remaining parameters are constructed. Specifically, the contributions for $[\boldsymbol{\sigma}_B]_1$ and $[\boldsymbol{\sigma}_V]_2$ are generated as follows:

$$\boldsymbol{\sigma}_V := (\gamma, \delta, \eta)$$

$$[\sigma_V]_2 = ([\gamma]_2,\ [\delta]_2,\ [\eta]_2,)$$

$$[\sigma_B]_1 = \left([\delta]_1,\ [\eta]_1,\ B_j^1,\ B_j^2,\ B_j^3,\ B_j^4,\ B_{i,j}^5,\ B_{i,j}^6,\ B_{h,k}^7,\ B_j^8,\ B_{i,k}^9\right)$$

$$
\begin{aligned}
B_j^1 &:= \gamma^{-1}\left(L_1(y)\sigma_j(x) + M_j(x)\right)\cdot G_1, && j \in \left[0,\ l_{\text{pub}}^{(\text{out})} - 1\right] \\
B_j^2 &:= \gamma^{-1}\left(L_0(y)o_j(x) + M_j(x)\right)\cdot G_1, && j \in \left[l_{\text{pub}}^{(\text{out})},\ l_{\text{pub}} - 1\right] \\
B_j^3 &:= \gamma^{-1}\left(L_3(y)\sigma_j(x)\right)\cdot G_1, && j \in \left[l_{\text{pub}},\ l_{\text{pub}} + l_{\text{prv}}^{(\text{out})} - 1\right] \\
B_j^4 &:= \gamma^{-1}\left(L_2(y)\sigma_j(x)\right)\cdot G_1, && j \in \left[l_{\text{pub}} + l_{\text{prv}}^{(\text{out})},\ l - 1\right] \\
B_{i,j}^5 &:= \eta^{-1}\left(L_i(y)\sigma_{j+l}(x) + \alpha^4 K_j(x)\right)\cdot G_1, && i \in [0, s_{\max} - 1],\ j \in [0, m_I - 1] \\
B_{i,j}^6 &:= \delta^{-1}L_i(y)\sigma_j(x)\cdot G_1, && i \in [0, s_{\max} - 1],\ j \in [l + m_I, m_D - 1] \\
B_{h,k}^7 &:= \delta^{-1}\alpha^k x^h t_n(x)\cdot G_1, && h \in [0, 2],\ k \in [1, 3] \\
B_j^8 &:= \delta^{-1}\alpha^4 x^j t_{m_I}(x)\cdot G_1, && j \in [0, 1] \\
B_{i,k}^9 &:= \delta^{-1}\alpha^k y^i t_{s_{\max}}(y)\cdot G_1, && i \in [0, 2],\ k \in [1, 4]
\end{aligned}
$$

Initialization:

During the initialization phase, we compute the following expressions for each index j as a prerequisite for entering the second phase of the setup ceremony. These computations can be independently performed by all participants, since all the necessary parameters were generated and published during the first phase.

As a brief reminder, G_1 denotes the generator of the elliptic curve group $\mathbb{G}_1$, and similarly, G_2 is the generator of the group $\mathbb{G}_2$.

1. $[\gamma]_1^{\mathbf{0}} := G_1, \quad [\gamma]_2^{\mathbf{0}} := G_2$
2. $[\delta]_1^{\mathbf{0}} := G_1, \quad [\delta]_2^{\mathbf{0}} := G_2$
3. $[\eta]_1^{\mathbf{0}} := G_1, \quad [\eta]_2^{\mathbf{0}} := G_2$
4. $[B_j^1]_1^{\mathbf{0}} := B_j^{1'}$, ▷ for each $j \in \left[0,\ l_{\text{pub}}^{(\text{out})} - 1\right]$
5. $[B_j^2]_1^{\mathbf{0}} := B_j^{2'}$, ▷ for each $j \in \left[l_{\text{pub}}^{(\text{out})},\ l_{\text{pub}} - 1\right]$
6. $[B_j^3]_1^{\mathbf{0}} := B_j^{3'}$, ▷ for each $j \in \left[l_{\text{pub}},\ l_{\text{pub}} + l_{\text{prv}}^{(\text{out})} - 1\right]$
7. $[B_j^4]_1^{\mathbf{0}} := B_j^{4'}$, ▷ for each $j \in \left[l_{\text{pub}} + l_{\text{prv}}^{(\text{out})},\ l - 1\right]$
8. $[B_{i,j}^5]_1^{\mathbf{0}} := B_j^{5'}$, ▷ for each $i \in [0, s_{\max} - 1],\ j \in [0, m_I - 1]$
9. $[B_{i,j}^6]_1^{\mathbf{0}} := B_{i,j}^{6'}$, ▷ for each $i \in [0, s_{\max} - 1],\ j \in [l + m_I,\ m_D - 1]$
10. $[B_{h,k}^7]_1^{\mathbf{0}} := B_{h,k}^{7'}$, ▷ for each $h \in [0, 2],\ k \in [1, 3]$
11. $[B_j^8]_1^{\mathbf{0}} := B_j^{8'}$, ▷ for each $j \in [0, 1]$
12. $[B_{i,k}^9]_1^{\mathbf{0}} := B_{i,k}^{9'}$, ▷ for each $i \in [0, 2],\ k \in [1, 4]$

Computations:

After the completion of the initialization phase, the main ceremony begins. The following computations are to be performed by each participant P_t, where $t \in [1, \ldots, N]$, as their individual contribution to the setup.

1. $[\gamma_t]_1 = \gamma_t \cdot G_1, \quad [\gamma_t]_2 = \gamma_t \cdot G_2,$ ▷ picks γ_t random number, $\gamma_t \in_R \mathbb{F}_p^*$
2. $[\delta_t]_1 = \delta_t \cdot G_1, \quad [\delta_t]_2 = \delta_t \cdot G_2,$ ▷ picks δ_t random number, $\delta_t \in_R \mathbb{F}_p^*$
3. $[\eta_t]_1 = \eta_t \cdot G_1, \quad [\eta_t]_2 = \eta_t \cdot G_2,$ ▷ picks η_t random number, $\eta_t \in_R \mathbb{F}_p^*$

4. $p_{\gamma_t} = \mathbf{POK}(\gamma_t, \mathrm{v}_{2,t-1})$
5. $p_{\delta_t} = \mathbf{POK}(\delta_t, \mathrm{v}_{2,t-1})$
6. $p_{\eta_t} = \mathbf{POK}(\eta_t, \mathrm{v}_{2,t-1})$
7. $[\gamma]_1^{\mathbf{t}} = \gamma_t \cdot [\gamma]_1^{\mathbf{t-1}}$,
8. $[\gamma]_2^{\mathbf{t}} = \gamma_t \cdot [\gamma]_2^{\mathbf{t-1}}$,
9. $[\delta]_1^{\mathbf{t}} = \delta_t \cdot [\delta]_1^{\mathbf{t-1}}$,
10. $[\delta]_2^{\mathbf{t}} = \delta_t \cdot [\delta]_2^{\mathbf{t-1}}$,
11. $[\eta]_1^{\mathbf{t}} = \eta_t \cdot [\eta]_1^{\mathbf{t-1}}$,
12. $[\eta]_2^{\mathbf{t}} = \eta_t \cdot [\eta]_2^{\mathbf{t-1}}$,
13. $[B_j^1]_1^{\mathbf{t}} = {\gamma_t}^{-1} \cdot [B_j^1]_1^{\mathbf{t-1}}$ ▷ for each $j \in \left[0,\ l_{\mathrm{pub}}^{(\mathrm{out})} - 1\right]$
14. $[B_j^2]_1^{\mathbf{t}} = {\gamma_t}^{-1} \cdot [B_j^2]_1^{\mathbf{t-1}}$ ▷ for each $j \in \left[l_{\mathrm{pub}}^{(\mathrm{out})},\ l_{\mathrm{pub}} - 1\right]$
15. $[B_j^3]_1^{\mathbf{t}} = {\gamma_t}^{-1} \cdot [B_j^3]_1^{\mathbf{t-1}}$ ▷ for each $j \in \left[l_{\mathrm{pub}},\ l_{\mathrm{pub}} + l_{\mathrm{prv}}^{(\mathrm{out})} - 1\right]$
16. $[B_j^4]_1^{\mathbf{t}} = {\gamma_t}^{-1} \cdot [B_j^4]_1^{\mathbf{t-1}}$ ▷ for each $j \in \left[l_{\mathrm{pub}} + l_{\mathrm{prv}}^{(\mathrm{out})},\ l - 1\right]$
17. $[B_{i,j}^5]_1^{\mathbf{t}} = {\eta_t}^{-1} \cdot [B_{i,j}^5]_1^{\mathbf{t-1}}$ ▷ for each $i \in [0, s_{\max} - 1]\,,\ j \in [0, m_I - 1]$
18. $[B_{i,j}^6]_1^{\mathbf{t}} = {\delta_t}^{-1} \cdot [B_{i,j}^6]_1^{\mathbf{t-1}}$ ▷ for each $i \in [0, s_{\max} - 1]\,,\ j \in [l + m_I,\ m_D - 1]$
19. $[B_{h,k}^7]_1^{\mathbf{t}} = {\delta_t}^{-1} \cdot [B_{h,k}^7]_1^{\mathbf{t-1}}$ ▷ for each $h \in [0, 2]\,,\ k \in [1, 3]$
20. $[B_j^8]_1^{\mathbf{t}} = {\delta_t}^{-1} \cdot [B_j^8]_1^{\mathbf{t-1}}$ ▷ for each $j \in [0, 1]$
21. $[B_{i,k}^9]_1^{\mathbf{t}} = {\delta_t}^{-1} \cdot [B_{i,k}^9]_1^{\mathbf{t-1}}$ ▷ for each $i \in [0, 2]\,,\ k \in [1, 4]$
22. This step is optional. In this phase, if desired, the randomness generated by the random beacon can be incorporated into the parameter generation process. After the final participant has submitted their contribution, this step may optionally be executed as follows: The procedures described from Step-7 to Step-21 (inclusive) are repeated using the newly generated randomness (via the random beacon) for the parameters γ', δ', η'.

    Note that $(\gamma', \delta', \eta') := \mathrm{RandomBeacon}(J, 3)$.

Verification:

The protocol verifier performs checks for each $t \in [1, ..., N]$. If any verification fails, the protocol will be halted, and the failure status will be communicated to all participants.

Additionally, each participant computes $[B_j^b]_1^{\mathbf{0}}$, where $b \in [1, 9]$, to verify the correctness of the agreed-upon circuit description that is used to generate the second-phase parameters of the ceremony. This ensures that a potentially malicious participant cannot tamper with the circuit representation during the setup.

1. Compute $[B_j^b]_1^{\mathbf{0}}$ using publicly known polynomials such that $o_j(x), L_i(x), M_j(x), K_j(x), t_n(x)$, for each $b \in [1, 9]$. Furthermore, for randomly chosen indices (b, j), a set of more than 1000 randomly sampled evaluation points from the corresponding parameter B_j^b will be compared with the initially published values to verify their correctness.
2. $\mathbf{CheckPOK}([\gamma_t]_1\,, \mathrm{v}_{2,t-1}, p_{\gamma_t})$
3. $r_{\gamma_t} \;=\; \mathcal{RO}([\gamma_t]_1\,, \mathrm{v}_{2,t-1})$
4. $\mathbf{Consistent}\left([\gamma]_1^{\mathbf{t-1}} - [\gamma]_1^{\mathbf{t}};\ (r_{\gamma_t,}, p_{\gamma_t})\right)$
5. $\mathbf{Consistent}\left([\gamma]_1^{\mathbf{t-1}} - [\gamma]_1^{\mathbf{t}};\ ([\gamma]_2^{\mathbf{t-1}}, [\gamma]_2^{\mathbf{t}})\right)$

6. $\mathbf{CheckPOK}([\delta_t]_1, \mathrm{v}_{2,t-1}, p_{\delta_t})$
7. $r_{\delta_t} = \mathcal{RO}([\delta_t]_1, \mathrm{v}_{2,t-1})$
8. $\mathbf{Consistent}\left([\delta]_1^{\mathbf{t-1}} - [\delta]_1^{\mathbf{t}};\ (r_{\delta_t,}, p_{\delta_t})\right)$
9. $\mathbf{Consistent}\left([\delta]_1^{\mathbf{t-1}} - [\delta]_1^{\mathbf{t}};\ ([\delta]_2^{\mathbf{t-1}}, [\delta]_2^{\mathbf{t}})\right)$
10. $\mathbf{CheckPOK}([\eta_t]_1, \mathrm{v}_{2,t-1}, p_{\eta_t})$
11. $r_{\eta_t} = \mathcal{RO}([\eta_t]_1, \mathrm{v}_{2,t-1})$
12. $\mathbf{Consistent}\left([\eta]_1^{\mathbf{t-1}} - [\eta]_1^{\mathbf{t}};\ (r_{\eta_t,}, p_{\eta_t})\right)$
13. $\mathbf{Consistent}\left([\eta]_1^{\mathbf{t-1}} - [\eta]_1^{\mathbf{t}};\ ([\eta]_2^{\mathbf{t-1}}, [\eta]_2^{\mathbf{t}})\right)$
14. $\mathbf{Consistent}\big([B_j^1]_1^{\mathbf{t}} - [B_j^1]_1^{\mathbf{t-1}}; [\gamma_t]_2\big)$, ▷ for each $j \in \left[0,\ l_{\mathrm{pub}}^{(\mathrm{out})} - 1\right]$
15. $\mathbf{Consistent}\big([B_j^2]_1^{\mathbf{t}} - [B_j^2]_1^{\mathbf{t-1}}; [\gamma_t]_2\big)$, ▷ for each $j \in \left[l_{\mathrm{pub}}^{(\mathrm{out})},\ l_{\mathrm{pub}} - 1\right]$
16. $\mathbf{Consistent}\big([B_j^3]_1^{\mathbf{t}} - [B_j^3]_1^{\mathbf{t-1}}; [\gamma_t]_2\big)$, ▷ for each $j \in \left[l_{\mathrm{pub}},\ l_{\mathrm{pub}} + l_{\mathrm{prv}}^{(\mathrm{out})} - 1\right]$
17. $\mathbf{Consistent}\big([B_j^4]_1^{\mathbf{t}} - [B_j^4]_1^{\mathbf{t-1}}; [\gamma_t]_2\big)$, ▷ for each $j \in \left[l_{\mathrm{pub}} + l_{\mathrm{prv}}^{(\mathrm{out})},\ l - 1\right]$
18. $\mathbf{Consistent}\big([B_{i,j}^5]_1^{\mathbf{t}} - [B_{i,j}^5]_1^{\mathbf{t-1}}; [\eta_t]_2\big)$, ▷ for each $i \in [0, s_{\max} - 1]$, $j \in [0, m_I - 1]$
19. $\mathbf{Consistent}\big([B_j^6]_1^{\mathbf{t}} - [B_j^6]_1^{\mathbf{t-1}}; [\delta_t]_2\big)$, ▷ for each $i \in [0, s_{\max} - 1]$, $j \in [l + m_I,\ m_D - 1]$
20. $\mathbf{Consistent}\big([B_j^7]_1^{\mathbf{t}} - [B_j^7]_1^{\mathbf{t-1}}; [\delta_t]_2\big)$, ▷ for each $h \in [0, 2]$, $k \in [1, 3]$
21. $\mathbf{Consistent}\big([B_j^8]_1^{\mathbf{t}} - [B_j^8]_1^{\mathbf{t-1}}; [\delta_t]_2\big)$, ▷ for each $j \in [0, 1]$
22. $\mathbf{Consistent}\big([B_j^9]_1^{\mathbf{t}} - [B_j^9]_1^{\mathbf{t-1}}; [\delta_t]_2\big)$, ▷ for each $i \in [0, 2]$, $k \in [1, 4]$

5 Conclusion

In this paper, we presented a general and practical Multi-Party Computation (MPC) setup protocol for generating structured reference strings in non-transparent, pairing-based zk-SNARKs. Our construction extends the MMORPG framework of BGM17 by supporting multivariate parameter structures, including the tri-variant monomials required in modern programmable SNARK designs.

To demonstrate the applicability of our framework, we applied it to the Tokamak zk-SNARK as a concrete use case. Since Tokamak naturally employs the same three-variant structure, our MPC ceremony integrates seamlessly with its setup procedure and instantiates its universal and circuit-specific CRS components with full correctness and efficiency. The resulting design provides modularity through a two-phase architecture—comprising a universal Powers-of-τ phase and a circuit-specialisation phase—together with formal compute and verify algorithms that enable decentralized participation, public verifiability, and soundness under the standard "one honest participant" assumption. We further showed how an optional random beacon can be incorporated to inject unbiased entropy and strengthen the trust model without altering the underlying protocol structure. The overall ceremony achieves scalability, robustness, and transparency, making it suitable for deployment in blockchain ecosystems and other real-world zero-knowledge applications.

Beyond the Tokamak use case, the technical components introduced in this work lay the groundwork for future MPC ceremonies capable of supporting increasingly expressive and programmable SNARK systems. Our framework thus provides both a practical blueprint for current deployments and a foundation for upcoming generations of multivariate and universal zk-SNARK constructions.

References

[1] Jens Groth. On the size of pairing-based non-interactive arguments. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*, page 305–326, Berlin, Heidelberg, 2016. Springer-Verlag.

[2] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Paper 2017/1050, 2017. `https://eprint.iacr.org/2017/1050`.

[3] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[4] Benedikt Bünz. *Verifiable Delay Function*, pages 1–4. Springer Berlin Heidelberg, Berlin, Heidelberg, 2019.

[5] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 379–407, Cham, 2019. Springer International Publishing.

[6] Qiang Wu, Liang Xi, Shiren Wang, Shan Ji, Shenqing Wang, and Yongjun Ren. Verifiable delay function and its blockchain-related application: A survey. *Sensors*, 22(19), 2022.

[7] Jehyuk Jang and Jamie Judd. An efficient SNARK for field-programmable and RAM circuits. Cryptology ePrint Archive, Paper 2024/507, 2024. `https://eprint.iacr.org/2024/507`.

[8] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304, 2015.

[9] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 98–127, Cham, 2021. Springer International Publishing.

[10] Aleo Engineering. A technical explanation of the aleo setup ceremony, 2021. `https://aleo.org/post/a-technical-explanation-of-the-aleo-setup-ceremony/` [accessed: (20/06/2025)].

[11] Filecoin. Trusted setup complete!, 2020. `https://filecoin.io/blog/posts/trusted-setup-complete/` [accessed: (20/06/2025)].

[12] AZTEC. Aztec crs: The biggest mpc setup in history has successfully finished, 2020. `https://aztec.network/blog/aztec-crs-the-biggest-mpc-setup-in-history-has-successfully-finished` [accessed: (20/06/2025)].

[13] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 105–134, Cham, 2024. Springer Nature Switzerland.

[14] Tokamak Network. Tokamak zk-evm: Mpc setup protocol. `https://github.com/tokamak-network/Tokamak-zk-EVM/tree/main/packages/backend/setup/mpc-setup`, 2025. Accessed: 2025-11-25.

[15] Mayank Raikwar. Competitive decentralized randomness beacon protocols. In *Proceedings of the Fourth ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '22, page 83–94, New York, NY, USA, 2022. Association for Computing Machinery.

[16] Ben Edgington. Upgrading ethereum: A technical handbook on ethereum's move to proof of stake and beyond, edition 0.3: Capella [wip]. `https://eth2book.info/capella/part2/building_blocks/randomness/`, 2025. Accessed: 2025-06-21.

[17] Chia Network Inc. Chia consensus: A new nakamoto consensus. Technical report, Chia Network, 2022. Version 0.9.

A MPC-Based Parameter Generation for the Groth16 Scheme

In this section, we present the application of BGM17's [2] multiparty computation (MPC) scheme, called MMORPG, on Groth16 [1], which is a well-known zk-SNARK scheme.

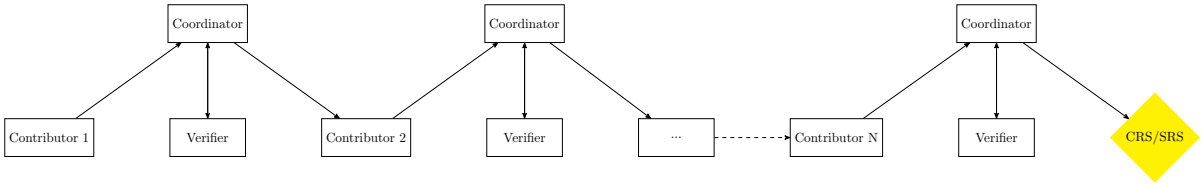

Figure 2: Multi-party Setup Ceremony Flow for Generating zk-SNARK Parameters

According the MPC protocol, the CRS is generated through two stages. The initial phase, known as "Powers of Tau", generates universal setup parameters applicable to all circuits within the scheme, up to a specified size. The "Powers of Tau" ceremony offers several improvements compared to previous schemes. Firstly, participants are not required to be pre-selected; instead, the protocol utilizes a random beacon that generates public, random values at regular intervals, enabling a continuous ceremony. This means that participants do not need to always be present and connected on-line. The use of the random beacon also guarantees the coordinator's public verifiability. Consequently, the protocol can theoretically accommodate hundreds or even thousands of participants. The subsequent phase transforms the results from the Powers of Tau phase into a CRS specific to the NP-relation.

In this protocol, a central *protocol coordinator* facilitates the communication of messages between participants. Instead, the protocol employs a random beacon that generates public random values at predetermined intervals to sustain an ongoing ceremony. However, there is no requirement to trust the *coordinator*, as any individual can subsequently confirm the accuracy of the protocol coordinator's messages within the protocol transcript. Specifically, the protocol verifier's responsibilities will encompass, beyond the explicitly outlined procedures, independently computing the protocol coordinator's messages and verifying their correctness.

This setup allows participants not always to be required to be online and available. The random beacon also guarantees the public verifiability of the coordinator. Consequently, the protocol can theoretically accommodate hundreds or even thousands of participants.

A.1 Preview of the setup phase in the Groth16 scheme

In this section, we provide an overview of the parameters used in the setup phase of the Groth16 protocol without going into detail. We present a pairing-based non-interactive zero-knowledge (NIZK) argument for quadratic arithmetic programs

$$\sum_{i=0}^{m} a_i u_i(X) \cdot \sum_{i=0}^{m} a_i v_i(X) = \sum_{i=0}^{m} a_i w_i(X) + h(X)t(X),$$

for some degree $n-2$ quotient polynomial $h(X)$.

Let $\{u_i,\, v_i,\, w_i\}_{i\in[0\ldots m]}$ and $\{t\}$ be the polynomials of a degree n QAP over $\mathbb{F}_p$, where t is the degree n target polynomial of the QAP and the other polynomials have degree smaller than n. Suppose that $1, \cdots, \ell < m$ are the indices of the public input.

A.2 Groth's Setup Phase:

Choose random $\alpha, \beta, \gamma, \delta, x \longleftarrow \mathbb{Z}_p^*$. $\tau = (\alpha, \beta, \gamma, \delta, x)$ and compute $([\sigma_1]_1, [\sigma_2]_2)$, $[\sigma_1]_1 = \sigma_1 \cdot G_1$ and $[\sigma_1]_2 = \sigma_1 \cdot G_2$ where

$$\sigma_1 = \left(\begin{array}{c} \alpha,\ \beta,\ \delta,\ \left\{x^i\right\}_{i=0}^{n-1},\ \left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}\right\}_{i=0}^{\ell} \\ \left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\right\}_{i=l+1}^{m},\ \left\{\frac{x^i t(x)}{\delta}\right\}_{i=0}^{n} \end{array} \right),\ \sigma_2 = \left(\beta, \gamma, \delta, \left\{x^i\right\}_{i=0}^{n-1}\right).$$

A.3 The Uni-variant MPC Protocol Description

We present the MMORPG MPC protocol designed to compute the common reference string (CRS) elements of Groth16.

The resulting output will be structured as follows:

$$\left\{\left[x^i\right]\right\}_{i\in[0,\ldots,n-1]}, \left\{\left[x^i\right]_1\right\}_{i\in[n,\ldots,2n-2]}, \left\{\left[\alpha x^i\right]_1\right\}_{i\in[0,\ldots,n-1]},$$

$$[\beta], \left\{\left[\beta x^i\right]_1\right\}_{i\in[1,\ldots,n-1]}, \left\{\left[x^i \cdot t(x)/\delta\right]_1\right\}_{i\in[0,\ldots,n-2]},$$
$$\left\{\left[\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\right]_1\right\}_{i\in[\ell+1,\ldots,m]}$$

We execute the protocol in two rounds, and each round we compute M_1 and M_2 set of parameters, respectively. M is an output in $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}$. Note that we use the notation $\mathbb{G} : \mathbb{G}_1 \times \mathbb{G}_2$ and $\mathbf{G} := (G_1, G_2)$.

We denote $[M]^j$ the "partial M" after N participants $P_1, P_2, ..., P_j$ have contributed their shares, where $j \in [N]$. $[M]^0$ will be initialized to a predetermined value as outlined in the protocol description. It is assumed that $\mathbf{G}$ is publicly known. We refer to $\text{transcript}_{i,j}$ as the record of the protocol up until the moment when participant j transmitted her message in phase i.

In this part, we will use some of the algorithms mentioned in the Section 3 such as Algorithm 3, Algorithm 4, Algorithm 5, Algorithm 6 called ***POK***, ***CheckPOK***, ***SameRatio***, ***Consistent***, respectively.

We assume the availability of a 'random beacon' denoted as RandomBeacon(.), which generates elements within $\mathbb{F}_p^*$. The function RandomBeacon(.) takes a time slot J and a positive integer k as input, and produces k elements $a_1, \ldots, a_k \in \mathbb{F}_p^*$. For convenience, we assume that RandomBeacon(J, k) is defined only for a specific subset of J values as its initial input.

This MPC protocol follows a round-robin structure, where each player P_i transmits a single message in each phase after receiving it from player P_{i-1}, and RandomBeacon(.) is triggered at the end of each phase, following the message from P_N. Consequently, we assume that the protocol specifies the time slot for player P_i to send their phase ℓ message as $J = (\ell - 1) \cdot (N + 1) + i$. In this setup, it is convenient to define RandomBeacon(J, k) only when J is a multiple of $N + 1$.

We assume that all parties can access to the same oracle $\mathcal{RO}$ during the ceremony protocol. The oracle $\mathcal{RO}$ takes as input strings of arbitrary length and output is a uniform independent elements of $\mathbb{G}_2^*$. We denote by $\text{transcript}_{\ell,i}$ the transcript of the protocol up to the point where player i sent his message in phase ℓ. $[\alpha]^{\mathbf{j}} \in \mathbb{G}_1$ or $[\alpha]^{\mathbf{j}} \in \mathbb{G}$, If $[\alpha]^{\mathbf{j}} \in \mathbb{G}_1$; then $[\alpha]^{\mathbf{j}} := \alpha_1 \cdot \alpha_2 \cdots \alpha_{j-1} \cdot \alpha_j \cdot G_1$ and $[\alpha x^i]^{\mathbf{j}} := (\alpha_0 x_0^i)(\alpha_1 x_1^i) \cdots (\alpha_j x_j^i) G_1$.

A.3.1 First Round: Powers of x

We will compute the M_1:

$$M_1 = \left\{ \begin{array}{l} \{[x^i]\}_{i\in[0,\ldots,n-1]}, \{[x^i]_1\}_{i\in[n,\ldots,n-2]}, \\ \{[\alpha x^i]_1\}_{i\in[0,\ldots,n-1]}, [\beta]\,\{[\beta x^i]_1\}_{i\in[1,\ldots,n-1]} \end{array} \right\}$$

Initialization:

We first initialize the parameters with the following public values.

1. $[x^i]^{\mathbf{0}} := \mathbf{G}$, $i \in [1, ..., n-1]$.
2. $[x^i]^{\mathbf{0}} := G_1$, $i \in [n, ..., 2n-2]$.
3. $[\alpha x^i]^{\mathbf{0}} := G_1$, $i \in [0, ..., n-1]$.
4. $[\beta]^{\mathbf{0}} := \mathbf{G}$.
5. $\left[\beta x^i\right]^{\mathbf{0}} := G_1,\ i \in [1, ..., n-1]$.

Computations:

Then, the participants $(j \in [N], P_j)$ perform the below computations:

1. $[\alpha_j]_1, [\beta_j]_1, [x_j]_1$
2. $y_{\alpha,j} := \mathbf{POK}(\alpha_j, \text{transcript}_{1,j-1})$
3. $y_{\beta,j} := \mathbf{POK}(\beta_j, \text{transcript}_{1,j-1})$
4. $y_{x,j} := \mathbf{POK}(x_j, \text{transcript}_{1,j-1})$
5. For each $i \in [1, .., 2n-2]$, $[x^i]^{\mathbf{j}} := x_j^i \cdot [x^i]^{\mathbf{j-1}}$

6. For each $j \in [0, .., n-1]$, $\left[\alpha x^i\right]^{\mathbf{j}} := \alpha_j x_j^i \cdot \left[\alpha x^i\right]^{\mathbf{j-1}}$
7. For each $j \in [0, .., n-1]$, $\left[\beta x^i\right]^{\mathbf{j}} := \beta_j x_j^i \cdot \left[\beta x^i\right]^{\mathbf{j-1}}$

Let $J-1$ be the time-slot where P_N sends their message. Let $(x', \alpha', \beta') := \text{RandomBeacon}(J, 3)$

1. $\left[x^i\right] := x'^i \cdot \left[x^i\right]^{\mathbf{N}}$, $i \in [1, .., 2n-2]$.
2. $\left[\alpha x^i\right] := \alpha' x'^i \cdot \left[\alpha x^i\right]^{\mathbf{N}}$, $i \in [0, .., n-1]$.
3. $\left[\beta x^i\right] := \beta' x'^i \cdot \left[\beta x^i\right]^{\mathbf{N}}$, $i \in [0, .., n-1]$.

Verifications:

The protocol verifier computes for each $j \in [N]$,

$$\begin{aligned} r_{\alpha,j} &:= \mathcal{RO}([\alpha_j]_1, \text{transcript}_{1,j-1}), \\ r_{\beta,j} &:= \mathcal{RO}([\beta_j]_1, \text{transcript}_{1,j-1}), \\ r_{x,j} &:= \mathcal{RO}([x_j]_1, \text{transcript}_{1,j-1}), \end{aligned}$$

and checks $j \in [N]$ that

1. $\mathbf{CheckPOK}([\alpha_{\mathrm{j}}]_1, \text{transcript}_{1,j-1}, y_{\alpha,j})$,
2. $\mathbf{CheckPOK}([\beta_{\mathrm{j}}]_1, \text{transcript}_{1,j-1}, y_{\beta,j})$,
3. $\mathbf{CheckPOK}([\mathrm{x_j}]_1, \text{transcript}_{1,j-1}, y_{x,j})$,
4. $\mathbf{Consistent}([\alpha]^{\mathbf{j-1}} - [\alpha]^{\mathbf{j}}; (r_{\alpha,j}, y_{\alpha,j}))$,
5. $\mathbf{Consistent}([\beta]^{\mathbf{j-1}} - [\beta]^{\mathbf{j}}; (r_{\beta,j}, y_{\beta,j}))$,
6. $\mathbf{Consistent}([x]^{\mathbf{j-1}} - [x]^{\mathbf{j}}; (r_{x,j}, y_{x,j}))$,
7. For each $i \in [1, \ldots, 2n-2]$, $\mathbf{Consistent}\left([x^{i-1}]^{\mathbf{j}} - [x^i]^{\mathbf{j}}; [x]^{\mathbf{j}}\right)$,
8. For each $i \in [1, \ldots, n-1]$, $\mathbf{Consistent}\left([x^i]_1^{\mathbf{j}} - [\alpha x^i]^{\mathbf{j}}; [\alpha]^{\mathbf{j}}\right)$,
9. For each $i \in [1, \ldots, n-1]$, $\mathbf{Consistent}\left([x^i]_1^{\mathbf{j}} - [\beta x^i]^{\mathbf{j}}; [\beta]^{\mathbf{j}}\right)$,

A.3.2 Second Round

During this subsequent phase of the protocol, parameters specific to the circuit will be generated.

We will compute the M_2:

$$\mathrm{M}_2 = \left\{[\delta], \left\{[K_i]_1\right\}_{i \in [\ell+1 \ldots m]}, \left\{[H_i]_1\right\}_{i \in [0, \ldots, n-2]}\right\}, \text{ where}$$

$$K_i := \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}, \text{ where } i \in [\ell+1, \ldots, m], \text{ and}$$
$$H_i := \frac{t(x)x^i}{\delta}, \text{ where } i \in [0, \ldots, n-2].$$

Also let
$K_i' = [\beta u_i(x) + \alpha v_i(x) + w_i(x)]_1$.
$H_i' = \left[t(x)x^i\right]_1$, where $\delta = 1$.

Initialization:

1. $[K_i]_1^{\mathbf{0}} := K_i'$, $i \in [\ell+1, \ldots, m]$
2. $[H_i]_1^{\mathbf{0}} := H_i'$, $i \in [0, \ldots, n-2]$
3. $[\delta]^{\mathbf{0}} := \mathbf{G}$.

Computations:

For $j \in [N], P_j$ outputs:

1. $[\delta_j]_1$.
2. $y_{\delta,j} := \mathbf{POK}(\delta_j, \text{transcript}_{2,j-1})$.
3. $[\delta]^{\mathbf{j}} := [\delta]^{\mathbf{j-1}}/\delta_{\mathbf{j}}$.
4. For each $i \in [\ell+1, \ldots, m]$, $[K_i]^{\mathbf{j}} := ([K_i]^{\mathbf{j-1}})/\delta_j$.
5. For each $i \in [\ell+0, \ldots, n-2]$, $[H_i]^{\mathbf{j}} := ([H_i]^{\mathbf{j-1}})/\delta_j$.

Finally, $J-1$ be the time-slot where P_N sends their message.
Let $\delta' := \text{RandomBeacon}(J, 1)$

1. $\big[\delta\big] := \big[\delta\big]^{\mathbf{N}}/\delta'$.
2. $[K_i]_1 := [K_i]^{\mathbf{N}}/\delta'$.
3. $[H_i]_1 := [H_i]^{\mathbf{N}}/\delta'$.

Verifications:

The protocol verifier(s) computes for each $j \in [N]$,

$$r_{\delta,j} := \mathcal{RO}([\delta_j]_1, \text{transcript}_{2,j-1}),$$

and for each $j \in [N]$ checks that

1. $\mathbf{CheckPOK}([\delta_{\mathrm{j}}]_1 \, \text{transcript}_{2,j-1}, y_{\delta,j})$.
2. For $j \in [N]$, $\mathbf{Consistent}\big([\delta]^{\mathbf{j-1}} - [\delta]^{\mathbf{j}}; (r_{\delta,j}, y_{\delta,j})\big)$,
3. For each $i \in [\ell+1, \ldots, m]$, $j \in [N]$, $\mathbf{Consistent}([K_i]^{\mathbf{j}} - [K_i]^{\mathbf{j-1}}; [\delta_j])$.
4. For each $i \in [\ell+0, \ldots, n-2]$, $j \in [N]$, $\mathbf{Consistent}([H_i]^{\mathbf{j}} - [H_i]^{\mathbf{j-1}}; [\delta_j])$.