Powers-of-Tau to the People: Decentralizing Setup Ceremonies

Valeria Nikolaenko[1], Sam Ragsdale[1], Joseph Bonneau[1,3], and Dan Boneh[2]

[1] A16Z Crypto Research Lab
[2] Stanford University
[3] New York University

Abstract. We propose several decentralized ceremonies for constructing a powers-of-tau structured reference string (SRS). Our protocols make use of a blockchain platform to run in a permissionless manner, where anyone can contribute randomness in exchange for paying the requisite transaction fees. The resulting SRS is secure as long as any single party participates honestly. We introduce several protocols optimized for different sized powers-of-tau setups and using an on-chain or off-chain data availability model to store the resulting string. We implement our most efficient protocol on top of Ethereum, demonstrating practical concrete performance.

1 Introduction

Many cryptographic protocols assume a *trusted setup ceremony*, a one-time procedure to generate public parameters which also generates an unwanted trapdoor as a byproduct. Perhaps the earliest example is the accumulator scheme of Benaloh and de Mare [12] which requires a public modulus N such that nobody knows its factorization $N = p \cdot q$, a trapdoor which allows forging a proof that any element is included in the accumulator.

In general, a setup ceremony consists of a randomized algorithm $\mathsf{Setup}() \xrightarrow{\$} (\mathsf{pp}, \tau)$. The *public parameters* (pp), also called a *structured reference string* (SRS), must be known to all users of the system, whereas the *trapdoor* (τ) must be discarded for the scheme to be secure. Such trapdoors have been called "toxic waste" due to the importance of destroying them after the setup is complete.

In the simplest case of a fully *trusted setup*, a single entity computes $\mathsf{Setup}()$ and is trusted to discard τ. Setup ceremonies have been conducted by several prominent cryptocurrency applications, which have pioneered the use of secure multiparty computation (MPC) ceremonies to avoid having any single party ever know the final trapdoor. These ceremonies have differed in the number of participants involved, the number of rounds, and the exact trust model, but so far all have been facilitated by a centralized coordinator. In particular, the coordinator has the ability to choose which parties are able to participate, making these protocols *permissioned*. We review setup ceremonies run in practice in Section 2.2.

In this work, we endeavor to remove the coordinator and build the first truly *decentralized* and *permissionless* setup ceremony. This approach is appropriate given a multiparty computation which requires only one honest participant (sometimes called an "anytrust" or "dishonest majority" model). In this model, there is no downside (beyond computational overhead) of allowing additional participants to contribute to the protocol. We call this the *more-the-merrier* property. A more-the-merrier protocol can safely be opened to the general public, enabling an interesting new security property: any individual can participate and therefore they can trust the final result (at least to the extent that they trust themselves to have participated correctly), even if they make no assumptions about any of the other participants.

Powers-of-tau. We focus on a common type of ceremony which constructs a powers-of-tau SRS. Working in elliptic curve groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order p with generators B_1 and B_2 respectively and an efficiently computable pairing, the goal of the setup is to produce a public parameter string:

$$\mathsf{pp} := \left(\tau B_1, \tau^2 B_1, \ldots, \tau^n B_1 \ ; \quad \tau B_2, \tau^2 B_2, \ldots, \tau^k B_2\right) \in \mathbb{G}_1^n \times \mathbb{G}_2^k.$$

The value $\tau \in \mathbb{Z}_p$ is the trapdoor: it should be randomly generated and unknown to anybody. The structure of this string enables efficient *re-randomization*. Without knowing τ, it is possible to take an existing string pp and produce a new randomized string pp' by choosing a new random value τ' and multiplying each component of pp by an appropriate power of τ'. The new trapdoor will be $\tau \cdot \tau'$, which is secure as long as *either* τ or τ' are unknown and neither of them is zero.

This re-randomizability leads to a simple serial MPC protocol in which each participant in turn re-randomizes the string. Note that this can be done on an ongoing (or "perpetual") basis, as new participants can continue to join and re-randomize the string for future use. As long as each participant re-randomizes correctly and at least one participant destroys their local randomness, the cumulatively constructed string will be secure.

Applications. Powers-of-tau setup is required for many protocols:

- The KZG polynomial commitment scheme [45] requires a setup of n powers of tau in any one of the groups (e.g., $\mathbb{G}_1$), plus one power of tau in the other group (e.g., $\mathbb{G}_2$).
- SNARKs built from the KZG univariate polynomial commitment scheme, such as Sonic [56], Plonk [36], and Marlin [27], require a powers-of-tau string proportional in length to the size of the statement being proved.
- KZG commitments are also used in Verkle trees [49,52], a bandwidth-efficient alternative to Merkle trees. Unlike a binary Merkle tree, a Verkle tree is a b-ary tree, where each node is a vector commitment to up to b children. While Merkle trees have $O(\log_2 n)$ inclusion proof size, where n is the number of nodes, Verkle trees have $O(\log_b n)$ inclusion proof size. The most efficient Verkle trees, e.g. BalanceProofs [64], are based on KZG polynomial commitments requiring a powers-of-tau setup.

- Fast proofs of multi-scalar multiplication (MSM) over arbitrary groups of size $O(\log d)$ are possible using a powers-of-tau setup of length $O(\sqrt{d})$, where d is the number of scalars and group elements [17].
- The recent Danksharding proposal [21] for sharding Ethereum relies on a powers-of-tau string with 4096 elements in $\mathbb{G}_1$ and 64 in $\mathbb{G}_2$.

Challenges to decentralization Historically, ceremonies have been administered by a centralized coordinator which ensures several important properties, all of which we seek to achieve in a decentralized fashion:

- **Consensus:** All participants should agree on the final value of pp.
- **Validity:** Each participant should only be able to re-randomize the current string (and not simply replace it with one for which the trapdoor is known).
- **Data Availability:** The final string must be available for all to download, as well as the history of prior versions and participants for auditability.
- **Censorship Resistance:** Any willing participant should be able to contribute.

In this work we demonstrate how to replace the centralized coordinator with a smart contract, observing that blockchain platforms are designed to provide most or all of our desired properties. In particular, blockchains inherently provide consensus, previously done by fiat of the central coordinator, as well as censorship resistance, which has not been an explicit goal of centrally coordinated ceremonies. Validity and data availability are more interesting and provide several design options. For validity, we can rely on on-chain (Layer-1) verification of zero-knowledge proofs that each update is valid, or (to reduce costs) use a Layer-2 approach. We also show that it is possible (and even cheaper) to defer this task to users, who will verify the string before using it, which may be preferable in some settings. Similarly, for data availability we might post the full string pp on chain or, for efficiency, post only a commitment and rely on an external data-availability layer.

Contributions. We design ceremonies with two data-availability models: one with the entire string pp posted on-chain, and one with only a commitment to pp, namely $c = \mathsf{H}(\mathsf{pp})$, posted on-chain and the full string stored in an external data-availability system. See Table 1 highlighting the properties of the two models that we develop. The latter can offer significant cost savings for large strings as on-chain data storage is expensive.

With data available on-chain, we present an efficient pairing-based proof construction for verifying each participant's contribution (Section 4). We implemented this protocol for the Ethereum blockchain, coding in Solidity and using the BN254 curve. We describe our implemention in Section 6; we have also released our open-source implementation (link). Participating in the ceremony costs 190,000 to 11,500,000 gas (about $5 to $315 at current Ethereum prices), depending on the size of the desired resulting parameters (in this case between 8 and 1024 powers-of-tau). The size of the setup is limited but can still be used

to power Verkle trees, data-availability sampling, and zero-knowledge SNARKs for small statements.

For larger strings, we develop methods that have on-chain verification, yet only store a short commitment to the full setup on-chain (see Section 5). We discuss how to make the data-availability solutions that can facilitate such setups light-weight. The data-availability service only needs to be able to produce a commitment over the data of an appropriate form and store at most two latest contributions.

Paper organization. We discuss related work and some historical notes on setup ceremonies in Section 2. In Section 4, we present our fully on-chain protocol for powers-of-tau setup. In Section 5, we discuss several protocols for powers-of-tau setup with off-chain data availability, supporting larger structured reference strings. In Section 6, we describe our practical implementation and performance evaluation of the fully on-chain protocol on top of Ethereum. Finally we conclude in Section 7 by discussing various practical concerns and possible extensions, including censorship resistance, incentives and methods to lower on-chain cost through roll-ups, optimistic verification, batching, IVC and other techniques.

Data availability	Commitment scheme	Section	Proof size	Verifier time
On-chain	none	4	$O_\lambda(1)$	$O_\lambda(n)$
Off-chain	Any commitment	5.1	$O_\lambda(\log n)$	$O_\lambda(\log n)$
	AFGHO unstructured commitment	5.2	$O_\lambda(\log n)$	$O_\lambda(\log n)$

Fig. 1. Comparing on-chain powers-of-tau of length n to off-chain powers-of-tau with an on-chain commitment. On-chain storage requires linear on-chain work to verify an update. With off-chain storage we require only logarithmic on-chain work to verify an update. The AFGHO-based proof in the third row performs better in practice than the generic proof in the second row.

2 Related work

2.1 Multiparty setup ceremonies

Generically, any trusted setup algorithm can be implemented via secure multi-party computation (MPC) to prevent any single entitity from learning the trap-door. Ben-Sasson et al. [11] proposed the first multi-party protocol to sample public parameters for a zero-knowledge proof scheme which was instantiated for Zcash Sprout. Although this ceremony was not instantiating the powers-of-tau, it paved the way for crowd-sourcing subsequent ceremonies.

Bowe et al. [15] designed a protocol for Groth16 [41], where constructing a powers-of-tau public string was part of one of two phases. The protocol however

required a random beacon, an auxiliary process that produces publicly verifiable unpredictable and unbiasable randomness. Kohlweiss, Maller, Siim, and Volkhov [48] removed the need for a random beacon in the setup by proving that the setup remains secure for use with zero-knowledge proofs even if the public parameters have some degree of bias. Cohen et al. [29] demonstrated that the KZG commitments also remain secure in case the public parameters have bounded bias, thus similarly eliminating the need to use the random beacons for setups to be used for KZG commitments. Ganesh et al. [37] gave a UC secure protocol for Groth16 setup. Kerber et al. [46] builds a UC secure protocol for updatable SRS into a Nakamoto consensus protocol itself: a block proposer updates the SRS, security relies on chain quality. In contrast, our protocol is chain agnostic, is open to wide participation from users (not just from miners or validators), and has optimized on-chain verification cost. The work of Groth, Kohlweiss, Maller, Meiklejohn, and Miers [42] introduced an updatable SRS model, they construct a SNARK where the SRS can be updated by anybody. The security is guaranteed as long as at least one of the contributors is honest. The generated setup string is different from the powers-of-tau, and the paper is not focusing on on-chain/off-chain deployments or optimizing the verification.

All of these protocols fall in a category of the more-the-merrier protocols, as they each require only a single one honest participant to be secure. However, all were built with the assumption of a central coordinator. Buterin [18] suggested a simple way to verify the update to the setup that, as we observe in this work, opens the possibility for a gas-efficient on-chain deployment which we base our on-chain protocol on.

Multiparty setup ceremonies have also been demonstrated for RSA-style parameter setup [14,55,38,35,43]. Chen et al. [26] demonstrated a multiparty protocol for sampling a 2,048 bit RSA modulus which can scale to thousands of participants and only requires a single honest participant for security.

2.2 Setup ceremonies in practice

Some of the most prominent ceremonies have been run by Zcash, a privacy-oriented blockchain project. Six participants carried out the first Zcash ceremony, Sprout, in 2016, and 90 participants built parameters for a Sapling upgrade in 2018.

The perpetual "powers-of-tau" ceremony was first run in a continuous mode, where contributions are still being accepted, by the team of the Semaphore project, a privacy preserving technology for anonymous signaling on Ethereum. The setup uses a BN254 elliptic curve and has had 71 participants so far. Other prominent projects later used this setup to run their own ceremonies on top, including Tornado.Cash [24] , Hermez network [44], and Loopring [32]. Similar ceremonies on other curves were run by Aztec [6] for zkSync, a "layer two" Ethereum scaling solution that uses zero knowledge rollups; by Filecoin [33], a decentralized data storage protocol; by Celo [25], a layer-1 blockchain, for their light-client Plumo; Aleo [3], a blockchain for private applications.

Ethereum is currently running [34] a smaller trusted setup ceremony for its upcoming ProtoDankSharding and DankSharding upgrades: the targeted sizes are $2^{12}, 2^{13}, 2^{14}, 2^{15}$ powers in $\mathbb{G}_1$ and 64 powers in $\mathbb{G}_2$, over the BLS12 -381 curve. Those two upgrades will increase the amount of data that the Ethereum chain provides to clients for storage. This data will have a suggested expiry 30–60 days, it will not be accessible for the smart contracts in full, except for short KZG-commitments to the data. With around 95,000 contributions since its start in Jan 13th, 2023, it is the largest trusted setup ceremony to date in terms of participation.

2.3 Proof systems with transparent setup

It is important to note that there has been considerable research effort aimed at building cryptographic systems with fully *transparent* setup; that is, setup in which there is no trapdoor at all and therefore no trust assumption is required for the setup ceremony. A notable effort in that direction comes from a partnership of Electric Coin Company, Protocol Labs, the Filecoin Foundation, and the Ethereum Foundation, who collaboratively work on the Halo2 proof-system [30] that does not require a trusted setup. Halo2 powers the ZCash cryptocurrency since Zcash Network Upgrade 5 (NU5) activated on mainnet on May 31, 2022.

Similarly, transparent setup is possible to replace RSA-style trusted setup, using *class groups* of imaginary quadratic order instead of the group $\mathbb{Z}_N^*$ for a large composite modulus N [54]. The Chia blockchain [28] utilizes class groups and randomly re-samples the group parameters periodically, avoiding the need for trusted setup.

However, known trustless systems don't match the efficiency of the ones based on a trusted setup: the zk-snarks have poly-logarithmic-time verification (e.g. Halo2 and STARKs) compared to constant-time (e.g. Groth16, Plonk, Marlin), and polynomial commitments have poly-logarithmic-size evaluation-opening proofs (e.g. FRI, Dory) compared to constant-size proofs (e.g. KZG). It remains to be an open problem and an impactful research direction to come up with a system for the aforementioned applications that does not require a trusted setup while providing constant-time verification, or alternatively prove an impossibility result in this regard. In the meanwhile, a unified framework for running setup ceremonies in a transparent, verifiable and censorship-resistant manner would help bootstrap more efficient cryptosystems.

3 A Powers-of-Tau System: definitions

Our goal is to construct a "powers of τ" SRS of the following form:

$$\mathsf{pp} = (\tau B_1, \tau^2 B_1, \tau^3 B_1, \ldots, \tau^n B_1; \quad \tau B_2, \tau^2 B_2, \ldots, \tau^k B_2) \in \mathbb{G}_1^n \times \mathbb{G}_2^k \ , \quad (3.1)$$

where τ is unknown. We will show below that a computationally-limited verifier (e.g. a smart contract) can use the pairing to efficiently verify that pp is *well formed*, namely there exists a $\tau \in \mathbb{Z}_p^*$ such that pp satisfies (3.1).

Note that some applications require powers-of-τ strings in slightly different forms. Our techniques can generally be adapted and we focus on this simplest form. A notable case is "punctured" powers-of-τ strings which are missing a specific element. We discuss this case in Appendix D.

Our goal is to construct pp using a sequential multi-party computation between m contributors in m rounds, such that contributor number j contributes only in round number j, and does nothing in all other rounds. Each contributor can efficiently prove that their participation was correct. The main challenge is to ensure that the value of τ is unknown even if all but one of the contributors are malicious. In this way it is possible to conduct a permissionless setup in which any contributor is free to contribute, mediated by a smart contract which verifies each participant's contribution. Using a smart contract as the mediator ensures that anyone who wants to contribute can.

Notation. We use $\lambda \in \mathbb{Z}^+$ to denote the security parameter. We use $x \leftarrow y$ to denote the assignment of the value of y to x, and write $x \xleftarrow{\$} S$ to denote sampling an element from the set S independently and uniformly at random. For a positive integer p we use $\mathbb{Z}_p$ to denote the ring $\mathbb{Z}/p\mathbb{Z}$. We write $\mathbb{Z}_p^*$ for the set of non-zero elements in $\mathbb{Z}_p$. For a positive integer m we use $[m]$ to denote the set $\{1, \ldots, m\}$. We use $\mathrm{poly}(\lambda)$ and $\mathrm{negl}(\lambda)$ to denote a polynomial function and a negiligible function in the security parameter λ, respectively.

Definition 1. *A Powers-of-Tau system is a triple of poly-time algorithms:*

- $\mathsf{GlobalSetup}(1^\lambda, n, k) \to \mathsf{par}$. *The algorithm generates global parameters* par *that describe the three bilinear groups* $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$*, each of prime order* p*, with generators* B_1, B_2, B_T *respectively, equipped with an efficiently computable non-degenerate bilinear pairing* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$*. These parameters are an implicit input to the remaining algorithms.*
- $\mathsf{Update}(\mathsf{pp}, r) \to (\mathsf{pp}', \pi)$*. The algorithm uses the provided randomness* $r \in \mathbb{Z}_p^*$ *to update the powers-of-tau* pp *to* pp' *along with a proof* π *that the update was done correctly.*
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pp}', \pi) \to \{0, 1\}$*. The algorithm checks the proof* π *and outputs 1 to accept the update.*

We require that for all supported (n, k)*, all* par *output by* $\mathsf{GlobalSetup}(1^\lambda, n, k)$*, all* $\mathsf{pp} \in \mathbb{G}_1^n \times \mathbb{G}_2^k$ *of the form* (3.1)*, and all* $r \in \mathbb{Z}_p^*$*, we have*

$$\text{if } (\mathsf{pp}', \pi) \leftarrow \mathsf{Update}(\mathsf{pp}, r) \text{ then } \mathsf{Verify}(\mathsf{pp}, \mathsf{pp}', \pi) = 1.$$

The $\mathsf{GlobalSetup}$ algorithm need only be run once and can be reused for multiple powers-of-tau setups. It is not a trusted setup in that no secret randomness is required. $\mathsf{GlobalSetup}$ utilizes an algorithm $\mathsf{GroupGen}(1^\lambda)$ to generate the three additive pairing groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and their generators.

The Verify algorithm runs on chain and must therefore be as efficient as possible to reduce transaction costs. We next define the initial state of the system and the security requirements.

Initialization. The Powers-of-Tau system begins with an initial state defined as:

$$\mathsf{pp}_0 := (B_1,B_1,B_1,\ldots,B_1;\quad B_2,B_2,\ldots,B_2) \in \mathbb{G}_1^n \times \mathbb{G}_2^k \ . \tag{3.2}$$

This is equivalent to an SRS with $\tau = 1$.

Security. We define security of a Powers-of-Tau system $(\mathsf{Setup}, \mathsf{Update}, \mathsf{Verify})$ using a game that captures a setting where the adversary controls all the contributors except for one honest contributor. The game is stated with respect to some predicate

$$\Pi : \mathbb{Z}_p \times \mathcal{W} \to \{0,1\} \ .$$

At the end of the game the adversary outputs some $w \in \mathcal{W}$ and wins the game if $\Pi(\tau, w) = 1$, where τ is the secret exponent used to define the final powers-of-tau. This w represents some information that $\mathcal{A}$ was able to learn about τ. We give examples of some important predicates Π after the definition.

Since the prime p is determined by the security parameter, we define security with respect to a family of predicates $\Pi = \left\{\Pi_p : \mathbb{Z}_p \times \mathcal{W} \to \{0,1\}\right\}_{p\in\mathcal{P}}$ where $\mathcal{P}$ is the set of all integer primes. We say that Π is poly-time if there is an algorithm that for all p, τ, w evaluates $\Pi_p(\tau, w)$ in polynomial time in the security parameter λ.

Formally, Π-security is defined using a game between an adversary $\mathcal{A}$ and a challenger. The game is parameterized by (n, k) and proceeds as follows:

- The challenger runs $\mathsf{GlobalSetup}(1^\lambda, n, k)$ and sends the resulting global parameters par to $\mathcal{A}$. This defines pp_0.
- $\mathcal{A}$ outputs a sequence of pairs $(\mathsf{pp}_1, \pi_1), \ldots, (\mathsf{pp}_\ell, \pi_\ell)$.
- The challenger samples $r \xleftarrow{\$} \mathbb{Z}_p^*$, runs $\mathsf{Update}(\mathsf{pp}_\ell, r)$ to get $(\mathsf{pp}_{\ell+1}, \pi_{\ell+1})$, and sends $(\mathsf{pp}_{\ell+1}, \pi_{\ell+1})$ to $\mathcal{A}$. This emulates an honest contributor.
- Adversary $\mathcal{A}$ outputs a further sequence of pairs $(\mathsf{pp}_{\ell+2}, \pi_{\ell+2}), \ldots, (\mathsf{pp}_m, \pi_m)$ along with a guess $w \in \mathcal{W}$.

The adversary wins if $\mathsf{Verify}(\mathsf{pp}_{i-1}, \mathsf{pp}_i, \pi_i) = 1$ for all $i \in [m]$, and either (i) $\Pi_p(\tau_m, w) = 1$, where τ_m is the secret exponent that defines pp_m, or (ii) pp_m is a malformed powers-of-tau.

We will show in Theorem 2 below how to use the pairing to efficiently test that pp_m is a *well formed* powers-of-tau. Hence, as long as Verify includes this test, the only way for $\mathcal{A}$ to win the game is to output some $w \in \mathcal{W}$ such that $\Pi_p(\tau_m, w) = 1$.

Definition 2. *Let $\Pi = \left\{\Pi_p : \mathbb{Z}_p \times \mathcal{W} \to \{0,1\}\right\}_{p\in\mathcal{P}}$ be a family of poly-time predicates. A Powers-of-Tau system is Π-secure if for all n, k that are poly(λ), and for all PPT adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ wins the Π-security game is a negligible function of the security parameter λ.*

Remark 1. Definition 2 requires that the adversary cannot compute some information about the final τ_m. It does not require τ_m to be close to uniform in $\mathbb{Z}_p^*$ because that is not possible to achieve in our settings. If the last contributor is malicious, it could cause τ_m to become non-uniform in $\mathbb{Z}_p^*$ by repeatedly running the update procedure until the resulting pp satisfies some property (for example, the first ten bits of the first element in pp are zero).

Despite Remark 1, our definitional framework is sufficient for many applications. For example, suppose that the powers-of-tau is to be used in a KZG polynomial commitment scheme [45], and we need to ensure *evaluation binding*, meaning that a committed polynomial cannot be convincingly opened to two different values at one input. To do so, let us define the family of predicates Π_p^{SDH} where

$$\Pi_p^{\text{SDH}}\big(\tau \in \mathbb{Z}_p,\ (c,T) \in \mathbb{Z}_p \times \mathbb{G}_1\big) = 1 \quad \Longleftrightarrow \quad T = \Big(\tfrac{1}{\tau+c}\Big) B_1\ . \tag{3.3}$$

Suppose that no PPT algorithm that takes a powers-of-tau string as input, can find a pair (c,T) that satisfies this predicate. Then it is not difficult to show that this implies evaluation binding for KZG. Hence, a powers-of-tau string that is generated by a Π^{SDH}-secure powers-of-tau system can be safely used to provide evaluation binding in KZG.

Note that the predicate Π_p^{SDH} can be checked in polynomial time using the element $Q_1 := \tau B_2$ from the powers-of-tau string because

$$\Pi_p^{\text{SDH}}\big(\tau,\ (c,T)\big) = 1 \quad \Longleftrightarrow \quad e(T,\ Q_1 + cB_2) = e(B_1, B_2)\ .$$

We will come back to this predicate when we analyze security of our powers-of-tau system. Other applications that require a powers-of-tau string can choose to use other predicates to argue security.

4 Powers-of-tau setup with full data on-chain

We now describe the Update and Verify algorithms for our powers-of-tau system, when the entire string pp is stored on chain. This is the simplest construction, though may carry high costs for large powers-of-τ strings as it requires the verifier to do linear work (in n and k) for each update.

Let pp be the current SRS string which is assumed to be:

$$\begin{aligned} \mathsf{pp} &= (\ P_1,\ \ P_2, \ldots,\ \ P_n;\ \ Q_1, \ldots,\ \ Q_k) \\ &= (\tau B_1, \tau^2 B_1, \ldots, \tau^n B_1;\ \ \tau B_2, \ldots, \tau^k B_2) \end{aligned} \tag{4.1}$$

for some (unknown) τ in $\mathbb{Z}_p^*$.

Let r be a random element in $\mathbb{Z}_p^*$. The $\mathsf{Update}(\mathsf{pp}, r)$ algorithm begins by computing the updated SRS string pp' as

$$\begin{aligned} \mathsf{pp}' &:= (P_1',\ \ P_2',\ \ \ldots, P_n';\ \ \ Q_1',\ \ \ldots, Q_k') \\ &= (rP_1, r^2 P_2,\ \ldots, r^n P_n;\ \ rQ_1,\ \ldots, r^k Q_k) \end{aligned} \tag{4.2}$$

Observe that

$$\begin{aligned}\mathsf{pp}' &= (r\tau B_1,\ r^2\tau^2 B_1,\ \ldots,\ r^n\tau^n B_1;\qquad r\tau B_2,\ \ldots,\ r^k\tau^k B_2)\\ &= (\tau' B_1,\ (\tau')^2 B_1,\ \ldots,\ (\tau')^n B_1;\qquad \tau' B_2,\ \ldots,\ (\tau')^k B_2)\end{aligned}$$

where $\tau' := r \cdot \tau$ is the secret exponent[4] for pp'. If an attacker knows τ but not r, and r was chosen uniformly at random from $\mathbb{Z}_p^*$ (meaning in particular that $r \neq 0$), then the attacker will have no information about τ. Consequently, if at least one of the contributors samples their update r randomly, and properly destroys it, then the final secret $\tau_m = r_1 \cdot r_2 \cdot \ldots \cdot r_m \in \mathbb{Z}_p^*$ is randomly distributed and unknown to anyone. This is assuming that none of the contributors set $r_i = 0$, which is easy to check for.

Update proofs. Next, the $\mathsf{Update}(\mathsf{pp}, r)$ algorithm needs to output a proof that the update was done correctly. In particular, the verify algorithm will need to convince itself of the following three claims:

Check #1 - the contributor knows r: this is needed to ensure that the latest update builds on the work of the preceding participants.

Check #2 - the new parameters pp' are well-formed: there is some $\tau' \in \mathbb{Z}_p$ such that pp' satisfies (3.1).

Check #3 - pp' is not degenerate, namely $r \neq 0$: defends against an update trying to erase the setup thus undermining the contributions of previous participants.

We will show that the verifier can efficiently check claims #2 and #3 on its own.

We first explain how to efficiently prove claim #1. To provide a zero-knowledge proof of knowledge of r, the $\mathsf{Update}(\mathsf{pp}, r)$ algorithm has two options: it can use a Fiat-Shamir version of Schnorr's Σ-protocol [58,59] or it can use a BLS-style proof of possession [57] for r. The latter is more expensive to verify on-chain as it requires the verifier to compute pairings. We therefore focus on the former approach which works as follows:

$\mathsf{Update}(\mathsf{pp}, r)$ samples a random $z \xleftarrow{\$} \mathbb{Z}_p^*$, computes

$$h \leftarrow \mathsf{HASH}(P_1' \,||\, P_1 \,||\, z \cdot P_1) \quad \text{and} \quad \pi \leftarrow (z \cdot P_1,\ \ z + h \cdot r) \in \mathbb{G}_1 \times \mathbb{Z}_p,$$

and outputs the proof $\pi \in \mathbb{G}_1 \times \mathbb{Z}_p$. Here HASH is a hash function that outputs elements in $\mathbb{Z}_p$. In the security proof we will model HASH as a random oracle.

The $\mathsf{Verify}(\mathsf{pp}, \mathsf{pp}', \pi)$ algorithm (an on-chain smart contract) verifies the proof $\pi = (\pi_1, \pi_2) \in \mathbb{G}_1 \times \mathbb{Z}_p$ by checking that:

[4] Note that it is also possible to compute an additive update to the tau ($\tau' \leftarrow r + \tau$), however it would require the contributor to compute many multi-scalar multiplications making it less efficient.

Check # 1:	$\pi_2 \cdot P_1 = \pi_1 + \mathsf{HASH}(P_1' \,\|\|\, P_1 \,\|\, \pi_1) \cdot P_1$

We next show how to verify claims #2 and #3.

Definition 3. *We say that the string* $\mathsf{pp} = (P_1, P_2, P_3, \ldots, P_n; Q_1, Q_2, \ldots, Q_k)$ *is well-formed if there exists* $\tau \in \mathbb{Z}_p$ *such that* $P_i = \tau^i B_1$ *and* $Q_\ell = \tau^\ell B_2$ *for all* $i = 1 \ldots n$ *and* $\ell = 1 \ldots k$.

To verify that pp is well-formed, the verifier samples two random scalars $\rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_p^*$ and checks that:

$$\text{Check \# 2:} \quad e\Big(\textstyle\sum_{i=1}^{n} \rho_1^{i\text{-}1} P_i,\ \ B_2 + \sum_{\ell=1}^{k-1} \rho_2^\ell Q_\ell\Big) = e\Big(B_1 + \sum_{i=1}^{n-1} \rho_1^i P_i,\ \ \sum_{\ell=1}^{k} \rho_2^{\ell\text{-}1} Q_\ell\Big) \tag{4.3}$$

For a well-formed string pp the check will always pass successfully, since:

$$e\left(\tau B_1 + \sum_{i=1}^{n-1} \left(\rho_1^i \cdot \tau^{i+1} B_1\right),\ \ B_2 + \sum_{\ell=1}^{k-1} \left(\rho_2^\ell \tau^\ell B_2\right)\right) =$$
$$e\left(B_1 + \sum_{i=1}^{n-1} \left(\rho_1^i \cdot \tau^i B_1\right),\ \ \tau \cdot \left(B_2 + \sum_{\ell=1}^{k-1} \left(\rho_2^\ell \tau^\ell B_2\right)\right)\right)$$

We prove that this check is sound in Theorem 2 below.

One complication is that an on-chain verifier does not have access to secure randomness. Instead, it will generate the scalars $\rho_1, \rho_2 \in \mathbb{Z}_p$ by hashing the string submitted by the contributor as $\rho_1 \leftarrow \mathsf{HASH}(\mathsf{pp}'||1)$ and $\rho_2 \leftarrow \mathsf{HASH}(\mathsf{pp}'||2)$.

Finally to ensure that the updated setup is non-degenerative, the verifier simply checks that the first element in pp' non-zero:

Check #3:	$P_1' \neq 0$	(4.4)

Correctness: it is easy to check that the Update and Verify algorithms satisfy our correctness requirement.

4.1 Security

We now argue that the powers-of-tau system in the previous section satisfies the security definition (Definition 2). Recall that security is defined with respect to a poly-time predicate family $\Pi = \left\{\Pi_p : \mathbb{Z}_p \times \mathcal{W} \to \{0,1\}\right\}_{p \in \mathcal{P}}$. Let us first define the (n,k)-Π-DH assumption. The assumption says that no PPT adversary that takes a powers-of-tau string with secret exponent $\tau \in \mathbb{Z}_p$ as input, can find a $w \in \mathcal{W}$ such that $\Pi_p(\tau, w) = 1$.

Definition 4. *Let* $\Pi = \{\Pi_p : \mathbb{Z}_p \times \mathcal{W} \to \{0,1\}\}_{p\in\mathcal{P}}$ *be a poly-time predicate family. We say that the* (n,k)**-**Π**-DH assumption** *holds for the bilinear group generator* GroupGen *if for all PPT algorithms* $\mathcal{A}$*,*

$$\Pr\Big[\Pi_p\big(\tau,\ \mathcal{A}\big(par,\ \tau B_1, \tau^2 B_1, \ldots, \tau^n B_1,\ \tau B_2, \tau^2 B_2, \ldots, \tau^k B_2\big) = 1\big)\Big] \leq negl(\lambda),$$

where $par \xleftarrow{\$} \mathsf{GroupGen}(1^\lambda)$ *and* $\tau \xleftarrow{\$} \mathbb{Z}_p^*$*.*

The (n,k)-Π-DH assumption encompasses a large class of standard cryptographic assumptions. For example, taking Π to be the predicate family Π^{SDH} from (3.3) gives the so called (n,k)-Strong Diffie-Hellman (SDH) assumption [13].

Definition 5. *We say that the predicate family* $\Pi = \{\Pi_p : \mathbb{Z}_p \times \mathcal{W} \to \{0,1\}\}_{p\in\mathcal{P}}$ *is* **self reducible** *if there is a PPT algorithm* Reduce *such that for all* $p \in \mathcal{P}$*, all* $\tau, r \in \mathbb{Z}_p^*$*, and all* $w \in \mathcal{W}$ *we have*

$$\Pi_p(\tau, w) = 1 \quad \Longrightarrow \quad \Pi_p\big(\tau \cdot r,\ \mathsf{Reduce}(r, w)\big) = 1\ .$$

In other words, given a valid w for τ, algorithm Reduce outputs a valid w' for $\tau \cdot r$. For example, the predicate family Π^{SDH} from (3.3) is self reducible. To see why, observe that for all $p \in \mathcal{P}$ and $r \in \mathbb{Z}_p^*$ we have

$$\Pi_p^{\text{SDH}}\big(\tau,\ (c, T)\big) = 1 \quad \Longrightarrow \quad \Pi_p^{\text{SDH}}\big(\tau r,\ \ (cr,\ (1/r)\cdot T)\big) = 1$$

because

$$T = \tfrac{1}{\tau + c} \cdot B_1 \quad \Longrightarrow \quad \tfrac{1}{r} \cdot T = \tfrac{1}{\tau r + cr} \cdot B_1\ .$$

With these definitions in place, we can now state the security theorem.

Theorem 1. *Let* $\Pi = \{\Pi_p : \mathbb{Z}_p \times \mathcal{W} \to \{0,1\}\}_{p\in\mathcal{P}}$ *be a poly-time self reducible predicate family. Then the powers-of-tau system in Section 4 is* Π*-secure, as in Definition 2, assuming the* (n,k)*-*Π*-DH assumption holds for* GroupGen *and the hash function* HASH *is modeled as a random oracle.*

We give the proof intuition and defer the proof to the full version of the paper.

Proof idea. For now, let us assume that the proof system used in the powers-of-tau system is zero knowledge and simulation extractable [39] even for a prover that proves multiple statements one after the other. We will justify these two assumptions later on.

We are given an adversary $\mathcal{A}$ that wins the attack game in Definition 2 with non-negligible probability. By Theorem 2 below, the only way for $\mathcal{A}$ to win the game is to output some $w_m \in \mathcal{W}$ such that $\Pi_p(\tau_m, w_m) = 1$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the (n,k)-Π-DH assumption. Algorithm $\mathcal{B}$ is given as input an (n,k)-Π-DH challenge

$$\mathsf{pp}_{\text{chal}} := (P_1, \ldots, P_n; Q_1, \ldots, Q_k) \in \mathbb{G}_1^n \times \mathbb{G}_2^k\ .$$

It needs to find some $w \in \mathcal{W}$ such that $\Pi_p(\tau, w) = 1$, where $\tau \in \mathbb{Z}_p^*$ is the secret exponent used define this challenge. Algorithm $\mathcal{B}$ begins by running adversary $\mathcal{A}$ and the following happens:

- $\mathcal{B}$ receives from $\mathcal{A}$ a sequence of ℓ pairs $(\mathsf{pp}_1, \pi_1), \ldots, (\mathsf{pp}_\ell, \pi_\ell)$.
- $\mathcal{B}$ sends to $\mathcal{A}$ the pair $(\mathsf{pp}_{\text{chal}}, \pi)$ where π is a simulated proof that $\mathsf{pp}_{\text{chal}}$ is a valid update. Here we are using the zero knowledge property of the proof system.
- $\mathcal{B}$ receives from $\mathcal{A}$ an additional sequence of pairs $(\mathsf{pp}_{\ell+2}, \pi_{\ell+2}), \ldots, (\mathsf{pp}_m, \pi_m)$ along with a guess $w_m \in \mathcal{W}$.

Now $\mathcal{B}$ will use the extractor to extract from $\mathcal{A}$ all the randomizers $r_{\ell+2}, \ldots, r_m$ in $\mathbb{Z}_p^*$ that the adversary used to update the SRS in the second set of pairs that it output. To do so we are using the simulation extractability property of the proof system. Now, if all the extracted randomizers are correct, then

$$\tau_m = \tau \cdot (r_{\ell+2} \cdots r_m) \ ,$$

where τ_m is the exponent used to define pp_m. Moreover, if w_m output by $\mathcal{A}$ indeed satisfies $\Pi_p(\tau_m, w_m) = 1$, then by the self reducibility of Π, our $\mathcal{B}$ can efficiently find a w such that $\Pi_p(\tau, w) = 1$, as required. □

It remains to argue that the proof system used in our powers-of-tau system is zero knowledge and simulation extractable. We first show that the verifier's Check #2 is sound, namely, a malformed string pp will fail the check with overwhelming probability.

Theorem 2. *Check #2 ensures the well-formedness of* pp. *In particular, let* p *be the size of the groups ouptut by* $\mathsf{GroupGen}(1^\lambda)$, *and let* n *and* k *be polynomial in the security parameter* λ. *Then a malformed* pp *will pass Check #2 with probability at most* $\frac{(n-1)(k-1)}{p}$, *which is negligible in* λ.

The proof of this theorem can be found in Appendix A.

We next briefly argue that the proof system used in our powers-of-tau system is zero knowledge and simulation extractable. The proof output by algorithm $\mathsf{Update}(\mathsf{pp}, r)$ is a standard Schnorr proof of knowledge of discrete log that is made non-interactive using the Fiat-Shamir transform. This proof system is known to be zero-knowledge in the random oracle model, and simulation extractable in the random oracle model even for a prover that proves multiple such statements one after the other [37]. Moreover, Theorem 2 shows that a witness extracted from a convincing prover will correspond to a valid witness with overwhelming probability.

5 Powers-of-tau setup protocol with data off-chain

The required number of powers of tau for some applications can be as high as 2^{24}–2^{28}, resulting in public parameters of size in the range 0.5GB–9GB. This rules out the possibility of storing the full parameters on chain, given limitations of today's Layer-1 smart contract platforms. However, it is still possible to take advantage of the anti-censorship properties of an L1 chain by posting a *commitment* to the

parameters on chain, while storing the parameters off chain. Each contributor who updates the on-chain commitment proves that the update to the current off-chain parameters is well-formed by submitting a ZK proof to the smart contract. The contract accepts the contribution if the proof is valid.

In more detail, let Alice be the i-th contributor to the powers-of-tau. Let pp_i be the powers-of-tau before Alice's contribution and let pp_{i+1} be the powers-of-tau after. Prior to Alice's contribution, the smart contract holds a short binding commitment to pp_i, namely $c_i := H(\mathsf{pp}_i)$, for some collision resistant hash function H. Alice will send to the contract $c_{i+1} := H(\mathsf{pp}_{i+1})$ along with a succinct ZK proof π that the transition from c_i to c_{i+1} is well formed, as discussed in more detail in the next subsection. If the proof is valid, the contract updates the stored hash to c_{i+1} and erases c_i. Note that the contract places c_{i+1} in its storage array; however the proof π need only be sent to the contract as call data and does not need to be written to the contract's storage.

We describe three ways to produce the proof π: in Section 5.1 using a generic transparent SNARK; in Section 5.2 using the Dory polynomial commitment scheme; and in Appendix C using an inferior method of inner-pairing product argument.

Data availability. If the L1 chain only holds a hash of the powers-of-tau, then the actual data must be kept elsewhere. One can use a centralized data-availability (DA) service, such as a cloud storage provider, or a decentralized one, such as EigenDA, Celestia, Polygon Avail, or Arweave. These data availability services vary in many respects, including the precise guarantees and pricing model, but they all commit to storing a large blob of data and making it publicly available, in exchange for fees. In the DA service the data is typically addressable by its hash-digest or a deterministic commitment. Updates can write a new copy of the data to the DA service and old versions will still exist. Regardless, of how the DA service is run, we only require it to attest to the availability of the data behind the on-chain commitment, we assume that the DA service is censorship-resistant and append-only. The DA service does not need to run any verification on the underlying data.

Note that the DA service can safely discard an old parameter set after the chain verifies a new parameter set, meaning that the DA service only needs to store at most two parameter sets at any given time, meaning it scales well to protocols with many participants.

5.1 Off-chain setup using a transparent succinct proof

Let pp be the current state of the powers-of-tau stored at some data availability service, and let $c := H(\mathsf{pp})$ be the commitment to pp stored in the smart contract on chain. Recall that

$$\begin{aligned}\mathsf{pp} &= \Big(P_1, P_2, P_3, \ldots, P_n; \quad Q_1, Q_2, \ldots, Q_k) = \\ &= \Big(\tau B_1, \tau^2 B_1, \tau^3 B_1, \ldots, \tau^n B_1; \quad \tau B_2, \tau^2 B_2, \ldots, \tau^k B_2\Big) \in \mathbb{G}_1^n \times \mathbb{G}_2^k\end{aligned}$$

for some secret $\tau \in \mathbb{Z}_p$ and public $B_1 \in \mathbb{G}_1$, $B_2 \in \mathbb{G}_2$.

Alice wants to re-randomize pp to obtain pp'. She chooses a random $r \in \mathbb{Z}_p$, computes

$$\mathsf{pp}' \leftarrow \Big(rP_1, r^2P_2, r^3P_3, \ldots, r^nP_n; \quad rQ_1, r^2Q_2, \ldots, r^kQ_k\Big) =$$
$$= \Big(P'_1, P'_2, P'_3, \ldots, P'_n; \quad Q'_1, Q'_2, \ldots, Q'_k\Big) \in \mathbb{G}_1^n \times \mathbb{G}_2^k$$

and sends pp' to the data availability service. Next, she computes the commitment $c' = H(\mathsf{pp}')$ and needs to convince the on-chain smart contract that the transition from c to c' is a valid transition. As explained in Section 4, Alice must produce a succinct zero-knowledge argument of knowledge (zk-SNARK) that the following relation holds, for random ρ_1, ρ_2 in $\mathbb{Z}_p$ chosen by the verifier:

$$\text{public statement: } c,\ c' \text{ and } \rho_1, \rho_2 \in \mathbb{Z}_p\ , \quad \text{witness: } \mathsf{pp},\ \mathsf{pp}', \text{ and } r \in \mathbb{Z}_p\ ,$$

and the relation is satisfied if and only if

$$c = H(\mathsf{pp}), \quad c' = H(\mathsf{pp}'), \quad P'_1 = rP_1, \quad P'_1 \neq 0, \quad \text{and}$$
$$e\Big(\sum_{i=1}^{n} \rho_1^i P'_i, \quad \rho_2 B_2 + \sum_{j=1}^{k-1} \rho_2^{j+1} Q'_j\Big) = e\Big(\rho_1 B_1 + \sum_{i=1}^{n-1} \rho_1^{i+1} P'_i, \quad \sum_{j=1}^{k} \rho_2^j Q'_j\Big).$$

Note that the zero-knowledge property is needed to keep r secret.

The simplest, though not the most efficient, way to produce a succinct proof for this relation is to use a generic zk-SNARK system (we describe better approaches in the next subsection). To use a generic zk-SNARK, we need a proof system with the following properties: (i) *transparent*, namely the zk-SNARK requires no trusted setup, since we cannot assume the existence of a trusted setup in our settings; (ii) short, to reduce the cost of posting the proof on-chain; and (iii) fast to verify, to reduce the on-chain gas costs for verification. The STARK system [10] meets these requirements. In practice, the resulting proof is about 100KB which may be too expensive to post on chain for every update. In Section 7 we discuss batching proofs, namely supporting multiple updates using a single proof. This may make STARKs a viable option.

Once Alice constructs the proof π, she sends (c, c', π) to the on-chain contract. The contract verifies the proof, and if valid, it replaces c by c'.

5.2 Off-chain setup using AFGHO commitments on-chain

In this section we describe a more efficient approach than the one in the previous section. We use the unstructured AFGHO commitments of Abe et al. [1] in combination with the Dory [51] inner-pairing product arguments. This leads to short and efficiently verifiable proofs on chain.

We again assume groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of a prime order p and a bilinear operation $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We adopt the product notation for pairing operations: for vectors $\mathbf{A} \in \mathbb{G}_1^n$ and $\mathbf{B} \in \mathbb{G}_2^n$ we write $\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{i=1}^{n} e(A_i, B_i)$. Let $\mathbf{\Gamma}_2 \in \mathbb{G}_2^n$

be generators of $\mathbb{G}_2$ and $\mathbf{\Gamma}_1 \in \mathbb{G}_1^k$ be generators of $\mathbb{G}_1$, all randomly chosen in a transparent way.

Instead of the full parameters $\mathsf{pp} = (\mathbf{P}; \mathbf{Q}) = ((P_1, P_2, \ldots, P_n); (Q_1, Q_2, \ldots, Q_k))$, the chain only stores P_1 and AFGHO commitments $(C_1, C_2) \in \mathbb{G}_T \times \mathbb{G}_T$ on chain, where $C_1 = \langle \mathbf{P}, \mathbf{\Gamma}_2 \rangle \in \mathbb{G}_T$ and $C_2 = \langle \mathbf{\Gamma}_1, \mathbf{Q} \rangle \in \mathbb{G}_T$.

The contributor submits a proof-of-knowledge of the discrete log of the update to P_1 as explained in Check #1 of Section 4 and a logarithmic-size proof for the following inner-pairing product (IPP) relations:

$$\begin{aligned} &C_1 = \langle \mathbf{P}, \mathbf{\Gamma}_2 \rangle \ \wedge \ C_2 = \langle \mathbf{\Gamma}_1, \mathbf{Q} \rangle \ \wedge \\ &\rho_1^n P_n Q_1 - B_1 Q_1 = \langle \mathbf{P}, (1, \rho_1, \rho_1^2, \ldots, \rho_1^{n-1}) \cdot (\rho_1 Q_1 - B_2) \rangle \ \wedge \\ &\rho_2^k P_1 Q_k - P_1 B_2 = \langle (1, \rho_2, \rho_2^2, \ldots, \rho_2^{k-1}) \cdot (\rho_2 P_1 - B_1), \mathbf{Q} \rangle \ \wedge \\ &P_n = \langle \mathbf{P}, (0, 0, \ldots, 0, 1) \rangle \ \wedge \ P_1 = \langle \mathbf{P}, (1, 0, \ldots, 0, 0) \rangle \\ &Q_k = \langle \mathbf{Q}, (0, 0, \ldots, 0, 1) \rangle \ \wedge \ Q_1 = \langle \mathbf{Q}, (1, 0, \ldots, 0, 0) \rangle \end{aligned} \tag{5.1}$$

We give further details on this construction in Appendix B.

6 Implementation and Evaluation on Ethereum

In this section, we analyse the practicality of our fully on-chain setup ceremony, presented in Section 4. We implemented our protocol on top of Ethereum [20], the most popular smart contract platform. Currently (as of May 2023), Ethereum natively supports only one group with bilinear pairing, BN254 (the initial EIP-197 [62] describes the curve equations). This group is foundational to multiple projects (e.g. Aztec, zkSync) although unfortunately its security has been lowered with recent attacks [7], and now estimated [47] to be at 100-bits level. Ethereum consensus layer uses BLS12-381, which is another pairing-friendly group, and also a popular choice for other projects (e.g. Aztec and Filecoin), has stronger security guarantees, however the precompiles for this curve are not available on Ethereum yet, though have been suggested (EIP-2537 [4]) alongside precompiles for other pairing-friendly curves BLS12-377 (EIP-2539 [63]) and BW6-761 (EIP-3026 [66]). The supported operations are scalar-multiplication and addition in $\mathbb{G}_1$ and a pairing precompile, which are priced as follows according to EIP-1108 [23]:

Name	Operation	Gas cost
ECADD	$A + B$ for $A, B \in \mathbb{G}_1$	150
ECMULT	αA for $\alpha \in \mathbb{Z}_p, A \in \mathbb{G}_1$	6,000
ECPAIR	$\sum_{i=1}^{k} e(A_i, B_i) = 0$ for $A_i \in \mathbb{G}_1, B_i \in \mathbb{G}_2$	$34,000 \cdot k + 45,000$

Each contribution is sent as calldata, which is a read-only byte array, currently priced at 16 gas per byte according to EIP-2028 [2].

n	8	16	32	64	128	256	512	1024
compute in gas units	179,000	227,000	323,000	515,000	899,000	1,667,000	3,203,000	6,275,000
compute cost	$5	$6	$9	$14	$25	$46	$89	$174
storage in gas units	8,192	16,384	32,768	65,536	131,072	262,144	524,288	1,048,576
storage cost	$0	$0	$1	$2	$4	$7	$15	$29
Total (estimates)	187,192	243,384	355,768	580,536	1,030,072	1,929,144	3,727,288	7,323,576
	$5	$7	$10	$16	$29	$54	$103	$203
Total (actual)	192,162	272,217	432,702	755,340	1,406,185	2,731,526	5,474,920	11,341,136
	$5	$8	$12	$21	$39	$76	$152	$315

Table 1. Estimates according to the Eq. 6.2 and actual costs. The pricing in USD is calculated based on rough numbers on 05/01/2023: 15 gwei per gas unit and 1 ETH = $1,850 (1 gwei = 10^{-9} ETH).

Fully on-chain setup for $k = 1$. We first consider a setup with a single element in $\mathbb{G}_2$. The following pre-computation will reduce the cost of the Check #2 to $n + 3$ scalar multiplications and one ECPAIR, though the check will remain to dominate the verification cost:

$$\text{Check \# 2 (more efficient): for } R := \sum_{i=1}^{n-1} \rho_1^{i\text{-}1} \cdot P_i,$$
$$\text{verify that } e(B_1 + \rho_1 R,\ Q_1) \quad = \quad e(R + \rho_1^{n-1} P_n,\ B_2) \tag{6.1}$$

The contributor submits $64 \cdot n + 224$ bytes of calldata: n elements of $\mathbb{G}_1$ (64 bytes, uncompressed[5]), 1 element in $\mathbb{G}_2$ (128 bytes, uncompressed), and a proof which consists of one element in $\mathbb{Z}_p$ and one element in $\mathbb{G}_1$. The cost of the contribution is therefore comprised of compute and calldata storage:

$$\begin{aligned} &\text{compute cost: } (n+3) \cdot 6,000 + 113,000 \text{ gas} \\ &\text{storage cost: } n \cdot 1,024 + 3,584 \text{ gas} \end{aligned} \tag{6.2}$$

It is instructive to notice that the cost of compute is roughly 6x the cost of storage. The compute is dominated by the multi-scalar multiplication. Most likely it is inevitable for each element of the setup to have to be multiplied by a scalar or be directly inserted into a pairing, it is therefore unlikely to be able to reduce the compute cost for the fully on-chain setup. However, using techniques of Bellare et al. [8] the scalar-multiplications might be substituted by λ-random

[5] Our evaluations showed that recovering element from a compressed form would cost significantly more than sending them in an uncompressed form directly.

subset sums for λ-security, however for Ethereum this trick does not bring any savings. Table 1 shows estimated and concrete pricing per contribution with a check from Eq. 6.1 based on our open-sourced implementation[6].

Fully on-chain setup for $k > 1$. Since Ethereum does not support addition and scalar multiplication in $\mathbb{G}_2$ the following alternative method for Check #2 targeting Ethereum can be used, it does one additional pairing per each power in $\mathbb{G}_2$:

Check #2 (alternative):

$$\text{For } R = \sum_{i=0}^{n-2} \rho^i \cdot P_{i+1,j} : e(B_1 + \rho R,\ Q_{1,j}) = e(R + \rho^{n-1} P_{n,j},\ B_2) \tag{6.3}$$

$$\text{For } t = 2..k\text{-}1 :\ e(P_{k\text{-}t,j}, Q_t) = e(P_{k,j}, B_2) \wedge e(B_1, Q_k) = e(P_{k,j}, B_2) \tag{6.4}$$

Note that the right-hand part of the equations 6.4 can be computed once. Note also that equations 6.3 and 6.4 are each checking the equalities of pairings, these checks can be batched using pseudorandom scalars $\alpha_0, \alpha_1, \ldots, \alpha_D \in \left(\mathbb{Z}_p^*\right)^n$ sampled as $\alpha_i = \mathsf{HASH}(\mathsf{pp}_j, i)$ to transform into a check of the sum of pairings which is cheaper to do on Ethereum (Ethereum has an opcode that allows to verify $e(A_1, B_1) + ... + e(A_m, B_m) = 0$):

$$\begin{array}{l} e(A_1, B_1) = e(C_1, D_1) \\ e(A_2, B_2) = e(C_2, D_2) \\ \quad \ldots \\ e(A_m, B_m) = e(C_m, D_m) \end{array} \Leftrightarrow \begin{cases} e(\alpha_1 A_1, B_1) - e(\alpha_1 C_1, D_1) + \\ \ e(\alpha_2 A_2, B_2) - e(\alpha_2 C_2, D_2) + \\ \ \ \ldots \\ \ e(\alpha_m A_m, B_m) - e(\alpha_m C_m, D_m) = 0 \end{cases} \tag{6.5}$$

Note on the use of hash functions for generation of scalars. For a 256-bits order groups, the hash function HASH needs to output 512-bits, should be given as inputs strings generated with invertible serialization method, and be domain-separated (i.e. the input should be prefixed with a fixed-length string indicating the step of the protocol and the purpose of hashing).

7 Concluding discussion and open problems

In conclusion, we note that our work shows the practicality of *decentralized* setup ceremonies for the first time. These protocols can scale to support an unlimited number of participants as blockchain performance continues to improve. Our protocols inherit (and rely on) the ability of the underlying blockchain to support open participation while managing potential spam and denial-of-service.

Given the more-the-merrier property of our protocols, these represent a qualitative security advance over the state of the art. While practical trusted setup ceremonies have attempted to recruit a diverse and trustworthy group of participants to convince the public that the results of the ceremony can be trusted,

[6] github.com/a16z/evm-powers-of-tau

decentralized setup ceremonies offer a stronger promise: if a participant doesn't trust the ceremony, they are free to contribute themselves. We hope that this model will inspire future setup ceremonies; it may also extend to other applications such as distributed randomness beacons which can be made decentralized and open to participation for all using blockchains.

We conclude with several open problems and directions for future work.

7.1 Incentives for participation

Several options are available to subsidize gas costs to encourage additional participation. The simplest solution is to load funds into the setup contract and reward each user who successfully updates the structured reference string pp, although users will still need to first pay the requisite gas fees. Alternately, transaction relay services, such as the nascent Gas Station Network (GSN), can pay transaction fees for users sending data to the setup contract. The upcoming account abstraction, EIP 4337 [19], should also help build an ecosystem of paymasters that would sponsor transactions for other users. This makes it possible for an end user to participate in setup even if that user owns no crypto to pay for gas. Finally, we note that a setup ceremony might give users a non-monetary reward such as an NFT as a badge of participation. A challenge in all cases is that users might pseudonymously participate many times via Sybil accounts; while this doesn't undermine security of the setup (assuming there was at least one honest contributor) it may enable them to claim rewards multiple times or drain the available budget for covering transaction fees, preventing other users from participating cheaply.

7.2 Verifying participation

Users may wish to see an authentic list of everyone who has contributed to the SRS. A lazy participant might see that enough participants that it trusts contributed, and choose to use the SRS without participating themselves. Fortunately, since every Ethereum transaction is signed by the party that initiates that transaction, any user can inspect the chain and construct a list of authenticated addresses that contributed to the ceremony since its inception.

7.3 Sequential participation and denial-of-service

Our ceremonies are designed to run without any centralized coordination, but they do require contributions in a serial manner. The j^{th} contributor must prove correctness of their update relative to the previous value pp_{j-1}. If two contributors independently submit transactions building on the same parameter set pp_{j-1}, only the one sequenced first will be executed successfully. The second will fail for referencing a stale parameter set. This means that, without off-chain coordination, at most one contribution per block is possible as contributors must first observe pp_{j-1}. For Ethereum this limits the ceremony to one contribution every 12 seconds or 219,000 contributions per month.

Worse, this also provides an avenue for denial-of-service and censorship: whenever an honest contribution arrives, an attacker can create an alternative contribution paying higher transaction fees, preempting the honest one. Such an attack could be detected off-chain via timing analysis. A stronger defense strategy against censorship could be to select one contribution among the conflicting ones in a random but publicly-verifiable way. To lower the transaction fees, a contributor could first register an intent to make a contribution, and only submit the actual data if it is selected. Alternatively, the setup contract can order the registered future contributors using a public randomness beacon, giving each user a random pre-assigned slot to contribute.

7.4 Verification with general-purpose roll-ups

Verification costs can be decreased using a general Layer-2 compute platform such as a *rollup server*. ZK-Rollups (also called verifiable rollups) provide succinct proofs of execution (in our case, verifying a contribution) and hence provide equivalent security to execution on Layer-1. The two common constructions today are zero-knowledge rollups and optimistic rollups, each of which brings unique design challenges. Many (though not all) ZK-rollups themselves rely on a (centralized) trusted setup. However, our protocol can be seen as a way to perform new decentralized trusted setups given a single centralized one. Or we might use a ZK-rollup which relies on a transparent setup. Alternately, optimistic rollups require watchful observers to submit fraud proofs to detect incorrect execution. Given the serial nature of our ceremony, general optimistic rollups require caution as they naively require waiting for a *challenge period* before accepting correct execution.

Rollups might offer significant cost savings, given that execution costs are roughly 100× cheaper on Layer-2, and execution costs (as opposed to storage) are over 75% of total transaction costs [50] in our evaluation. Combined with off-chain data availability, total costs can be greatly reduced. The result of a Layer-2 construction would be a 75% reduction in per transaction cost. The remainder of the transaction cost is due to the storage of elliptic curve points on Ethereum Layer-1. There are several proposals in process to decrease the cost of Layer-2 storage on Ethereum, potentially further decreasing setup cost (see EIP-4844 or EIP-4444). As of this writing, all production rollup servers rely on a single centralized sequencing server, undermining the censorship resistance benefits of an on-chain trusted setup. When these optimistic rollup Layer-2s have decentralized their sequencing, we expect the costs outlined for a trusted setup can be decreased 75–95%. In the interim, one could also implement a hybrid design which allows updates via the rollup server (to save gas) but also directly on-chain in the event of a censoring rollup sequencer.

7.5 Protocol-specific ZK rollups via proof batching

Rather than relying on a general-purpose rollup server, we can design a specific one optimized for our application. In our ceremony, every contribution is ac-

companied by a proof of correctness, requiring a linear number of proofs in the number of updates. We can improve things using a coordinator which compiles a sequence of update proofs from multiple participants and aggregates them all into a single proof that all the received updates are valid. This can be done using proof recursion [61] or accumulation [16,9,22]. This coordinator will then post the aggregate proof on chain along with the aggregate update to the parameters. This coordinator can censor particular participants by refusing to accumulate their proofs into the batch. However, since anyone can act as a coordinator, an affected participant can find another coordinator. In the worst case, if all coordinators are censoring, the participant can post their own update and proof directly on chain, bypassing the censoring coordinators.

7.6 Protocol-specific optimistic verification and checkpointing

Another mode of operation which may offer improved performance would have users post proofs (or even commitments to proofs with off-chain data availability), but not rely on on-chain verification in the optimistic case. Instead, users can post a fidelity bond which is forfeited (within a set challenge period) if another user determines off-chain that their proof is incorrect and challenges it on-chain. A caveat is that any invalidated update will also invalidate all subsequent updates due to the chained nature of the protocol. With this approach, users should verify recent contributions themselves before participating to avoid building on top of a contribution that is later invalidated.

To avoid requiring users to verify too many recent contributions before participating, it is possible to *checkpoint* certain updates by including a proof that all updates since the last checkpoint were valid. This checkpoint can be created via proof batching as discussed above. We note that, in our protocol in Section 4, only Check #1 needs to be repeated for each update since the last checkpoint; the more expensive Check #2 only needs to be done once on the latest version of the structured reference string.

7.7 Fully off-chain verification via IVC/PCD.

Another potential optimization is to conduct a ceremony with no on-chain proof verification, but where each update includes a succinct proof that *every* update since the start of the ceremony was well-formed. These proofs can be constructed using any incrementally verifiable computation scheme (IVC). In this case the parameters plus proof are an instantiation of proof-carrying data (PCD). With such a protocol, it is possible to execute the ceremony using a blockchain which only provides data availability and consensus (and no verification). Each user can verify the succinct proof of the latest parameters before using or updating them. The ceremony is only using the chain for its persistent storage and anti-censorship properties.

7.8 Forking/re-starting

Throughout the paper we assumed that updates to the powers-of-tau are applied sequentially and each update is applied to the latest state. It is also possible that a project may build on an existing powers-of-tau string, but fork it for its own use. A forking community can continue to re-randomize their own powers-of-tau branch, while the rest of the world continues to re-randomize the main branch. As such, the on-chain contract could be set up to handle forks in the update process, where multiple powers-of-tau are continuously updated independently of one another. Some powers-of-tau may even start afresh from scratch, perhaps to support different tower lengths and possibly different groups.

Acknowledgments

We would like to thank Lúcás Meier, Yashvanth Kondi, Mary Maller, and Justin Thaler for useful feedback on the early ideas underlying this work. The last author is supported by the Simons Foundation and NTT Research.

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Annual Cryptology Conference. pp. 209–236. Springer (2010)
2. Akhunov, A., Sasson, E.B., Brand, T., Guthmann, L., Levy, A.: EIP-2028: Transaction data gas cost reduction. `https://eips.ethereum.org/EIPS/eip-2028` (2019)
3. Aleo: Announcing aleo setup. `https://www.aleo.org/post/announcing-aleo-setup` (2021)
4. Alex Vlasov, K.O.: EIP-2537: Precompile for bls12-381 curve operations. `https://eips.ethereum.org/EIPS/eip-2537` (2020)
5. Attema, T., Cramer, R.: Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In: CRYPTO'20. Lecture Notes in Computer Science, vol. 12172, pp. 513–543. Springer (2020)
6. Aztec: Universal crs setup. `https://docs.zksync.io/userdocs/security/#universal-crs-setup` (2020)
7. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Journal of cryptology **32**(4), 1298–1336 (2019)
8. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: International conference on the theory and applications of cryptographic techniques. pp. 236–250. Springer (1998)
9. Bellare, M., Garay, J.A., Rabin, T.: Fast Batch Verification for Modular Exponentiation and Digital Signatures. In: Eurocrypt (1998)
10. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptol. ePrint Arch. p. 46 (2018)
11. Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., Virza, M.: Secure sampling of public parameters for succinct zero knowledge proofs. In: IEEE Symposium on Security and Privacy (2015)

12. Benaloh, J., Mare, M.d.: One-way accumulators: A decentralized alternative to digital signatures. In: Eurocrypt (1993)
13. Boneh, D., Boyen, X.: Short signatures without random oracles. In: EUROCRYPT 2004. Lecture Notes in Computer Science, vol. 3027, pp. 56–73. Springer (2004)
14. Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. In: CRYPTO '97. Lecture Notes in Computer Science, vol. 1294, pp. 425–439. Springer (1997)
15. Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive (2017)
16. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: TCC (2020)
17. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 65–97. Springer (2021)
18. Buterin, V.: How do trusted setups work? `https://vitalik.ca/general/2022/03/14/trustedsetup.html` (2022)
19. Buterin, V., Weiss, Y., Tirosh, D., Nacson, S., Forshtat, A., Gazso, K., Hess, T.: ERC-4337: Account abstraction using alt mempool. link (2021)
20. Buterin, V., et al.: Ethereum: A next-generation smart contract and decentralized application platform. `https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf` (2014)
21. Buterin, V.: What is Danksharding (2020)
22. Camenisch, J., Hohenberger, S., Pedersen, M.Ø.: Batch verification of short signatures. J. Cryptol. **25**(4), 723–747 (2012)
23. Cardozo, A.S., Williamson, Z.: EIP-1108: Reduce alt_bn128 precompile gas costs. `https://eips.ethereum.org/EIPS/eip-1108` (2018)
24. Cash, T.: Tornado.cash trusted setup ceremony. `https://tornado-cash.medium.com/tornado-cash-trusted-setup-ceremony-b846e1e00be1` (2020)
25. Celo: Plumo ceremony. `https://celo.org/plumo` (2020)
26. Chen, M., Hazay, C., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., Shelat, A., Venkitasubramaniam, M., Wang, R.: Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In: IEEE Security and Privacy (2021)
27. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Eurocrypt (2020)
28. Cohen, B., Pietrzak, K.: The Chia Network Blockchain. `https://www.chia.net/wp-content/uploads/2022/07/ChiaGreenPaper.pdf` (2019)
29. Cohen, R., Doerner, J., Kondi, Y., et al.: Guaranteed output in o(sqrt(n)) rounds for round-robin sampling protocols. Cryptology ePrint Archive (2022)
30. Company, T.E.C.: Halo2. `https://github.com/zcash/halo2`
31. DeMillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Tech. rep., Georgia Tech (1977)
32. Devos, B.: Loopring starts zkSNARK trusted setup multi-party computation ceremony. link (2019)
33. FileCoin: Trusted setup complete! `https://filecoin.io/blog/posts/trusted-setup-complete/` (2020)
34. Foundation, E.: Ethereum: Powers of tau specification. `https://github.com/ethereum/kzg-ceremony-specs` (2022)
35. Frederiksen, T.K., Lindell, Y., Osheter, V., Pinkas, B.: Fast distributed RSA key generation for semi-honest and malicious adversaries. In: CRYPTO 2018. Lecture Notes in Computer Science, vol. 10992, pp. 331–361. Springer (2018)

36. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019)
37. Ganesh, C., Khoshakhlagh, H., Kohlweiss, M., Nitulescu, A., Zajac, M.: What makes fiat–shamir zksnarks (updatable srs) simulation extractable? Cryptology ePrint Archive, Paper 2021/511 (2021), https://eprint.iacr.org/2021/511, https://eprint.iacr.org/2021/511
38. Gilboa, N.: Two party RSA key generation. In: CRYPTO '99. Lecture Notes in Computer Science, vol. 1666, pp. 116–129. Springer (1999)
39. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4284, pp. 444–459. Springer (2006)
40. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: ASIACRYPT'10. Lecture Notes in Computer Science, vol. 6477, pp. 321–340. Springer (2010)
41. Groth, J.: On the size of pairing-based non-interactive arguments. In: Eurocrypt (2016)
42. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-snarks. In: Annual International Cryptology Conference. pp. 698–728. Springer (2018)
43. Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T., Nicolosi, A.A.: Efficient RSA key generation and threshold paillier in the two-party setting. J. Cryptol. **32**(2), 265–323 (2019)
44. Hermez, P.: Hermez zero-knowledge proofs. https://blog.hermez.io/hermez-zero-knowledge-proofs/ (2020)
45. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: International conference on the theory and application of cryptology and information security. pp. 177–194. Springer (2010)
46. Kerber, T., Kiayias, A., Kohlweiss, M.: Mining for privacy: How to bootstrap a snarky blockchain. In: Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25. pp. 497–514. Springer (2021)
47. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: Annual international cryptology conference. pp. 543–571. Springer (2016)
48. Kohlweiss, M., Maller, M., Siim, J., Volkhov, M.: Snarky ceremonies. In: AsiaCrypt (2021)
49. Kuszmaul, J.: V(ery short m)erkle trees. verkle trees. https://math.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf (2018)
50. "l2 fees". https://l2fees.info/ (2022)
51. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In: Theory of Cryptography Conference. pp. 1–34. Springer (2021)
52. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: Theory of Cryptography Conference. pp. 499–517. Springer (2010)
53. Lipmaa, H., Siim, J., Zajac, M.: Counting vampires: From univariate sumcheck to updatable zk-snark. Cryptology ePrint Archive (2022)

54. Long, L.: Binary Quadratic Forms. https://github.com/Chia-Network/vdf-competition/blob/main/classgroups.pdf (2019)
55. Malkin, M., Wu, T.D., Boneh, D.: Experimenting with shared generation of RSA keys. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, San Diego, California, USA. The Internet Society (1999)
56. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2111–2128 (2019)
57. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 228–245. Springer (2007)
58. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Conference on the Theory and Application of Cryptology. pp. 239–252. Springer (1989)
59. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
60. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM (JACM) **27**(4), 701–717 (1980)
61. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: TCC (2008)
62. Vitalik Buterin, C.R.: EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. https://eips.ethereum.org/EIPS/eip-197 (2017)
63. Vlasov, A.: EIP-2539: Bls12-377 curve operations. https://eips.ethereum.org/EIPS/eip-2539 (2020)
64. Wang, W., Ulichney, A., Papamanthou, C.: BalanceProofs: Maintainable Vector Commitments with Fast Aggregation. Cryptology ePrint Archive, Paper 2022/864 (2022)
65. Waters, B., Wu, D.: Batch arguments for NP and more from standard bilinear group assumptions. In: CRYPTO'22 (2022)
66. Youssef El Housni, Michael Connor, A.G.: EIP-3026: Bw6-761 curve operations. https://eips.ethereum.org/EIPS/eip-3026 (2020)
67. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: International symposium on symbolic and algebraic manipulation. pp. 216–226. Springer (1979)

A Proof of Theorem 2

In this section we prove Theorem 2 of Section 4 which guarantees that Check #2 guards the setup from malformed contributions.

Proof. Suppose the contributor generated a parameter set pp that passed Check #2. We write

$$\begin{aligned}\mathsf{pp} &= (P_1, P_2, P_3, \ldots, P_n \ ; \ Q_1, \ldots, Q_k) = \\ &= (a_1 B_1, a_2 B_1, \ldots, a_n B_1 \ ; \ b_1 B_2, b_2 B_2, \ldots, b_k B_2).\end{aligned}$$

If check # 2 passed, then for two random scalars $x = \rho_1$ and $y = \rho_2$ in $\mathbb{Z}_p$ chosen by the verifier the following equation holds:

$$(1 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}) \cdot (b_1 + b_2 y + \ldots + b_k y^{k-1}) - (a_1 + a_2 x + \ldots + a_n x^{n-1}) \cdot (1 + b_1 y + b_2 y^2 + \ldots + b_{k-1} y^{k-1}) = 0 \quad \text{(A.1)}$$

Let us define a 2-variate polynomial $f(x, y)$ to match the left-hand side of Eq. A.1. By the DeMillo-Lipton-Schwartz–Zippel (DLSZ) lemma [31,67,60], if f is a non-zero polynomial, then the number of zeros of f is bounded by $d \cdot p$ where $d = (n-1)(k-1)$ is the degree of $f(x, y)$. Equivalently, the probability that $f(x, y) = 0$ for x and y selected uniformly at random from $\mathbb{Z}_p$ is bounded above by d/p. Therefore, the probability that the polynomial f defined in Eq. A.1 is a zero polynomial is overwhelming: it is at least $1 - (k-1)(n-1)/p$. For a zero polynomial $f \equiv 0$, its coefficients are all zero. In particular the constant term $b_1 - a_1$ is 0 implying that $a_1 = b_1$, and we denote that by $\tau = a_1$. The rest of the coefficients being zero implies that

$$\begin{array}{lll} \text{coefficient of } x: & a_1 b_1 - a_2 = 0 & \Rightarrow \quad a_2 = \tau^2 \\ \text{coefficient of } x^2: & a_2 b_1 - a_3 = 0 & \Rightarrow \quad a_3 = \tau^3 \\ \ldots & & \\ \text{coefficient of } x^{n-1}: & a_{n-1} b_1 - a_n = 0 & \Rightarrow \quad a_n = \tau^n \end{array}$$

Applying the same argument to the coefficients of y^i in Eq. A.1 we obtain:

$$\begin{array}{lll} \text{coefficient of } y: & b_2 - a_1 b_1 = 0 & \Rightarrow \quad b_2 = \tau^2 \\ \text{coefficient of } y^2: & b_3 - a_1 b_2 = 0 & \Rightarrow \quad b_3 = \tau^3 \\ \ldots & & \\ \text{coefficient of } y^k: & b_k - a_1 b_{k-1} = 0 & \Rightarrow \quad b_k = \tau^k \end{array}$$

Therefore we obtain that a setup that successfully passes check #2 is well-formed with probability at least $1 - (k-1)(n-1)/p$, as required.

Note on soundness for a punctured setup. At the end of Section 7 we explained how to modify Check # 2 to be able to handle powers-of-tau setups with one point missing. The soundness proof for this modified check is analogous: for random scalars $x = \rho_1, y = \rho_2$ in $\mathbb{Z}_p$ we define the polynomial $f(x, y)$ to match the left-hand side of Eq. D.3:

$$\left(\sum_{\substack{i=1 \\ i \neq N+1 \\ i \neq N+2}}^{2N} a_i x^{i-1} \right) \cdot \left(1 + \sum_{i=1}^{N-1} b_i y^i \right) - \left(1 + \sum_{\substack{i=1 \\ i \neq N \\ i \neq N+1}}^{2N-1} a_i x^i \right) \cdot \left(\sum_{i=1}^{N} b_i y^{i-1} \right) = 0 \quad \text{(A.2)}$$

The probability that the polynomial f is zero is at least $1 - 2N^2/p$. For a zero polynomial all of its coefficients are zero, hence the constant term $b_1 - a_1 = 0$ (denote $\tau = a_1$) and analogously we get $b_i = \tau^i$ for $i = 1 \ldots N$ and $a_i = \tau^i$ for $i = 1 \ldots 2N$ where $i \neq N+1$. The only difference in the argument, is that we use the second pairing check (D.2) to get $a_{N+2} = a_N b_2$ which implies $a_{N+2} = \tau^{N+2}$.

B Inner-pairing product arguments for Section 5.2

We restate Eq. 5.1 of Section 5.2 again for convenience:

$$C_1 = \langle \mathbf{P}, \mathbf{\Gamma}_2 \rangle \tag{B.1}$$
$$C_2 = \langle \mathbf{\Gamma}_1, \mathbf{Q} \rangle \tag{B.2}$$
$$\rho_1^n P_n Q_1 - B_1 Q_1 = \langle \mathbf{P}, (1, \rho_1, \rho_1^2, \ldots, \rho_1^{n-1}) \cdot (\rho_1 Q_1 - B_2) \rangle \tag{B.3}$$
$$\rho_2^k P_1 Q_k - P_1 B_2 = \langle (1, \rho_2, \rho_2^2, \ldots, \rho_2^{k-1}) \cdot (\rho_2 P_1 - B_1), \mathbf{Q} \rangle \tag{B.4}$$
$$P_n = \langle \mathbf{P}, (0, 0, \ldots, 0, 1) \rangle \tag{B.5}$$
$$P_1 = \langle \mathbf{P}, (1, 0, \ldots, 0, 0) \rangle \tag{B.6}$$
$$Q_k = \langle \mathbf{Q}, (0, 0, \ldots, 0, 1) \rangle \tag{B.7}$$
$$Q_1 = \langle \mathbf{Q}, (1, 0, \ldots, 0, 0) \rangle \tag{B.8}$$

We first prove the soundness, namely we show that with an overwhelming probability a setup $\mathsf{pp} = (\mathbf{P}; \mathbf{Q})$ that satisfies the set of equations above for random scalars ρ_1 and ρ_2 chosen by the verifier has to be well-formed according to Definition 3. We denote by $x = \rho_1$, and we write $\mathbf{P} = (a_1 B_1, a_2 B_1, \ldots, a_n B_1)$ and $\mathbf{Q} = (b_1 B_2, b_2 B_2, \ldots, b_k B_2)$ for some $a_1, \ldots, a_n, b_1, \ldots, b_k \in \mathbb{Z}_p$ and we rewrite Eq. B.3 equivalently into the following equation:

$$x^n a_n b_1 - b_1 - (a_1 + x a_2 + x^2 a_3 + \ldots + x^{n-1} a_n) \cdot (x b_1 - 1) = 0 \iff$$
$$(a_1 - b_1) + (a_2 - a_1 b_1) x + (a_3 - a_2 b_1) x^2 + \ldots + (a_n - a_{n-1} b_1) x^{n-1} = 0 \tag{B.9}$$

We denote the left-hand side of Eq. B.9 by $f(x)$, where f is a polynomial of degree $n-1$ over $\mathbb{Z}_p$. We apply the DeMillo-Lipton-Schwartz–Zippel (DLSZ) lemma [31,67,60], if f is a non-zero polynomial, then the number of zeros of f is bounded by $d \cdot p$ where $d = n - 1$ is the degree of $f(x)$. Equivalently, the probability that $f(x) = 0$ for x selected uniformly at random from $\mathbb{Z}_p$ is bounded above by d/p. Therefore, the probability that the polynomial f defined in Eq. B.9 is a zero polynomial is overwhelming: it is at least $1 - (n-1)/p$. For a zero polynomial $f \equiv 0$, its coefficients are all zero:

$$\text{free term} : a_1 - b_1 = 0 \Rightarrow a_1 = b_1 \text{we denote that by } a_1 = \tau$$
$$\text{coefficient of } x : a_2 - a_1 b_1 = 0 \Rightarrow a_2 = \tau^2$$
$$\text{coefficient of } x^2 : a_3 - a_2 b_1 = 0 \Rightarrow a_3 = \tau^3$$
$$\ldots$$
$$\text{coefficient of } x^{n-1} : a_n - a_{n-1} b_1 = 0 \Rightarrow a_n = \tau^n$$

With analogous analysis of Eq. B.4 we get that $b_i = \tau^i$ for all $i = 1..k$ with probability at least $1 - (k-1)/p$. This proves Theorem 3:

Theorem 3. *A probabilistic polynomial-time contributor will satisfy Eq. B.3 and Eq. B.4 with a malformed setup string with probability at most* $\frac{(n-1)+(k-1)}{p}$, *which is negligible in the security parameter* λ *(where we assume* $p \approx 2^{2\lambda}$ *and* n, k *being polynomial-size in* λ*).*

The IPP protocol. We now explain the interactive version of the protocol that can be made non-interactive with a Fiat-Shamir heuristic to be run with a verifier as an on-chain smart-contract.

1. The prover submits $C_1, C_2, P_1, P_n, Q_1, Q_k \in \mathbb{G}_T^2 \times \mathbb{G}_1^2 \times \mathbb{G}_2^2$ to the verifier.
2. The prover shows that it knows the discrete log to the update of P_1 (knowledge of discrete log of P_1 base the previous value of P_1 that is currently stored on-chain) as explained in Section 4, Eq. 4.
3. The verifier checks that the update is non-degenerative: $P_1 \neq 0$ and if so replies with two random scalars $\rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_p$.
4. The prover sends $E_1 \in \mathbb{G}_1$ and $E_2 \in \mathbb{G}_2$ to the verifier, where $E_1 = \langle \mathbf{P}, (1, \rho_1, \rho_1^2, \ldots, \rho_1^{n-1}) \rangle$ and $E_2 = \langle \mathbf{Q}, (1, \rho_2, \rho_2^2, \ldots, \rho_2^{k-1}) \rangle$.
5. The prover runs six Dory-IPP arguments in batch to produce a proof π that it sends to the verifier. As we explain below.
6. The verifier checks that $E_1(\rho_1 Q_1 - B_2) = P_n \rho^n Q_1 - B_1 Q_1$, and $E_1(\rho_2 P_1 - B_1) E_2 = \rho_2^k P_1 Q_k - P_1 B_2$.
7. The verifier checks π and, if correct, updates the setup that it stores to $(C_1, C_2, P_1) \in \mathbb{G}_T^2 \times \mathbb{G}_1$.

We now show how to construct a succinct (logarithmic-size) proof π for Eq. B.1-B.8 using Dory inner product argument of Jonathan Lee [51]. Those arguments allow to prove the following general relation (where the vectors of scalars $\vec{s_1}$ and $\vec{s_2}$ are public and have multiplicative structure):

$$(D, C_1, C_2, E_1, E_2) \in \mathcal{L}_{n,\Gamma_1,\Gamma_2}(\vec{s_1}, \vec{s_2}) \in \mathbb{G}_T^3 \times \mathbb{G}_1 \times \mathbb{G}_2 \iff$$
$$\text{Exists witnesses } \vec{v_1} \in \mathbb{G}_1 \text{ and } \vec{v_2} \in \mathbb{G}_2 : C_1 = \langle \vec{v_1}, \Gamma_2 \rangle \quad C_2 = \langle \Gamma_1, \vec{v_2} \rangle$$
$$E_1 = \langle \vec{v_1}, \vec{s_1} \rangle \quad E_2 = \langle \vec{v_2}, \vec{s_2} \rangle \quad D = \langle \vec{v_1}, \vec{v_2} \rangle$$

We invoke the argument six times (the arguments are batchable and allow to squash six proofs into a single one) to prove the following less general statements, we show two of those for Eq. B.3 and Eq. B.5 as the rest are analogous:

- For Eq. B.3: $(0, C_1, 0, E_1, 0) \in \mathcal{L}_{n,\Gamma_1,\Gamma_2}(\vec{s_1}, \vec{s_2})$ for scalars $\vec{s_1} = (1, \rho_1, \rho_1^2, \ldots, \rho_1^{n-1})$ and $\vec{s_2} = \vec{0}$ and witnesses $v_1 = \mathbf{P}$, $v_2 = \vec{0}$.
- For Eq. B.5: $(0, C_1, 0, P_n, 0) \in \mathcal{L}_{n,\Gamma_1,\Gamma_2}(\vec{s_1}, \vec{s_2})$ for scalars $s_1 = (0, 0, 0, \ldots, 0, 1)$, $s_2 = \vec{0}$ and witnesses $v_1 = \mathbf{P}$, $v_2 = \vec{0}$.

The verifier in [51] is set up with $4\log(n) + 1$ pre-computed elements of $\mathbb{G}_T$. Those values are inner-products between subvectors of the vectors of generators $\mathbf{\Gamma}_1$ and $\mathbf{\Gamma}_2$ and can be pre-computed in linear-time.

Note that in this type of setup, the secret is only used to update the setup and prove knowledge of the discrete log of P_1. The bulk of the computation, namely proof generation, is independent of the secret chosen by the contributor. Thus, the contributor may outsource this computation to an untrusted helper.

C Off-chain setup from IPP arguments with a smaller setup

For completeness, we briefly explain the inner-product pairing (IPP) method of Bünz et al. [17]. It relies on a powers-of-tau SRS of a smaller size stored by the verifier in full:

$$\mathbf{\Gamma}_1 = (\alpha B_1, \alpha^2 B_1, \ldots, \alpha^{2n} B_1), \qquad \mathbf{\Gamma}_2 = (\beta B_2, \beta^2 B_2, \ldots, \beta^{2n} B_2)$$

The contributor can then commit to a larger setup of length $N = \eta \times n$ in $\mathbb{G}_1$ and $\mathbb{G}_2$ with structured AFGHO commitments of of Abe et al. [1] as follows:

$$\begin{aligned} &\text{For } \mathbf{P} = (\mathbf{P}_1, \ldots, \mathbf{P}_\eta) \in (\mathbb{G}_1^n, \ldots, \mathbb{G}_1^n) \text{ and} \\ &\quad \text{for } \mathbf{Q} = (\mathbf{Q}_1, \ldots, \mathbf{Q}_\eta) \in (\mathbb{G}_2^n, \ldots, \mathbb{G}_2^n) : \\ &\qquad \mathbf{C}_1 = (\langle \mathbf{P}_1, \mathbf{\Gamma}_{1,\mathsf{even}} \rangle, \ldots, \langle \mathbf{P}_\eta, \mathbf{\Gamma}_{1,\mathsf{even}} \rangle) \in \mathbb{G}_T^\eta \\ &\qquad \mathbf{C}_2 = (\langle \mathbf{\Gamma}_{2,\mathsf{even}}, \mathbf{Q}_1 \rangle, \ldots, \langle \mathbf{\Gamma}_{1,\mathsf{even}}, \mathbf{Q}_\eta \rangle) \in \mathbb{G}_T^\eta \end{aligned}$$

The contributor submits commitments $\mathbf{C}_1, \mathbf{C}_2$ to the verifier and creates TIPP-proofs of a set of inner-pairing-product relations similar to the ones described in Section 5.2. The resulting proofs add up to be of cumulative size $O(\eta \log(n))$ and can be verified in $O(\eta \log(n))$ time.

This method leads to worse practical efficiency compared to the method described in Section 5.2, although it might yield better concrete costs if an on-chain setup is extended by a small multiple making the resulting length N be far from the power of two.

D Powers-of-tau with a punctured point

Some systems require a powers-of-tau string where one power in the sequence is absent, namely

$$\mathsf{pp} = \left[(P_i)_{i=1, i \neq N+1}^{2N}, \quad (Q_i)_{i=1}^{N} \right] = \left[(\tau^i B_1)_{i=1, i \neq N+1}^{2N}, \quad (\tau^i B_2)_{i=1}^{N} \right],$$

where the point $P_{N+1} = \tau^{N+1} B_1$ is absent from pp. Example systems that use a punctured sequence include Groth'10 [40], Attema and Cramer [5], Lipmaa, Siim, and Zajac's Vampire scheme [53], and Waters and Wu [65]. The absence of the point P_{N+1} from pp is necessary for security. Check #2 in (4) can be modified to handle this case: the verifier will sample two random scalars ρ_1, ρ_2 in $\mathbb{Z}_p^*$ and carry out the following check that now consists of two equations:

Check # 2 for punctured setup:

$$e\Big(\sum_{\substack{i=1\\ i\neq N+1\\ i\neq N+2}}^{2N} \rho_1^{i\text{-}1} P_i,\quad B_2 + \sum_{\ell=1}^{N-1} \rho_2^{\ell} Q_\ell\Big) =$$

$$= e\Big(B_1 + \sum_{\substack{i=1\\ i\neq N\\ i\neq N+1}}^{2N-1} \rho_1^{i} P_i,\quad \sum_{\ell=1}^{N} \rho_2^{\ell\text{-}1} Q_\ell\Big) \tag{D.1}$$

$$e\Big(P_{N+2}, B_2\Big) = e\Big(P_N, Q_2\Big) \tag{D.2}$$

It is not difficult to see that a well-formed setup will pass the check successfully. The soundness proof for this modified check is analogous: for random scalars $x = \rho_1, y = \rho_2$ in $\mathbb{Z}_p$ we define the polynomial $f(x, y)$ to match the left-hand side of Eq. D.3:

$$\left(\sum_{\substack{i=1\\ i\neq N+1\\ i\neq N+2}}^{2N} a_i x^{i-1}\right) \cdot \left(1 + \sum_{i=1}^{N-1} b_i y^i\right) - \left(1 + \sum_{\substack{i=1\\ i\neq N\\ i\neq N+1}}^{2N-1} a_i x^i\right) \cdot \left(\sum_{i=1}^{N} b_i y^{i-1}\right) = 0 \tag{D.3}$$

The probability that the polynomial f is zero is at least $1 - 2N^2/p$. For a zero polynomial all of its coefficients are zero, hence the constant term $b_1 - a_1 = 0$ (denote $\tau = a_1$) and analogously we get $b_i = \tau^i$ for $i = 1 \ldots N$ and $a_i = \tau^i$ for $i = 1 \ldots 2N$ where $i \neq N+1$. The only difference in the argument, is that we use the second pairing check (D.2) to get $a_{N+2} = a_N b_2$ which implies $a_{N+2} = \tau^{N+2}$.