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

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

[1] *Andreesen-Horowitz - a16z crypto research*
[2] *Stanford University*
[3] *New York University*

Abstract

We introduce the first decentralized trusted setup protocols for constructing a powers-of-tau structured reference string. Facilitated by a blockchain platform, our protocols can run in a permissionless manner, with anybody able to participate in exchange for paying requisite transaction fees. The result 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 numbers.

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 [BM93] 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, trusted setup consists of a randomized algorithm $\mathsf{Setup}() \xrightarrow{\$} \mathsf{pp}, \tau$ whose public parameters (pp) are needed, but for which 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 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*.

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), 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 structured reference string (SRS). Working in elliptic curve groups $\mathbb{G}_1, \mathbb{G}_2$ 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:

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

The value τ 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 if 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, including:

- The KZG polynomial commitment scheme [KZG10] require 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 [MBKM19], Plonk [GWC19], and Marlin [CHM+20], require a powers-of-tau string. The number of powers of tau needed is proportional to the size of the statement being proved.
- KZG commitments are also used in Verkle trees [Kus18, LY10], 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 [WUP22], 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 [BMM+21].
- The recent Danksharding proposal [But20] 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 run through a centralized coordinator which fulfills several important functions, all of which we seek to replicate in a decentralized fashion:

- **Consensus** Participants should agree on the final value of pp.
- **Verification** Each participant must provide a zero-knowledge proof that their contribution is a valid re-randomization (and not simply replacing the string 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. Verification and data availability are more interesting and provide several design options. For verification, we can rely on on-chain (Layer-1) verification, or (to reduce costs) use a Layer-2 verification solution or leave this task

Data availability	Commitment scheme	Section	Proof size	Verifier time
On-chain	none	3	$O_\lambda(1)$	$O_\lambda(n)$
Off-chain	Any commitment	4.1	$O_\lambda(\lg n)$	$O_\lambda(\lg n)$
	AFGHO unstructured commitment	4.2	$O_\lambda(\lg n)$	$O_\lambda(\lg n)$

Figure 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.

to users of the string to verify before using. 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.

It would be prudent to note that there are considerable research efforts 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 of 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 [Comb, Coma] 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. However, known trustless systems don't match the efficiency of the ones based on the 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 a meanwhile, a unified framework for running setup ceremonies in a transparent, verifiable and censorship-resistant manner would help bootstrap more efficient cryptosystems.

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, $c = H(pp)$, 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 3). We implemented this protocol (and have released our code open-source: github.com/a16z/evm-powers-of-tau) on the Ethereum blockchain with the BN254 curve. Participating in the ceremony costs 190,000 to 11,500,000 gas (about \$4 to \$250 at current 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 small zero-knowledge SNARKs, data-availability sampling, and Verkle trees.

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 4). 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.

We discuss censorship resistance, incentives and methods to lower on-chain cost through roll-ups, optimistic verification, batching, IVC and other techniques in Section 5.

2 Related work

Ben-Sasson et al. [BSCG$^+$15] 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 floor for crowd-sourcing subsequent ceremonies.

Bowe et al. [BGM17] designed a protocol for Groth16, 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 [KMSV21] 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. [CDK$^+$22] 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.

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 [But22] suggested a simple way to verify the update to the setup that opens the possibility of a gas-efficient on-chain deployment which we base our on-chain protocol on.

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 [Cas20] , Hermez network [Her20], and Loopring [Dev19]. Similar ceremonies on other curves were run by Aztec [Azt20] for zkSync, a "layer two" Ethereum scaling solution that uses zero knowledge rollups; by Filecoin [Fil20], a decentralized data storage protocol; by Celo [Cel20], a layer-1 blockchain, for their light-client Plumo; Aleo [Ale21], a blockchain for private applications.

Ethereum plans [Fou22] to run 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 $\mathbb{G}_2$. Those two upgrades will increase the amount of data that the Ethereum chain provides to clients for storage. This data will have a suggetsed expiry 30–60 days. The ceremony is under active development, and is planned to run for six weeks in 2023. It is shaping up to be the largest trusted setup ceremony in terms of participation for blockchains run so far.

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

We assume three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, each of prime order p, with generators B_1, B_2, B_T respectively, addition as a group operation in all of them, and a bilinear pairing operation $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, s.t. for any $a, b \in \mathbb{Z}_p^*$ the following holds: $e(aB_1, bB_2) = ab \cdot e(B_1, B_2)$. Our goal is to construct a "powers of τ" structured reference string (SRS) of the form:

$$\mathsf{pp} = [\tau B_1, \tau^2 B_1, \tau^3 B_1, \ldots, \tau^n B_1; \quad \tau B_2, \tau^2 B_2, \ldots, \tau^k B_2]$$

It is essential that τ be kept secret in the final string, pp. The protocol for constructing pp will be a sequential multi-party computation between m contributors in m rounds, such that each contributor, C_j, contributes only in the j^{th} round. Each contributor can efficiently prove that their participation was correct. The protocol should be secure as long as any individual contributor used good randomness in their round and was honest, i.e. only used locally generated secrets as intended by the protocol and destroyed them successfully after the protocol's completion. 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.

Initialization The initial state (after round 0) consists of the string:

$$\begin{aligned}\mathsf{pp} &= [P_{1,0},\quad P_{2,0},\quad P_{3,0},\quad \ldots,\quad P_{n,0};\quad Q_{1,0},\quad Q_{2,0},\quad \ldots,\quad Q_{k,0}]\\ &= [\ B_1,\quad B_1,\quad B_1,\quad \ldots,\quad B_1;\quad B_2,\quad B_2,\quad \ldots,\quad B_2]\end{aligned} \tag{3.1}$$

That is, n copies of the generator B_1 plus k copies of the generator B_2. This is equivalent to an SRS with $\tau = 1$. This is trivially insecure as everybody knows τ, but is trivially easy to check for well-formedness. Note that it is not strictly necessary to have $k > 1$, but it benefits the efficiency of certain applications, e.g. multi-point evaluation proofs in KZG polynoimal commitments.

Update procedure At the beginning of round j the current string assumed to be:

$$\begin{aligned}\mathsf{pp} &= [\ P_{1,j\text{-}1},\ P_{2,j\text{-}1}, \ldots,\ P_{n,j\text{-}1};\quad Q_{1,j\text{-}1}, \ldots,\ Q_{k,j\text{-}1}]\\ &= [\tau_{j\text{-}1}B_1,\ \tau_{j\text{-}1}^2 B_1, \ldots,\ \tau_{j\text{-}1}^n B_1;\quad \tau_{j\text{-}1}B_2, \ldots,\ \tau_{j\text{-}1}^k B_2]\end{aligned} \tag{3.2}$$

The value τ_{j-1} is of course hidden. Contributor C_j chooses a random value $r_j \xleftarrow{\$} \mathbb{Z}_p^*$ and publishes a new string:

$$\begin{aligned}\mathsf{pp} =&[P_{1,j},\quad P_{2,j},\quad P_{3,j},\quad \ldots, P_{n,j};\quad Q_{1,j},\quad \ldots, Q_{k,j}]\\ =&[r_j P_{1,j\text{-}1},\ r_j^2 P_{2,j\text{-}1},\quad r_j^3 P_{3,j\text{-}1},\ \ldots,\ r_j^n P_{n,j\text{-}1};\quad r_j Q_{1,j\text{-}1},\ \ldots,\ r_j^k Q_{k,j\text{-}1}]\\ =&[r_j \tau_{j\text{-}1} B_1,\ r_j^2 \tau_{j\text{-}1}^2 \cdot B_1,\ r_j^3 \tau_{j\text{-}1}^3 B_1,\ \ldots,\ r_j^n \tau_{j\text{-}1}^n B_1;\quad r_j \tau_{j\text{-}1} B_2,\ \ldots,\ r_j^k \tau_{j\text{-}1}^k B_2]\\ =&[\tau_j B_1,\quad \tau_j^2 B_1,\quad \tau_j^3 B_1,\quad \ldots, \tau_j^n B_1;\quad \tau_j B_2,\quad \ldots, {\tau_j}^k B_2]\end{aligned} \tag{3.3}$$

The new setup has $\tau_j = r_j \cdot \tau_{j\text{-}1}$ as its secret[1]. If an attacker knows τ_{j-1} but not r_j, and r_j was chosen uniformly at random from $\mathbb{Z}_p^*$ (meaning in particular that $r_j \neq 0$), then the attacker will have no information about τ_j (since the operations are done modular a large prime p of roughly 256-bits length). In other words, each new honest contributor randomizes the setup completely. If at least one of the contributors supplies their update, r_j, randomly and properly destroys it (and forgets), then the resulting secret ($\tau_m = r_1 \cdot r_2 \cdot \ldots \cdot r_m$) is randomly distributed and unknown to anybody.

Update proofs Contributor C_j must convince the verifier (the smart contract) that the following three statements are true about its contribution:

Check #1 - the contributor knows r_j: a proof that the latest contribution to the ceremony builds on the work of the preceding participants.

Check #2 - the new parameters, pp_j, are well-formed: the contract should verify that newly submitted pp consists of consecutive powers of some τ_j.

Check #3 - the update is non-degenerative, $r_j \neq 0$: a defense against attackers trying to erase the setup thus undermining the contributions of previous participants.

Only if the verifier (smart contract) is convinced that all of the above is true, it updates the setup pp with the contribution from C_j. We now give the details of how each of these statements is verified on-chain and what proofs (if any) the contributor needs to send to facilitate the verification.

[1]Note that it is also possible to compute an additive update to the tau ($\tau_j = r_j + \tau_{j-1}$), however it would require the contributor to compute many multi-scalar multiplications making it less efficient.

The contributor computes a zero-knowledge proof π demonstrating that it knows r_j s.t. $P_{1,j} = P_{1,j-1} \cdot r_j$. This proof could be either a Fiat-Shamir version of Schnorr's Σ-protocol [Sch89, Sch91] or a BLS-style proof of possession [RY07] of the secret key. The latter is effectively, the BLS signature over the public key, but it is more expensive to verify on-chain as it requires to compute pairings, so we will focus on the former approach which works as follows.

The prover, the contributor C_j, samples a random $z \xleftarrow{\$} \mathbb{Z}_p^*$ and computes

$$h = \mathsf{HASH}(P_{1,j} \mid\mid P_{1,j-1} \mid\mid z \cdot P_{1,j-1}), \qquad \pi = (z \cdot P_{1,j-1},\ \ z + h \cdot r_j)$$

Here HASH is a collision-resistant hash function whose outputs are uniformly distributed in $\mathbb{Z}_p$. The smart contract verifies the proof $\pi = (\pi_1, \pi_2)$ as follows:

$$\boxed{\text{Check } \# \text{ 1:} \qquad \pi_2 \cdot P_{1,j-1} = \pi_1 + \mathsf{HASH}(P_{1,j} \mid\mid P_{1,j-1} \mid\mid \pi_1) \cdot P_{1,j} \qquad (3.4)}$$

Definition 3.1. *We say that the setup 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_j is well-formed, the verifier will sample two random scalars $\rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_p^*$ and check that:

Check # 2:

$$e\Big(\sum_{i=1}^{n} \rho_1^{i\text{-}1} P_{i,j},\ B_2 + \sum_{\ell=1}^{k-1} \rho_2^{\ell} Q_{\ell,j}\Big) = e\Big(B_1 + \sum_{i=1}^{n-1} \rho_1^{i} P_{i,j},\ \sum_{\ell=1}^{k} \rho_2^{\ell\text{-}1} Q_{\ell,j}\Big) \qquad (3.5)$$

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

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

In practice the random scalars ρ_1, ρ_2 can be generated using the Fiat-Shamir heuristic by hashing the string submitted by the contributor, namely $\rho_1 \leftarrow \mathsf{HASH}(\mathsf{pp}||1)$ 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 the new setup is non-zero:

$$\boxed{\text{Check } \#3: \qquad P_{1,j} \neq 0. \qquad (3.6)}$$

We now argue the properties of the scheme: correctness, honest-verifier zero-knowledge, soundness and knowledge-soundness.

Correctness: it is easy to see that an honest prover that updated the setup correctly, assuming the previous setup was well-formed, and produced correct proof π will convince the verifier about the correctness of its update.

Zero-knowledge: only the first check requires the contributor to submit a proof, and it is necessary to demonstrate that this proof leaks no information about the contributor's secret. Since for the proof π we suggest to use Schnorr's identification protocol with Fiat-Shamir's transform, proving that π is zero-knowledge and leaks no information about the underlying secret is standard, nonetheless we sketch the proof next for completeness. A satisfying proof with the same distribution for Eq. (3.4) without knowing r_j can be constructed by programming the random oracle (using the random-oracle (RO) assumption) as follows. Choose random $w, h \xleftarrow{\$} \mathbb{Z}_p^*$, and set $\pi_1 = P_{1,j-1}^w / P_{1,j}^h$, $\pi_2 = w$ and program the random oracle:

$\mathsf{HASH}(P_{1,j} \mid\mid P_{1,j-1} \mid\mid P^{w}_{1,j-1}) = h$, such a proof will leak no information about the secret and for a public observer will look identical to a real proof.

Knowledge soundness: to show that a witness r_j can be extracted from a convincing prover, we apply folklore techniques and exploit knowledge soundness of the Σ-protocol of the proof π_j to extract the discrete log of $\tau_j B_1$ to the base $\tau_{j-1} B_1$: $r_j = \tau_j / \tau_{j-1}$ by rewinding the prover with different outputs of its random oracle queries [PS00] and use the forking-lemma to extract the witness r_j.

Soundness: we argue that a contributor can not pass Check #2 with a malformed setup other than with some negligible probability, and we prove the following theorem in Appendix A.

Theorem 3.1. *Check # 2 ensures the well-formedness of the setup. In particular, a probabilistic polynomial-time contributor will pass Check #2 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*).*

3.1 Implementation and Evaluation on Ethereum

In this section, we analyse the practicality of a fully on-chain implementation on Ethereum. Ethereum currently (as of Oct'22) natively supports only one group with bilinear pairing, BN254 (the initial EIP-197 [VB17] 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 [BD19], and now estimated [KB16] 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 [AV20]) alongside precompiles for other pairing-friendly curves BLS12-377 (EIP-2539 [Vla20]) and BW6-761 (EIP-3026 [YEH20]). 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 [ASC18]:

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

Each contribution is sent as calldata, which is a read-only byte array, currently priced at 16 gas per byte according to EIP-2028 [ASB+19].

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:

Check # 2 (more efficient):

$$\text{For } R = \sum_{i=1}^{n-1} \rho_1^{i\text{-}1} \cdot P_{i,j} : \quad e(B_1 + \rho_1 R,\ Q_1) \ = \ e(R + \rho_1^{n-1} P_{n,j},\ B_2) \tag{3.7}$$

The contributor submits $64 \cdot n + 224$ bytes of calldata: n elements of $\mathbb{G}_1$ (64 bytes per uncompressed[2] element), 1 element in $\mathbb{G}_2$ (128 bytes per uncompressed element), 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:

$$\begin{aligned} &\text{compute cost: } (n+3) \cdot 6,000 + 113,000 \text{ gas} \\ &\text{storage cost: } n \cdot 1,024 + 3,584 \text{ gas} \end{aligned} \tag{3.8}$$

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

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 a techniques of Bellare et al. [BGR98a] the scalar-multiplications might be substituted by λ-random 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. 3.7 based on our open-sourced implementation (github.com/a16z/evm-powers-of-tau).

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	\$3	\$4	\$6	\$10	\$18	\$33	\$62	\$122
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	\$1	\$3	\$5	\$10	\$20
Total (estimates)	187,192	243,384	355,768	580,536	1,030,072	1,929,144	3,727,288	7,323,576
	\$4	\$5	\$7	\$11	\$20	\$38	\$73	\$143
Total (actual)	192,162	272,217	432,702	755,340	1,406,185	2,731,526	5,474,920	11,341,136
	\$4	\$5	\$8	\$15	\$27	\$53	\$108	\$221

Table 1: Estimates according to the Eq. 3.8 and actual costs. The pricing in USD is calculated based on rough numbers on 10/11/2022: 15 gwei per gas unit and 1 ETH = \$1,300.

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):

$$\text{For } R = \sum_{i=0}^{n-2} \rho^i \cdot P_{i+1,j} : \quad e(B_1 + \rho R,\ Q_{1,j}) \quad = \quad e(R + \rho^{n-1} P_{n,j},\ B_2) \tag{3.9}$$

$$\text{For } t = 2..k\text{-}1 : \ e(P_{k\text{-}t,j}, Q_t) = e(P_{k,j}, B_2) \wedge e(B_1, Q_k) = e(P_{k,j}, B_2) \tag{3.10}$$

Note that the right-hand part of the equations 3.10 can be computed once. Note also that equations 3.9 and 3.10 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 $e(A_1, B_1)$+...+$e(A_m, B_m) = 0$).:

$$\begin{matrix} e(A_1, B_1) = e(C_1, D_1) \\ e(A_2, B_2) = e(C_2, D_2) \\ \cdots \\ e(A_m, B_m) = e(C_m, D_m) \end{matrix} \quad \Leftrightarrow \begin{cases} e(\alpha_1 A_1, B_1) + e(-\alpha_1 C_1, D_1) + \\ \quad e(\alpha_2 A_2, B_2) + e(-\alpha_2 C_2, D_2) + \\ \quad \cdots \\ \quad e(\alpha_m A_m, B_m) + e(-\alpha_m C_m, D_m) = 0 \end{cases} \tag{3.11}$$

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).

4 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: (i) using a generic transparent SNARK and (ii) using the Dory polynomial commitment scheme, and (iii) using an inferior method of inner-pairing product argument (see Appendix C).

On 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 Celestia, Polygon Avail, or Arweave. 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. 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.

4.1 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

$$\begin{aligned}\mathsf{pp}' &\leftarrow \Big(rP_1, r^2 P_2, r^3 P_3, \ldots, r^n P_n; \quad rQ_1, r^2 Q_2, \ldots, r^k Q_k\Big) = \\ &= \Big(P_1', P_2', P_3', \ldots, p_n'; \quad Q_1', Q_2', \ldots, Q_k'\Big) \in \mathbb{G}_1^n \times \mathbb{G}_2^k\end{aligned}$$

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 3, 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:

public statement: c, c' and $\rho_1, \rho_2 \in \mathbb{Z}_p$, witness: pp, pp', 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 [BBHR18] 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 5 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'.

4.2 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. [AFG+10] in combination with the Dory [Lee21] 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 3 and a logarithmic-size proof for the following inner-pairing product (IPP) relations:

$$\begin{aligned}
&C_1 = \langle \mathbf{P}, \mathbf{\Gamma}_2 \rangle \qquad C_2 = \langle \mathbf{\Gamma}_1, \mathbf{Q} \rangle \\
&\rho_1^n P_n Q_1 - B_1 Q_1 = \langle \mathbf{P}, (1, \rho_1, \rho_1^2, \ldots, \rho_1^{n-1}) \cdot (\rho_1 Q_1 - B_2) \rangle \\
&\rho_2^k P_1 Q_k - P_1 B_2 = \langle (1, \rho_2, \rho_2^2, \ldots, \rho_2^{k-1}) \cdot (\rho_2 P_1 - B_1), \mathbf{Q} \rangle \\
&P_n = \langle \mathbf{P}, (0, 0, \ldots, 0, 1) \rangle \qquad Q_k = \langle \mathbf{Q}, (0, 0, \ldots, 0, 1) \rangle \\
&P_1 = \langle \mathbf{P}, (1, 0, \ldots, 0, 0) \rangle \qquad Q_1 = \langle \mathbf{Q}, (1, 0, \ldots, 0, 0) \rangle
\end{aligned} \tag{4.1}$$

We give more details on constructing the proof for these IPPs in Appendix B.

5 Discussion and open problems

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. 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 it may enable them to claim rewards multiple times or drain the available budget for covering transaction fees, preventing other users from participating cheaply.

Censorship-resistance. Our ceremonies are designed to run without any centralized coordination, but they do require contributions in a serial manner. At time j, the next contributor must prove correctness 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 contributors in a verifiable random way and then execute the protocol according to the list, with each user given a pre-assigned slot to contribute.

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. However, caution is in order as many (though not all) ZK-rollups themselves rely on a trusted setup, leading to a circular dependency. 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 typically average 100x cheaper on Layer-2, and execution costs (as opposed to storage) are about 75% of total transaction costs [l2f22]. Combined with off-chain data availability, total costs can be greatly reduced. 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.

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 accompanied 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 [Val08] or accumulation [BCMS20]. 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.

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 [BGR98b, CHP12]. We note that, in our protocol in Section 3, 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 structure reference string.

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.

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.

Proving 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.

Generating a powers-of-tau setup with a punctured point Some systems require a powers-of-tau string where one power in the sequence is absent, namely

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

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 [Gro10], Attema and Cramer [AC20], Lipmaa, Siim, and Zajac's Vampire scheme [LSZ22], and Waters and Wu [WW22]. The absence of the point P_{N+1} from pp is necessary for security. Check #2 in (3.5) 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:

$$e\Big(\sum_{\substack{i=1\\ i\neq N+1\\ i\neq N+2}}^{2N} \rho_1^{i\text{-}1} P_i, \quad B_2 + \sum_{\ell=1}^{N-1} \rho_2^{\ell} Q_\ell\Big) = e\Big(B_1 + \sum_{\substack{i=1\\ i\neq N\\ i\neq N+1}}^{2N-1} \rho_1^{i} P_i, \quad \sum_{\ell=1}^{N} \rho_2^{\ell\text{-}1} Q_\ell\Big) \tag{5.1}$$

$$e\Big(P_{N+2}, B_2\Big) = e\Big(P_N, Q_2\Big) \tag{5.2}$$

It is not difficult to see that a well-formed setup will pass the check successfully. We argue soundness, i.e. that this check guarantees the well-formedness of the setup, in the note at the end of Appendix A.

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. Boneh is supported by the Simons Foundation and NTT Research.

References

[AC20] Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In *CRYPTO'20*, volume 12172 of *Lecture Notes in Computer Science*, pages 513–543. Springer, 2020.

[AFG$^+$10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In *Annual Cryptology Conference*, pages 209–236. Springer, 2010.

[Ale21] Aleo. Announcing aleo setup. `https://www.aleo.org/post/announcing-aleo-setup`, 2021.

[ASB$^+$19] Alexey Akhunov, Eli Ben Sasson, Tom Brand, Louis Guthmann, and Avihu Levy. Eip-2028: Transaction data gas cost reduction. `https://eips.ethereum.org/EIPS/eip-2028`, 2019.

[ASC18] Zachary Williamson Antonio Salazar Cardozo. Eip-1108: Reduce alt_bn128 precompile gas costs. `https://eips.ethereum.org/EIPS/eip-1108`, 2018.

[AV20] Kelly Olson Alex Vlasov. Eip-2537: Precompile for bls12-381 curve operations. `https://eips.ethereum.org/EIPS/eip-2537`, 2020.

[Azt20] Aztec. Universal crs setup. `https://docs.zksync.io/userdocs/security/#universal-crs-setup`, 2020.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, page 46, 2018.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In *TCC*, 2020.

[BD19] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of cryptology*, 32(4):1298–1336, 2019.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *Cryptology ePrint Archive*, 2017.

[BGR98a] Mihir Bellare, Juan A Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *International conference on the theory and applications of cryptographic techniques*, pages 236–250. Springer, 1998.

[BGR98b] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *Eurocrypt*, 1998.

[BM93] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Eurocrypt*, 1993.

[BMM$^+$21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 65–97. Springer, 2021.

[BSCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *IEEE Symposium on Security and Privacy*, 2015.

[But20] Vitalk Buterin. What is Danksharding, 2020.

[But22] Vitalik Buterin. "How do trusted setups work?". https://vitalik.ca/general/2022/03/14/trustedsetup.html, 2022.

[Cas20] Tornado Cash. Tornado.cash trusted setup ceremony. https://tornado-cash.medium.com/tornado-cash-trusted-setup-ceremony-b846e1e00be1, 2020.

[CDK+22] Ran Cohen, Jack Doerner, Yashvanth Kondi, et al. Guaranteed output in o(sqrt(n)) rounds for round-robin sampling protocols. *Cryptology ePrint Archive*, 2022.

[Cel20] Celo. Plumo ceremony. https://celo.org/plumo, 2020.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: preprocessing zkSNARKs with universal and updatable SRS. In *Eurocrypt*, 2020.

[CHP12] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. *J. Cryptol.*, 25(4):723–747, 2012.

[Coma] The Electric Coin Company. Halo2. https://github.com/zcash/halo2.

[Comb] The Electric Coin Company. The halo2 book. https://zcash.github.io/halo2/.

[Dev19] Brecht Devos. Loopring starts zksnark trusted setup multi-party computation ceremony. https://medium.loopring.io/loopring-starts-zksnark-trusted-setup-multi-party-computation-ceremony-6582874f7a5b, 2019.

[DL77] Richard A DeMillo and Richard J Lipton. A probabilistic remark on algebraic program testing. Technical report, GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION AND COMPUTER SCIENCE, 1977.

[Fil20] FileCoin. Trusted setup complete! https://filecoin.io/blog/posts/trusted-setup-complete/, 2020.

[Fou22] Ethereum Foundation. Ethereum: Powers of tau specification. https://github.com/ethereum/kzg-ceremony-specs, 2022.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT'10*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2010.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019.

[Her20] Polygon Hermez. Hermez zero-knowledge proofs. https://blog.hermez.io/hermez-zero-knowledge-proofs/, 2020.

[KB16] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *Annual international cryptology conference*, pages 543–571. Springer, 2016.

[KMSV21] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In *AsiaCrypt*, 2021.

[Kus18] John Kuszmaul. V(ery short m)erkle trees. verkle trees. `https://math.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf`, 2018.

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*, pages 177–194. Springer, 2010.

[l2f22] "l2 fees". `https://l2fees.info/`, 2022.

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography Conference*, pages 1–34. Springer, 2021.

[LSZ22] Helger Lipmaa, Janno Siim, and Michal Zajac. Counting vampires: From univariate sumcheck to updatable zk-snark. *Cryptology ePrint Archive*, 2022.

[LY10] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *Theory of Cryptography Conference*, pages 499–517. Springer, 2010.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 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*, pages 2111–2128, 2019.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.

[RY07] Thomas Ristenpart and Scott Yilek. 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*, pages 228–245. Springer, 2007.

[Sch80] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC*, 2008.

[VB17] Christian Reitwiessner Vitalik Buterin. Eip-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. `https://eips.ethereum.org/EIPS/eip-197`, 2017.

[Vla20] Alex Vlasov. Eip-2539: Bls12-377 curve operations. `https://eips.ethereum.org/EIPS/eip-2539`, 2020.

[WUP22] Weijie Wang, Annie Ulichney, and Charalampos Papamanthou. BalanceProofs: Maintainable Vector Commitments with Fast Aggregation. Cryptology ePrint Archive, Paper 2022/864, 2022.

[WW22] Brent Waters and David Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO'22*, 2022.

[YEH20] Aurore Guillevic Youssef El Housni, Michael Connor. Eip-3026: Bw6-761 curve operations. `https://eips.ethereum.org/EIPS/eip-3026`, 2020.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979.

A Proof of Theorem 3.1

In this section we prove Theorem 3.1 of Section 3 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:

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

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 [DL77, Zip79, Sch80], 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 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 5 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. A.2:

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

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 (5.2) to get $a_{N+2} = a_N b_2$ which implies $a_{N+2} = \tau^{N+2}$.

B Inner-pairing product arguments for Section 4.2

We restate Eq. 4.1 of Section 4.2 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 3.1. 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:

$$\begin{aligned} & x^n a_n b_1 - b_1 - (a_1 + x a_2 + x^2 a_3 + \ldots + x^{n-1} a_n) \cdot (x b_1 - 1) = 0 \Longleftrightarrow \\ & \quad (a_1 - b_1) + (a_2 - a_1 b_1) x + (a_3 - a_2 b_1) x^2 + \ldots + (a_n - a_{n-1} b_1) x^{n-1} = 0 \end{aligned} \tag{B.9}$$

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 [DL77, Zip79, Sch80], 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:

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

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 B.1:

Theorem B.1. *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 3, Eq. 3.4.
3. The verifier checks that the update is non-degenerative: $P_1 \neq 0$ and if so replies with two random scalars $\rho_1, \rho_2 \overset{\$}{\leftarrow} \mathbb{Z}_p$.
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 [Lee21]. 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):

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

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 [Lee21] 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. [BMM+21]. 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. [AFG+10] as follows:

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

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 4.2. 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 4.2, 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.