

KZG ceremony

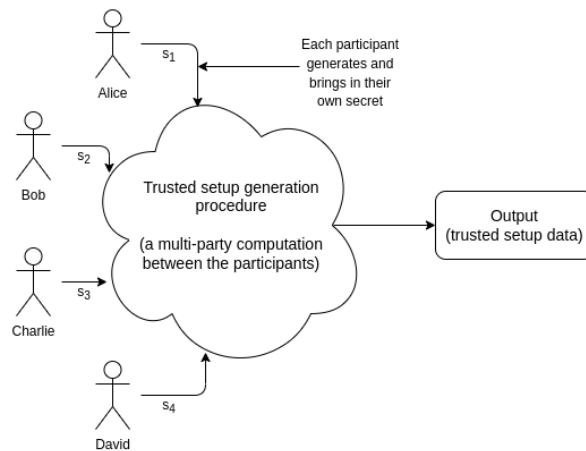
Sofia Martellozzo (10623060)

1 Setup ceremony

A trusted setup ceremony is a one-time procedure to generate public parameters/piece of data that must then be used every time some cryptographic protocol is run.

Generating this data requires some secret information: the "trust".

The trust comes from the fact that some person or some group of people has to generate these secrets, use them to generate the data, and then publish the data and forget the secrets. But once the data is generated, and the secrets are forgotten, no further participation from the creators of the ceremony is required (Vitalik Buterin, [3]).



In the simplest case of a fully trusted setup a single entity computes $\text{Setup}()$ and is trusted to discard τ (the trapdoor).

Many cryptographic protocols, especially in the areas of data availability sampling and zkSNARKs depend on trusted setups.

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.

There are many types of trusted setups: the earliest instance being used in a major protocol is the original Zcash ceremony in 2016. This ceremony was very complex, and required many rounds of communication, so it could only have six participants. Everyone using Zcash at that point was effectively trusting that at least one of the six participants was honest.

More modern protocols usually use the powers-of-tau setup, which has a 1-of-N trust model with N typically in the hundreds, it means that: hundreds of people participate in generating the data together, and only one of them needs to be honest and not publish their secret for the final output to be secure. Well-executed setups like this are often considered "close enough to trustless" in practice.

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.

2 Power-of-tau

We focus on a common type of ceremony which constructs a power-of-tau [8] Structured Reference String (SRS). It works with elliptic curve groups \mathbb{G}_1 and \mathbb{G}_2 , with generators B_1 and B_2 respectively and an efficiently computable pairing. The goal is to produce a public parameter string in the form:

$$\text{pp} := (\tau B_1, \tau^2 B_1, \dots, \tau^n B_1; \tau B_2, \tau^2 B_2, \dots, \tau^m B_2) \in \mathbb{G}_1^n \times \mathbb{G}_2^m$$

n and m are the lengths of the \mathbb{G}_1 and \mathbb{G}_2 sides of the setup.

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.

Historically, ceremonies have been run through a centralized coordinator which fulfills several important functions, all of which were intended to be replaced with a decentralized fashion:

- Consensus: participants should agree on the final value of pp .

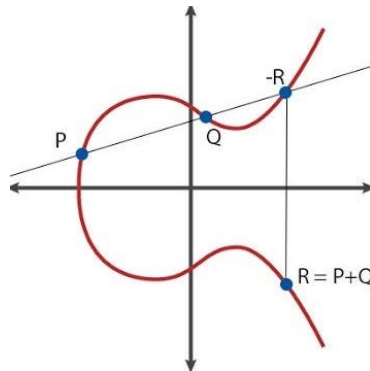
- **Verification:** each participant must provide a zero-knowledge proof that their contribution is a valid re-randomization (and not simply replacing the string with one for which the trapdoor is known).
- **Data Availability:** the final string must be available for all to download, as well as the history of prior versions and participants for auditability.
- **Censorship Resistance:** any willing participant should be able to contribute.

Ethereum trusted setup ceremony for its upcoming ProtoDankSharding and DankShardings upgrades with targeted sizes: $2^{12}, 2^{13}, 2^{14}, 2^{15}$ 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 suggested expiry 30–60 days. The ceremony run for a few weeks and is the the largest trusted setup ceremony in terms of participation for blockchains so far. We will deep into it in the next chapters.

2.1 Elliptic Curve Pairings

Elliptic curve pairings (or "bilinear maps") are a recent addition to a 30-year-long history of using elliptic curves for cryptographic applications including encryption and digital signatures; pairings introduce a form of "encrypted multiplication", greatly expanding what elliptic curve-based protocols can do.

Elliptic curve cryptography involves mathematical objects called "points" (these are literal two-dimensional points with (x,y) coordinates), with special formulas for adding and subtracting them (i.e. for calculating the coordinates of $R = P + Q$) and you can also multiply a point by an integer (i.e. $P \cdot n = P + P + \dots + P$, though there's a much faster way to compute it if n is big) (Vitalik Buterin, [2]).



There exists a special point called the "point at infinity" (O), the equivalent of zero in point arithmetic; it's always the case that $P + O = P$.

Also, a curve has an "order"; there exists a number n such that $P \cdot n = O$ for any P . There is also some commonly agreed upon "generator point" G , which is understood to in some sense represent the number 1. Theoretically, any point on a curve (except O) can be G ; all that matters is that G is standardized.

Pairings go a step further in that they allow you to check certain kinds of more complicated equations on elliptic curve points — for example, if $P = G \cdot p, Q = G \cdot q$ and $R = G \cdot r$, you can check whether or not $p \cdot q = r$, having just P, Q and R as inputs.

It seems like here the security of elliptic curves are being broken (as information about p is leaking from just knowing P) but it turns out that the leakage is highly contained; the computational Diffie Hellman problem (knowing P and Q in the above example, computing $R = G \cdot p \cdot q$) and the discrete logarithm problem (recovering p from P) remain computationally infeasible.

A third way to look at what pairings do is that if you view elliptic curve points as one-way encrypted numbers (that is, $encrypt(p) = p \cdot G = P$).

Traditional elliptic curve math lets you check linear constraints on the numbers, pairings let you check quadratic constraints (eg. checking $e(P, Q) \cdot e(G, G \cdot 5) = 1$ is really checking that $p \cdot q + 5 = 0$).

The operator $e(P, Q)$ is the pairing, it is also called bilinear map because it satisfies the constraints:

$$\begin{aligned} e(P, Q + R) &= e(P, Q) \cdot e(P, R) \\ e(P + S, Q) &= e(P, Q) \cdot e(S, Q) \end{aligned}$$

Note that $+$ and \cdot can be arbitrary operators.

It turns out that it is possible to make a bilinear map over elliptic curve points — that is, come up with a function $e(P, Q)$ where the inputs P and Q are elliptic curve points and where the output is what's called an $(F_p)^{12}$ element.

Instead of using regular real number we use numbers in a prime field: it is a set of numbers $0, 1, 2, \dots, p - 1$ where p is prime, and the various operations are defined in modulo p :

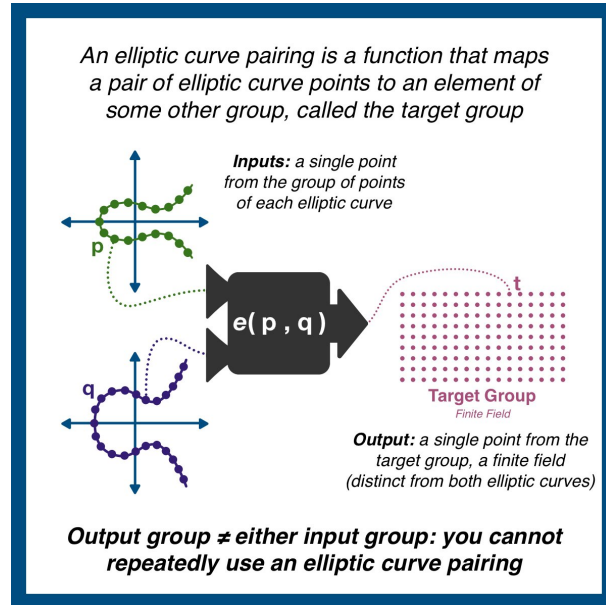
$$\begin{aligned} a + b &: (a + b) \% p \\ a - b &: (a - b) \% p \\ a \cdot b &: (a \cdot b) \% p \\ a / b &: (a / b) \% p \end{aligned}$$

About the Modular Arithmetic, imagine a clock being mod 12, adding it to elliptic curve it is very difficult to solve discrete logarithm problem (that is, if you add two points over and over again using modular arithmetic, it takes a lot of work to find out how many times you did it).

To illustrate this concept, consider a clock that resets after 12 hours. Imagine you're posed with a question like, "Alice left at 4 o'clock and arrived at 6 o'clock. How many hours did she spend traveling?" The answer could be 2 hours, 14 hours, 26 hours. The only way to arrive at the correct answer is by making guesses.

This situation precisely mirrors the discrete logarithm problem.

The following image and explanation from (Haym Salomon, [9]) help to better understand the concept of elliptic curve pairings:



Actually there are more types of pairings, it is a map $G_1 \times G_2 \rightarrow G_t$, where:

- G_1 is an elliptic curve where points satisfy an equation of the form $y^2 = x^3 + b$ and where both coordinates are elements of F_p
- G_2 is an elliptic curve, where points satisfy the same equation as G_1 , except where the coordinates are elements of $(F_p)^{12}$.
- G_t is the type of object that the result of the elliptic curve goes into. In the curves that we look at G_t is $(F_p)^{12}$.

As said before, the main property that it must satisfy is bilinearity, but also: efficient computability and non-degeneracy.

Every elliptic curve has a value called an embedding degree (k).

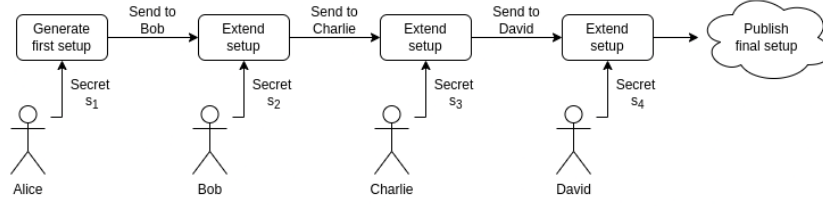
If $k = 1$, then the "discrete logarithm" problem for elliptic curves can be reduced into a similar math problem over $(F_p)^{12}$, where the problem becomes much easier (this is called the MOV attack). Using curves with an embedding degree of 12 or higher ensures that this reduction is either unavailable, or that solving the discrete log problem over pairing results is at least as hard as recovering a private key from a public key "the normal way" (i.e. computationally infeasible).

3 Multi-participant setup

It is easy to see how one participant can generate a setup: just pick a random s (secret, trapdoor) and generate the elliptic curve points using that s .

However, a single-participant trusted setup is insecure since a specific person must be trusted.

The solution to this is multi-participant trusted setups (Vitalik Buterin, [3]), where the number of participants is usually over 100 and for smaller setups it's possible to get over 1000. Here is how a multi-participant powers-of-tau setup works.



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 and a bilinear pairing operator $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$. The goal is to construct a "powers of τ " structured reference string (SRS) of the form:

$$pp = [\tau B_1, \tau^2 B_1, \dots, \tau^n B_1; \tau B_2, \tau^2 B_2, \dots, \tau^k B_2]$$

It is essential that τ be kept secret in the final string, pp .

The protocol to construct pp is 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.

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

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

That is n copies of the generator B_1 plus k copies of the generator B_2 .

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

Then the update procedure works as follows: at the beginning of round j the current string, assumed to be:

$$\begin{aligned} pp &= [P_{1,j-1}, P_{2,j-1}, \dots, P_{n,j-1}; Q_{1,j-1}, Q_{2,j-1}, \dots, Q_{k,j-1}] \\ &= [\tau_{j-1} B_1, \tau_{j-1}^2 B_1, \dots, \tau_{j-1}^n B_1; \tau_{j-1} B_2, \tau_{j-1}^2 B_2, \dots, \tau_{j-1}^k B_2] \end{aligned}$$

The value of τ_{j-1} is of course hidden.

Contributor C_j chooses a random value r_j and publishes a new string:

$$\begin{aligned} pp &= [P_{1,j}, P_{2,j}, \dots, P_{n,j}; Q_{1,j}, Q_{2,j}, \dots, Q_{k,j}] \\ &= [r_j P_{1,j-1}, r_j^2 P_{2,j-1}, \dots, r_j^n P_{n,j-1}; r_j Q_{1,j-1}, r_j^2 Q_{2,j-1}, \dots, r_j^k Q_{k,j-1}] \\ &= [r_j \tau_{j-1} B_1, r_j^2 \tau_{j-1}^2 B_1, \dots, r_j^n \tau_{j-1}^n B_1; r_j \tau_{j-1} B_2, r_j^2 \tau_{j-1}^2 B_2, \dots, r_j^k \tau_{j-1}^k B_2] \\ &= [\tau_j B_1, \tau_j^2 B_1, \dots, \tau_j^n B_1; \tau_j B_2, \tau_j^2 B_2, \dots, \tau_j^k B_2] \end{aligned}$$

The new setup has $\tau_j = r_j \cdot \tau_{j-1}$ as its secret.

If an attacker knows τ_{j-1} but not r_j , and r_j was chosen uniformly at random, then the attacker will have no information about τ_j .

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 ($\tau_m = r_1 \cdot r_2 \cdot \dots \cdot r_m$) is randomly distributed and unknown to anybody.

3.1 Verification of the setup

To verify that each participant actually participated (Vitalik Buterin, [3]), each participant can provide a proof that consists of:

- the $B_1\tau$ that they received
- B_2r with r the secret they introduce.

The list of these proofs can be used to verify that the final setup combines together all the secrets (as opposed to, say, the last participant just forgetting the previous values and outputting a setup with just their own secret, which they keep so they can cheat in any protocols that use the setup).

In the following image the $G_1 = B_1$, $G_2 = B_2$, $s_i = r_j$ each new contribution: s_1 is the first participant secret, s_2 the second participant's secret ...

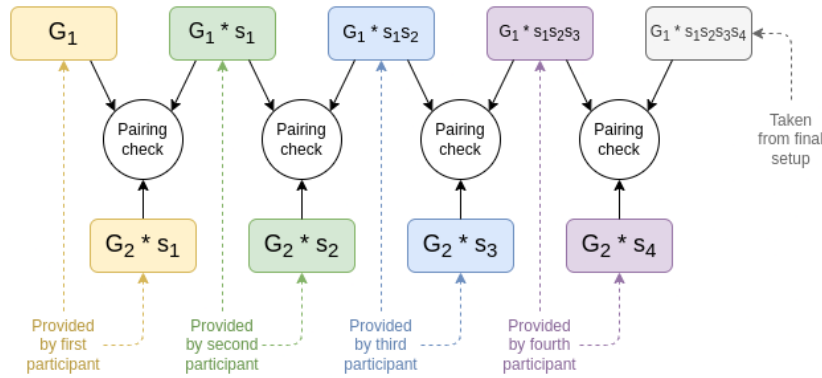


Figure 1: The pairing check at each step proves that the setup at each step actually came from a combination of the setup at the previous step and a new secret known by the participant at that step.

This mechanism does not prevent someone from claiming to have participated at some index where someone else has (assuming that other person has revealed their proof), but it's generally considered that this does not matter: if someone is willing to lie about having participated, they would also be willing to lie about having deleted their secret. As long as at least one of the people who publicly claim to have participated is honest, the setup is secure.

A code implementation of a trusted setup can be found [here](#).

4 Background

In recent times, there has been growing dissatisfaction with Ethereum (Joseph Harris, [5]). Ethereum transactions often become slow and costly, particularly when compared to more competitive alternatives like Solana. These drawbacks are not a result of Ethereum's age or outdated technology. Rather, performance issues and high fees are a result of Ethereum's decentralized nature. Ethereum's primary objective is to lower the barriers of entry for individuals who want to actively participate in the network. Consequently, the

Ethereum software must be capable of running on various types of computers, including budget-friendly, low-powered ones. As a result, Ethereum blocks cannot handle an excessive number of transactions or data because doing so would overwhelm the less powerful computers within the network. Consequently, even a modest increase in transaction volume can overwhelm Ethereum's blocks. When this occurs, users are compelled to outbid one another in a bid to ensure their transactions are given priority and included in a block. This competitive environment leads to the high transaction fees that frequently draw criticism. To address this challenge, the Ethereum community has devised a plan known as "Layered Scaling." This strategy seeks to maintain the core principles of accessibility and decentralization while simultaneously expanding capacity and reducing transaction costs.

4.1 Layered Scaling

Layered Scaling primarily revolves around consolidating a significant volume of activity into a limited number of Ethereum transactions (Joseph Harris, [5]). The idea is to shift the majority of day-to-day transactions away from Ethereum and onto innovative 'Layer Two' (L2) platforms.

These Layer Two platforms are designed with a strong emphasis on performance, ensuring that transaction fees remain affordable even during periods of high demand. Periodically, all recent L2 transactions are grouped, condensed, and represented within a single transaction on the Ethereum blockchain. This method allows all activities to still occur on the Ethereum blockchain, although in a more indirect manner. Crucially, it guarantees that all L2 transactions are verified and secured by Ethereum. This results in an ideal situation for end users – they can still enjoy all the advantages of Ethereum and its robust, decentralized architecture without enduring its low processing capacity, slow transactions, and exorbitant fees.

While this approach is the optimal way to construct and expand a blockchain, it is worth noting that Ethereum was not initially designed with this methodology in mind. Consequently, certain adjustments are necessary to maximize the potential of a more modular design. One of the main challenges in this context is data storage. Consider how Layer Twos consolidate all their recent transactions and represent them in a single Ethereum transaction. This Ethereum transaction must contain a substantial amount of data, effectively summarizing all the recent activity on L2s.

The issue is the fact that Ethereum offers only one type of data storage: permanent. When an L2, or any other entity for that matter, uploads a substantial amount of data to Ethereum, it necessitates that everyone within the Ethereum network maintains that data indefinitely. This is not straightforward and places a considerable burden on the network's computers. This situation isn't favorable for Ethereum because its priority has always been minimising the burden on those computers to keep the network decentralised.

This situation also poses challenges for the L2 platforms. Ethereum is keen on avoiding data overload that could overwhelm the network's low-end computers, leading to high charges for data storage. Consequently, L2s are forced to incur significant costs to execute their settlement transactions on Ethereum, and the users of L2 platforms must incur higher fees to accommodate these expenses. This is paradoxical, particularly when the fundamental objective of L2 platforms is to provide a low-fee environment. Hence, there is a pressing need for an improved solution for data storage, and this is where Proto-Danksharding enters the picture.

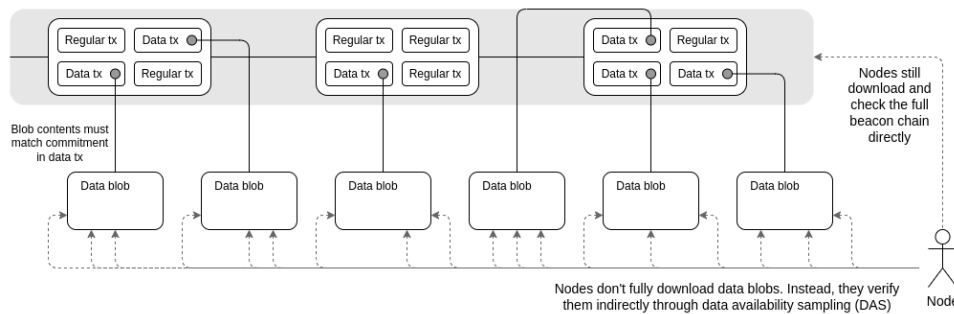
5 Proto-Danksharding

Proto-Danksharding (officially named EIP-4844) will introduce a new type of transaction to Ethereum, and these transactions will unlock a new kind of short-term storage.

The main difference between these recent Ethereum sharding proposal and most non-Ethereum sharding proposals is Ethereum's rollup-centric roadmap: instead of providing more space for transactions, it provides more space for blobs of data (which Ethereum protocol itself does not attempt to interpret).

These new transactions are called 'blob-carrying transactions'. 'Blob' is short for Binary Large Object, which refers to an arbitrarily large chunk of data. So, when we say 'blob-carrying transaction', we just mean a transaction that is associated and travelling with some piece of data, called blob. Data that might, for example, summarise a whole bunch of L2 transactions.

Verify a blob in danksharding [1] simply requires checking that the blob is available - that it can be downloaded from the network. The data space in these blobs is expected to be used by layer-2 rollup protocols that supports high-throughput transactions.



In a proto-danksharding implementation, all validators and users still have to directly validate the availability of the full data.

The term 'blob' is quite fitting in this context. To simplify this concept, think of these data segments as sticky, tangible "blobs" that can attach themselves to an Ethereum block, much like barnacles adhering to a boat's hull. Just as barnacles are periodically removed from boats to prevent issues, these data blobs are periodically removed from blocks and discarded before they overburden the network.

These "blobs" are notably large, around 125 kilobytes, and are often more cost-effective than equivalent amounts of calldata. However, it's important to note that the Ethereum Virtual Machine (EVM) can't directly access blob data; it can only observe a commitment to the blob.

In the context of proto-danksharding, validators and clients must still download the complete contents of these blobs, but the data bandwidth is reduced to 1 megabyte per slot, rather than the full 16 megabytes. This approach yields significant scalability improvements because this data doesn't compete with the gas consumption of existing Ethereum transactions. This marks a significant enhancement introduced by Proto-Danksharding. Any data contained within a blob will only be retained on Ethereum for a short duration, typically a few weeks. After this period, anyone in the Ethereum network can safely remove and forget this data, lightening the load on all devices.

This approach is feasible because not all parties need to retain this data indefinitely. It is crucial that the Layer 2 (L2) data is widely distributed and accessible to those who

require it. Attaching it to the exterior of an Ethereum block and disseminating it across an uncensorable global network is an effective means of ensuring accessibility.

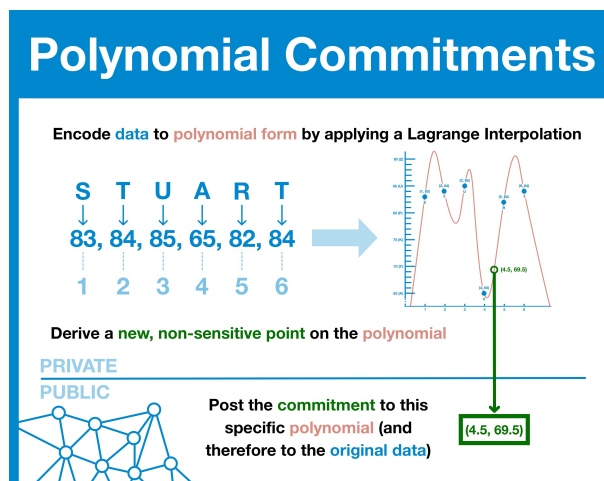
After a few weeks, interested parties should have had ample time to create permanent copies or carry out any necessary actions with the data. Transactions that carry these blobs are a substantial benefit for Ethereum.

Short-term storage eases the strain on the Ethereum network and doesn't pose the same decentralization risks as permanent storage. Furthermore, it benefits Layer 2 solutions, as the reduced cost of short-term storage allows them to offer more at a significantly lower price, ultimately passing on these savings to end users.

6 KZG Commitments

6.1 Polynomial Commitments

Before delving into KZG, it is essential to establish a foundational understanding of polynomial commitments. These commitments are a fundamental concept that underlies cryptographic techniques, ensuring data integrity and confidentiality [7] [10]:



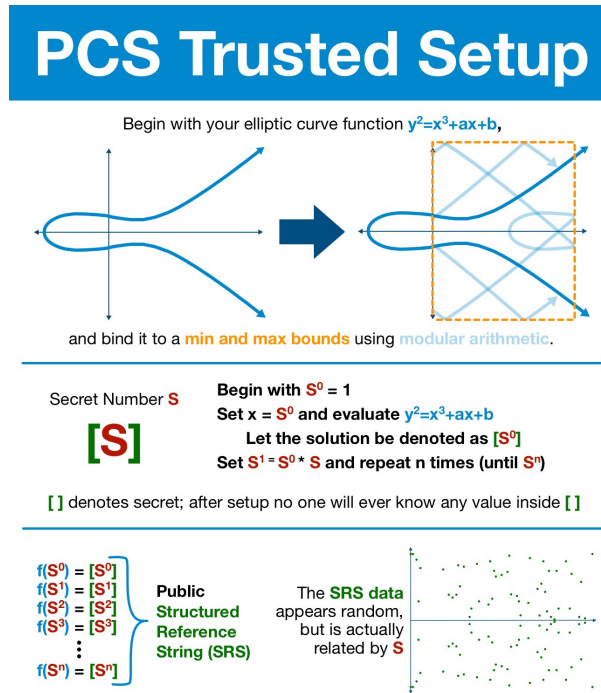
In this context, we can simplify the concept as follows:

1. **Data Chunking:** The first step involves dividing the dataset into discrete segments or chunks. This process essentially breaks down the data into more manageable parts.
2. **Graphical Representation:** Following data chunking, these chunks are plotted on a graph, and a line is drawn through these points. The purpose of this line is to find a mathematical formula that accurately represents the relationship between the data points.
3. **Polynomial Formula:** The derived formula is known as a polynomial, a specific type of mathematical equation with unique properties. This polynomial captures the same information as the original dataset. Moreover, it enables the generation of additional data points.

4. Commitment: This polynomial serves as a commitment mechanism. It acts as proof that a party possesses complete knowledge of the dataset, as they can recreate the polynomial. Importantly, this commitment does not expose the actual data it represents, maintaining its confidentiality.

In essence, polynomial commitments are a cryptographic tool that verifies data ownership without revealing the underlying data, ensuring data security and trustworthiness in various applications.

6.2 PCS Trusted setup



Any cryptographic system has the same two part goal:

1. transform data into digital-nonsense, indistinguishable from random noise
2. allow specific individuals (and only those individuals) to reverse the process and recover the original data

Solving the elliptic curve discrete logarithm problem is a highly challenging task. This difficulty serves as the foundation for constructing secure systems, beginning with the concealment of a confidential number within an elliptic curve.

To illustrate, consider selecting a random number, denoted as S , which will be discarded at the process's conclusion. Next, apply modular arithmetic to constrain the elliptic curve within specific minimum and maximum bounds. Subsequently, employ this modular elliptic curve in conjunction with the secret value S to produce a series of values, ranging from S^0 to S^n . Each value is generated by inputting it into our elliptic curve formula, yielding a new point on the curve.

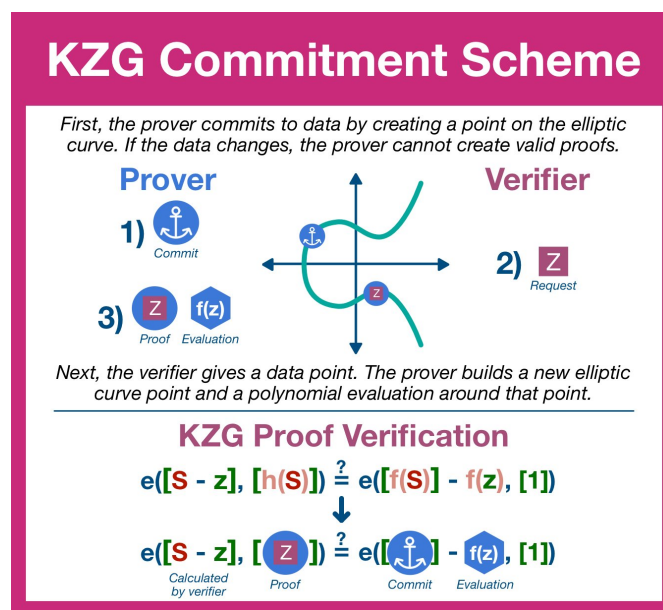
In the absence of S , the collection of points appears to be like noise. While there is apparent horizontal symmetry, the overall pattern seems almost random. However, this

is intentional; it is not truly random. These points are all that remains of S - a concealed number buried within a scattered array of points.

This seemingly arbitrary array of points is virtually indistinguishable from random data. The outcome of this trusted setup is a Structured Reference String (SRS). The SRS provides sufficient information to access the concealed random number for our polynomial (and, consequently, our polynomial commitment scheme) while maintaining the value's complete secrecy.

It is worth noting that a single SRS can serve multiple individuals, requiring only one trusted setup (or one trusted setup per specific application) for all KZG commitments (Haym Salomon,[10]).

The following image from (Haym Salomon, [10]) provide a scheme of the KZG Commitment:



This is an interactive method for proving the truth of a statement, involving two parties: the prover and the verifier.

The prover responds with two pieces of information: the result of applying a polynomial to an elliptic curve, and the output of this polynomial at that specific point. The former is represented by a point on the elliptic curve, while the latter corresponds to the value of the polynomial at that particular position.

It's important to note that the polynomial describes the visual representation of our data when plotted on a graph. Evaluating this function simply means obtaining the data at that specific position.

To conclude this proof process, the verifier, who lacks access to the entire dataset and is unable to generate the data or quotient polynomial, must ensure that the prover's response matches the initial commitment. The verifier accomplishes this by employing elliptic curve pairings to combine the components sent by the prover. Two expressions are constructed and compared for equality. If they match, the verifier can be confident that the prover performed the (polynomial) division, confirming their possession of the original data.

We have just seen that a ‘blob-carrying transaction’ is a transaction sent alongside and associated with some arbitrary piece of data. As well as travelling together, the transaction and blob are associated through a special reference or summary that is embedded in the transaction. It is worth noting, because it is part of a transaction and included in an Ethereum block, this reference does become a permanent part of Ethereum’s history — even after the blob itself is discarded.

The reference is something called a KZG Commitment (Dankrad Feist , [4]) to the blob. To be more precise, it’s a versioned hash of a KZG Commitment to the blob. KZG Commitments, and commitment schemes in general, serve the purpose of demonstrating possession or awareness of specific information without disclosing the information itself. Subsequently, once the information becomes public, the commitment can serve as a means of validating the authenticity of that information. However, this validation would fail if there were even the slightest variance between the committed information and the disclosed information.

In the case of Proto-Danksharding, KZG Commitments give users an efficient way to verify that blobs contain all the information they supposedly contain and that L2s aren’t misbehaving. KZG Commitments can also be used to prove that a certain piece of data was contained in a blob. This is essential for a type of L2 called an Optimistic Rollup, as they rerun disputed transactions on Ethereum to verify the results. Of course, there’s no point rerunning a transaction that never actually happened on the L2, which is where the KZG proof comes in. When someone raises a dispute, they must also provide a KZG proof to show the disputed transaction took place and was included in a blob.

KZG Commitments have just two noteworthy drawbacks.

First, they are not quantum-resistant, so there could be security issues if someone develops a powerful quantum computer. However, this concern is not particularly pressing at the moment, and when such a development occurs, there will likely be numerous other significant concerns to contend with, making this less of a priority.

The second issue is more immediately concerning. KZG Commitments require what is known as a "trusted setup". This is where the Summoning Ceremony plays a crucial role.

6.3 KZG Trusted setup

To summarize the procedure to generate a KZG Commitment (Joseph Harris, [5]), the data to be committed is initially transformed into a polynomial, which appears as follows:

$$a + bx + cx^2 + dx^3 + \dots$$

Subsequently, this polynomial is assessed at specific points, which essentially involves substituting certain values to observe the resulting outcome. This resulting outcome constitutes the KZG Commitment.

When a party intends to verify the integrity of the data at a later time, they can essentially replicate the process. By evaluating the polynomial at the same set of points, they can compare the output with the original commitment. If the two match, it signifies that the data remains unchanged. In case of any disparities, it indicates alterations to the data. It is crucial to emphasize that the values input into the equation must be kept confidential.

If these values were to become known, an individual could potentially manipulate the system by constructing a new polynomial that produces the same output. In simpler terms, they could initially commit to one dataset and subsequently modify it completely while still providing a valid proof.

To maintain the confidentiality of these numbers, elliptic curves offer a solution. These curves constitute a distinct category characterized by equations in the following form:

$$y^2 = x^3 + ax + b$$

Elliptic curves possess unique attributes, enabling the addition of points on the curve to yield a third point, also residing on the curve. When this process is carried out, the location of the resulting third point becomes challenging to predict. Moreover, when points are repeatedly combined, it becomes exceptionally complex to discern the number of times these points have been aggregated. Mathematicians employ this property to conceal numerical values within an elliptic curve.

In essence, they take the "secret" number and iteratively combine it with elliptic curve points. The outcome of this operation is an array of elliptic curve points, which, when graphically represented, appears entirely random. This array is termed a Structured Reference String (SRS), and it can be thoroughly analyzed by a potential adversary without revealing any information regarding the concealed secrets. Remarkably, the SRS still contains adequate data about the secret values, making it valuable for constructing KZG Commitments.

Now, we must have questions about the origin of the Structured Reference String (SRS). Who creates it, and how can they assure others that they do not possess any confidential information?

The solution to this matter lies in the implementation of a mechanism known as a "trusted setup." The general objective of this industry is, in essence, to eliminate the dependency on singular trusted entities and instead substitute them with extensive, diverse networks including individual operators. This transformation serves to heighten the overall trustworthiness of the system.

In simple terms, a trusted setup involves the active participation of a large number of individuals in the construction of the Structured Reference String. Rather than a single person contributing the entire secret and then assuring that they do not know its content, a trusted setup engages a multitude of individuals, each providing a small portion of the overall secret. These individual contributions are carefully combined in such a way that no one can deduce what any other person added. Consequently, even if someone were aware of their own contribution, it would not bring them any closer to discovering the ultimate, aggregated secret. In fact, the only way to uncover the combined secret would be for every participant to collaborate and reconstruct it.

To put it differently, as long as the group includes at least one honest participant, the final secret remains concealed. This concept is termed a "1-of-N trust model." As the group size increases, the likelihood of having at least one honest participant also rises. Therefore, the primary objective of a trusted setup is to maximize the number of trustworthy, independent contributors. (Joseph Harris, [5])

To better understand how the process actually work, think of it as a game of pass-the-parcel played backwards. In this scenario, the "parcel" represents the collection of elliptic curve points. The term "played backwards" conveys that each participant makes an additive contribution rather than a subtractive one. Notably, the element being contributed

is the participant's confidential random number. Central to this process is a "sequencer" or "coordinator" who maintains order and ensures smooth operations. In the context of the KZG Ceremony, this coordinator is a server administered by the Ethereum Foundation. Its responsibilities encompass keeping track of participant requests, arranging their sequence, and validating their contributions. If the ceremony is compared to a game of pass-the-parcel, the coordinator would be similar to the overseer, resembling a parental figure ensuring the game's proper execution.

As asked in this interview on a podcast episode ZeroKnowledge Podcast: Episode 262, Ethereum's KZG Ceremony with Trent and Carl from the Ethereum Foundation:

- "You just mentioned a sequencer. In other trusted setups, we usually had something called a coordinator. Are we talking about the same thing?"
- "Yes"
- "So, why the new name?"
- "I think it's more just to like, not overemphasize the power that this entity has. They're not some privileged entity in the space, and in essence, all they're really doing is deciding who goes next. So they decide on the sequence of things. Coordinating sounds a little bit more like there's someone pulling the strings in the background. When this entity server really has much less power than that. I guess sequencer is more of a passive framing of just like, okay, we take stuff, we pass it to somebody else. We are not, like, I guess, I mean obviously there is some coordinating type activity happening, but maybe it's just a personal choice, not personal, but like, we just sort of settled on that."

In practice the sequencer is a server, with no much power, it just receive all the requests to contribute and decide the order of which users can contribute. If is not your turn it will put you in a lobby and when it is you will be able to provide your contribution in the ceremony.

When a participant enrolls in the Ceremony, they generate a confidential random number (Joseph Harris, [5]). Subsequently, when it is their turn as directed by the sequencer, they receive a list of over 4000 numbers. This list comprises the set of elliptic curve points, encompassing the amalgamation of all preceding participants' secret numbers. The participant then incorporates their own secret number into this list through a process of multiplication. Specifically, they multiply their secret number with each of the elliptic curve points individually. Furthermore, the exponent to which they raise their secret number during multiplication depends on the position of the respective point in the list. For instance, the exponent is 0 for the first point, 1 for the second, 2 for the third, and so forth. This entire operation takes place within a finite field, which operates like a clock, whereby numbers reach a maximum value and subsequently wrap around to a lower bound. This implies that the multiplication may yield a smaller number. Nonetheless, it remains a fundamental process of multiplication. Upon completing this operation, the participant discards their secret number and forwards the updated list of numbers to the sequencer. The sequencer conducts a swift validation and subsequently dispatches the list to the next participant in line, initiating a recursive repetition of this process. The

multi-participant setup and its functionality were elucidated in the third chapter.

The final secret, produced at the very end of the Ceremony, is just a product of every individual participant's secrets. And, each elliptic curve point is related through those secrets, despite appearing as a random selection of numbers. As long as each individual keeps their secret to themselves, it would be extremely difficult to determine what numbers were multiplied together to produce the final set of elliptic curve points. In fact, because the finite field keeps everything looping around, it's impossible to work it out.

Participants in Ethereum's KZG Summoning Ceremony will be doing this for two sets of elliptic curve points each time. That's because the KZG Scheme requires two sets of related but slightly different points to work.

The plan is to run in parallel four ceremonies (with different secrets) with sizes $(n_1=4096, n_2=16), (n_1=8192, n_2=16), (n_1=16384, n_2=16), (n_1=32768, n_2=16)$ [1]. Theoretically, only the first is needed, but running more with larger sizes improves future-proofness by allowing us to increase blob size. We can not just have a larger setup, because we want to be able to have a hard limit on the degree of polynomials that can be validly committed to, and this limit is equal to the blob size.

In the future, it may be desirable to make certain changes to blobs — changes that would necessitate a longer set of elliptic curve points to produce secure commitments. If that happened, a new Ceremony would be required to produce those points, creating unnecessary delay and hassle. Ethereum developers solved this issue by running multiple setups simultaneously: in this way, the process does not need to be entirely repeated whenever a change is needed.

Another way to see the KZG ceremony is the following, as described by Hagen Hübel [6]: "Imagine you are playing a game of hide and seek with your friends. Now, in this game, there's a magic box that can tell if you're telling the truth or not. But to make this magic box work, all your friends need to help create a unique key."

This is kind of like the KZG Ceremony in Ethereum.

- The Magic Box (KZG Polynomial Commitments): In Ethereum, there is something called KZG Polynomial Commitments. This is like our magic box. It's a special kind of math that can help make sure people are telling the truth when they're playing the Ethereum game.
- The Special Key (Trusted Setup Ceremony): To make the magic box work, we need a special key. This key is created in a ceremony called the Trusted Setup Ceremony. In this ceremony, a bunch of people (like your friends in the game) each do a little bit of work to create the key. This is like the KZG Ceremony.
- Why We Need the Ceremony: The reason we need this ceremony is to make sure no one can cheat. If one person made the key by themselves, they could make the magic box say whatever they want. But if everyone helps make the key, then no one person can control the magic box.
- What Happens After the Ceremony: Once the key is made, it's used in the Ethereum game to make sure everyone is playing fair.

7 Security

This ceremony has the one of n trust assumption, so only a single person has to destroy the entropy or the randomness that are putting into the ceremony, only a single person has to be destroying their randomness or behaving honestly (in the general sense) to it to be secure. So it can be run publicly, you have the assurance it will be secure (just one honest is needed).

Another think is that, once the ceremony is ended, the output is out, there will be a verification process: the compunity will be encouraged to run a script and verify that that was the actual output of the ceremony. So if you were part of the ceremony, you will have a verifiable proof that your Ethereum address was included as part of the ceremony.

The unique think of this ceremony compare to the others is that you have multiple implementation that you can contribute through, and this helps the entire event to be more secure (kind of multi-client ethos of the Ethereum execution layer or consensus layer, the more different contributors the less probability of failure) here having multiple implementation means less probability to have bug in all of them.

8 How to contribute

To contribute is very easy (Joseph Harris, [5]), it is needed to have:

1. A computer to contribute from. Unfortunately, mobile devices will not work here.
2. A valid browser.
3. Either:
 - An Ethereum Account that had made at least 8 transactions before January 13th 2023
 - A GitHub account

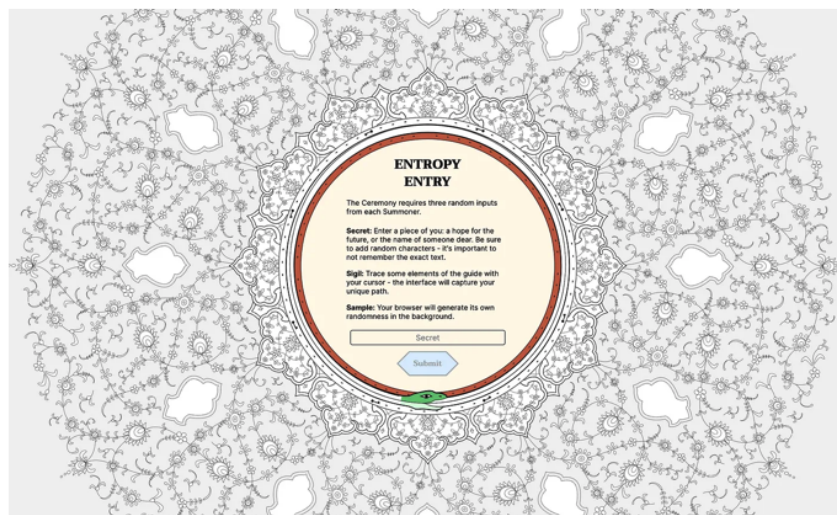
These restrictions are to try and prevent massive Sybil attacks. When the Ceremony first opened, the lobby was frequently flooded with airdrop farmers each trying to contribute with multiple addresses. While that's not necessarily a problem, it was limiting the number of unique users that could take part. As a result, the rules were adjusted so that most participants are relatively unique humans and not tens of thousands of identical robots.

Go to link and click begin.

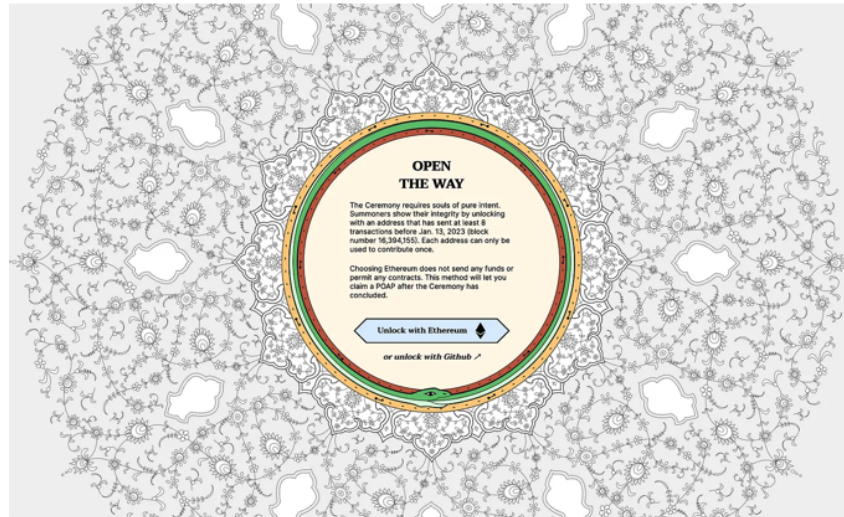


A new tab will open. This is where you generate your secret random number. The randomness comes from three sources:

1. The Secret: which is whatever you type into the text box.
2. The Sigil: which is taken from your cursor's movements. You have to move the cursor until the entire green ring (representing a snake) is all colored, until it eats itself.
3. The Sample: which your browser will generate in the background.

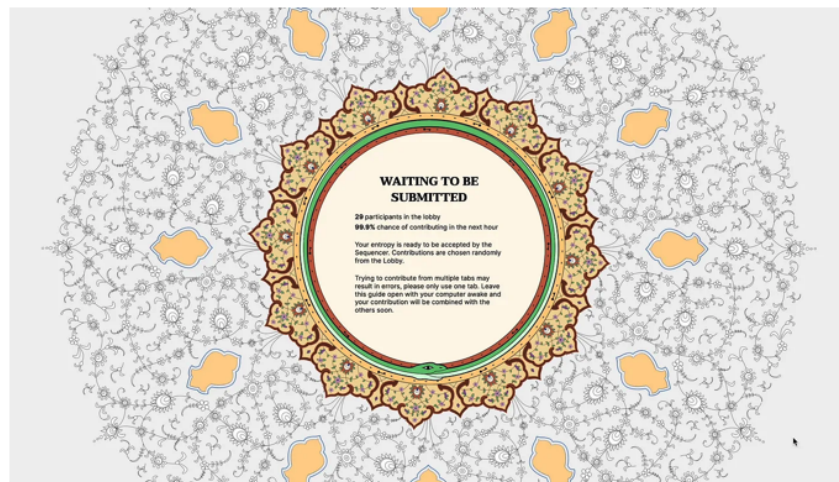


Then you'll be asked to sign in, as mentioned earlier you can either use a Github or your Ethereum address.



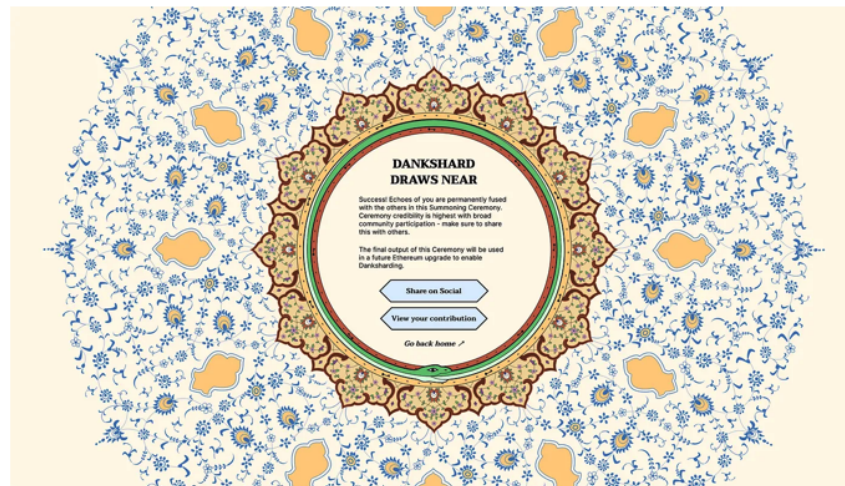
Now, you just have to wait until receive a confirmation message. The screen will tell you how many others are in this waiting room with you. The coordinator will make random selections from this room when choosing who to go next, rather than selecting on any kind of first-come-first-served basis. That makes it impossible to know quite how long you'll have to wait.

It is possible for you to enter and submit immediately or wait a few minutes, hours, days...




Eventually, the screen will change to let you know you have received the Structured Reference String — here called the Powers of Tau — and that your secrets are being mixed in.

The screen will change again once the calculations have completed and the Coordinator has received your updated SRS. When you see this, congratulations: you have just contributed to Ethereum's next upgrade!



9 How to verify that your contribution is part of the result

When the ceremony is over, there is a ceremony transcript that contains all the necessary to verify who participated. If you used your Ethereum address to sign-in, then there will be cryptographic proof that you participated via a signed message from your account. So to verify if your contribution is in the ceremony just enter your ETH address or GitHub username used during this event.



+ Sequencer Online





141,416 total contributions
0 participants in lobby

English

CONTRIBUTIONS TRANSCRIPT

Lobby size: 0
Contributions: 141416
Sequencer address: 0xfAA3A871325D44E33C994556f772AC7193710
Transcript hash: 0x8ed1c73857e77ae98ea23e36cdc828ccbf32b423fddc7480de658f9d116c848

[Download full transcript ↗](#)
[Verify Transcript](#)

#	Participant ID	Signatures
0		
1	git 106971636 limitranger	
2	git 1500888 skylenet	
3	0x79063f7730bbc39bd8c09b3ad11ce246a33caef8	


In the link on the left (Download full transcript) you can see the output of all the 4 ceremonies. if you search on the bar for your Ethereum wallet address (or HitHub) account you will find the following page


CONTRIBUTION DETAILS


3 ← →


Participant ID:
0x79063f7730bbc39bd8c09b3ad11ce246a33caef8

Powers of Tau Pubkeys:

 **(2¹²):** 0x82e176ac98c927a6eaf42778a7b4c382d78c102ccf18c0219ca584fd0e8c12c044c94566daea1a1d3831b05450ec2a9214eace8fee2e3244974bdc3de65a5bbd33c31006ca5729c0b50be346dd562c01d481c7083224fb56146c7b14822692f

 **(2¹³):** 0xb88cd806386a6a89a665951b298e0cc7f6ab2fb8b41406aa3088e0480268ec7541856933a392e2b04d9e468ea56f1fdf157a3e2118b113e955e72cce8037efe392a27f968d8afcc0b3e5cbce69d26d4f051e3ffde105c9d727d8aa0d306d04b6

 **(2¹⁴):** 0xb5be66fd64088e7a3d2969f5726a6d164f588a27be95a08433931f3e342fda1cae57ec3beef00e1184efe28303fc33ab0e793a1f74e882d44051d980434c917916f17617a40afddaa4f14588b2644f2c532bb072d99a0012451b5a0f60030828

 **(2¹⁵):** 0xb78a522dbcab7c8453fbaeb2a7f50fd929d27a035e8d71e2d3d3fe1a91f6a34ceaced6cfcae9b2df4fb6f79c133115a0d4dabe05f15be8052953609a7e0f0186422e197882f9ed248057c2b08999f28bf6fb2fef7f68654acbfef80224e25699

BLS Signatures:

1. 0x8a306cf1cbbd5e9335dd6f0aa2c67bde9e9fd051f0ed305f3365d5f0d4f436ba1560bc49002666a83a0ca4adc6e1a0c6
2. 0x99a4a6081bdadcfc3d21f4e59ba160107a5c1c5ad72147cba6328469ece2eec1d12ab51041c0bb67c202ccc8b925da3a
3. 0x950afdadff54a989b7f2bb22b64161484a430a8a4e683c5ac97ff93762418025f3b5f454bb8a633a57eb8ef4b3f017a6
4. 0xab484eb57833671bc889cd7a9d4b857f279f5e20f3288f23f2fe39f0fb8894658d9a6f8602d9ade052a30ecd9d261b49

ECDSA Signature (optional):
0x48a2ebc295e543e6c90f5291dcc773ed23df944059d0cc26912db4bf031ade78403b30920773277b68cd04dd1b31b1576343e500893ba240defc4e0e101522921c

That shows your contribution details. The #3 is because is the third contribution in the ceremony, made by a Ethereum address: 0x79063... and the four powers of tau PUBLIC key(because as said before 4 ceremonies).

The output SRS consists of points on BLS12-381, the same curve used for BLS signatures in Ethereum PoS.

As seen before we have two points from two different elliptic curve generator (G_1 and G_2) then is performed a pairing (the function that maps these two points into a new one in G_T). The BLS is a cryptographic procedure that generate a signature, keeping as input a private key (the secret) and a message and return a signature (that is a G_1 point). To verify it it is needed as input the public key, the message and the BLS signature; the function returns True if the signature is valid.

The specifics can be found here BLS. There is also an optional additional signature: ECDSA to sign the user's PoT PubKeys (only if the user is signed up with an Ethereum address). Also BLS signature is optional.

References

- [1] Proto-danksharding notes. https://notes.ethereum.org/@vbuterin/proto_danksharding_faq#Proto-Danksharding-FAQ, 2022.
- [2] Vitalik Buterin. Exploring elliptic curve pairings. https://vitalik.ca/general/2017/01/14/exploring_ecp.html.
- [3] Vitalik Buterin. How do trusted setups work? <https://vitalik.ca/general/2022/03/14/trustedsetup.html>.
- [4] Dankrad Feist. Zkp polynomial commitment. <https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html>, 2020.
- [5] Joseph Harris. Ethereum’s kzg summoning ceremony explained. <https://medium.com/topic-crypto/ethereums-kzg-summoning-ceremony-explained-ecb258830826>, 2022.
- [6] Hagen Hübel. Explain kzg ceremony on ethereum like i am 5 — lia5. <https://0xhagen.medium.com/explain-kzg-ceremony-on-ethereum-like-i-am-5-lia5-4439a3950446>, 27.05.
- [7] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [8] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. Cryptology ePrint Archive, Paper 2022/1592, 2022. <https://eprint.iacr.org/2022/1592>.
- [9] Haym Salomon. Elliptic curve pairings. <https://inevitableeth.com/home/concepts/elliptic-curve-pairings>.
- [10] Haym Salomon. Kzg commitment scheme. <https://inevitableeth.com/home/concepts/kzg-commitment>.