

KZG ceremony

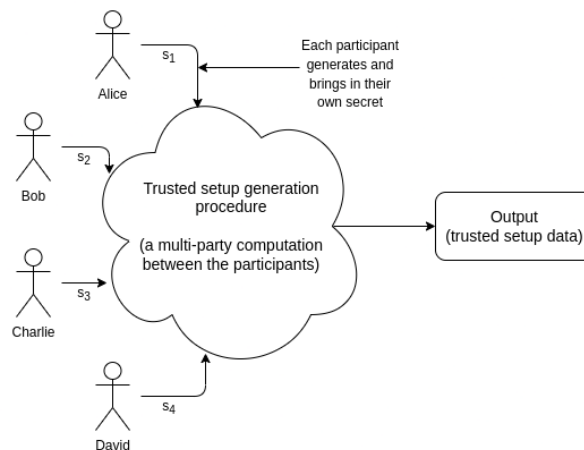
Sofia Martellozzo (10623060)

1 Setup ceremony

A trusted setup ceremony [3] 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.



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 [7] 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).

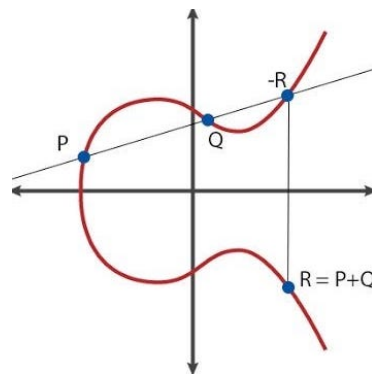
- **Data Availability:** the final string must be available for all to download, as well as the history of prior versions and participants for auditability.
- **Censorship Resistance:** any willing participant should be able to contribute.

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 [2] (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 (ie. for calculating the coordinates of $R = P + Q$) and you can also multiply a point by an integer (ie. $P \cdot n = P + P + \dots + P$, though there's a much faster way to compute it if n is big).



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:

$$e(P, Q + R) = e(P, Q) \cdot e(P, R)$$

$$e(P + S, Q) = e(P, Q) \cdot e(S, Q)$$

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 :

$$a + b : (a + b) \% p$$

$$a - b : (a - b) \% p$$

$$a \cdot b : (a \cdot b) \% p$$

$$a/b : (a/b) \% p$$

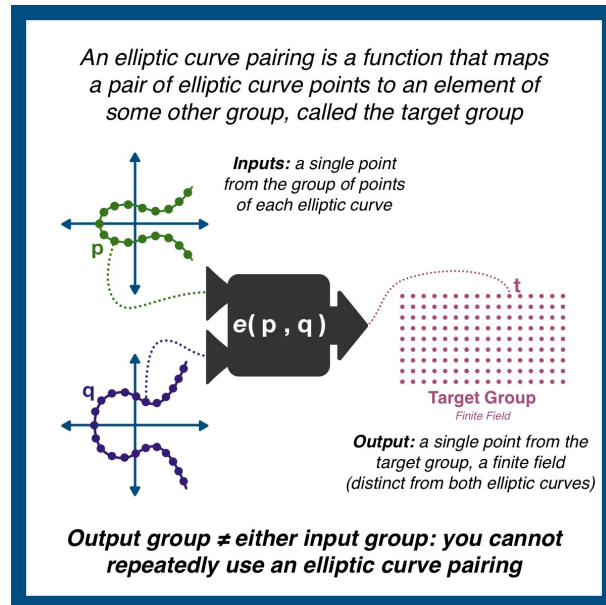
About the Modular Arithmetic, imagine a clock being mod 12, adding it to elliptic curve it is very difficult to solve discrete logarithm problem (that is, if you add two points over and over again using modular arithmetic, it takes a lot of work to find out how many times you did it).

If a clock is mod 12, then consider the following question: "Alice left at 4 and arrived at 6. How many hours did she spend traveling?"

2 hours? 14 hours? 26 hours? The only way to figure it out is to start guessing.

This is the discrete logarithm problem.

The following image from [8] 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).

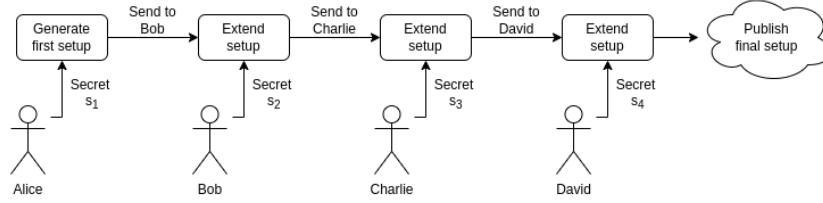
If $k = 1$, then the "discrete logarithm" problem for elliptic curves can be reduced into a similar math problem over $(F_p)^{12}$, where the problem becomes much easier (this is called the MOV attack). Using curves with an embedding degree of 12 or higher ensures that this reduction is either unavailable, 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" (ie. computationally infeasible).

3 Multi-participant setup

It's 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 .

But a single-participant trusted setup is insecure: you have to trust one specific person! The solution to this is multi-participant trusted setups, where by "multi" we mean a lot

of participants: over 100 is normal, 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, 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 ...

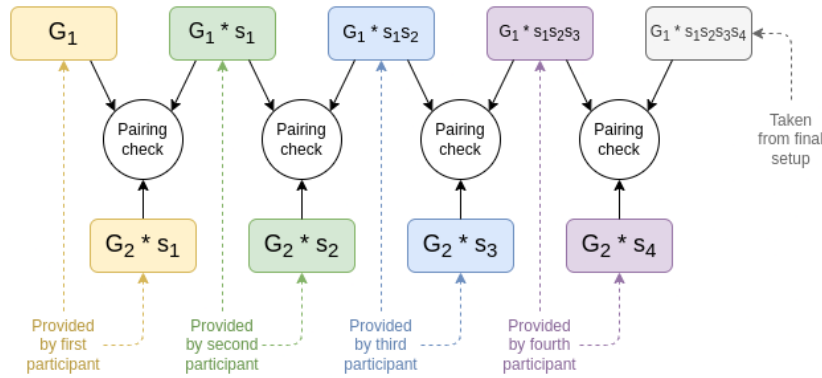


Figure 1: The pairing check at each step proves that the setup at each step actually came from a combination of the setup at the previous step and a new secret known by the participant at that step.

Note that 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: [Code](#)

4 Background

Lately, many people are complaining about Ethereum. That's because, during times of high demand, Ethereum transactions are both slow and expensive — especially when compared to competitors like Solana.

It is not due to the fact that Ethereum is older, and and outdated technology.

In reality, Ethereum's poor performance and high fees are entirely intentional.

You might think that's crazy. After all, why would anyone choose to provide a terrible

user experience? In reality, it's because Ethereum is simply prioritising something else: decentralisation. Ethereum is designed to minimise the barrier to entry for anyone wanting an active role in the network. In practice, that means the Ethereum software must be able to run on all sorts of computers — including fairly cheap, low-powered ones.

Therefore, Ethereum blocks cannot contain a large quantity of transactions or data because that would overwhelm the low-end computers in the network. As a result, it doesn't take a significant increase in transaction volume to saturate Ethereum's blocks.

When that happens, users are forced to outbid one another to ensure their transaction is prioritised and included in a block. And that very quickly spirals into the sky-high transaction fees we all love to complain about.

Of course, Ethereumers recognise the problem here. They certainly don't want to create a system that anyone can help run but only the rich can afford to use. And so they've come up with a plan. One that will allow them to retain the accessible, decentralised core of the system while also increasing capacity and reducing transaction fees and that plan is called 'Layered Scaling'.

4.1 Layered Scaling

Layered scaling is ultimately about aggregating as much activity as possible into a small number of Ethereum transactions. The idea is to shift most day-to-day transactions away from Ethereum and onto new 'Layer Two' (L2) platforms.

These platforms are built with performance in mind, ensuring transaction fees remain low even when demand is high. Then, every so often, all of the recent L2 transactions will be batched, compressed, and represented in a single transaction on Ethereum.

In this way, all the activity still takes place on the Ethereum blockchain, just in a less direct manner. All the same, it means all of the L2 transactions are still verified and secured by Ethereum. That leads to the ultimate win-win scenario for end users. They still enjoy all the benefits of Ethereum and its robust, decentralised design but they don't suffer with its low throughput, slow transactions, and exorbitant fees.

This approach is the best way to build and scale a blockchain, but the problem here is that Ethereum wasn't built with this methodology in mind. So, some modifications are required to get the most out of a more modular design.

One of the biggest issues is storing data.

Remember how Layer Twos aggregate all their recent transactions and represent them in a single Ethereum transaction. Well, that Ethereum transaction must contain a fair amount of data — essentially a summary of all the L2's recent activity.

And, here's the problem, Ethereum only has one kind of data storage: permanent. When an L2 — or anyone else, for that matter — posts a bunch of data to Ethereum, everyone in the Ethereum network must hold onto that data forever. That's not easy to do, and it places a significant burden on the computers in the network. So, this isn't good for Ethereum. Especially because its priority has always been minimising the burden on those computers to keep the network decentralised. The situation is also not good for the L2s. Ethereum doesn't want people filling blocks with data and overwhelming the low-end computers in the network, so it charges a high price for data storage. As a result, L2s must pay a lot of money to make their settlement transaction on Ethereum, and the L2's users must pay higher fees to account for that. Again, this is particularly silly when the whole point of the L2s is to provide a low-fee environment. So, we need a better solution for storing data. And, this is where Proto-Danksharding comes in.

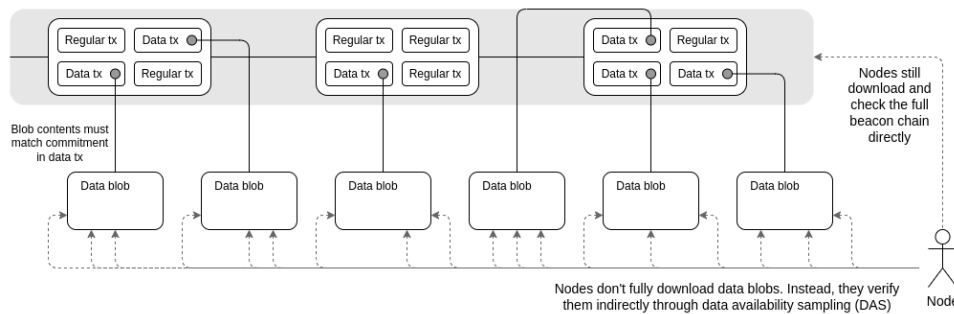
5 Proto-Danksharding

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 word ‘blob’ is also quite appropriate here.

I find it useful to think of these data chunks as literal sticky blobs that can cling to an Ethereum block like a barnacle on a boat. And, just as barnacles are scraped off of boats before they cause problems, these blobs will be scraped off of blocks and discarded before they overwhelm the network.

Blobs are extremely large (125 kB), and can be much cheaper than similar amounts of calldata. However, blob data is not accessible to EVM execution; the EVM can only view a commitment to the blob.

Because validators and clients still have to download full blob contents, data bandwidth in proto-danksharding is targeted to 1 MB per slot instead of the full 16 MB. However, there are nevertheless large scalability gains because this data is not competing with the gas usage of existing Ethereum transactions.

Yes, this is the big improvement that Proto-Danksharding will bring. Any data contained in a blob will only be stored on Ethereum for a few weeks. After that, anyone in the Ethereum network is free to delete and forget about that data forever, reducing the burden on all those devices. It’s safe to do this because that information doesn’t need to be stored for eternity. At least, not by all parties. The important thing is that the L2 data is distributed and made available to anyone who might need it. Sticking it onto the outside of an Ethereum block and distributing it across an un-censorable global network

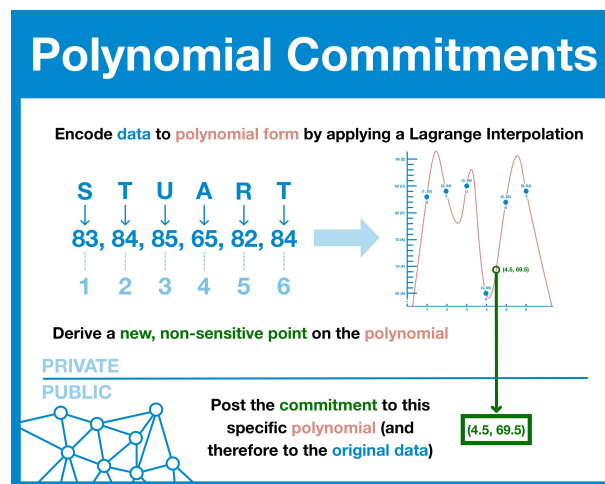
is a pretty good way to ensure all interested parties have access. After a few weeks, those parties should have had plenty of time to make permanent copies or do whatever they need to do with that data.

Blob-carrying transactions are a big win for Ethereum. The short-term storage minimises the strain placed on the Ethereum network and doesn't threaten its decentralisation in the way permanent storage might. It's also good for the L2s. Because short-term storage is less burdensome than permanent storage, a greater quantity can be offered at a much lower price. Therefore, Layer Twos won't need to pay anywhere near the costs they currently pay to settle on Ethereum, and they can pass those savings on to end users.

6 KZG Commitments

6.1 Polynomial Commitments

Before getting into KZG, let's first cover the basics: polynomial commitments [6]:

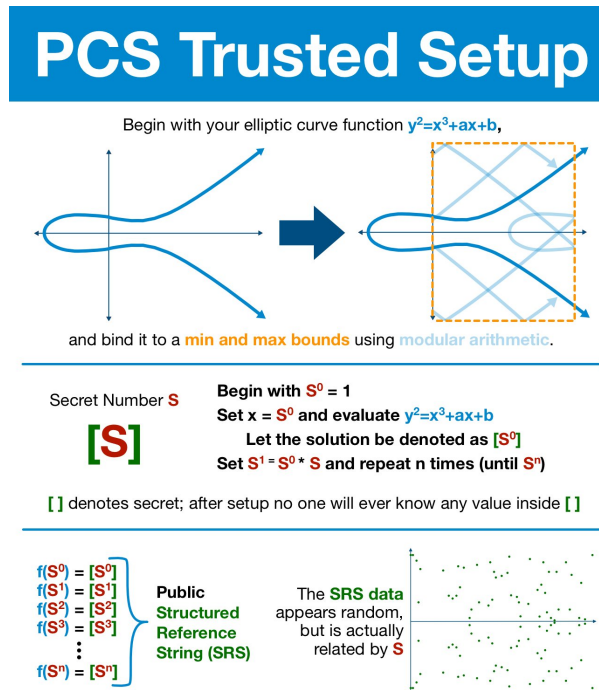


First, take the dataset and break it into discrete chunks.

Then, plot them on a graph and draw a line through those point. Derive the formula for that line.

This formula is a polynomial, a special type of equation that has some very powerful mathematical properties. This polynomial holds the same information as the raw data, but we can also use it to generate new points. In fact, these 'non-real' points will serve as a commitment: it confirms a party has a complete knowledge of the dataset (else they would not be able to generate the polynomial) but it does not leak any of the underlying data.

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

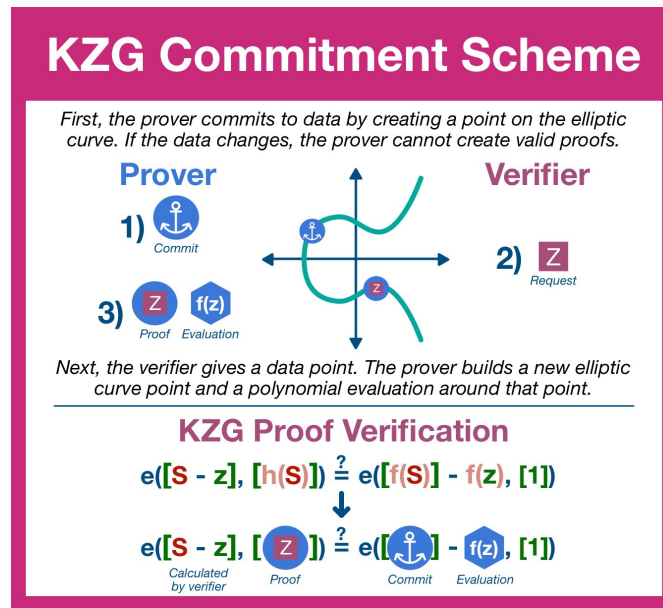
It is very hard to solve the elliptic curve discrete logarithm problem. This is the property to build the system out of, starting by hiding a secret number inside an elliptic curve.

As an example imagine to pick a number S completely at random, that will be discarded at the end of the process (forgotten), then bind the elliptic curve to a minimum and a maximum bounds bounds using modular arithmetic. Then use this (modular) elliptic curve and secret (number) S to generate a series of values, starting with S^0 and ending with S^n , each time feeding it into our elliptic curve formula and generate a new value, representing a point on the curve.

Without S the list of points look like noise. Yes, we can see the clear horizontal symmetry, but otherwise it seems near random. But that's the point: it's NOT random. This is all that remains of S . A secret number lost in a scattered mess of points. An arbitrary splattering of points, indistinguishable from random data.

The result of the trusted setup is a Structured Reference String (SRS). The SRS conveys enough information that the random number is accessible for our polynomial (and therefore our polynomial commitment scheme) while still keeping the value entirely secret. Note: one SRS can be used by an arbitrary number of people; only one trusted setup (or one trusted setup per application) are needed for ALL KZG commitments.

The following image from [9] provide a scheme of the KZG Commitment:



It is an interactive proof system: a method by which one party (prover) can prove to another (verifier) that a statement is true.

The prover replies back with two items: the evaluation of quotient polynomial using the elliptic curve and the result of the polynomial evaluated at that position. The former is an elliptic curve point, the later the data polynomial evaluation.

Remember, the polynomial is an expression that describes how our data looks when plotted on a graph. The evaluation of that function is simply just the data at that position! To finish our proof cycle, the verifier (who does not have access to the full dataset and therefore cannot generate the data or quotient polynomial) is going to ensure the provers response reconciles with the commitment. To combine the pieces sent by the prover the verifier uses the elliptic curve pairings. The verifier constructs two expressions and checks them for equivalence. If they are equivalent, the verifier can be certain that the prover did the (polynomial) division and therefore still has the original data.

We've 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's worth noting, because it's 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 [4] to the blob. Well, more accurately it's a versioned hash of a KZG Commitment to the blob. KZG Commitments, and commitment schemes in general, can be used to prove ownership or knowledge of some information without revealing the information itself. Later, after the information is revealed, the commitment can be used to authenticate that information. That authentication would fail if there was even a single, tiny difference between the information committed to and the information revealed.

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.

There are only two notable downsides to KZG Commitments. First, they're not quantum-resistant, so there could be security issues if someone develops a powerful quantum computer. Frankly, that's not worth worrying about today, and when it does happen there will be so many other things at risk it probably won't matter much then either.

The second issue is more immediately concerning. KZG Commitments require something called a trusted setup. And, that's where the Summoning Ceremony comes in.

6.3 KZG Trusted setup

As a recap, we have seen that to generate a KZG Commitment, the data being committed to is first converted into a polynomial. Something like this:

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

That polynomial is then evaluated at certain points — which is to say, some numbers are plugged in to see what comes out. And, that stuff that comes out is the KZG Commitment. When someone wants to verify the data later on, they can effectively repeat the process. By evaluating the polynomial at the same points, they can see if the output matches the original commitment. If it does, everything is fine and the data was the same. If it doesn't, then something has been changed.

The numbers that are plugged into the equation should remain secret. If someone did know the numbers, they could cheat the system by crafting a new polynomial that produced the same output. In other words, they could commit to one set of data, then change it entirely but still provide a valid proof.

To keep the numbers secret the elliptic curve is the solution. There are a special family of curves defined by equations that look like:

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

and have some unique properties, such that points on an elliptic curve can be added together to produce a third point that is also on the curve. When this is done, it's difficult to predict where the third point will be. And, when points are repeatedly added together, it's incredibly difficult to determine how many times the points were added. Mathematicians can use this to hide numbers inside an elliptic curve.

Essentially, they take the 'secret' number and add elliptic curve points to themselves many times. The output of this calculation is a list of elliptic curve points that appear totally random when plotted on a graph. This is called a Structured Reference String (SRS), and an attacker could study it to their heart's content without ever learning anything about the secrets held inside. However, quite amazingly, the SRS still contains

enough information about the secret values to be useful in constructing KZG Commitments.

Now the question is, where does the SRS come from? Who produces it, and how can they reassure others that they don't know the secrets held within?

The answer here is to use something called a trusted setup. After all, this industry's entire purpose is to remove singular trusted entities and replace them with large, disparate networks of individual operators, thereby increasing overall trust in the system.

Yes, the idea of a trusted setup is to involve as many people as possible in crafting the Structured Reference String. So, instead of one person contributing the entire secret, and then promising they don't know what that secret is, a trusted setup involves a large number of people each contributing a small portion of the overall secret. Each individual contribution is cleverly mixed in so that no one can determine what anyone else added. That way, even if someone knew exactly what they had entered, they wouldn't be any closer to discovering the ultimate, combined secret. In fact, the only way to discover the combined secret would be for every single participant to work together to recreate it.

Put another way, provided the group contains at least one honest actor, the final secret can never be known. This is called a 1-of-N trust model. As the group gets larger, the probability that it contains at least one honest actor also increases. And so, the goal of a trusted setup is to maximise the number of honest, independent participants.

How does it actually work?

it's kind of like a game of pass-the-parcel played backwards. In this case, the parcel is the set of elliptic curve points. By "played backwards", I mean each player adds something to the mix instead of removing something. And, as you can probably guess, the thing being added is the participant's secret random number. In the middle of all of this is a 'sequencer' or 'coordinator', who keeps everything organised and running smoothly.

For the KZG Ceremony, the coordinator is a server run by the Ethereum Foundation. It keeps track of who wants to participate, orders participants, and verifies their contributions. If the ceremony is like a game of pass-the-parcel, then perhaps the coordinator is the parent overseeing the game and ensuring it's played properly.

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 joins the Ceremony, they generate a secret random number. Later, when the sequencer says it's their turn, they will receive a long list of more than 4000 numbers. This is the set of elliptic curve points containing the combination of all previous participants' secrets. Now, the participant can mix their secret in as well, and they do this through multiplication. They take their secret number, and essentially just multiply each of the elliptic curve points by that number. More precisely, they multiply by their number to a power according to where the point is in the list. So, it's to the power of 0 at the first point, the power of 1 at the second, the power of 2 at the third, and so on. Plus, importantly, this is all done over something called a finite field. These work a bit like a clock in that numbers reach some maximum value and then loop back around to a lower bound. That means two numbers could be multiplied together to produce a smaller number. Still, essentially just multiplication. Once they've done that, they discard their secret and send the updated list of numbers back to the sequencer. The sequencer performs a quick validation and then sends the list to the next participant to repeat the process. The multy participants setup, so how it works was shown in the third chapter.

he 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 $(n_1 = 4096, n_2 = 16), (n_1 = 8192, n_2 = 16), (n_1 = 16384, n_2 = 16), (n_1 = 32768, n_2 = 16)$. Theoretically, only the first is needed, but running more with larger sizes improves future-proofness by allowing us to increase blob size. We can't 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. So, Ethereum's developers had an idea: why not run multiple setups at the same time? That way, everything is ready and waiting for if/when those changes are made.

Another way to see the KZG ceremony is the following:

Imagine you're 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's 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. And just like in hide and seek, the game is much more fun when everyone plays by the rules!

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 [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

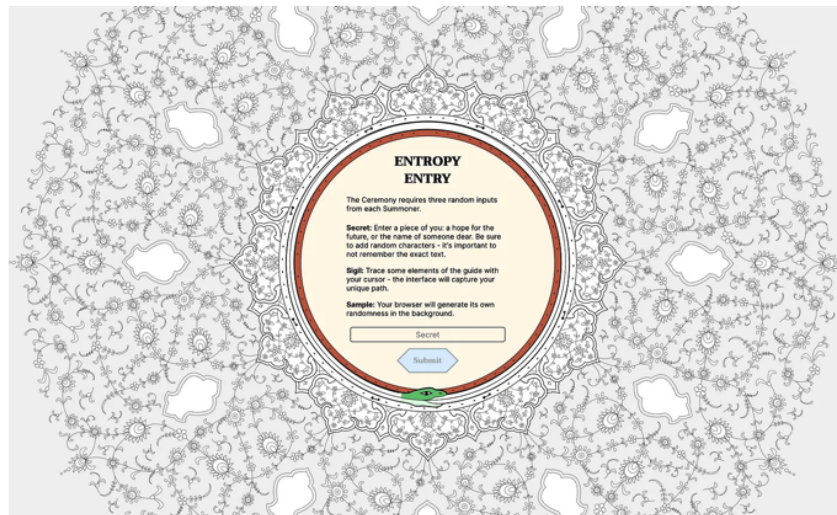
These restrictions are to try and prevent massive Sybil attacks. When the Ceremony first opened, the lobby was frequently flooded with airdrop farmers each trying to contribute with multiple addresses. While that's not necessarily a problem, it was limiting the number of unique users that could take 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.

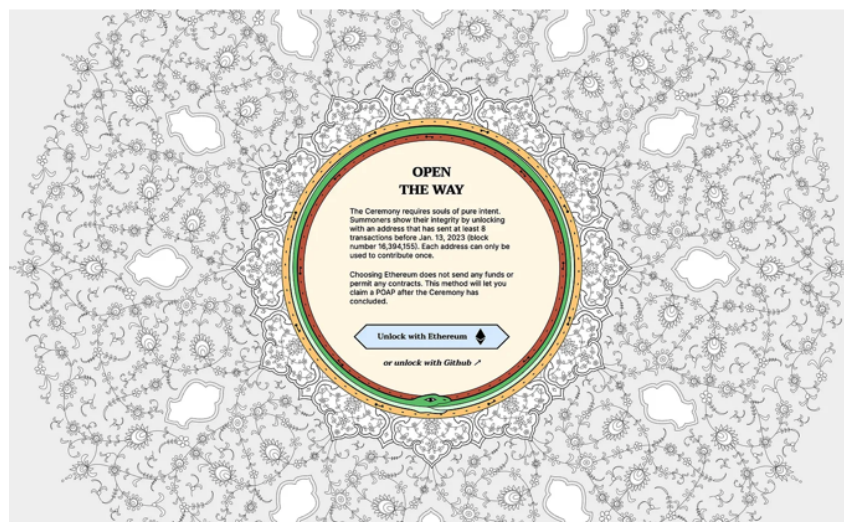


A new tab will open. This is where you generate your secret random number. The randomness comes from three sources:

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.

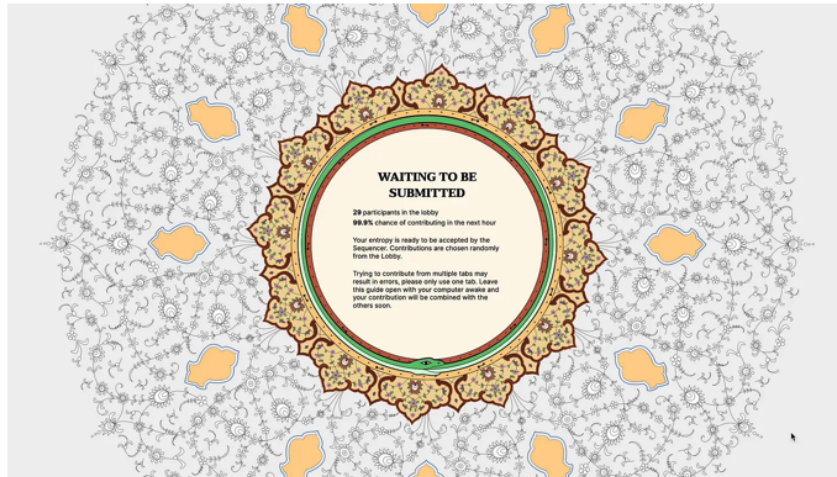


Then you'll be asked to sign in, as mentioned earlier you can either use a Github or your Ethereum address.



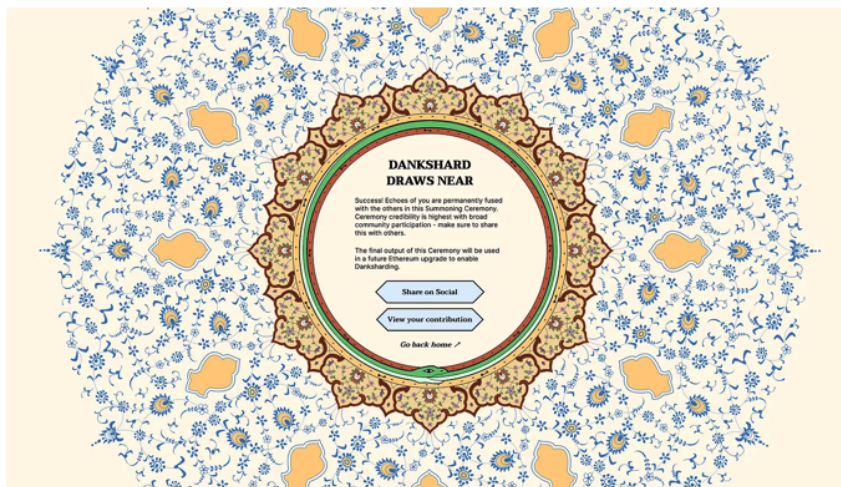
Now, you just have to wait until receive a confirmation message. The screen will tell you how many others are in this waiting room with you. The coordinator will make random selections from this room when choosing who to go next, rather than selecting on any kind of first-come-first-served basis. That 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: 0xfAA3A87713253D44E33C994556f772AC71937f0
 Transcript hash:
 0x8ed1c73857e77ae98ea23e36cdc628ccb32b423fddc7480de58f9d116c848

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

#	Participant ID	Signatures
0		
1	git 106971636 limitranger	
2	git 1500888 skynet	
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): 0x82e176ac98c927a6eaf42778a7b4c382d78c102ccf18c0219ca584fd0e8c12c044c94566daea1a1d3831b05450ec2a9214eace8fee2e3244974bdc3de65a5bbd33c31006ca5729c0b50be346dd562c01d481c7083224bf56146c7b14822692f



(2¹³): 0xb88cd806386a6a89a665951b298e0cc7f6ab2fb8b41406aa3088e0480268ec7541856933a392e2b04d9e468ea56f1fdf157a3e2118b113e955e72cce8037efe392a27f968d8afcc0b3e5cbce69d26d4f051e3ffde105c9d727d8aa0d306d04b6



(2¹⁴): 0xb5be66fd64088e7a3d2969f5726a6d164f588a27be95a08433931f3e342fda1cae57ec3beef00e1184efe28303fc33ab0e793a1f74e882d44051d980434c917916f17617a0afddaa4f14588b2644f2c532bb072d99a0012451b5a0f60030828



(2¹⁵): 0xb78a522dbcab7c8453fbaeb2a7f50fd929d27a035e8d71e2d3d3fe1a91f6a34ceaced6cfcae9b2df4fb6f79c133115a0d4dabe05f15be8052953609a7e0f0186422e197882f9ed248057c2b08999f28bf6b2fef7f68654acbf80224e25699

BLS Signatures:

- 0x8a306cf1cbbd5e9335dd6f0aa2c67bde9e9fd051f0ed305f3365d5f0d4f436ba1560bc49002666a83a0ca4adc6e1a0c6
- 0x99a4a6081bdadcf3d21f4e59ba160107a5c1c5ad72147cba6328469ece2eecd1d12ab51041c0bb67c202ccc8b925da3a
- 0x950afdadff54a989b7f2bb22b64161484a430a8a4e683c5ac97ff93762418025f3b5f454bb8a633a57eb8ef4b3f017a6
- 0xab484eb57833671bc889cd7a9d4b857f279f5e20f3288f23f2fe39f0fb8894658d9a6f8602d9ade052a30ecd9d261b49

ECDSA Signature (optional):
0x48a2ebc295e543e6c90f5291dcc773ed23df944059d0cc26912db4bf031ade78403b3092077327b68cd04dd1b31b1576343e500893ba240defc4e0e101522921c

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, 2022.
- [2] Vitalik Buterin. Exploring elliptic curve pairings.
- [3] Vitalik Buterin. How do trusted setups work?
- [4] Dankrad Feist. Zkp polynomial commitment, 2020.
- [5] Joseph Harris. Ethereum's kzg summoning ceremony explained, 2022.
- [6] 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.
- [7] 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>.
- [8] Haym Salomon. Elliptic curve pairings.
- [9] Haym Salomon. Kzg commitment scheme.