



Haym @SalomonCrypto

Oct 22 · 25 tweets · [SalomonCrypto/status/1583705993300492288](https://twitter.com/SalomonCrypto/status/1583705993300492288)

Tr

(1/24) KZG Polynomial Commitments: The Complete Guide

Our goal: 1) prove we are committed to a specific set of data and 2) allow others to verify specific points within that dataset.

Want to see some mathematical magic? This megathread is for you!

KZG Commitment Scheme

First, the prover commits to data by creating a point on the elliptic curve. If the data changes, the prover cannot create valid proofs.

Prover

1) Commit

3) Proof Evaluation

Verifier

2) Request

Next, the verifier gives a data point. The prover builds a new elliptic curve point and a polynomial evaluation around that point.

KZG Proof Verification

$$e([S - z], [h(S)]) \stackrel{?}{=} e([f(S)] - f(z), [1])$$

$$\downarrow$$

$$e([S - z], [\text{Z}]) \stackrel{?}{=} e([\text{Anchor}] - f(z), [1])$$

Calculated by verifier
Proof
Commit
Evaluation

(2/24) Elliptic curve cryptography and KZG polynomial commitments are HARD. This thread aims to give you a high level understanding of how KZG commitments work, while the linked threads go MUCH deeper.

Just remember: pink border = summary, blue border = details

(3/24) Before we get to KZG, let's first cover the basics: polynomial commitments.

First, take your dataset and break it into discrete chunks. Then, plot them on a graph and draw a line through those point. Now derive the formula for that line.

Haym
@SalomonCrypto · Follow

(1/17) Cryptography Basics: Polynomial Commitments

Creating unique digital fingerprints is core to cryptography. We use tools like hashing to provide cryptographic-certainty without revealing any info.

But hashing is so destructive, is there a better way to commit to data?

Polynomial Commitments

Encode data to polynomial form by applying a Lagrange Interpolation

S T U A R T
↓ ↓ ↓ ↓ ↓ ↓
83, 84, 85, 65, 82, 84
1 2 3 4 5 6

Derive a new, non-sensitive point on the polynomial

PRIVATE
PUBLIC

Post the commitment to this specific polynomial (and therefore to the original data)

(4.5, 69.5)

1:50 AM · Oct 16, 2022

Read the full conversation on Twitter

371 Reply Copy link

Read 15 replies

(4/24) 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... maybe a point in between two "real" points.

(5/24) In fact, one of these "non-real" points will serve as our commitment: it confirms a party has 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.

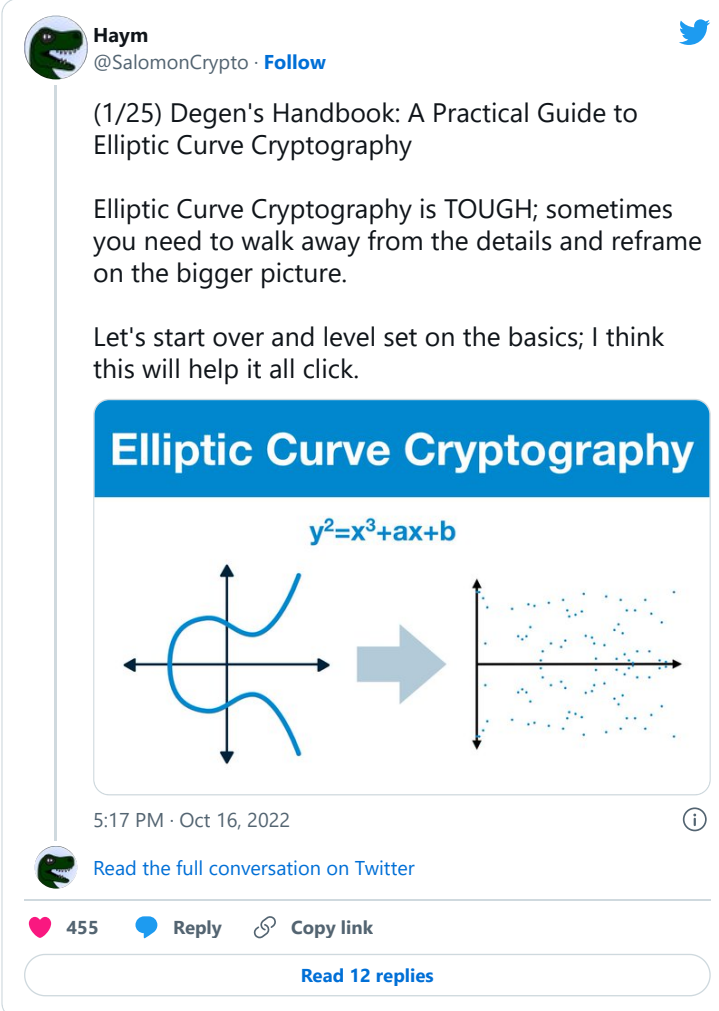
Now, which "non-real" point to use...

(6/24) Ideally we'd like to use a completely random point, one that no malicious party could know beforehand. Then the prover couldn't precompute some points that look valid, but don't actually check out.

But where are we going to find a random number?

(7/24) Well, dear reader, at this point you may be asking yourself "where are the elliptic curves?!?" This is your time!

We are going to use an elliptic curve!



The image is a screenshot of a tweet from a user named Haym (@SalomonCrypto). The tweet is titled "(1/25) Degen's Handbook: A Practical Guide to Elliptic Curve Cryptography". The text of the tweet reads: "Elliptic Curve Cryptography is TOUGH; sometimes you need to walk away from the details and reframe on the bigger picture. Let's start over and level set on the basics; I think this will help it all click." Below the text is a graphic with a blue header that says "Elliptic Curve Cryptography". Underneath the header is the equation $y^2 = x^3 + ax + b$. To the left of the equation is a diagram of a blue elliptic curve on a coordinate system. A large grey arrow points from the curve to the right, where there is a scatter plot of blue dots on a coordinate system, representing the discrete points of the curve. The tweet is dated "5:17 PM · Oct 16, 2022" and has 455 likes. There are also options to "Reply" and "Copy link", and a button that says "Read 12 replies".

(8/24) (Modular) elliptic curves have many incredibly useful properties that we will be relying on. In particular, we care that:

- elliptic curve points can be added together
- when added repeatedly, it is INCREDIBLY difficult to figure how many times a point has been added

(9/24) Using these properties, we can hide a secret (random) number inside an elliptic curve. After we go through the trusted process and the secret number is permanently destroyed, the only knowledge of that value will be permanently hidden in the folds of the elliptic curve.

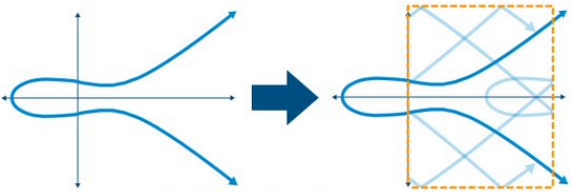
Haym
@SalomonCrypto · Follow

(1/19) Elliptic Curve Cryptography: Trusted Setups

Successful cryptography is cryptography that transforms legible data into digital-static. Before we go big, let's wrap our mind around something simple.

An instruction manual for creating a permanently secret number.

Begin with your elliptic curve function $y^2=x^3+ax+b$,



and bind it to a **min and max bounds** using **modular arithmetic**.

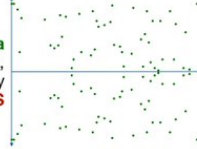
Secret Number **S** **Begin with $S^0 = 1$**
[S] **Set $x = S^0$ and evaluate $y^2=x^3+ax+b$**
 Let the solution be denoted as $[S^0]$
 Set $S^1 = S^0 * S$ and repeat n times (until S^n)

[] denotes secret; after setup no one will ever know any value inside []

$f(S^0) = [S^0]$
 $f(S^1) = [S^1]$
 $f(S^2) = [S^2]$
 $f(S^3) = [S^3]$
 \vdots
 $f(S^n) = [S^n]$

Public Structured Reference String (SRS)

The **SRS data** appears random, but is actually related by **S**



4:27 AM · Oct 17, 2022

[Read the full conversation on Twitter](#)

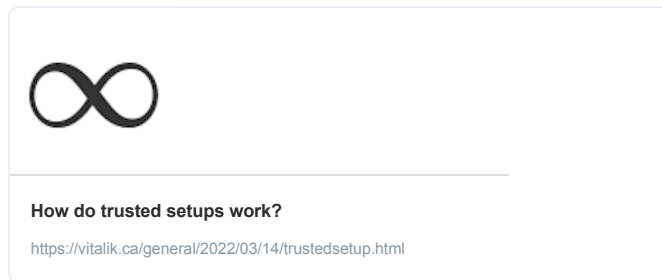
248 Reply Copy link

[Read 7 replies](#)

(10/24) 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.

(11/24) 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.

For those interested in real world implementations, Vitalik has you covered:



(12/24) Another note, terminology.


Interactive Proof System: a method by which one party (prover) can prove to another (verifier) that a statement is true.

KZG: a prover can confirm a specific piece of data is in a dataset without revealing any other info about the dataset.


(13/24) Once we have our SRS and we've generated the polynomial from our data, we are ready to begin.

The prover is going to begin by applying the SRS to the data polynomial, generating a point on the elliptic curve.

We call this point a commitment.



Haym
@SalomonCrypto · [Follow](#)



(1/23) KZG Commitments Part 1: Commit

Our goal: 1) prove we are committed to specific data and 2) allow others to verify specific points within that dataset.

Before we get started, we need to prepare the data and elliptic curve.... Then, it's time for some magic!

KZG Polynomial Commitments

Red: Secret Green: Public Blue: Elliptic Curve (Public) Pink: Data (Varies)

Step 0: Preperation

shared between all commitments
specific to each commitment

Trusted Setup

Secret Number
S

Elliptic Curve:
 $y^2 = x^2 + ax + b$

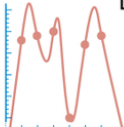
Data

Plaintext Data
STUART

UTF-8 Encoding:
83, 84, 85, 65, 84

$f(S^0) = [S^0] = S^0G$
 $f(S^1) = [S^1] = S^1G$
 $f(S^2) = [S^2] = S^2G$
 $f(S^3) = [S^3] = S^3G$
 \vdots
 $f(S^n) = [S^n] = S^nG$

Public Structured Reference String (SRS)




Lagrange Polynomial:
 $f(x) = a_0x^0 + a_1x^1 + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$


Step 1: Commit




Commitment: **f(S)** — single value that serves as the polynomial commitment

$f(S) = [a_0S^0 + a_1S^1 + a_2S^2 + a_3S^3 + a_4S^4 + a_5S^5]$

$f(S) = a_0[S^0] + a_1[S^1] + a_2[S^2] + a_3[S^3] + a_4[S^4] + a_5[S^5]$

1:06 AM · Oct 19, 2022



[Read the full conversation on Twitter](#)

 195  Reply  Copy link

[Read 6 replies](#)

(14/24) The commitment will serve as an anchor, tying the prover to the original dataset. Any changes to the underlying data will result in a new polynomial and therefore a new commitment.

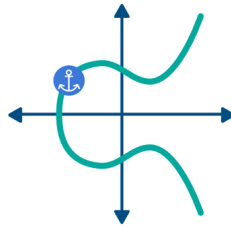
Changes to the data = previous commitments will generate invalid proofs.

KZG Commitment Scheme

First, the prover commits to data by creating a point on the elliptic curve. If the data changes, the prover cannot create valid proofs.

Prover


1) 
Commit




Verifier

(15/24) Now it's time for the verifier to test the prover. The verifier will select a position and request a proof based around that specific chunk of data.

The prover will divide the data polynomial to create a new quotient polynomial.



Haym
@SalomonCrypto · [Follow](#)



(1/13) KZG Commitments Part 2: Open

Previously, we committed to our data by evaluating a polynomial using an elliptic curve. Now, it's time to test that commitment; it's time for the verifier to see if the prover knows a specific piece of data.

It's time to open the commitment.

KZG Polynomial Commitments

Red: Secret Green: Public Blue: Elliptic Curve (Public) Pink: Data (Varies)

Step 2: Open

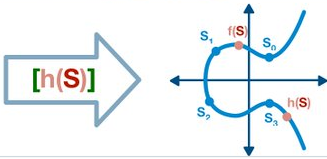
Prover	Verifier
Data → Polynomial → Commitment	Step 0 & Step 1
Step 2a	Given commitment, create proof for z
With z, calculate [h(S)] and f(z) and return the values < z, [h(S)], f(z)>	Step 2b

Proof: < z, [h(S)], f(z) >


Calculating [h(S)]

h(x) is a polynomial that can be generated by an honest prover with quick and (relatively) simple math

$$h(x) = \frac{f(x) - f(z)}{x - z}$$



2:27 AM · Oct 21, 2022 i



[Read the full conversation on Twitter](#)

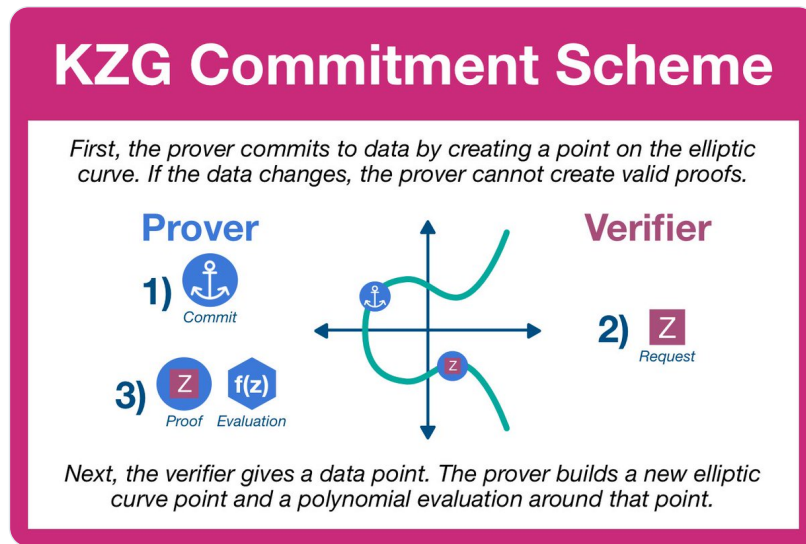
♥ 113
💬 Reply
🔗 Copy link

[Read 2 replies](#)

(16/24) This division is the crux of the KZG commitment scheme. Polynomial division is a (relatively) easy procedure; as long as an honest prover has full access to the data (and therefore the data polynomial), he can create and evaluate this new quotient polynomial.

(17/24) 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.



(18/24) Before we move on, let's consider the 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!

(19/24) 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.

But first, we are going to need one more tool.

(20/24) In order to combine the pieces sent by the prover, the verifier is going to need to multiply elliptic curve points. Unfortunately, this just isn't possible; that's not how elliptic curves work.

Fortunately, we have a substitute: elliptic curve pairings.

 **Haym**
@SalomonCrypto · [Follow](#) 

(1/24) Degen's Handbook: A Practical Guide to Elliptic Curve Pairings

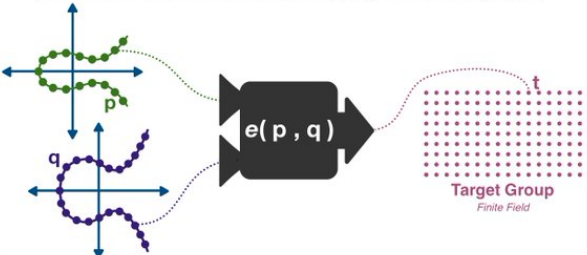
Magic is real, elliptic curve pairings are proof. Through incredibly advanced math, pairings allow us to multiply two polynomials... through elliptic curves!

Here's all you need to know.

Elliptic curve pairings are the elliptic point equivalent of multiplication

$$e(p, q) = t \approx p * q = t$$

An elliptic curve pairing is a function that takes a pair of elliptic curve points returns an element of some other group, called the target group





Think of a pairing as a black box that takes elliptic points. Pairings cannot be used consecutively; the target group points don't match the input points




Elliptic curve pairings are bilinear, holding to the following property:

$$e(p+r, q) = e(p, q) * e(r, q)$$
$$e(p, q+r) = e(p, q) * e(p, r)$$

Translation: you can pull additive component out of a pairing by multiplication

6:50 PM · Oct 20, 2022 

 [Read the full conversation on Twitter](#)

 363  Reply  Copy link

[Read 14 replies](#)

(21/24) And now, finally, we are ready to verify our proof.

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.

Haym
@SalomonCrypto · Follow

(1/22) KZG Commitments Part 3: Verify

After the prover has committed to the data, the verifier issued a challenge: "prove yourself based on this value." The prover has obliged, generating and transmitting a proof... now it's time to verify.

The conclusion of the KZG scheme.

KZG Polynomial Commitments

Red: Secret Green: Public Blue: Elliptic Curve (Public) Pink: Data (Varies)

Step 3: Verify

Prover	Verifier
1) Commit to $[f(S)] = C$	2a) Request proof of z
2b) Calculate $f(z)$, $[h(S)] = H$	3) Verify $e(H, [S - z]) = e(C - f(z), [1])$

What is $e(C - f(z)) = e(H, [S - z])$?

Goal: verify that the prover actually did the polynomial division to create $h(x)$, and that $[h(S)]$ the commitment of $h(x)$ at S .

What is $h(x)$?	Curve Points	Elliptic Curve Pairings
$h(x) = \frac{f(x) - f(z)}{x - z}$ <p>$h(x)$ is a polynomial created by the prover through the (relatively) simple process of polynomial division.</p>	<p>$[f(S)]$ and $[h(S)]$ are points on an elliptic curve</p>	<p>Pairings are functions that act as the elliptic point-equivalent of multiplication</p>

$$h(x) = \frac{f(x) - f(z)}{x - z} \rightarrow (x - z) \cdot h(x) = f(x) - f(z) \rightarrow e([x - z], [h(x)]) = e([f(x)] - [f(z)], [1])$$

$$e([S - z], H) = e(C - f(z), [1])$$

7:36 PM · Oct 21, 2022

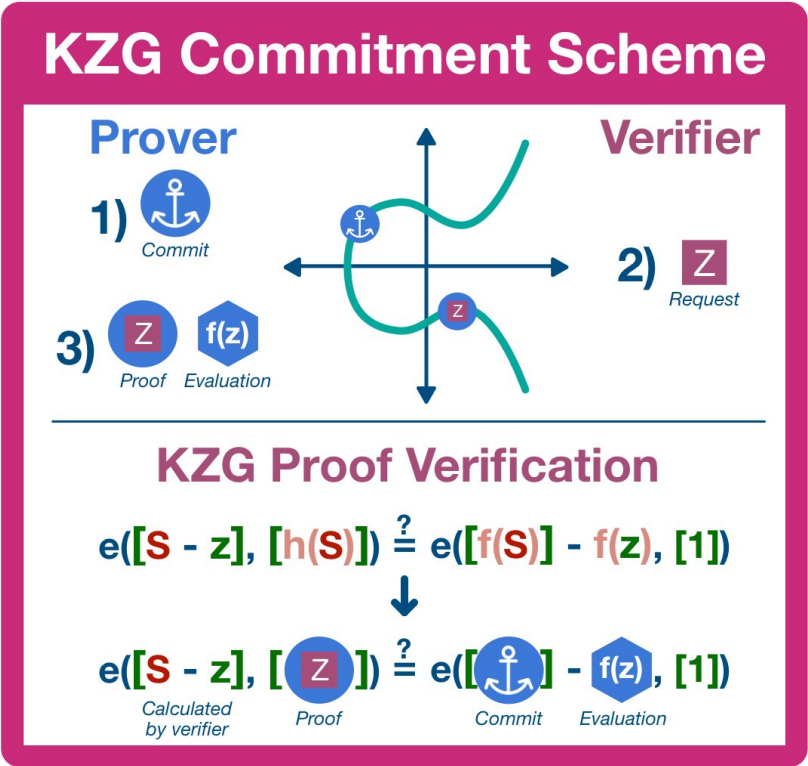
[Read the full conversation on Twitter](#)

62 Reply Copy link

Read 2 replies

(22/24) The commitment acts as an anchor, a proof entails quick polynomial division and verification is a simple check.

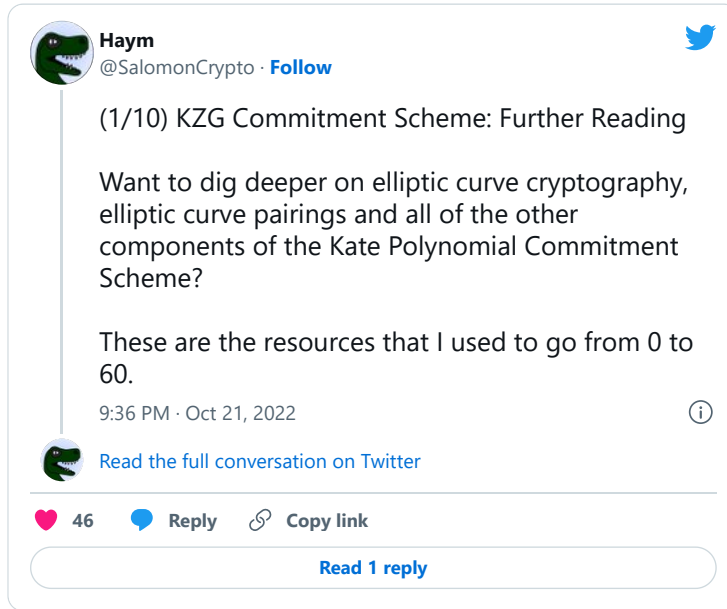
The verifier can know with mathematical certainty that the prover is honoring their commitment; whatever data is there is the same that created the commitment.



(23/24) So, in summary, a KZG commitment is a polynomial commitment scheme used to bind a prover to a specific set of data.

KZG commitments also have a very useful feature: they can be opened (audited) at any position without revealing any unrelated data.

(24/24) Looking to go deeper? Here are the resources I turned to again and again as I wrote this series:



Haym
@SalomonCrypto · [Follow](#)

(1/10) KZG Commitment Scheme: Further Reading

Want to dig deeper on elliptic curve cryptography, elliptic curve pairings and all of the other components of the Kate Polynomial Commitment Scheme?

These are the resources that I used to go from 0 to 60.

9:36 PM · Oct 21, 2022


[Read the full conversation on Twitter](#)

46 [Reply](#) [Copy link](#)

[Read 1 reply](#)

Like what you read? Help me spread the word by retweeting the thread (linked below).

Follow me for more explainers and as much alpha as I can possibly serve.

 **Haym**
@SalomonCrypto · [Follow](#)

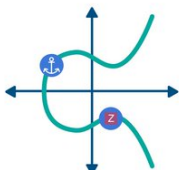
(1/24) KZG Polynomial Commitments: The Complete Guide

Our goal: 1) prove we are committed to a specific set of data and 2) allow others to verify specific points within that dataset.




Want to see some mathematical magic? This megathread is for you!

KZG Commitment Scheme


First, the prover commits to data by creating a point on the elliptic curve. If the data changes, the prover cannot create valid proofs.



Prover

- 1)  Commit
- 3)   Proof Evaluation

Verifier


- 2)  Request


Next, the verifier gives a data point. The prover builds a new elliptic curve point and a polynomial evaluation around that point.




KZG Proof Verification

$$e([S - z], [h(S)]) \stackrel{?}{=} e([f(S) - f(z)], [1])$$
$$\downarrow$$
$$e([S - z], [Z]) \stackrel{?}{=} e([\text{Anchor}] - f(z), [1])$$

Calculated by verifier Proof Commit Evaluation

6:25 AM · Oct 22, 2022 

 [Read the full conversation on Twitter](#)

 20  Reply  Copy link

[Read 1 reply](#)

...