KZG Commitment Scheme: Fundamentals & build other schemes based on.

Introduction of KZG Commitment scheme

One of the commitment schemes that a KGZ commitment scheme uses polynomial for commitment and its Processes.

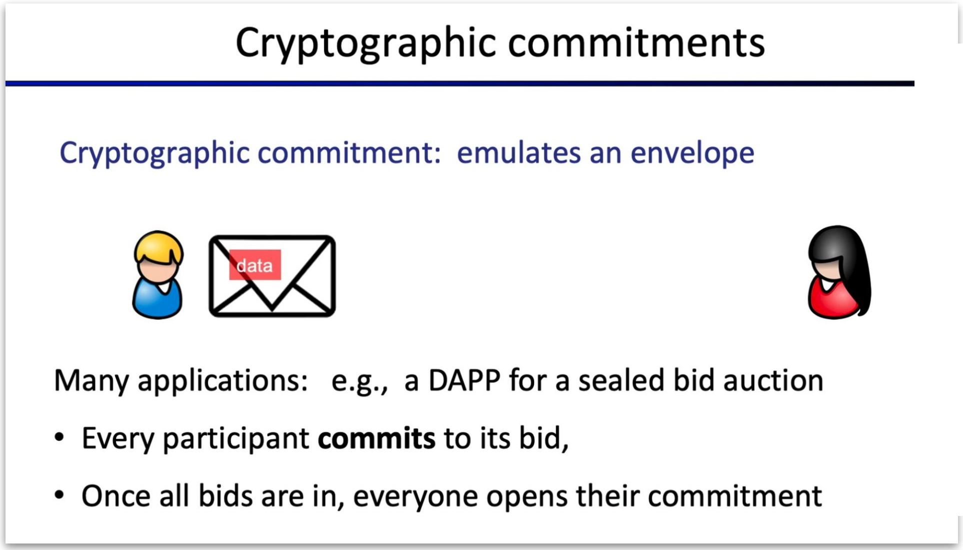
It enables committing to a polynomial and revealing its value at a specific point with a proof, without exposing the polynomial itself. Notable for its efficiency, small proof sizes, and flexibility

Introduction of KZG Commitment scheme

Key features of KZG Commitment scheme;

- Bilinear pairings
- Hidden order groups
- Fast Fourier Transform
- Small proof size
- Flexible commitment and proof generation

Commitment scheme?



Cryptographic commitments

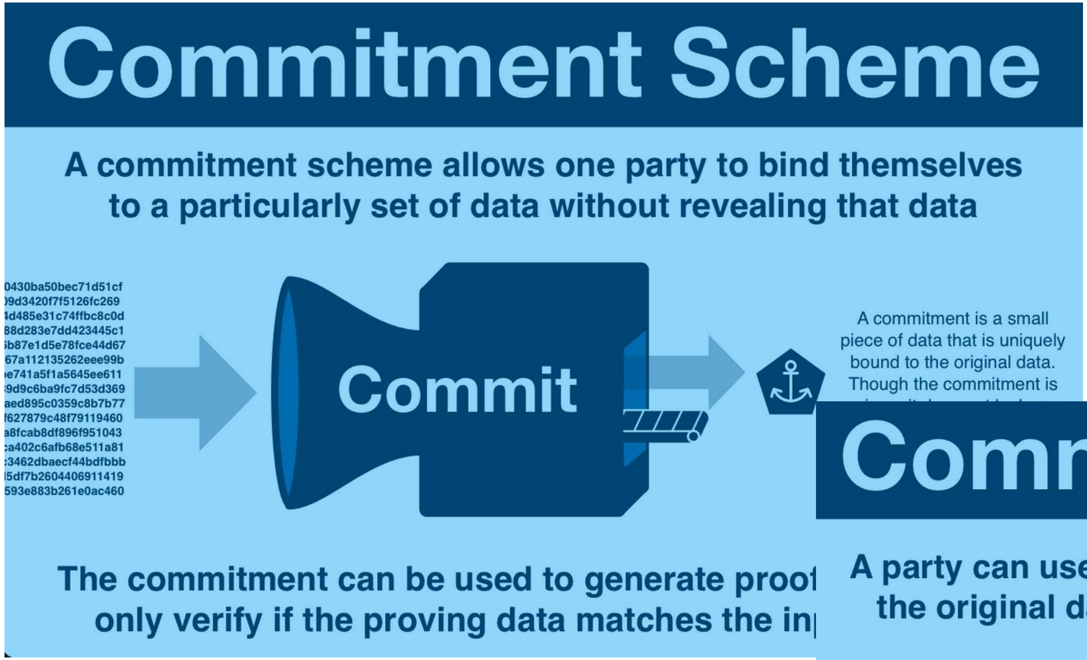
Cryptographic commitment: emulates an envelope

Many applications: e.g., a DAPP for a sealed bid auction

- Every participant **commits** to its bid,
- Once all bids are in, everyone opens their commitment

Commitment schemes: cryptographic primitive that allows one to commit to a chosen value (or chosen statement) while keeping it hidden to others, with the ability to reveal the committed value later.

Commitment scheme?

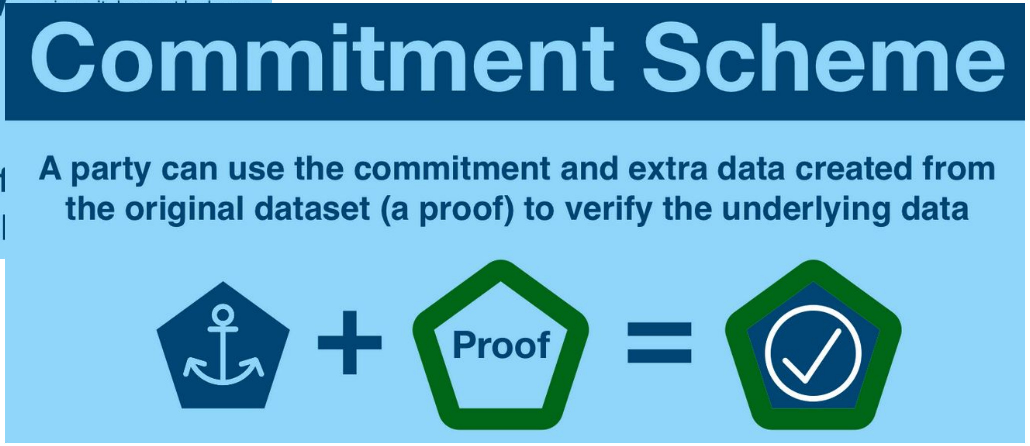


Commitment Scheme

A commitment scheme allows one party to bind themselves to a particularly set of data without revealing that data

A commitment is a small piece of data that is uniquely bound to the original data. Though the commitment is

The commitment can be used to generate proof only verify if the proving data matches the in



Commitment Scheme

A party can use the commitment and extra data created from the original dataset (a proof) to verify the underlying data

A commitment scheme is about creating commitment that is anchored to a piece of data. Though the commitment is unique, it doesn’t leak any information about it,

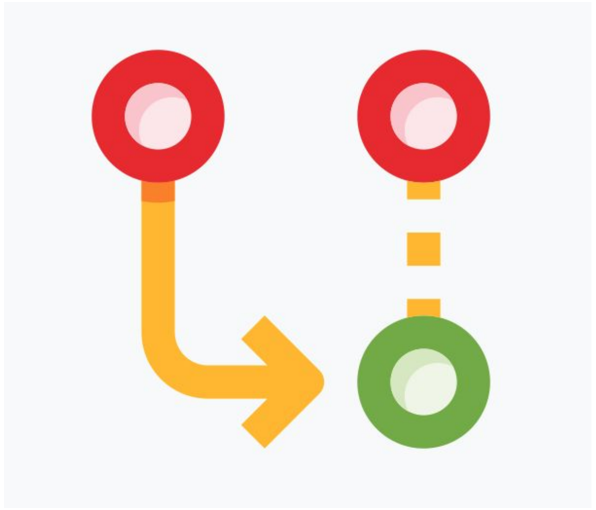
Schemes that can be opened at specific points are particularly useful.

Commitment scheme property: Binding & Hiding

- Binding: The binding property ensures that once a value is committed to, it cannot be changed. This property is crucial for ensuring the integrity of the commitment, as it prevents the committer from changing their committed value.

- Hiding: The hiding property guarantees that the committed value remains hidden until the committer chooses to reveal it. Meant that ,given only the commitment, attacker couldn’t get any information for value which committed. (related with Randomness)

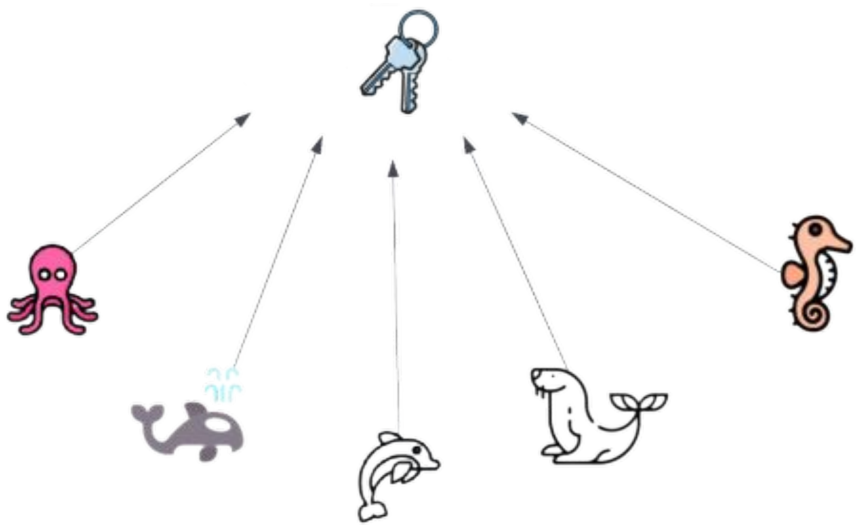
Introduction about KZG commitment scheme process



Commit

Verify

KZG Commitment's scheme process : Keygen/Trusted setup



someone (the prover) can prove the correctness of a statement to someone else (the verifier) without disclosing any information.

both the prover and the verifier need to commit to some shared secret and use this shared secret to generate public parameters.

$G1 \times G2 \rightarrow GT$

KZG Commitment's scheme process : Keygen/Trusted setup

Step 1: Preparing the building blocks In the setup phase, we need to prepare some essential components for the KZG commitment scheme. This includes:

1. Choosing a large prime number 'p': This number will be used in all our calculations to ensure security.
2. Selecting a group 'G': We pick a special mathematical group, typically an elliptic curve group, which has certain properties that make it suitable for the KZG commitment scheme.
3. Picking generator points 'g' and 'h': We choose two special points, called generator points, in our group 'G'. These points have unique properties that make them ideal for creating commitments.

Step 2: Generating the secret value 's' and SRS/CRS

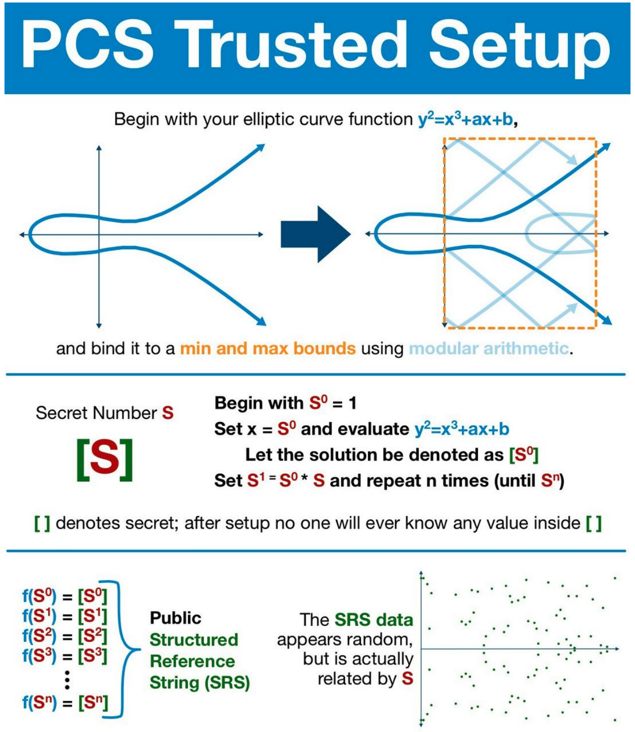
1. We need to create a secret value 's' that will be used during the setup process. This value should be randomly chosen and kept secret. It's important to note that 's' should be discarded after the setup process to ensure the security of the scheme.
2. In a secure multiparty computation, several parties collaborate to generate the secret value 's' and the corresponding SRS/CRS without any single party knowing 's'. This helps maintain the security of the system.

Step 3: Computing 'g^s' and 'h^s' Using the secret value 's', we compute two new values:

1. 'g^s': This is the result of raising the generator point 'g' to the power of 's'.
2. 'h^s': This is the result of raising the generator point 'h' to the power of 's'.

Step 4: Sharing public parameters We share the public parameters with everyone who will be using the KZG commitment scheme. These parameters include the large prime number 'p', the group 'G', the generator points 'g' and 'h', and the computed values 'g^s' and 'h^s'. However, the secret value 's' must not be shared and should be discarded after the setup process.

KZG Commitment's scheme process : Keygen/Trusted setup



PCS Trusted Setup

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

[S]

Begin with $S^0 = 1$

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

Generate the necessary public parameters and secret key. This step typically involves choosing a secret point 's' and computing public parameters related to 's'. The secret key remains with the prover, while the public parameters are shared with the verifier.

Why need trusted setup?

In the KZG commitment scheme's trusted setup, a secret value 's' is chosen from an elliptic curve that is pairing-friendly, establishing a hidden mathematical relationship. The public parameters are computed using this secret value 's', bilinear pairings, and hidden order groups, forming the Structured Reference String (SRS).

Compared to Pedersen-style commitment schemes, the KZG commitment scheme leverages this mathematical relationship and elliptic curve properties to achieve more efficient verification. It allows for constant-time verification instead of linear-time (O(n)) computation, providing a significant advantage in terms of efficiency.

TR DR; Good efficiency to calculate verification & Proof

Why need trusted setup? ; Bilinear pairings

$$e : G \times G \rightarrow G_T$$

$$e(g^a, g^b) = e(g, g)^{ab} \forall a, b \in \mathbb{Z}$$

$$e(g, g) \neq 1$$

In the context of elliptic curves, a bilinear pairing is a map that takes two points from an elliptic curve and produces an element in a finite field.

Why need trusted setup? ; Bilinear pairings

1. Bilinearity: The pairing operation is linear in both arguments.
2. Non-degeneracy: The pairing of the generator points of G1 and G2 (or their representatives) results in a non-identity element in GT.
3. Computability: It is computationally efficient to compute the bilinear pairing for any given input points from G1 and G2.

Why need trusted setup?; Hidden order Group

In the KZG commitment scheme, hidden order groups refer to a specific type of group in which the order (the number of elements in the group) is not easily computable. In the context of KZG, hidden order groups are typically derived from elliptic curves with a pairing-friendly structure. These groups provide added security for the commitment scheme, making it more difficult for an attacker to break the binding and hiding properties.

KZG Commitment’s scheme process : Commit

Step 1: Representing the data as a polynomial

- In the KZG commitment scheme, we represent the data we want to commit to as a polynomial 'P(x)'. This polynomial has coefficients that correspond to the pieces of data we want to commit to. The degree of the polynomial depends on the number of data pieces.

Step 1.1: Creating the Lagrange polynomial

- To represent the data as a polynomial, we use the Lagrange interpolation method. This method helps us construct a polynomial P(x) that passes through all the data points we want to commit to.

Step 2: Fast Fourier Transform (FFT)

- The FFT is an efficient algorithm for computing the coefficients of P(x) from the data points using the Lagrange interpolation method. By applying the FFT, we can compute the coefficients in O(n log n) time, which is much faster than the naive approach, which takes O(n^2) time.

KZG Commitment's scheme process : Commit

Step 3: Committing to the polynomial

1. To create a commitment to the polynomial 'P(x)', we compute a commitment 'C' by evaluating the polynomial at the secret value 's' using the public parameters from the setup phase (SRS or CRS).
2. The commitment 'C' is computed as: C = P(s) * g, where 'g' is the generator point from the setup phase.
3. The commitment 'C' does not reveal any information about the polynomial 'P(x)' or the secret value 's', making it secure and private.

Step 4: Sharing the commitment

- The commitment 'C' is shared with other parties, who can later use it to verify proofs related to the data committed in 'P(x)'.

KZG Commitment's scheme process : Commit

Step 1: Commit

Commitment: $\boxed{[f(S)]}$ single value that serves as the polynomial commitment

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

$$[f(S)] = a_0[S^0] + a_1[S^1] + \underbrace{a_2[S^2]}_{\boxed{a_i}\boxed{[S^i]}} + a_3[S^3] + a_4[S^4] + a_5[S^5]$$

single value generated during polynomial creation — $\boxed{a_i}$ $\boxed{[S^i]}$ — single value generated during trusted setup

Commitment: $[f(S)]$

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

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

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

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

$$\underbrace{[f(S)]}_{\text{Commitment}} = \underbrace{f(S) \cdot G}_{\text{Point on elliptic curve}}$$

KZG Commitment's scheme process : Open/Evaluation

Step 1: Receiving the commitment

- During the evaluation phase, a verifier receives a commitment 'C' created during the commit phase. The verifier does not know the polynomial 'P(x)' or the secret value 's', but has access to the public parameters from the setup phase (SRS or CRS).

Step 2: Requesting a proof for a specific point

- The verifier requests a proof from the prover that the committed polynomial 'P(x)' has a specific value 'y' at a point 'x_0'. The prover knows the polynomial 'P(x)' and the secret value 's'.

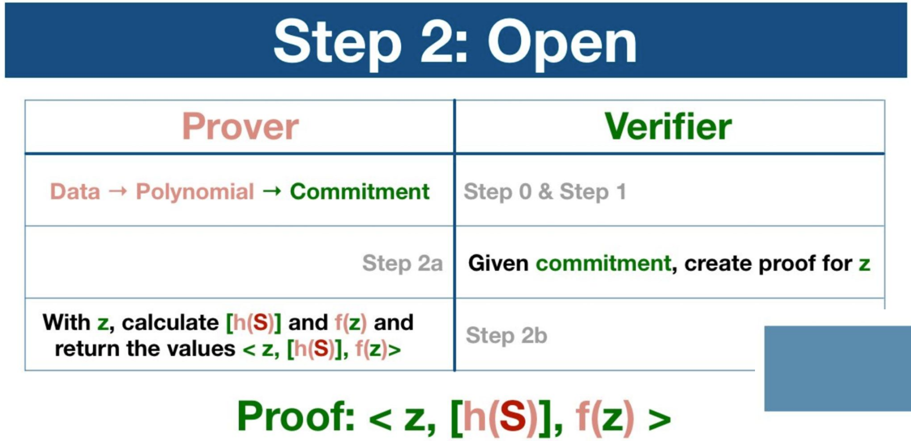
Step 3: Generating the proof

1. The prover computes a new polynomial 'Q(x)' such that 'Q(x) = (P(x) - y) / (x - x_0)'. This polynomial has the property that 'Q(x_0) = (P(x_0) - y) / (x_0 - x_0) = 0'.
2. The prover then evaluates 'Q(x)' at the secret value 's' to obtain 'Q(s)'.
3. The proof 'π' is computed as: π = Q(s) * g, where 'g' is the generator point from the setup phase.
4. The prover sends the proof 'π' to the verifier.

Step 4: Verifying the proof

1. The verifier checks if the following equation holds true: C / (y * g) = π * (x_0 - x_0 * g)^(s-1).
2. If the equation holds true, the verifier accepts the proof, concluding that 'P(x_0) = y'. Otherwise, the verifier rejects the proof.

KZG Commitment's scheme process : Open



Step 2: Open

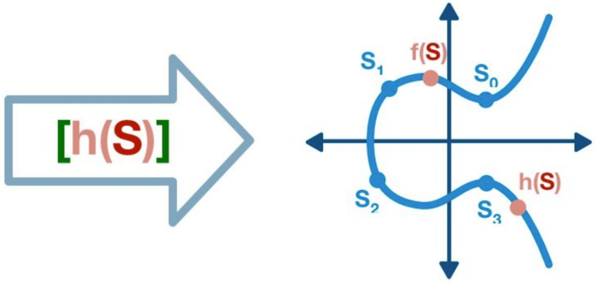
Prover	Verifier
Data → Polynomial → Commitment	Step 0 & Step 1
Step 2a	Given commitment, create proof for z
With z, calculate [h(S)] and f(z) and return the values < z, [h(S)], f(z)>	Step 2b

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

Caclulating [h(S)]

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

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



[h(S)] is also a point on the elliptic curve

KZG Commitment’s scheme process : Verify

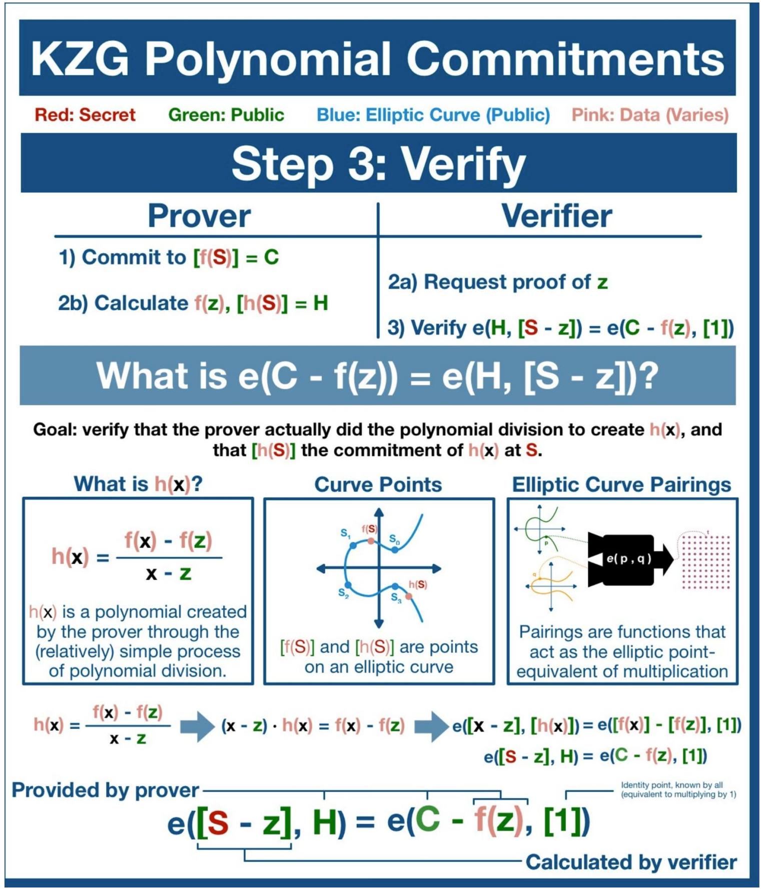
Step 1

- Receiving the commitment and proof During the verification phase, a verifier receives a commitment 'C' created during the commit phase and a proof 'π' created during the evaluation phase. The verifier does not know the polynomial 'P(x)' or the secret value 's', but has access to the public parameters from the setup phase (SRS or CRS).

Step 2: Verifying the proof

1. The verifier checks if the following equation holds true: e(C / (y * g), g) = e(π, x_0 * g - g), where 'e' denotes a bilinear pairing operation.
2. If the equation holds true, the verifier accepts the proof, concluding that 'P(x_0) = y'. Otherwise, the verifier rejects the proof.

KZG Commitment's scheme process : Verify



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 $h(x)$?

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

$h(x)$ is a polynomial created by the prover through the (relatively) simple process of polynomial division.

Curve Points

$[f(S)]$ and $[h(S)]$ are points on an elliptic curve

Elliptic Curve Pairings

Pairings are functions that act as the elliptic point-equivalent of multiplication

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

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

Provided by prover

Identity point, known by all (equivalent to multiplying by 1)

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

Calculated by verifier

Bilinear pairings & Hidden order group

1. Bilinear Pairings: A bilinear pairing is a special type of function that takes two elements from two different groups and maps them to a third group, while preserving certain mathematical properties. In the context of the KZG scheme, bilinear pairings are used to enable efficient operations on committed polynomials and to facilitate the verification of evaluation proofs.

- Type 3 Pairings : BLS curve, BN curve

2. Hidden Order Groups: A hidden order group is a group whose order (the number of elements in the group) is unknown or hard to compute. In the KZG scheme, hidden order groups provide the foundation for the commitment scheme's security. The difficulty of computing the group's order makes it challenging for an attacker to exploit the group's structure and break the binding and hiding properties of the commitment scheme.

- Derived from pairing friendly elliptic curve.

Bilinear pairings & Hidden order group : when these are using?

1. Setup Phase: In the initial setup phase, a bilinear pairing is chosen, and a hidden order group is used to generate the public parameters. This includes a structured reference string (SRS) containing the group generator g, the secret evaluation point s, and the corresponding public evaluation points g^s, g^(s^2), g^(s^3), ..., g^(s^d) for a maximum polynomial degree d.

2. Commitment Phase: To commit to a polynomial f(x), the sender computes the commitment C = g^f(s), where g is the group generator and s is the secret evaluation point from the SRS. Bilinear pairings are used in this computation to ensure that the commitment preserves the structure of the polynomial.

3. Evaluation Proofs: When the sender wants to reveal a specific evaluation f(z) for some point z without disclosing the entire polynomial, they provide an evaluation proof π. The proof is a commitment to the polynomial quotient q(x) = (f(x) - f(z))/(x - z). Bilinear pairings are used to verify the evaluation proof efficiently, ensuring that the revealed evaluation is consistent with the commitment.

Comparisons with other polynomial commitment scheme

- Inner Product Argument Scheme is a protocol used to prove that the inner product of two vectors is equal to a certain value, without revealing the vectors themselves. It is an essential building block in bulletproofs and other efficient zero-knowledge proof systems.

- KZG Commitment Scheme, also known as Kate Commitments, is a polynomial commitment scheme based on bilinear pairings.

Comparisons with other polynomial commitment scheme / Trade offs

1. Efficiency:
    - Inner Product Argument Scheme is efficient in terms of proof size and computation time, particularly when applied to large vectors. However, it has limited use cases, primarily focusing on inner product proofs.
    - KZG Commitment Scheme is more versatile, as it can be used to commit to polynomials and prove evaluations. While it is also efficient, it might have higher computational complexity due to the use of bilinear pairings.
2. Versatility:
    - Inner Product Argument Scheme is mainly focused on inner product proofs, which can be a limitation in terms of the variety of applications it can support.
    - KZG Commitment Scheme is more versatile, allowing for commitments to polynomials and proving evaluations, making it suitable for a broader range of cryptographic applications.
3. Assumptions:
    - Inner Product Argument Scheme is based on the discrete logarithm problem, which is a well-studied assumption in cryptography.
    - KZG Commitment Scheme relies on bilinear pairings and requires a trusted setup. The trusted setup can be a drawback in some applications, as it might introduce trust assumptions and potential vulnerabilities.

Comparisons with other polynomial commitment scheme; KZG commitment's unique features.

Feature 1: Small proof size

- Achieved through: Polynomial commitments
- The KZG commitment scheme uses polynomials, which are math equations involving variables raised to different powers. By representing secrets as polynomials, we can create proofs that only involve a few special values. This keeps the proof size small, no matter how big or complex the original secret is.

Feature 2: Quick verification

- Achieved through: Bilinear pairings
- Bilinear pairings are special math functions that help us quickly check proofs in the KZG commitment scheme. These functions allow us to combine and compare values in a way that's efficient and secure. By using bilinear pairings, we can quickly verify if a proof is valid without needing to know the original secret.

Comparisons with other polynomial commitment scheme; KZG commitment's unique features.

Feature 3: Keeping secrets safe

- Achieved through: Commitments and hidden coefficients
- When you create a KZG commitment, you're essentially locking your secret polynomial inside a box using a special math equation. The commitment only reveals a single value, which doesn't give away any information about the polynomial's coefficients. This way, your secret stays safe and hidden from others.

Feature 4: Mixing and matching

- Achieved through: Homomorphic property
- The KZG commitment scheme has a homomorphic property, allowing you to combine commitments and their corresponding secrets. This means you can add or subtract polynomials and their commitments without revealing the original secrets. This property is useful for advanced cryptographic applications, such as secure multi-party computation.

How to build another complex scheme based on the Kate commitment.

- Sonic, Marlin, Halo, Plonk
  - Sonic: Sonic is a zk-SNARK system that uses a universal and updatable structured reference string (SRS). It does not explicitly use hidden order groups; instead, it relies on a combination of elliptic curves and polynomial commitments for its cryptographic operations.

  - Marlin: Marlin is a zk-SNARK that uses a polynomial commitment scheme called "KZG10," which is similar to the KZG commitment scheme. Marlin uses elliptic curves with pairing-friendly properties, which can provide hidden order groups in some instances.

  - Halo: Halo is a zk-SNARK that employs a recursive proof composition, allowing for the construction of proofs for arbitrary-depth circuits. It relies on elliptic curve pairings but does not explicitly use hidden order groups. Instead, it uses an accumulator-based construction that achieves similar goals.

  - Plonk: Plonk is a zk-SNARK that uses a universal SRS and an efficient polynomial commitment scheme based on the KZG commitment scheme. Like Marlin, Plonk also employs elliptic curves with pairing-friendly properties, which may involve hidden order groups in some cases.

Compare with another commitment schemes without using polynomial commitment

Step 1: Common Points of Plonk, Sonic and Marlin

1. Polynomial Commitments: All three protocols (PLONK, Sonic, and Marlin) use the KZG commitment scheme for creating efficient polynomial commitments. This allows them to commit to polynomials without revealing the actual coefficients and enables efficient zero-knowledge proofs.
2. Zero-Knowledge Proofs: PLONK, Sonic, and Marlin are all zero-knowledge proof systems, which allow a prover to convince a verifier that a statement is true without revealing any information about the statement itself.
3. Universal and Updatable Setup: Sonic and Marlin both have a universal and updatable setup. PLONK also has a universal setup, meaning the setup phase can be performed once and then used for multiple proofs across different circuits.

Step 2: Differences of Plonk, Sonic and Marlin

1. PLONK: PLONK uses a specific permutation argument for checking the consistency of wire values in arithmetic circuits. This makes PLONK more efficient than Sonic but less general than Marlin, as it focuses on a specific type of constraint system.
2. Sonic: Sonic uses a more general constraint system, allowing it to handle a wider range of problems compared to PLONK. However, this comes at the cost of increased proof size and computational complexity compared to PLONK.
3. Marlin: Marlin combines the benefits of both PLONK and Sonic, offering a more efficient and general-purpose zero-knowledge proof system. It leverages a combination of polynomial commitment schemes, including the KZG commitment scheme, to achieve better performance compared to Sonic and Groth16.

THE END. THANK YOU