Bonus Lecture #4:
KZG Commitments

COMS 4995-001:
The Science of Blockchains

URL: https://timroughgarden.org/s25/

Tim Roughgarden

Goals for Bonus Lecture #4

1. Fun facts about polynomials.
   – roots of polynomials, encoding data with a polynomial
2. KZG commitments: the basic idea.
   – commitment, proofs via polynomial evaluation
3. Making it real: structured reference string + group pairings.
   – implementing the idea of “evaluation at an unknown random input”
4. Trusted setup ceremonies.
   – where does the structured reference string come from?

Roots of Polynomials

Recall: a (degree-d) polynomial has the form

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{d} a_i x^i.$$

Roots of Polynomials

Recall: a (degree-d) polynomial has the form

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{d} a_i x^i.$$

Fact 1: if polynomial f has the form $f(x) = (x - r) \cdot q(x)$, where q is a degree-(d-1) polynomial, then $f(r) = 0$.

Roots of Polynomials

Recall: a (degree-d) polynomial has the form

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{d} a_i x^i.$$

Fact 1: if polynomial f has the form $f(x) = (x - r) \cdot q(x)$, where q is a degree-(d-1) polynomial, then $f(r) = 0$.

Fact 2: if polynomial f satisfies $f(r) = 0$, then f has the form $f(x) = (x - r) \cdot q(x)$, where q is a degree-(d-1) polynomial.

- example: can write $x^3 - 6x^2 + 11x - 6$ as $(x - 1)(x^2 - 5x + 6)$
- for proof(s), see “polynomial factor theorem” (e.g., use long division)

Roots of Polynomials

Recall: a (degree-d) polynomial has the form

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{d} a_i x^i.$$

Fact 2: if polynomial f satisfies $f(r) = 0$, then f has the form $f(x) = (x - r) \cdot q(x)$, where q is a degree-(d-1) polynomial.

Roots of Polynomials

Recall: a (degree-d) polynomial has the form

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{d} a_i x^i.$$

Fact 2: if polynomial f satisfies $f(r) = 0$, then f has the form $f(x) = (x - r) \cdot q(x)$, where q is a degree-(d-1) polynomial.

Corollary 1: a (non-zero) degree-d polynomial has ≤ d roots.

– after applying Fact 2 d times, left with a (non-zero) constant

Roots of Polynomials

Recall: a (degree-d) polynomial has the form

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{d} a_i x^i.$$

Fact 2: if polynomial f satisfies $f(r) = 0$, then f has the form $f(x) = (x - r) \cdot q(x)$, where q is a degree-(d-1) polynomial.

Corollary 1: a (non-zero) degree-d polynomial has ≤ d roots.

– after applying Fact 2 d times, left with a (non-zero) constant

Corollary 2: if p,q are distinct degree-d polynomials, then p(x)=q(x) for at most d points x. [because p-q has ≤ d roots]

Roots of Polynomials

Recall: a (degree-d) polynomial has the form

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{d} a_i x^i.$$

Fact 2: if polynomial f satisfies $f(r) = 0$, then f has the form $f(x) = (x - r) \cdot q(x)$, where q is a degree-(d-1) polynomial.

Corollary 1: a (non-zero) degree-d polynomial has ≤ d roots.

– after applying Fact 2 d times, left with a (non-zero) constant

Corollary 2: if p,q are distinct degree-d polynomials, then p(x)=q(x) for at most d points x. [because p-q has ≤ d roots]

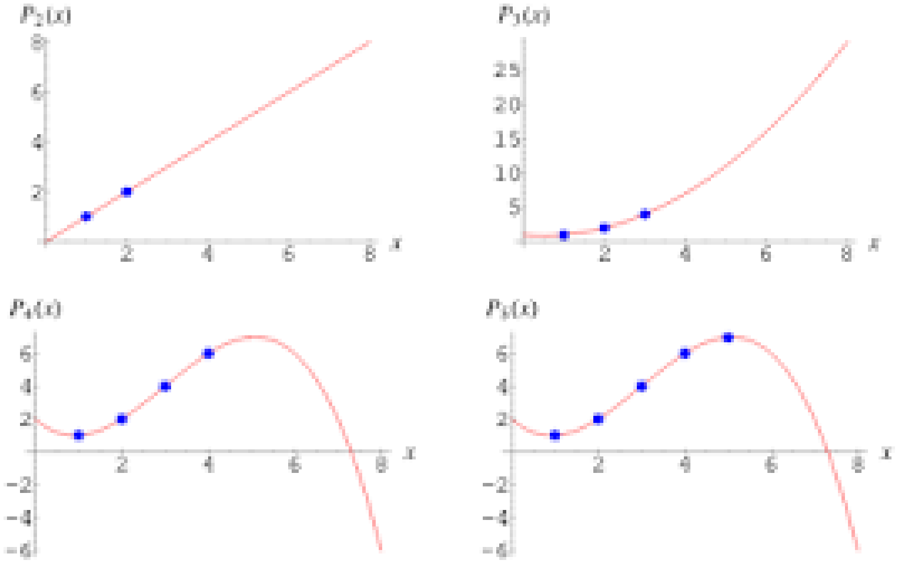
– for randomly chosen x (from big set), p(x),q(x) almost certainly differ

Polynomial Interpolation

Fact 3: given points $(x_0, y_0), (x_1, y_1), \ldots, (x_d, y_d)$, can easily compute a degree-d polynomial f s.t. $f(x_i) = y_i$ for all i=0,1,..,d.

Polynomial Interpolation

Fact 3: given points $(x_0, y_0), (x_1, y_1), \dots, (x_d, y_d)$, can easily compute a degree-d polynomial f s.t. $f(x_i) = y_i$ for all i=0,1,..,d.



Polynomial Interpolation

Fact 3: given points $(x_0, y_0), (x_1, y_1), \ldots, (x_d, y_d)$, can easily compute a degree-d polynomial f s.t. $f(x_i) = y_i$ for all i=0,1,..,d.

Example: $y_0 = 1, y_1 = y_2 = \cdots = y_d = 0.$

Polynomial Interpolation

Fact 3: given points $(x_0, y_0), (x_1, y_1), \ldots, (x_d, y_d)$, can easily compute a degree-d polynomial f s.t. $f(x_i) = y_i$ for all i=0,1,..,d.

Example: $y_0 = 1, y_1 = y_2 = \cdots = y_d = 0.$

- interpolating function should have form

$$f(x) = c \cdot (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_d)$$ [for scalar c]

Polynomial Interpolation

Fact 3: given points $(x_0, y_0), (x_1, y_1), \ldots, (x_d, y_d)$, can easily compute a degree-d polynomial f s.t. $f(x_i) = y_i$ for all i=0,1,..,d.

Example: $y_0 = 1, y_1 = y_2 = \cdots = y_d = 0.$

- interpolating function should have form
  $$f(x) = c \cdot (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_d)$$ [for scalar c]
- for $f(x_0) = 1$, take $c = 1/[(x_0 - x_1) \cdot (x_0 - x_2) \cdot \ldots \cdot (x_0 - x_d)]$

Polynomial Interpolation

Fact 3: given points $(x_0, y_0), (x_1, y_1), \dots, (x_d, y_d)$, can easily compute a degree-d polynomial f s.t. $f(x_i) = y_i$ for all i=0,1,..,d.

Example: $y_0 = 1, y_1 = y_2 = \cdots = y_d = 0.$

- interpolating function should have form

  $f(x) = c \cdot (x - x_1) \cdot (x - x_2) \cdot \dots \cdot (x - x_d)$ [for scalar c]

- for $f(x_0) = 1$, take $c = 1/[(x_0 - x_1) \cdot (x_0 - x_2) \cdot \dots \cdot (x_0 - x_d)]$

Note: can interpolate arbitrary points via linear combinations $f(x) = \sum_{i=0}^{d} y_i \cdot L_i(x)$ of these "Lagrange basis functions."

Polynomial Interpolation

Fact 3: given points $(x_0, y_0), (x_1, y_1), \ldots, (x_d, y_d)$, can easily compute a degree-d polynomial f s.t. $f(x_i) = y_i$ for all i=0,1,..,d.

Example: $y_0 = 1, y_1 = y_2 = \cdots = y_d = 0.$

- interpolating function should have form

  $f(x) = c \cdot (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_d)$ [for scalar c]

- for $f(x_0) = 1$, take $c = 1/[(x_0 - x_1) \cdot (x_0 - x_2) \cdot \ldots \cdot (x_0 - x_d)]$

Note: can interpolate arbitrary points via linear combinations $f(x) = \sum_{i=0}^{d} y_i \cdot L_i(x)$ of these “Lagrange basis functions.”

– also: can redundantly encode polynomial via > d+1 evaluations

Encoding Data as a Polynomial

Idea: encode list of objects as evaluations of a polynomial.

- cf., Merkle trees: encode (hashes of) objects as leaves of a tree

Encoding Data as a Polynomial

Idea: encode list of objects as evaluations of a polynomial.

- cf., Merkle trees: encode (hashes of) objects as leaves of a tree

Example: list $a_0, a_1, \ldots, a_d$ of objects, each a_i with length 256 bits.

- interpret as elements $y_0, y_1, \ldots, y_d$ of Z_p={0,1,…,p-1} for ≈256-bit prime p
- can compute degree-d polynomial f s.t. $f(i)=y_i$ for all i=0,1,2,…,d
  - all arithmetic (including division) done modulo p

Encoding Data as a Polynomial

Idea: encode list of objects as evaluations of a polynomial.

– cf., Merkle trees: encode (hashes of) objects as leaves of a tree

Example: list $a_0, a_1, \ldots, a_d$ of objects, each a_i with length 256 bits.

– interpret as elements $y_0, y_1, \ldots, y_d$ of Z_p={0,1,…,p-1} for ≈256-bit prime p

– can compute degree-d polynomial f s.t. f(i)=y_i for all i=0,1,2,…,d

- all arithmetic (including division) done modulo p

Ethereum blob: as above, with d = 4096 (size ≈ 125 Kb).

– validators store blobs for 2 weeks, (KZG) commitments to blobs forever

KZG Commitments (v1)

Goal: short, binding commitment to a polynomial f.

- where evaluations of f on 0,1,…,d correspond to list of d+1 objects
- cf., guarantees provided by a Merkle tree (no false positives/negatives)

KZG Commitments (v1)

Goal: short, binding commitment to a polynomial f.

- where evaluations of f on 0,1,…,d correspond to list of d+1 objects
- cf., guarantees provided by a Merkle tree (no false positives/negatives)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

- note: τ generally >> d, does not correspond to one of encoded objects
- commitment clearly short, but how to compute $f(\tau)$ if τ is unknown?

KZG Commitments (v1)

Goal: short, binding commitment to a polynomial f.

- where evaluations of f on 0,1,…,d correspond to list of d+1 objects
- cf., guarantees provided by a Merkle tree (no false positives/negatives)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

- note: τ generally >> d, does not correspond to one of encoded objects
- commitment clearly short, but how to compute $f(\tau)$ if τ is unknown?

Key idea #2: to prove that $f(z) = v$:

KZG Commitments (v1)

Goal: short, binding commitment to a polynomial f.

– where evaluations of f on 0,1,…,d correspond to list of d+1 objects

– cf., guarantees provided by a Merkle tree (no false positives/negatives)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

– note: τ generally >> d, does not correspond to one of encoded objects

– commitment clearly short, but how to compute $f(\tau)$ if τ is unknown?

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]

KZG Commitments (v1)

Goal: short, binding commitment to a polynomial f.

- where evaluations of f on 0,1,…,d correspond to list of d+1 objects
- cf., guarantees provided by a Merkle tree (no false positives/negatives)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

- note: τ generally >> d, does not correspond to one of encoded objects
- commitment clearly short, but how to compute $f(\tau)$ if τ is unknown?

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = “witness polynomial” (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⬄ $f(\tau) - v = (\tau - z) \cdot \pi$

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⇔ $f(\tau) - v = (\tau - z) \cdot \pi$ [question: isn't τ unknown?]

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⇔ $f(\tau) - v = (\tau - z) \cdot \pi$ [question: isn't τ unknown?]

Correctness:

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⬄ $f(\tau) - v = (\tau - z) \cdot \pi$ [question: isn't τ unknown?]

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $\pi = w(\tau)$

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⇔ $f(\tau) - v = (\tau - z) \cdot \pi$ [question: isn't τ unknown?]

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $\pi = w(\tau)$
➔ $(\tau - z) \cdot \pi = (\tau - z) \cdot w(\tau)$

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⇔ $f(\tau) - v = (\tau - z) \cdot \pi$ [question: isn't τ unknown?]

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $\pi = w(\tau)$
➔ $(\tau - z) \cdot \pi = (\tau - z) \cdot w(\tau) = f(\tau) - v$ [no matter what τ is]

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⇔ $f(\tau) - v = (\tau - z) \cdot \pi$ [question: isn't τ unknown?]

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $\pi = w(\tau)$
➔ $(\tau - z) \cdot \pi = (\tau - z) \cdot w(\tau) = f(\tau) - v$ [no matter what τ is]

(ii) [intuition] if $f(z) \neq v$

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⇔ $f(\tau) - v = (\tau - z) \cdot \pi$ [question: isn't τ unknown?]

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $\pi = w(\tau)$
➔ $(\tau - z) \cdot \pi = (\tau - z) \cdot w(\tau) = f(\tau) - v$ [no matter what τ is]

(ii) [intuition] if $f(z) \neq v$ ➔ for any w, $(x - z) \cdot w(x) \neq f(x) - v$

KZG Commitments (v1)

Key idea #1: commitment = $f(\tau)$ on unknown random input τ.

Key idea #2: to prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $w(\tau)$ on same unknown random input τ

Verification: given commitment $f(\tau)$, alleged proof π that $f(z) = v$:

- accept proof ⇔ $f(\tau) - v = (\tau - z) \cdot \pi$ [question: isn't τ unknown?]

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $\pi = w(\tau)$
➔ $(\tau - z) \cdot \pi = (\tau - z) \cdot w(\tau) = f(\tau) - v$ [no matter what τ is]

(ii) [intuition] if $f(z) \neq v$ ➔ for any w, $(x - z) \cdot w(x) \neq f(x) - v$ ➔ for almost all τ, $(\tau - z) \cdot w(\tau) \neq f(\tau) - v$ [Corollary 2]

Structured Reference String

Note: if τ is known, easy to forge false proofs!

- to "prove" that $f(z) = v$, just set $\pi = (f(\tau) - v)/(\tau - z)$

Structured Reference String

Note: if τ is known, easy to forge false proofs!

- to "prove" that $f(z) = v$, just set $\pi = (f(\tau) - v)/(\tau - z)$

Question: how to compute $f(\tau)$ without knowing τ?

Structured Reference String

Note: if τ is known, easy to forge false proofs!

- to "prove" that $f(z) = v$, just set $\pi = (f(\tau) - v)/(\tau - z)$

Question: how to compute $f(\tau)$ without knowing τ?

Idea: compute $g^{f(\tau)}$, where g = generator of some cyclic group.

- e.g., of an elliptic curve group
- i.e., compute $f(\tau)$ only in "encrypted form" (i.e., in exponent)
- cf., Schnorr signatures

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, computing g^x ?

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- given g^x, computing x?

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- [if discrete log is hard] given g^x, hard to compute x

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- [if discrete log is hard] given g^x, hard to compute x
- computing g^{x+y} from g^x and g^y ?

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- [if discrete log is hard] given g^x, hard to compute x
- easy to compute g^{x+y} from g^x and g^y

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- [if discrete log is hard] given g^x, hard to compute x
- easy to compute g^{x+y} from g^x and g^y
- computing g^{ax} from a and g^x ?

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- [if discrete log is hard] given g^x, hard to compute x
- easy to compute g^{x+y} from g^x and g^y
- easy to compute g^{ax} from a and g^x

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- [if discrete log is hard] given g^x, hard to compute x
- easy to compute g^{x+y} from g^x and g^y
- easy to compute g^{ax} from a and g^x
- computing g^{xy} from g^x and g^y?

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- [if discrete log is hard] given g^x, hard to compute x
- easy to compute g^{x+y} from g^x and g^y
- easy to compute g^{ax} from a and g^x
- generally hard to compute g^{xy} from g^x and g^y
  - "computational Diffie-Hellman (CDH)" assumption

Aside: Computing in the Exponent

Recall: given generator g of a cyclic group G (of known order):

- given x, easy to compute g^x [repeated squaring]
- [if discrete log is hard] given g^x, hard to compute x
- easy to compute g^{x+y} from g^x and g^y
- easy to compute g^{ax} from a and g^x
- generally hard to compute g^{xy} from g^x and g^y
  - "computational Diffie-Hellman (CDH)" assumption

Tl;dr: easy to add or scale in the exponent, but hard to multiply.

Structured Reference String

Question: how to compute $f(\tau)$ without knowing τ?

Idea: compute $g^{f(\tau)}$, where g = generator of some cyclic group.

Structured Reference String

Question: how to compute $f(\tau)$ without knowing τ?

Idea: compute $g^{f(\tau)}$, where g = generator of some cyclic group.

- write $g^{f(\tau)} = g^{a_d\tau^d + a_{d-1}\tau^{d-1} + \cdots + a_1\tau + a_0}$

Structured Reference String

Question: how to compute $f(\tau)$ without knowing τ?

Idea: compute $g^{f(\tau)}$, where g = generator of some cyclic group.

- write $g^{f(\tau)} = g^{a_d\tau^d + a_{d-1}\tau^{d-1} + \cdots + a_1\tau + a_0}$
- issue: easy to add/scale in exponent, but hard to multiply

Structured Reference String

Question: how to compute $f(\tau)$ without knowing τ?

Idea: compute $g^{f(\tau)}$, where g = generator of some cyclic group.

- write $g^{f(\tau)} = g^{a_d\tau^d + a_{d-1}\tau^{d-1} + \cdots + a_1\tau + a_0}$
- issue: easy to add/scale in exponent, but hard to multiply
- solution: assume "powers of tau" $\sigma = (g^{\tau}, g^{\tau^2}, g^{\tau^2}, \ldots, g^{\tau^d})$ are known (e.g., part of description of commitment scheme)
  - σ called a "structured reference string," a form of "trusted setup"

KZG Commitments (v2)

Assume: powers of tau σ publicly known, no one knows τ.

KZG Commitments (v2)

Assume: powers of tau σ publicly known, no one knows τ.

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

KZG Commitments (v2)

Assume: powers of tau σ publicly known, no one knows τ.

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

KZG Commitments (v2)

Assume: powers of tau σ publicly known, no one knows τ.

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]

KZG Commitments (v2)

Assume: powers of tau σ publicly known, no one knows τ.

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

KZG Commitments (v2)

Assume: powers of tau σ publicly known, no one knows τ.

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

KZG Commitments (v2)

Assume: powers of tau σ publicly known, no one knows τ.

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = “witness polynomial” (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

- accept proof ⇔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$

KZG Commitments (v2)

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

- accept proof ⇔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$

Correctness: (i)

KZG Commitments (v2)

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = “witness polynomial” (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

- accept proof ⇔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $u = w(\tau)$

KZG Commitments (v2)

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = “witness polynomial” (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

- accept proof ⇔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $u = w(\tau)$ ➔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$ [no matter what τ is]

KZG Commitments (v2)

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = “witness polynomial” (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

- accept proof ⇔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $u = w(\tau)$ ➔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$ [no matter what τ is]

(ii) [intuition] if $f(z) \neq v$

KZG Commitments (v2)

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

- accept proof ⇔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $u = w(\tau)$ ➔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$ [no matter what τ is]

(ii) [intuition] if $f(z) \neq v$, would need to compute $\pi = g^{(f(\tau)-v)/(\tau-z)}$ knowing only σ

KZG Commitments (v2)

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

- accept proof ⇔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$

Correctness: (i) if $f(z) = v$, $f(x) - v = (x - z) \cdot w(x)$, and $u = w(\tau)$ ➔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$ [no matter what τ is]

(ii) [intuition] if $f(z) \neq v$, would need to compute $\pi = g^{(f(\tau)-v)/(\tau-z)}$ knowing only σ, which seems hard (dividing by τ in exponent)

Checking Multiplication in the Exponent

Issue: in verification, how to check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

Checking Multiplication in the Exponent

Issue: in verification, how to check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

- know $g^{f(\tau)}$ (commitment)

Checking Multiplication in the Exponent

Issue: in verification, how to check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

- know $g^{f(\tau)}$ (commitment), can compute $g^{f(\tau)-v}$

Checking Multiplication in the Exponent

Issue: in verification, how to check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

- know $g^{f(\tau)}$ (commitment), can compute $g^{f(\tau)-v}$
- know g^{τ} (from structured reference string)

Checking Multiplication in the Exponent

Issue: in verification, how to check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

- know $g^{f(\tau)}$ (commitment), can compute $g^{f(\tau)-v}$
- know g^{τ} (from structured reference string), can compute $g^{(\tau-z)}$

Checking Multiplication in the Exponent

Issue: in verification, how to check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

- know $g^{f(\tau)}$ (commitment), can compute $g^{f(\tau)-v}$
- know g^{τ} (from structured reference string), can compute $g^{(\tau-z)}$
- know $\pi = g^{u}$ (alleged proof)

Checking Multiplication in the Exponent

Issue: in verification, how to check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

- know $g^{f(\tau)}$ (commitment), can compute $g^{f(\tau)-v}$
- know g^{τ} (from structured reference string), can compute $g^{(\tau-z)}$
- know $\pi = g^{u}$ (alleged proof), but how to compute $g^{(\tau-z)\cdot u}$?

Checking Multiplication in the Exponent

Issue: in verification, how to check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

- know $g^{f(\tau)}$ (commitment), can compute $g^{f(\tau)-v}$
- know g^{τ} (from structured reference string), can compute $g^{(\tau-z)}$
- know $\pi = g^{u}$ (alleged proof), but how to compute $g^{(\tau-z)\cdot u}$?

Key insight: don't need to *compute* $g^{(\tau-z)\cdot u}$ from $g^{(\tau-z)}$ and g^{u} (hard by CDH), only to *verify* whether $g^{f(\tau)-v}$ is in fact what you'd get if you *could* "multiply in the exponent" starting from $g^{(\tau-z)}$, g^{u} !

Checking Multiplication in the Exponent

Issue: in verification, how check if $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$?

- know $g^{f(\tau)}$ (commitment), can compute $g^{f(\tau)-v}$
- know g^{τ} (from structured reference string), can compute $g^{(\tau-z)}$
- know $\pi = g^{u}$ (alleged proof), but how to compute $g^{(\tau-z)\cdot u}$?

Key insight: don't need to *compute* $g^{(\tau-z)\cdot u}$ from $g^{(\tau-z)}$ and g^{u} (hard by CDH), only to *verify* whether $g^{f(\tau)-v}$ is in fact what you'd get if you *could* "multiply in the exponent" starting from $g^{(\tau-z)}$, g^{u} !

Fact: there exist groups ("elliptic curve groups with pairings") in which can efficiently verify multiplication in the exponent.

- given input $x = g^{a}$, $y = g^{b}$, $z = g^{c}$, reports whether $c = a \cdot b$

KZG Commitments

[Kate/Zaverucha/Goldberg 2010]

Assume: powers of tau σ publicly known, no one knows τ.

Commitment to a polynomial f: $g^{f(\tau)}$ [given σ, easy to compute]

To prove that $f(z) = v$:

1. write $f(x) - v$ as $(x - z) \cdot w(x)$ [w = "witness polynomial" (Fact 2)]
2. proof = $g^{w(\tau)}$ [given σ, easy to compute]

To verify alleged proof $\pi = g^u$ that $f(z) = v$:

- accept proof ⇔ $g^{f(\tau)-v} = g^{(\tau-z)\cdot u}$
  - check with one pairing operation (w/inputs $g^{(\tau-z)}$, π, and $g^{f(\tau)-v}$)

Trusted Setup Ceremonies

Question: where does structured reference string σ come from?

Trusted Setup Ceremonies

Question: where does structured reference string σ come from?

Answer: use a "trusted setup ceremony."

Trusted Setup Ceremonies

Question: where does structured reference string σ come from?

Answer: use a “trusted setup ceremony.”

- N participants ($N \approx 141000$ in Ethereum’s KZG ceremony)

Trusted Setup Ceremonies

Question: where does structured reference string σ come from?

Answer: use a “trusted setup ceremony.”

- N participants (N ≈ 141000 in Ethereum’s KZG ceremony)
- participant #1 chooses τ_1, publishes $\sigma_1 = (g^{\tau_1}, g^{\tau_1^2}, \ldots\ g^{\tau_1^d})$
  - – can use pairings to check that published vector indeed of this form

Trusted Setup Ceremonies

Question: where does structured reference string σ come from?

Answer: use a "trusted setup ceremony."

- N participants (N ≈ 141000 in Ethereum's KZG ceremony)
- participant #1 chooses τ_1, publishes $\sigma_1 = (g^{\tau_1}, g^{\tau_1^2}, \ldots g^{\tau_1^d})$
  - can use pairings to check that published vector indeed of this form
- participant #2 chooses τ_2, publishes $\sigma_2 = (g^{(\tau_1\tau_2)}, g^{(\tau_1\tau_2)^2}, \ldots g^{(\tau_1\tau_2)^d})$ [note: possible knowing only σ_1]

Trusted Setup Ceremonies

Question: where does structured reference string σ come from?

Answer: use a “trusted setup ceremony.”

- N participants (N ≈ 141000 in Ethereum’s KZG ceremony)
- participant #1 chooses τ_1, publishes $\sigma_1 = (g^{\tau_1}, g^{\tau_1^2}, \ldots g^{\tau_1^d})$
  - can use pairings to check that published vector indeed of this form
- participant #2 chooses τ_2, publishes $\sigma_2 = (g^{(\tau_1\tau_2)}, g^{(\tau_1\tau_2)^2}, \ldots g^{(\tau_1\tau_2)^d})$ [note: possible knowing only σ_1]
- etc. [final $\tau = \tau_1 \cdot \tau_2 \cdot \cdots \cdot \tau_N$]

Trusted Setup Ceremonies

Question: where does structured reference string σ come from?

Answer: use a "trusted setup ceremony."

- N participants (N ≈ 141000 in Ethereum's KZG ceremony)
- participant #1 chooses τ_1, publishes $\sigma_1 = (g^{\tau_1}, g^{\tau_1^2}, \dots g^{\tau_1^d})$
  - can use pairings to check that published vector indeed of this form
- participant #2 chooses τ_2, publishes $\sigma_2 = (g^{(\tau_1\tau_2)}, g^{(\tau_1\tau_2)^2}, \dots g^{(\tau_1\tau_2)^d})$ [note: possible knowing only σ_1]
- etc. [final $\tau = \tau_1 \cdot \tau_2 \cdot \cdots \cdot \tau_N$]

Guarantee: ≥ 1 honest participant (i.e., chooses its τ_i randomly and deletes it forever) ➔ τ is effectively random and unknown!