Existing research highlights promising use cases for Kate-Zaverucha-Goldberg (KZG10) commitments in ETH2, particularly as a better accumulator of block data than Merkle trees Despite strong interest from the research community, however, there are few, if any, implementations of this commitment scheme which smart contract developers can use today.

In this post, I present [libkzg

](http://github.com/weijiekoh/libkzg), an implementation written in Typescript, and which contains a Solidity verifier. I also describe in detail the choice of trusted setup, and how libkzg

uses the EIP-197 precompiled contract for pairing checks to verify a single-point KZG proof.

libkzg

currently supports the following features:

1. The ability to convert an arbitrary list of values to the coefficients of a polynomial.
2. The ability to generate a KZG commitment function to said coefficients.
3. The ability to generate a proof (or witness) that the polynomial evaluates to a particular value at at given point (or index).
4. The ability to verify said proof in Typescript and the EVM.

It currently does not support proofs of evaluation at multiple points, but this feature may be built sometime after this post is published. Do note that multi-point proofs may not be feasible to implement in the EVM, and a follow-up post will discuss this.

Implementation details

Typescript functions

genCoefficients(values: BigInt[]): Coefficient[]

Performs polynomial interpolation on the given values and returns a new polynomial, which is represented as a list of coefficients. Note that the new polynomial's coefficients, not values

, should be fed into commit()

in order to commit to values

.

commit(coefficients: Coefficient[]): Commitment

Given a list of coefficients, this function should output their KZG commitment.

genProof(coefficients: Coefficient[], index: number): Proof

Generates a proof that the polynomial evaluates to coefficients[index]

at the x-value index

.

verify(c: Commitment, proof: Proof, index: number, value: BigInt): boolean

Returns true if the proof (that for the polynomial committed to, the evaluation at the given index equals the given value) is valid, and false otherwise.

The trusted setup

The KZG scheme requires a trusted setup. To abstract this away from developers, libkzg

comes packaged with values from the 46th contribution to thePerpetual Powers of Tau Ceremony (PPOT). As PPOT is a multi-party trusted setup, as long as just one participant had safely discarded their toxic waste during the ceremony, nobody can generate fake proofs.

The number of so-called powers of tau from this ceremony is the maximum number of coefficients this library supports. For the sake of portability, libkzg

only supports up to 65536 coefficients. That said, anyone can choose any existing contribution to PPOT and modify libkzg

to support up to a maximum of 2 ^ {28}

coefficients (although this would require an unwieldly amount disk space in the range of dozens of gigabytes for off-chain use, and be limited by gas costs for deploying te verifier contract on-chain).

Verification in the EVM

libkzg

provides a Solidity contract that can verify single-point proofs. The contract code is here. Each invocation consumes about 178k gas. Its function signature is:

function verifyKZG(uint256 _commitmentX, uint256 _commitmentY, uint256 _proofX, uint256 _proofY, uint256 _index, uint256 _value) public view returns (bool)

The verifier uses the EIP-197 precompiled contract to perform a pairing check. According to the KZG10 paper, this pairing check must pass in order for the proof to be valid. It states:

[

1604×466

](https://hackmd.io/_uploads/rJn8bFWJD.png)

It is crucial that this equation is converted into a form that can be efficiently computed on-chain. On the one hand, the EIP-197 precompiled contract accepts a list of G1 and G2 group elements (a_1, b_1 ... a_n, b_n)

, and checks if they fulfill this equation:

e(a_1, b_1) \cdot \ldots \cdot e(a_n, b_n) = 1

On the other hand, all on-chain operations cost gas, so it is important to minimise the number of a_n, b_n

elements, as well as group operations on said elements, particularly curve point addition and multiplication. Furthermore, we want to avoid performing these operations on G2 group elements entirely. Unlike curve operations in G1, which can be done using EIP-196 precompiled contracts, those for G2 have to be implemented in Solidity, and this increases code complexity and steepens gas costs.

As such, to achieve gas-efficiency and simplicity, we have to perform some algebra on the above equations. To begin this process, here is a legend to its mathematical symbols:

Symbol

Description

e

The pairing function.

w_i

or g^{\psi_i(\alpha)}

The proof, which is also associated with the _index

.

g

The generator of either G1 or G2. In the expression e(g, P)

, g

is the generator of G1. In e(P, g)

, g

is the generator of G2.

g^\alpha

The generator g

(either G1 or G2) raised to the first power of tau. In other words, this is the second

value from the trusted setup's TauG1 or TauG2 outputs. (The first is simply g^{\alpha ^ 0} = g^{1} = g

.).

g^{\alpha(x)}

or g^\alpha

The KZG commitment to the polynomial with coefficients [0, 1]

which is (1)x^1 - (0)x^0 = x

g^{i}

or g ^{1} \cdot i

The KZG commitment to the polynomial with coefficients [_index]

which is ix^0 = i

\phi(x)

The committed polynomial.

\phi(i)

This is _value

, which is the evaluation of the committed polynomial at the x-value _index

.

\psi_i(\alpha)

The polynomial \frac{\phi(x) - \phi(i)}{x-i}

, also known as the quotient polynomial.

Next, recall some fundamental properties of elliptic curve pairings:

[a]

: e(P, Q)^{x} = e(x \cdot P, Q) = e(P, x\cdot Q)

[b]

: e(x \cdot P, y \cdot Q) = e(y \cdot P, x \cdot Q)

[c]

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

[d]

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

Now, simplify this equation from the original paper:

e(g^{\psi_i(\alpha)}, g^{\alpha - i}) \cdot e(g, g)^{\phi(i)} = e(C, g)

Subsitute C

with g^{\phi(x)}

and flip sides:

e(g^{\phi(x)}, g) = e(g^{\psi_i(\alpha)}, g^{\alpha - i}) \cdot e(g, g)^{\phi(i)}

Apply [a] to shift \phi(i)

:

$$e(g^{\phi(x)}, g) = e(g^{\psi_i(\alpha)}, g^{\alpha - i}) \cdot e(\phi(i) \cdot g, g)$$

Multiply both sides by $e(\phi(i) \cdot g, g)^{-1}$

:

$$e(g^{\phi(x)}, g) \cdot e(\phi(i) \cdot g, g)^{-1} = e(g^{\psi_i(\alpha)}, g^{\alpha - i})$$

Apply [a] to shift -1

:

$$e(g^{\phi(x)}, g) \cdot e(-\phi(i) \cdot g, g) = e(g^{\psi_i(\alpha)}, g^{\alpha - i})$$

Apply [c] to reduce the number of pairings from 3 to 2:

$$e(g^{\phi(x)}-g \cdot \phi(i), g) = e(g^{\psi_i(\alpha)}, g^{\alpha - i})$$

Substitute $g^{\alpha - i}$

with $g^\alpha \cdot g^{-i}$

:

According to the original paper:

$$g^{\alpha - i} = \frac{g^\alpha}{g^i}$$

So, $g^{\alpha - i}$

is the generator to the power of $\alpha -i$

, and can also be expressed as $g^\alpha \cdot g^{-i}$

.

$$e(g^{\phi(x)}-g \cdot \phi(i), g) = e(g^{\psi_i(\alpha)}, g^\alpha \cdot g^{-i})$$

Apply [a] to shift $\alpha - i$

:

$$e(g^{\phi(x)}-g \cdot \phi(i), g) = e(g^{\psi_i(\alpha)}, g)^{\alpha -i}$$

$$e(g^{\phi(x)}-g \cdot \phi(i), g) = e(g^{\psi_i(\alpha)}, g)^{\alpha} \cdot e(g^{\psi_i(\alpha)}, g)^{-i}$$

$$e(g^{\phi(x)}-g \cdot \phi(i), g) = e(\alpha \cdot g^{\psi_i(\alpha)}, g) \cdot e(-i \cdot g^{\psi_i(\alpha)}, g)$$

$$e(g^{\phi(x)}-g \cdot \phi(i), g) \cdot e(-i \cdot g^{\psi_i(\alpha)}, g)^{-1} = e(\alpha \cdot g^{\psi_i(\alpha)}, g)$$

$$e(g^{\phi(x)}-g \cdot \phi(i), g) \cdot e(i \cdot g^{\psi_i(\alpha)}, g) = e(\alpha \cdot g^{\psi_i(\alpha)}, g)$$

Apply [c] to reduce the number of pairings from 3 to 2:

$$e((g^{\phi(x)}-g \cdot \phi(i)) + (i \cdot g^{\psi_i(\alpha)}), g) = e(\alpha \cdot g^{\psi_i(\alpha)}, g)$$

Divide both sides by $e(\alpha \cdot g^{\psi_i(\alpha)}, g)$

to make it EIP-197 friendly:

$$e((g^{\phi(x)}-g \cdot \phi(i)) + (i \cdot g^{\psi_i(\alpha)}), g) \cdot e(\alpha \cdot g^{\psi_i(\alpha)}, g)^{-1} = 1$$

Apply [a] to shift -1

:

$$e((g^{\phi(x)}-g \cdot \phi(i)) + (i \cdot g^{\psi_i(\alpha)}), g) \cdot e(-\alpha \cdot g^{\psi_i(\alpha)}, g) = 1$$

Apply [a] to remove one G1 multiplication step:

$$e((g^{\phi(x)}-g \cdot \phi(i)) + (i \cdot g^{\psi_i(\alpha)}), g) \cdot e(- g^{\psi_i(\alpha)}, \alpha \cdot g) = 1$$

As a result, the expensive operations that the verifier contract needs to perform are:

- one G1 curve point addition.
- one G1 curve point scalar multiplication.
- one pairing check (involving 2 pairings).

The only elements from the trusted setup it must store on-chain are:

- The generator of G1.
- The generator of G2.
- \alpha \cdot g

of G2, which is the generator of G2 raised to the first power of tau from the contribution file.

As such, with EIP-1108 already on mainnet, the verifyKZG()

contract function only consumes 178k gas.

Credits

Many thanks to @barryWhiteHat, @kobigurk, @liangcc, @yingtong, and @lsankar4033 for their feedback.

Resources

Constant-Size Commitments to Polynomials and Their Applications by Kate, Zaverucha, and Goldberg.

Kate polynomial commitments by @dankrad.

Kate-Zaverucha-Goldberg (KZG) Constant-Sized Polynomial Commitments by Alin Tomescu.