Fast amortized KZG proofs

Dankrad Feist*1 and Dmitry Khovratovich†2

1,2Ethereum Foundation

January 11, 2023

Abstract

In this note we explain how to compute \( n \) KZG proofs for a polynomial of degree \( d \) in time superlinear of \((t + d)\). Our technique is used in lookup arguments and vector commitment schemes.

1 Preliminaries

1.1 Setup

Let \( F \) be a field and let \( G \) be a group with a designated element \( g \), called a generator. We denote \([a] = a \cdot g\) for integer \( a \).

1.2 KZG Commitment Scheme

Setup. In a KZG commitment scheme [KZG10] for polynomials of degree up to \( d \), a Verifier or a trusted third party first selects a secret \( s \) and then constructs \( d \) elements of \( G \): \([s], [s^2], \ldots, [s^m]\).

Commitment. Let \( f(X) = \sum_{0 \leq i \leq d} f_i X^i \in \mathbb{F}[X] \) be a polynomial of degree \( d \). Then a commitment \( C_f \in G \) is defined as

\[
C_f = \sum_{0 \leq i \leq d} f_i [s^i],
\]

being effectively the evaluation of \( f \) at point \( s \) multiplied by \( g \).

Proof. Note that for any \( y \) we have that \((X - y) \) divides \( f(X) - f(y) \). Then the proof that \( f(y) = z \) is defined as

\[
\pi[f(y) = z] = C_{T_y},
\]

where \( T_y(X) = \frac{f(X) - z}{X - y} \) is a polynomial of degree \((d - 1)\).

Note that a proof can be constructed using \( d \) scalar multiplications in the group. The coefficients of \( T \) are computed with one multiplication each:

\[
T_y(X) = \sum_{0 \leq i \leq d-1} t_i X^i;
\]

\[
t_{d-1} = f_d;
\]

\[
t_j = f_{j+1} + y \cdot t_{j+1}.
\]

Expanding on the last equation, we get

\[
T_y(X) = f_d X^{d-1} + (f_{d-1} + yf_d) X^{d-2} + (f_{d-2} + yf_{d-1} + y^2f_d) X^{d-3} + \cdots + (f_1 + yf_2 + y^2f_3 + \cdots + y^{d-1}f_d).
\]
1.3 Discrete Fourier Transform

Let \( n \) be a positive integer. Then \( \omega \in \mathbb{F} \) is called \( n \)-th root of unity if \( \omega^n = 1 \) and \( \omega^i \neq 1 \) for \( i < n \).

Discrete Fourier Transform for vectors in \( \mathbb{F}^n \) is defined as

\[
\text{DFT}_n(a_0, a_1, \ldots, a_{n-1}) = (b_0, b_1, \ldots, b_{n-1})
\]

where

\[
b_i = \sum_{0 \leq j \leq n-1} a_j \omega^{ij}.
\]

It is easy to see that \( b_i \) are essentially evaluations of polynomial \( a(X) = \sum_j a_j X^j \) in points \( \omega^0, \omega^1, \ldots, \omega^{n-1} \). As a polynomial of degree \( n - 1 \) is defined by its values in \( n \) points, DFT is invertible. We denote its inverse by \( \text{IDFT}_n \).

In a vast majority of finite fields with characteristic bigger than \( n \), the DFT can be computed in \( O(n \log n) \) time with an algorithm called FFT (Fast Fourier Transform) \cite{CT65}. An overview of such methods can be found in \cite{DV90}.

2 Multiple KZG proofs

In this section we derive our main result.

**Theorem 1.** Let \( \{[s^i]\} \) be KZG setup of size at least \( d \), and \( f_i \) be the coefficients of polynomial \( f(X) \) of degree \( d \). Let \( \{\xi_i\}_{1 \leq i \leq n} \subset \mathbb{F} \) be field elements, and suppose that FFT with complexity \( n \log n \) is available for \( n \)-sized vectors. Then KZG proofs for evaluating \( f \) at \( \{\xi_i\} \) can be obtained

- In \( O((n + d) \log(n + d)) \) group operations (scalar multiplications) if \( \{\xi_i\} \) are \( n \)-th roots of unity.
- In \( O(n \log^2 n + d \log d) \) group operations in other cases.

2.1 Formula for multiple proofs

Let \( \xi_1, \xi_2, \ldots, \xi_n \) be field elements and let \( f(\xi_k) = z_k \). We show how to construct KZG proofs for all these \((\xi_k, z_k)\) pairs simultaneously.

**Proposition 1.** Let \( \{[s^i]\} \) be KZG setup of size at least \( d \), and \( f_i \) be the coefficients of polynomial \( f(X) \) of degree \( d \). Let \( \{\xi_i\} \subset \mathbb{F} \) be field elements. Then KZG proofs for evaluating \( f \) at \( \{\xi_i\} \) are evaluations of polynomial

\[
h(X) = h_1 + h_2X + \ldots + h_dX^{d-1}.
\]

where

\[
h_i = (f_d[s^{d-1}] + f_{d-1}[s^{d-2}] + f_{d-2}[s^{d-3}] + \cdots + f_{i+1}[s] + f_i).
\]

**Proof.** Note that a proof for \( \xi_k \) is

\[
\pi[f(\xi_k) = z_k] = C_{T_{\xi_k}} = f_d[s^{d-1}] + (f_{d-1} + \xi_k f_d)[s^{d-2}] + (f_{d-2} + \xi_k f_{d-1} + \xi_k^2 f_d)[s^{d-3}] + \cdots + f_{i+1}[s] + f_i + 1)
\]

Regrouping the terms, we get:

\[
C_{T_{\xi_k}} = (f_d[s^{d-1}] + f_{d-1}[s^{d-2}] + f_{d-2}[s^{d-3}] + \cdots + f_2[s] + f_1) +
\]

\[
+ (f_d[s^{d-2}] + f_{d-1}[s^{d-3}] + f_{d-2}[s^{d-4}] + f_3[s] + f_2) \xi_k +
\]

\[
+ (f_d[s^{d-3}] + f_{d-1}[s^{d-4}] + f_{d-2}[s^{d-5}] + f_4[s] + f_3) \xi_k^2 +
\]

\[
+ (f_d[s^{d-4}] + f_{d-1}[s^{d-5}] + f_{d-2}[s^{d-6}] + f_5[s] + f_4) \xi_k^3 +
\]

\[
\cdots
\]

\[
+ (f_d[s] + f_{d-1}) \xi_k^{d-2} + f_d \xi_k^{d-1}.
\]

Let for \( 1 \leq i \leq d \) denote

\[
h_i = (f_d[s^{d-1}] + f_{d-1}[s^{d-2}] + f_{d-2}[s^{d-3}] + \cdots + f_{i+1}[s] + f_i).
\]
Then
\[ C_{T_k} = h_1 + h_2\xi_k + h_3\xi_k^2 + \cdots + h_d\xi_k^{d-1}. \] (13)

Let us denote
\[ C_T = [C_{T_{\xi_1}}, C_{T_{\xi_2}}, \ldots, C_{T_{\xi_n}}]. \]
Therefore, \( C_T \) is the evaluation of \( h(X) = \sum_{0 \leq i \leq d-1} h_{i+1}X^i \) at points \( \xi_1, \xi_2, \ldots, \xi_n \). □

### 2.2 Computing \( h \)

Now we demonstrate that \( h \) can be also computed efficiently from \( \{f_i\} \).

**Proposition 2.** The coefficients \( h_i \) can be computed in \( O(d \log d) \) time if FFT is available for vectors of size \( d \).

**Proof.** Indeed, by definition
\[
\begin{bmatrix}
  h_1 \\
  h_2 \\
  h_3 \\
  \vdots \\
  h_{d-1} \\
  h_d
\end{bmatrix} =
\begin{bmatrix}
  f_{d} & f_{d-1} & f_{d-2} & f_{d-3} & \cdots & f_1 \\
  0 & f_{d} & f_{d-1} & f_{d-2} & \cdots & f_2 \\
  0 & 0 & f_{d} & f_{d-1} & \cdots & f_3 \\
  \vdots & & & & & \ddots \\
  0 & 0 & 0 & 0 & \cdots & f_{d-1} \\
  0 & 0 & 0 & 0 & \cdots & f_d
\end{bmatrix}
\begin{bmatrix}
  [s^{d-1}] \\
  [s^{d-2}] \\
  [s^{d-3}] \\
  \vdots \\
  [s] \\
  [1]
\end{bmatrix}
\]

The matrix
\[
A =
\begin{bmatrix}
  f_{d} & f_{d-1} & f_{d-2} & f_{d-3} & \cdots & f_1 \\
  0 & f_{d} & f_{d-1} & f_{d-2} & \cdots & f_2 \\
  0 & 0 & f_{d} & f_{d-1} & \cdots & f_3 \\
  \vdots & & & & & \ddots \\
  0 & 0 & 0 & 0 & \cdots & f_{d-1} \\
  0 & 0 & 0 & 0 & \cdots & f_d
\end{bmatrix}
\]
is a **Toeplitz** matrix. It is known that a multiplication of a vector by a \( d \times d \) Toeplitz matrix costs \( O(d \log d) \) operations for FFT-friendly fields (see Section 3 for derivation). Let \( \nu \) be the 2\( d \)-th root of unity. Then the algorithm is as follows:

1. Compute
   \[
   y = \text{DFT}_{2d}(\hat{s}) \quad \text{where} \quad \hat{s} = ([s^{d-1}], [s^{d-2}], [s^{d-3}], \ldots, [s], [1], [0], [0], \ldots, [0])
   \]
   \( d \) neutral elements

2. Compute
   \[
   v = \text{DFT}_{2d}(\hat{c}) \quad \text{where} \quad \hat{c} = (f_{d}, f_{d-1}, \ldots, f_1, 0, 0, \ldots, 0)
   \]

3. Compute
   \[
   u = y \circ v \circ (1, \nu, \nu^2, \ldots, \nu^{2d-1})
   \]

4. Compute
   \[
   \hat{h} = i\text{DFT}_{2d}(u)
   \]

5. Take first \( d \) elements of \( \hat{h} \) as \( h \).

Therefore, we can compute \( h \) from the KZG setup using \( O(d \log d) \) scalar multiplications in \( \mathbb{G} \). □

### 2.3 Proof of Theorem 1

Now we can prove the statement of Theorem 1. It remains to show the complexity of evaluating \( h(X) \) in \( \{\xi_i\} \).

\( \{\xi_i\} \) are \( n \)-th roots of unity. When evaluation points are \( n \)-th roots of unity, the polynomial \( h(X) \) can be evaluated in \( n \log n \) time using FFT.
\{\xi_i\} are arbitrary values. In this case we adapt the generic fast evaluation algorithm [vzGG13, Algorithm 10.4], which is known to have complexity $O(n \log^2 n)$ whenever FFT for $n$-sized vectors is available. For the sake of completeness we provide a full description of the algorithm in Section A.

3 Circulant and Toeplitz matrix-vector product computation

3.1 Circulant multiplication

A matrix-vector product with a circulant matrix $B$ and vector $c$ is equivalent to polynomial multiplication. Concretely, let

$$b(X) = \sum_i b_i X^i, \quad c(X) = \sum_i c_i X^i, \quad a(X) = \sum_i a_i X^i$$

Then $a_i = \sum_{j+k=i-1} (\text{mod } n) b_j c_k$ and so

$$a(X) \equiv X \cdot b(X) \cdot c(X) \quad (\text{mod } X^n - 1) \quad (14)$$

Denote the $n$-th root of unity by $\omega$, then $a(\omega^i) = \omega^i \cdot b(\omega^i) \cdot c(\omega^i)$ since $\omega^n = 1$. We know that all $b(\omega^i), c(\omega^i)$ can be computed in $n \log n$ time using FFT. Therefore we have the following algorithm for $a$:

1. Compute $\hat{b} = \text{DFT}_n(b_0, b_1, b_2, \ldots, b_{n-1})$.
2. Compute $\hat{c} = \text{DFT}_n(c_0, c_1, c_2, \ldots, c_{n-1})$.
3. Compute $\tilde{a} = \hat{b} \circ \hat{c} \circ (1, \omega, \omega^2, \ldots, \omega^{n-1})$.
4. Compute $a = \text{iDFT}_n(\tilde{a})$.

3.2 Toeplitz multiplication

A matrix-vector product with a Toeplitz matrix $F$ and vector $c$ is reduced to the circulant case by padding the matrix $F$ to size $2n \times 2n$ and vector $c$ accordingly:

$$F' = \begin{bmatrix}
    f_{n-1} & f_{n-2} & f_{n-3} & f_{n-4} & \cdots & f_0 & 0 & 0 & \cdots & 0 \\
    0 & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & f_1 & f_0 & 0 & \cdots & 0 \\
    0 & 0 & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & f_2 & f_1 & f_0 & \cdots & 0 \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
    0 & 0 & 0 & 0 & \cdots & f_{n-2} & f_{n-3} & f_{n-4} & \cdots & 0 \\
    0 & 0 & 0 & 0 & \cdots & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & 0 \\
    0 & 0 & 0 & 0 & \cdots & 0 & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & 0 \\
    f_0 & 0 & 0 & 0 & \cdots & 0 & 0 & f_{n-1} & \cdots & f_1 \\
    f_1 & f_0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & f_2 \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
    f_{n-2} & f_{n-3} & f_{n-4} & f_{n-5} & \cdots & 0 & 0 & 0 & \cdots & f_{n-1}
\end{bmatrix} \quad \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$
As a result the product of $F'$ and $c'$ has all the elements of $a$:

$$F' \cdot c' = a' = \begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-2} \\
a_{n-1} \\
a_n \\
\vdots \\
a_{2n-1}
\end{bmatrix}$$

Therefore, to compute $F \cdot c$ we compute $F' \cdot c'$ using DFT and then select the top $n$ elements of the resulting vector.

4 Applications

Our technique is useful whenever a large number of KZG openings is required by a protocol. Examples are

- **Lookup arguments.** When a table is encoded as polynomial evaluations over roots of unity, the $O(n \log n)$ version of Theorem 1 applies [ZBK+22, ZGK+22, EFG22]. In contrast, when a table is encoded as the set of roots of a polynomial, then individual proofs are no longer at roots of unity and so require the $O(n \log^2 n)$ version of Theorem [GK22].

- **Vector commitment schemes based on KZG.** Preparing many (or all) proofs is done with our technique [WUP22, Tom20]. Another application is speeding up the trusted setup phase [TAB+20].

References


A Fast evaluation algorithm

This section is a straightforward adaptation of fast polynomial algorithms from [vzGG13] to the case where the argument is a group element.

A.1 Fast evaluation algorithm

Input: $F \in \mathbb{F}^d[X]$, $A = (a_1, a_2, \ldots, a_d) \in G$.
Output: $C = (c_1, c_2, \ldots, c_d) \in G$ such that $f(a_i) = c_i$ for all $i$.

Construction.

• If $d = 1$ compute $F(a_1)$ in constant time and return.
• Else split $A$ into $A_1$ and $A_2$.
• Let $g_1(X) = \prod_{a \in A_1} (X - a)$ be vanishing poly of degree $d/2$ for $A_1$, and $g_2(X)$ be vanishing poly of degree $d/2$ for $A_2$.
• Compute $f_1(X) = F(X) \mod g_1(X)$ and $f_2(X) = F(X) \mod g_2(X)$ of degree $d/2$ using fast division algorithm (Section A.2).
• Evaluate $f_1$ on $A_1$ and get $C_1$ recursively (go to step 1). Evaluate $f_2$ on $A_2$ and get $C_2$. Return $C_1 \cup C_2$.

Complexity. The algorithm is divide-and-conquer. At the combination step we apply the fast division algorithm of complexity $O(d \log d)$. The cost of computing all vanishing polynomials is $d \log^2 d$ (see below). Thus for the complexity $C(d)$ of the evaluation algorithm without it we have an equation

$$C(d) = d \log d + 2C(d/2)$$

Thus the total complexity is $O(d \log^2 d)$.

Constructing all vanishing polys We construct all vanishing polynomials in the monomial form from low degree to high degree. To compute a vanishing poly of degree $r$, we multiply two vanishing polys of degree $r/2$ using fast multiplication algorithm. The complexity of the combination step is $r \log r$ so we have for the complexity $V(r)$ an equation:

$$V(r) = r \log r + 2V(r/2)$$

This yields total complexity of $r \log^2 r$.

A.2 Fast division algorithm

Input: $f \in \mathbb{F}^n[X]$, $g \in \mathbb{F}^m[X]$.
Output: $q \in \mathbb{F}^{n-m}[X]$, $r \in \mathbb{F}^{m-1}[X]$ such that

$$f(X) = q(X)g(X) + r(X)$$
Idea  For \( f(X) = f_0 + f_1 X + \cdots + f_n X^n \) define

\[
\text{rev}(f) = f_d + f_{n-1} X + \cdots + f_0 X^n
\]

Note that

\[
x^n f(1/x) = x^{n-m} q(1/x) x^m g(1/x) + x^{n-m+1} x^{m-1} r(1/x).
\]

In terms of reverses:

\[
\text{rev}(f) = \text{rev}(q) \cdot \text{rev}(g) + x^{n-m+1} \text{rev}(r).
\]

Then

\[
\text{rev}(f) \equiv \text{rev}(q) \cdot \text{rev}(g) \pmod{x^{n-m+1}}.
\]

And

\[
\text{rev}(q) \equiv \text{rev}(f) \cdot \text{rev}(g)^{-1} \pmod{x^{n-m+1}}.
\]

Construction

1. Compute \( \text{rev}(f), \text{rev}(g) \).
2. Compute \( \text{rev}(g)^{-1} \pmod{x^{n-m+1}} \) using fast inversion algorithm (section A.3).
3. Find \( \text{rev}(q) \), then \( q \) and \( r \) using fast polynomial multiplication.

Complexity  Both fast inversion algorithm and fast multiplication algorithm have complexity \( O(d \log d) \) (see below) so the total complexity is \( O(d \log d) \).

A.3 Fast Inversion Algorithm

Input: \( f \in \mathbb{F}[X], \ l. \)
Output: \( g \in \mathbb{F}[X] \) such that

\[
f(X) g(X) \equiv 1 \pmod{X^l}
\]

Idea  We find a "root" of an equation \( \frac{1}{g} - f = 0 \) using Newton iteration for \( \phi(g) = 0 \):

\[
g_{i+1} = g_i - \frac{\phi(g_i)}{\phi'(g_i)}
\]

which in our case is

\[
g_{i+1} = g_i - \frac{1/g_i - f}{-1/g_i^2} = 2 g_i - f g_i^2
\]

Construction

1. Initialize \( g_0 = \frac{1}{f(0)} \).
2. Compute for \( i \) up to \( \log l \):

\[
g_{i+1} = (2g_i - f g_i^2) \mod x^{2i+1}
\]
3. Return \( g_{\log l+1} \).

Complexity  At each step we do 3 fast polynomial multiplications of degree \( 2^i \). Using that

\[
\sum_{1 \leq i \leq r} c \cdot 2^i \cdot i \leq 2c r 2^r
\]

the total cost is still \( O(d \log d) \) as reduction modulo \( x^{2i+1} \) is easy.
A.4 Fast multiplication Algorithm

We multiply 2 polynomials of degree $d$ in $O(d \log d)$ time using FFT:

1. Compute $2d$-FFT of both polys. Note that we do not evaluate the polynomials at a group element here, but rather remain in the field $\mathbb{F}$.
2. Multiply pairwise.
3. Compute inverse FFT.