Fast amortized KZG proofs

Dankrad Feist*[1] and Dmitry Khovratovich†[2]

[1,2]Ethereum Foundation

May 15, 2025

Abstract

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

1 Preliminaries

1.1 Setup

Let $\mathbb{F}$ be a field and let $\mathbb{G}$ 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 $\mathbb{G}$:

$$[s], [s^2], \ldots, [s^m].$$

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

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

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

Single-point 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 $(d - 1)$. To verify the proof π we check the equation

$$e(\pi, [s - y]_2) = e(C_f - [z]_1, [1]_2) \tag{1}$$

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

$$T_y(X) = \sum_{0 \le i \le d-1} t_i X^i; \tag{2}$$

$$t_{d-1} = f_d; \tag{3}$$

$$t_j = f_{j+1} + y \cdot t_{j+1}. \tag{4}$$

*dankrad.feist@ethereum.org

†khovratovich@gmail.com

Expanding on the last equation, we get

$$T_y(X) = f_d X^{d-1} + (f_{d-1} + yf_d)X^{d-2} + (f_{d-2} + yf_{d-1} + y^2 f_d)X^{d-3} + \\ + (f_{d-3} + yf_{d-2} + y^2 f_{d-1} + y^3)X^{d-4} + \cdots + (f_1 + yf_2 + y^2 f_3 + \cdots + y^{d-1} f_d). \quad (5)$$

Multi-point proof. Let $I = \{y_1, y_2, \ldots, y_l\}$ be a set of elements of $\mathbb{F}$. Let $Z_I(X)$ be its vanishing polynomial i.e. $Z_I = \prod_{y \in I}(X - y)$. We write

$$f(X) = T_I(X)Z_I(X) + R_I(X)$$

where T_I has degree $(d - l)$ and R_I has degree $l - 1$. We see that f and R_I take the same values on I. Then the proof that $f(y_i) = z_i$ for all i is defined as

$$\pi[\{f(y) = z\}_{y \in I}] = C_{T_I},$$

To verify the proof π we check the equation

$$e(\pi, [Z_I]_2) = e(C_f - C_{R_I}, [1]_2) \quad (6)$$

Let us consider the special case where $Z_I(X)$ has special form $X^l - \theta$ i.e. where I is the set of l-roots of θ. Then the coefficients of the quotient polynomial $T_I(X)$ can be computed by a compact formula of θ and the coefficients of $f(X)$. This is done as follows:

$$T_I(X) = \sum_{0 \le k \le d-l} t_k X^k; \quad (7)$$

$$t_{d-l} = f_d; \quad (8)$$

$$\cdots \quad (9)$$

$$t_{d-2l+1} = f_{d-l+1} \quad (10)$$

$$t_k = f_{k+l} + \theta \cdot t_{k+l}, \quad 0 \le k \le d - 2l. \quad (11)$$

Expanding on the last equation, we get

$$T_I(X) = f_d X^{d-l} + f_{d-1} X^{d-l+1} + \ldots + f_{d-l+1} X^{d-2l+1} + \\ (f_{d-l} + \theta f_d)X^{d-2l} + (f_{d-l-1} + \theta f_{d-1})X^{d-2l-1} + \cdots + (f_{d-2l+1} + \theta f_{d-l+1})X^{d-3l+1} + \\ (f_{d-2l} + \theta f_{d-l} + \theta^2 f_d)X^{d-3l} + (f_{d-2l-1} + \theta f_{d-l-1} + \theta^2 f_{d-1})X^{d-3l-1} + \cdots + \\ (f_{d-3l+1} + \theta f_{d-2l+1} + \theta^2 f_{d-l+1})X^{d-4l+1} + \cdots + (f_l + \theta f_{2l} + \theta^2 f_{3l} + \cdots + \theta^{r-2} f_{(r-1)l}) \cdot 1. \quad (12)$$

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 $i < n$.

Dicrete Fourier Transform for vectors in $\mathbb{F}^n$ is defined as

$$\mathrm{DFT}_n(a_0, a_1, \ldots, a_{n-1}) = (b_0, b_1, \ldots, b_{n-1})$$

where

$$b_i = \sum_{0 \le j \le n-1} a_j \omega^{ij}.$$

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 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) [CT65]. An overview of such methods can be found in [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 $\{\xi_i\}_{1\le i\le n} \subset \mathbb{F}$ 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*[1]*.*

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 (ξ_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 $\Xi = \{\xi_i\} \subset \mathbb{F}$ be field elements. Then KZG proofs for evaluating f at $\{\xi_i\}$ are evaluations of polynomial $h(X) \in \mathbb{G}^{d-1}[X]$ at Ξ with*

$$h(X) = h_1 + h_2 X + \ldots + h_d X^{d-1}. \tag{13}$$

where

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

Proof. Note that a proof for ξ_k is

$$\begin{aligned}\pi[f(\xi_k) = z_k] = C_{T_{\xi_k}} = f_d[s^{d-1}] + (f_{d-1} + \xi_k f_d)[s^{d-2}] + (f_{d-2} + \xi_k f_{d-1} + \xi_k^2 f_d)[s^{d-3}] + \\ + (f_{d-3} + \xi_k f_{d-2} + \xi_k^2 f_{d-1} + \xi_k^3)[s^{d-4}] + \cdots + (f_1 + \xi_k f_2 + \xi_k^2 f_3 + \cdots + \xi_k^{(d-1)} f_d).\end{aligned} \tag{14}$$

Regrouping the terms, we get:

$$C_{T_{\xi_k}} = \left(f_d[s^{d-1}] + f_{d-1}[s^{d-2}] + f_{d-2}[s^{d-3}] + \cdots + f_2[s] + f_1\right) + \tag{15}$$

$$+ \left(f_d[s^{d-2}] + f_{d-1}[s^{d-3}] + f_{d-2}[s^{d-4}] + \cdots + f_3[s] + f_2\right)\xi_k + \tag{16}$$

$$+ \left(f_d[s^{d-3}] + f_{d-1}[s^{d-4}] + f_{d-2}[s^{d-5}] + \cdots + f_4[s] + f_3\right)\xi_k^2 + \tag{17}$$

$$+ \left(f_d[s^{d-4}] + f_{d-1}[s^{d-5}] + f_{d-2}[s^{d-6}] + \cdots + f_5[s] + f_4\right)\xi_k^3 + \tag{18}$$

$$\cdots \tag{19}$$

$$+ (f_d[s] + f_{d-1})\xi_k^{d-2} + f_d\xi_k^{d-1}. \tag{20}$$

Let for $1 \le i \le d$ denote

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

Then

$$C_{T_{\xi_k}} = h_1 + h_2\xi_k + h_3\xi_k^2 + \cdots + h_d\xi_k^{d-1}. \tag{21}$$

Let us denote

$$\mathbf{C}_T = [C_{T_{\xi_1}}, C_{T_{\xi_2}}, \ldots, C_{T_{\xi_n}}]$$

Therefore, $\mathbf{C}_T$ is the evaluation of $h(X) = \sum_{0\le i\le d-1} h_{i+1}X^i$ at points $\xi_1, \xi_2, \ldots, \xi_n$. □

[1] A similar statement was also obtained in [GK22]

2.2 Computing h

Now we demonstrate that $\mathbf{h}$ 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

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_{d-1} \\ h_d \end{bmatrix} = \begin{bmatrix} f_d & f_{d-1} & f_{d-2} & f_{d-3} & \cdots & f_1 \\ 0 & f_d & f_{d-1} & f_{d-2} & \cdots & f_2 \\ 0 & 0 & f_d & f_{d-1} & \cdots & f_3 \\ & & \ddots & & & \\ 0 & 0 & 0 & 0 & \cdots & f_{d-1} \\ 0 & 0 & 0 & 0 & \cdots & f_d \end{bmatrix} \cdot \begin{bmatrix} [s^{d-1}] \\ [s^{d-2}] \\ [s^{d-3}] \\ \vdots \\ [s] \\ [1] \end{bmatrix} \tag{22}$$

The matrix

$$A = \begin{bmatrix} f_d & f_{d-1} & f_{d-2} & f_{d-3} & \cdots & f_1 \\ 0 & f_d & f_{d-1} & f_{d-2} & \cdots & f_2 \\ 0 & 0 & f_d & f_{d-1} & \cdots & f_3 \\ & & \cdots & & & \\ 0 & 0 & 0 & 0 & \cdots & f_{d-1} \\ 0 & 0 & 0 & 0 & \cdots & f_d \end{bmatrix}$$

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 4 for derivation). Let ν be the $2d$-th root of unity. Then the algorithm is as follows:

1. Compute
$$\mathbf{y} = \mathrm{DFT}_{2d}(\widehat{\mathbf{s}}) \quad \text{where} \quad \widehat{\mathbf{s}} = ([s^{d-1}], [s^{d-2}], [s^{d-3}], \cdots, [s], [1], \underbrace{[0], [0], \ldots, [0]}_{d \text{ neutral elements}})$$
2. Compute[2]
$$\mathbf{v} = \mathrm{DFT}_{2d}(\widehat{\mathbf{c}}) \quad \text{where} \quad \widehat{\mathbf{c}} = (f_d, \underbrace{0, 0, \ldots, 0}_{d \text{ zeroes}}, f_1, f_2, \ldots, f_{d-1})$$
3. Compute
$$\mathbf{u} = \mathbf{y} \circ \mathbf{v}.$$
4. Compute
$$\widehat{\mathbf{h}} = \mathrm{iDFT}_{2d}(\mathbf{u})$$
5. Take first d elements of $\widehat{\mathbf{h}}$ as $\mathbf{h}$.

Therefore, we can compute $\mathbf{h}$ 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.

[2] A previous version of this note had an incorrect form for $\widehat{\mathbf{c}}$. We thank a reviewer for pointing it out.

3 Computing multiproofs

Proposition 3. *Let r, l be integers. Let $\{[s^k]\}$ be KZG setup of size at least $d = rl - 1$, and f_k be the coefficients of polynomial $f(X)$ of degree d. Let $\theta \in \mathbb{F}$ be a l-power field element with I denoting the set of l-roots of θ. Then KZG multiproofs for evaluating f at I is the evaluation of polynomial $v(X) \in \mathbb{G}^{r-2}[X]$ at θ with*

$$v(X) = v_1 + v_2 X + \ldots + v_{r-1} X^{r-2}. \tag{23}$$

where

$$v_j = \left(f_d[s^{d-jl}] + f_{d-1}[s^{d-jl-1}] + f_{d-2}[s^{d-jl-2}] + \cdots + f_{jl+1}[s] + f_{jl}\right).$$

Proof. The proof mostly follows the proof of Proposition 1. The only major difference is the computation of the multiproof C_{T_I} for evaluating f at l-roots of θ. Following the definition we obtain

$$\begin{aligned}
C_{T_I} = &\left(f_d[s^{d-l}] + f_{d-1}[s^{d-l-1}] + f_{d-2}[s^{d-l-2}] + \cdots + f_{l+1}[s] + f_l\right) + \\
&+ \left(f_d[s^{d-2l}] + f_{d-1}[s^{d-2l-1}] + f_{d-2}[s^{d-2l-2}] + \cdots + f_{2l+1}[s] + f_{2l}\right)\theta + \\
&+ \left(f_d[s^{d-3l}] + f_{d-1}[s^{d-3l-1}] + f_{d-2}[s^{d-3l-2}] + \cdots + f_{3l+1}[s] + f_{3l}\right)\theta^2 + \\
&+ \left(f_d[s^{d-4l}] + f_{d-1}[s^{d-4l-1}] + f_{d-2}[s^{d-4l-2}] + \cdots + f_{4l+1}[s] + f_{4l}\right)\theta^3 + \\
&\cdots \\
&+ \left(f_d[s^{d-(r-1)l}] + f_{d-1}[s^{d-(r-1)l-1}] + f_{d-2}[s^{d-(r-1)l-2}] + \cdots + f_{(r-1)l+1}[s] + f_{(r-1)l}\right)\theta^{r-2}.
\end{aligned}$$

Let for $1 \le j \le r-1$ denote

$$v_j = \left(f_d[s^{d-jl}] + f_{d-1}[s^{d-jl-1}] + f_{d-2}[s^{d-jl-2}] + \cdots + f_{jl+1}[s] + f_{jl}\right).$$

Then

$$C_{T_I} = v_1 + v_2\theta + v_3\theta^2 + \cdots + v_{r-1}\theta^{r-2}. \tag{24}$$

Therefore, C_{T_I} is the evaluation of $v(X) = \sum_{0 \le j \le r-2} v_{j+1} X^j$ at θ.

□

Now we generalize Proposition 2.

Proposition 4. *The coefficients $\mathbf{v} = \{v_1, v_2, \ldots, v_{r-1}\}$ can be computed using $2rl \log 2r$ scalar multiplications in $\mathbb{G}_1$.*

Proof. We note that the representation (22) does not work naively since the matrix representation of (13) is not Toeplitz. We, however, can represent the needed vector $\mathbf{v}$ as a *sum* of Toeplitz multiplications:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{r-2} \\ v_{r-1} \end{bmatrix} = \sum_{0 \le i < l} \begin{bmatrix} f_{d-i} & f_{d-l-i} & f_{d-2l-i} & f_{d-3l-i} & \cdots & f_{d-(r-2)l-i} \\ 0 & f_{d-i} & f_{d-l-i} & f_{d-2l-i} & \cdots & f_{d-(r-3)l-i} \\ 0 & 0 & f_{d-i} & f_{d-l-i} & \cdots & f_{d-(r-4)l-i} \\ & & \ddots & & & \\ 0 & 0 & 0 & 0 & \cdots & f_{d-l-i} \\ 0 & 0 & 0 & 0 & \cdots & f_{d-i} \end{bmatrix} \cdot \begin{bmatrix} [s^{d-l-i}] \\ [s^{d-2l-i}] \\ [s^{d-3l-i}] \\ \vdots \\ [s^{d-(r-2)l-i}] \\ [s^{d-(r-1)l-i}] \end{bmatrix}$$

Following Proposition 2, we compute $\mathbf{v}$ as follows.

1. Compute for $0 \le i < l$

$$\widehat{\mathbf{s}}_i = ([s^{d-l-i}], [s^{d-2l-i}], [s^{d-3l-i}], \ldots, [s^{d-(r-2)l-i}], [s^{d-(r-1)l-i}], \underbrace{[0], [0], \ldots, [0]}_{(r-1) \text{ neutral elements}}) \tag{25}$$

$$\mathbf{y}_i = \mathrm{DFT}_{2(r-1)}(\widehat{\mathbf{s}}_i) \tag{26}$$

2. Compute for $0 \le i < l$

$$\widehat{\mathbf{c}}_i = (f_{d-i}, \underbrace{0, 0, \ldots, 0}_{(r-1) \text{ zeroes}}, f_{d-(r-2)l-i}, f_{d-(r-3)l-i}, \ldots, f_{d-l-i}) \tag{27}$$

$$\mathbf{w}_i = \mathrm{DFT}_{2(r-1)}(\widehat{\mathbf{c}}_i) \tag{28}$$

3. Compute

$$\mathbf{u} = \left(\sum_{0 \le i < l} \mathbf{w}_i \circ \mathbf{y}_i \right)$$

4. Compute

$$\widehat{\mathbf{v}} = \mathrm{iDFT}_{2r-2}(\mathbf{u})$$

5. Take first $r-1$ elements of $\widehat{\mathbf{v}}$ as $\mathbf{v}$.

It is easy to see that steps 1 and 4 take $2rl \log r$ scalar multiplications each, step 2 needs field operations only, and step 3 needs $2rl$ scalar multiplications. □

Combining two propositions, we arrive at the main theorem for the case when the multiproofs are constructed for the set of roots of unity.

Theorem 2. *Let r, l be integers. Let $\{[s^k]\}$ be KZG setup of size at least $d = rl - 1$, and f_k be the coefficients of polynomial $f(X)$ of degree d. Let $\Theta = \{\theta_j\}_{1 \le j \le n} \subset \mathbb{F}$ be n-root of unity with I_j denoting the set of l-roots of θ_j. Then KZG multiproofs for evaluating f at I_j can be computed in $2rl \log 2r + n \log n$ scalar multiplications.*

Proof. In order to compute the multiproofs for Θ, we need to evaluate $v(X)$ at r points. If Θ is the set of roots of unity, it takes additional $n \log n$ scalar multiplications. □

4 Circulant and Toeplitz matrix-vector product computation

4.1 Circulant multiplication

A matrix-vector product with a circulant matrix B and vector

$$B = \begin{bmatrix} b_{n-1} & b_{n-2} & b_{n-3} & b_{n-4} & \cdots & b_0 \\ b_0 & b_{n-1} & b_{n-2} & b_{n-3} & \cdots & b_1 \\ b_1 & b_0 & b_{n-1} & b_{n-2} & \cdots & b_2 \\ & & \cdots & & & \\ b_{n-3} & b_{n-4} & b_{n-5} & b_{n-6} & \cdots & b_{n-2} \\ b_{n-2} & b_{n-3} & b_{n-4} & b_{n-5} & \cdots & b_{n-1} \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} \quad B\mathbf{c} = \mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{bmatrix}$$

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 $a_i = \sum_{j+k=i-1 \pmod n} b_j c_k$ and so

$$a(X) \equiv X \cdot b(X) \cdot c(X) \pmod{X^n - 1} \tag{29}$$

Denote the n-th root of unity by ω, then $a(\omega^i) = \omega^i \cdot b(\omega^i) \cdot c(\omega^i)$ since $\omega^n = 1$. Therefore we have the following algorithm for $\mathbf{a}$:

1. Compute $\widehat{\mathbf{b}} = \mathrm{DFT}_n(b_{n-1}, b_0, b_1, b_2, \ldots, b_{n-2})$ where we apply the DFT to $b'(X) = Xb(X)$.
2. Compute $\widehat{\mathbf{c}} = \mathrm{DFT}_n(c_0, c_1, c_2, \ldots, c_{n-1})$.
3. Compute $\widehat{\mathbf{a}} = \widehat{\mathbf{b}} \circ \widehat{\mathbf{c}}$.
4. Compute $\mathbf{a} = \mathrm{iDFT}_n(\widehat{\mathbf{a}})$.

If the FFT exists for n-th root of unity then the circulant matrix-vector multiplication can be done in $n(3 \log n + 1)$ time.

4.2 Toeplitz multiplication

A matrix-vector product with a Toeplitz matrix D and vector

$$F = \begin{bmatrix} f_{n-1} & f_{n-2} & f_{n-3} & f_{n-4} & \cdots & f_0 \\ 0 & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & f_1 \\ 0 & 0 & f_{n-1} & f_{n-2} & \cdots & f_2 \\ & & \cdots & & & \\ 0 & 0 & 0 & 0 & \cdots & f_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & f_{n-1} \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} \quad F\mathbf{c} = \mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{bmatrix}$$

is reduced to the circulant case by padding the matrix F to size $2n \times 2n$ and vector $\mathbf{c}$ accordingly:

$$F' = \begin{bmatrix} f_{n-1} & f_{n-2} & f_{n-3} & f_{n-4} & \cdots & f_0 & 0 & 0 & \cdots & 0 \\ 0 & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & f_1 & f_0 & 0 & \cdots & 0 \\ 0 & 0 & f_{n-1} & f_{n-2} & \cdots & f_2 & f_1 & f_0 & \cdots & 0 \\ & & \vdots & & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & f_{n-2} & f_{n-3} & f_{n-4} & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & f_{n-1} & f_{n-2} & \cdots & f_0 \\ f_0 & 0 & 0 & 0 & \cdots & 0 & 0 & f_{n-1} & \cdots & f_1 \\ f_1 & f_0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & f_2 \\ & & \vdots & & & & & & & \\ f_{n-2} & f_{n-3} & f_{n-4} & f_{n-5} & \cdots & 0 & 0 & 0 & \cdots & f_{n-1} \end{bmatrix} \quad \mathbf{c}' = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

As a result the product of F' and $\mathbf{c}'$ has all the elements of $\mathbf{a}$:

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

Therefore, to compute $F \cdot \mathbf{c}$ we compute $F' \cdot \mathbf{c}'$ using DFT and then select the top n elements of the resulting vector. If the FFT exists for the size $2n$, then the Toeplitz matrix-vector multiplication can be done in $\sim 6n \log n$ time.

If the FFT does not exist for $2n$ but exists for $k > 2n$ then one can pad the matrix F to size $k \times k$ as follows, with every column and row having exactly $n - k$ zeros:

$$F'' = \begin{bmatrix} f_{n-1} & f_{n-2} & f_{n-3} & f_{n-4} & \cdots & f_0 & 0 & 0 & \cdots & 0 \\ 0 & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & f_1 & f_0 & 0 & \cdots & 0 \\ 0 & 0 & f_{n-1} & f_{n-2} & \cdots & f_2 & f_1 & f_0 & \cdots & 0 \\ & & \vdots & & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & f_{n-2} & f_{n-3} & f_{n-4} & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & f_{n-1} & f_{n-2} & f_{n-3} & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & f_{n-1} & f_{n-2} & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & f_{n-1} & \cdots & 0 \\ & & \vdots & & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & f_0 \\ f_0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & f_1 \\ f_1 & f_0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & f_2 \\ & & \vdots & & & & & & & \\ f_{n-2} & f_{n-3} & f_{n-4} & f_{n-5} & \cdots & 0 & 0 & 0 & \cdots & f_{n-1} \end{bmatrix}$$

5 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. For this reason [GK22] proved the special case of the $O(n \log^2 n)$ case of Theorem 1 where the evaluations are all zero.
- 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].
- Data availability sampling schemes based on KZG [HASW23].

References

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[DV90] Pierre Duhamel and Martin Vetterli. Fast fourier transforms: a tutorial review and a state of the art. *Signal processing*, 19(4):259–299, 1990.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Paper 2022/1763, 2022. `https://eprint.iacr.org/2022/1763`.

[GK22] Ariel Gabizon and Dmitry Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. Cryptology ePrint Archive, Paper 2022/1447, 2022. `https://eprint.iacr.org/2022/1447`.

[HASW23] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Foundations of data availability sampling. *Cryptology ePrint Archive*, 2023. available at `https://eprint.iacr.org/2023/1079.pdf`.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[TAB+20] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In *SCN*, volume 12238 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2020.

[Tom20] Alin Tomescu. How to compute all pointproofs. Cryptology ePrint Archive, Paper 2020/1516, 2020. `https://eprint.iacr.org/2020/1516`.

[vzGG13] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra, Third edition*. 2013.

[WUP22] Weijie Wang, Annie Ulichney, and Charalampos Papamanthou. Balanceproofs: Maintainable vector commitments with fast aggregation. Cryptology ePrint Archive, Paper 2022/864, 2022. `https://eprint.iacr.org/2022/864`.

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *CCS*, pages 3121–3134. ACM, 2022.

[ZGK+22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Paper 2022/1565, 2022. `https://eprint.iacr.org/2022/1565`.

A Fast evaluation algorithm for group polynomials

This section is an adaptation of fast polynomial algorithms from [vzGG13] to the case when coefficients of one of polynomials are *group elements*. We first define what it to means to multiply polynomials from different domains.

Let $F = \sum_i F_i X^i \in \mathbb{G}^n[X], g = \sum_j g_j X^j \in \mathbb{F}^m[X]$. Then $F \cdot g = H \in \mathbb{G}^{m+n}[X]$ is defined as

$$H = \sum_k H_k X^k = \sum_k \left(\sum_{i \le k} [g_{k-i}] F_i \right) X^k$$

A.1 Fast evaluation algorithm

Input: $F \in \mathbb{G}^d[X],\ A = (a_1, a_2, \ldots, a_d) \in \mathbb{F}$.

Output: $C = (c_1, c_2, \ldots, c_d) \in \mathbb{G}^d$ 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 $g_1(X) = \prod_{a \in A_1}(X - a) \in \mathbb{F}^{d/2}[X]$ be vanishing poly of degree $d/2$ for A_1, and $g_2(X) \in \mathbb{F}^{d/2}[X]$ be vanishing poly of degree $d/2$ for A_2.
- Compute $F_1(X) = F(X) \bmod g_1(X)$ and $F_2(X) = F(X) \bmod g_2(X)$ 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 $O(d \log^2 d)$ group operations.

Constructing all vanishing polys We construct all vanishing polynomials in the monomial form from low degree to high degree. Recall that these polynomials belong to $\mathbb{F}[X]$ i.e. their coefficients are field elements. In order 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: $F \in \mathbb{G}^n[X],\ g \in \mathbb{F}^m[X]$.

Output: $Q \in \mathbb{G}^{n-m}[X],\ R \in \mathbb{G}^{m-1}[X]$ such that

$$F(X) = Q(X)g(X) + R(X)$$

Idea For $F(X) = F_0 + F_1X + \cdots + F_nX^n$ define

$$\text{rev}(F) = F_n + F_{n-1}X + \cdots + F_0X^n$$

Note that

$$X^nF(1/x) = X^{n-m}Q(1/X)X^m g(1/X) + X^{n-m+1}X^{m-1}R(1/X).$$

In terms of reverses:

$$\text{rev}(F) = \text{rev}(Q) \cdot \text{rev}(g) + X^{n-m+1}\text{rev}(R).$$

Then

$$\text{rev}(F) \equiv \text{rev}(Q) \cdot \text{rev}(g) \pmod{X^{n-m+1}}.$$

where reduction modulo X^{n-m+1} means dropping terms of degree $(n-m+1)$ and higher. This is consistent with regular modular reduction for polynomials.

Finally we obtain

$$\text{rev}(Q) \equiv \text{rev}(F) \cdot \text{rev}(g)^{-1} \pmod{X^{n-m+1}}.$$

Construction

1. Compute $\text{rev}(F) \in \mathbb{G}^n[X], \text{rev}(g) \in \mathbb{F}^m[X]$.
2. Compute $\text{rev}(g)^{-1} \bmod X^{n-m+1}$ 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 $O(d \log d)$ group operations.

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

$$g_{i+1} = g_i - \frac{1/g_i - f}{-1/g_i^2} = 2g_i - fg_i^2$$

Construction

1. Initialize $g_0 = \frac{1}{f(0)}$.
2. Compute for i up to $\log l$:
$$g_{i+1} = (2g_i - fg_i^2) \bmod x^{2^{i+1}}$$
3. Return $g_{\log l+1}$.

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

$$\sum_{1 \le i \le r} c \cdot 2^i \cdot i \le 2cr2^r$$

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

A.4 Fast multiplication Algorithm for Group Polynomials

Input: $F \in \mathbb{G}^n[X]$, $g \in \mathbb{F}^m[X]$.
Output: $H \in \mathbb{G}^{n-m}[X]$ such that

$$H(X) = F(X)g(X)$$

The algorithm is as follows:

1. Evaluate F on $2d$-roots of unity using FFT and obtain tuple $\widetilde{F} \in \mathbb{G}^{2d}$. We multiply group elements by field elements here.
2. Evaluate g on $2d$-roots of unity using FFT and obtain tuple $\widetilde{g} \in \mathbb{F}^{2d}$.
3. Multiply $\widetilde{F}$ by $\widetilde{g}$ componentwise and obtain $\widetilde{H}$.
4. Apply inverse FFT to $\widetilde{H}$ and obtain H.

The complexity is $2d \log d$ group operations.
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.