Polynomial Evaluation in the Lagrange Basis

Armando Faz Hernandez

armfazh@cloudflare.com

November 6th, 2025

Internet Engineering Task Force (IETF-124)
Crypto Forum Research Group (CFRG)

Outline

1. Motivation
2. New Results
3. Application to Prio
4. Polynomials
   - Polynomial Basis
   - Polynomial Evaluation
   - Polynomial Evaluation in the Pólya Basis
   - Polynomial Multiplication: Stolon vs Rhizome
5. Remarks

Motivation

A Common Pattern when Working with Polynomials

Definition

Let P be a polynomial such that

$$p(x_i) = y_i \text{ for } 0 \leq i \leq n$$

for some $x_i, y_i \ldots$

Evaluation

Evaluate P at a random value r.

1. Interpolate P (values $\rightarrow$ coefficients)
2. Evaluate $P(r)$

Storage/Wire Format

Interpolation

- General Case
  Lagrange interpolation.
- Roots of Unity
  Fast Fourier Transform (FFT/NTT)

$$x_i = {\omega_N}^i, \text{ such that}$$

ω_N is an Nth root of unity (${\omega_N}^N = 1$),
N is a power of 2.

Evaluation

- Horner's algorithm.

Motivation

Bottleneck: Interpolate + Evaluation

For n-degree polynomials:

1. Interpolate: FTT/NTT takes $O(n \log n)$ operations.
2. Evaluation: Horner's algorithm takes $O(n)$ operations.

Our Idea: Evaluate polynomials directly in the value representation (*Lagrange basis*).

Task: Find an $O(n)$ time algorithm for polynomial evaluation in the Lagrange basis.

New Results

New Results from Latincrypt Paper — (ia.cr/2025/1727)

An $O(n)$ time algorithm for polynomial evaluation in the Lagrange basis.

- Use of Pólya polynomial basis (known in numerical analysis).
- Similar pattern as Horner's.
- Better than the barycentric formula: No field inverses required.
- Admits batched evaluation that amortizes the cost.

Polynomial multiplication in the Lagrange basis.

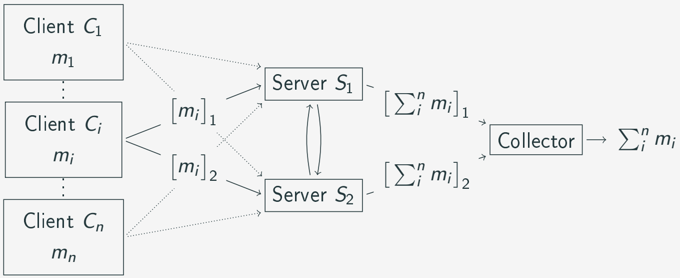
- Textbook vs Stolon vs Rhizome

Speeds up Prio's fully-linear proof system operations.

Application to Prio

VDAF: Prio

The draft-irtf-cfrg-vdaf specifies Prio for private aggregation of measurements.



- Prio uses a fully-linear Probabilistically-Checkable Proof (FLPCP) system to detect invalid measurements.
- A measurement is valid if $C(x, w) = 0$ for some validity circuit C with input x and witness w.

Changes to the VDAF-Prio Specification

Prover

Evaluates $C(x, w)$ and defines $f_1, \ldots, f_L$ s.t.

- $f_i(0) \leftarrow_\$ \mathbb{F}$.
- $f_i(j)$ is the ith input wire of the jth gate.

Set the polynomial $p = G(f_1, \ldots, f_L)$ s.t.

- $p(j)$ is the output wire of the jth gate.

The proof is $\pi = (w,\ f_1(0), \ldots, f_L(0),\ p)$.

Verifier

Makes linear queries to additive shares of the proof.

- Interpolates $f_1, \ldots, f_L$ evaluating C.
- Evaluates $f_1(r), \ldots, f_L(r)$ at $r \leftarrow_\$ \mathbb{F}$.

Accepts if $p(r) = G(f_1(r), \ldots, f_L(r))$.

Proposed Changes to draft-irtf-cfrg-vdaf (v16)

Maintain polynomials in the Lagrange basis.

1. Prover sends p in the Lagrange basis. [changes test vectors] Issue #574
2. Verifier uses our polynomial evaluation algorithm. [no breaking changes] Issue #573

Changes to the VDAF-Prio Specification

Prover

Evaluates $C(x, w)$ and defines $f_1, \dots, f_L$ s.t.

- $f_i(0) \leftarrow_\$ \mathbb{F}$.
- $f_i(j)$ is the ith input wire of the jth gate.

Set the polynomial $p = G(f_1, \dots, f_L)$ s.t.

- $p(j)$ is the output wire of the jth gate.

The proof is $\pi = (w,\ f_1(0), \dots, f_L(0),\ p)$.

Verifier

Makes linear queries to additive shares of the proof.

- Interpolates $f_1, \dots, f_L$ evaluating C.
- Evaluates $f_1(r), \dots, f_L(r)$ at $r \leftarrow_\$ \mathbb{F}$.

Accepts if $p(r) = G(f_1(r), \dots, f_L(r))$.

Proposed Changes to draft-irtf-cfrg-vdaf (v16)

Maintain polynomials in the Lagrange basis.

1. Prover sends p in the Lagrange basis. [changes test vectors] Issue #574
2. Verifier uses our polynomial evaluation algorithm. [no breaking changes] Issue #573

Changes to the VDAF-Prio Specification

Prover

Evaluates $C(x, w)$ and defines $f_1, \dots, f_L$ s.t.

- $f_i(0) \leftarrow_\$ \mathbb{F}$.
- $f_i(j)$ is the ith input wire of the jth gate.

Set the polynomial $p = G(f_1, \dots, f_L)$ s.t.

- $p(j)$ is the output wire of the jth gate.

The proof is $\pi = (w,\ f_1(0), \dots, f_L(0),\ p)$.

Verifier

Makes linear queries to additive shares of the proof.

- Interpolates $f_1, \dots, f_L$ evaluating C.
- Evaluates $f_1(r), \dots, f_L(r)$ at $r \leftarrow_\$ \mathbb{F}$.

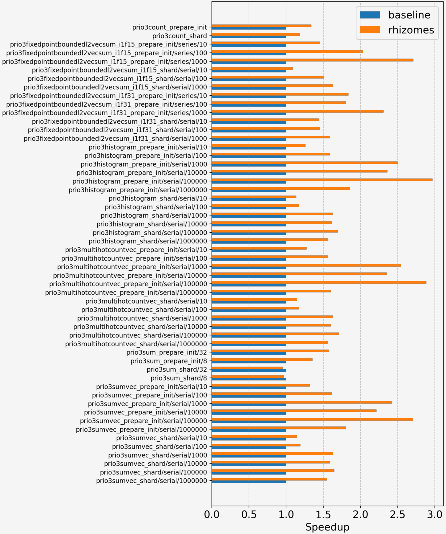
Accepts if $p(r) = G(f_1(r), \dots, f_L(r))$.

Proposed Changes to draft-irtf-cfrg-vdaf (v16)

Maintain polynomials in the Lagrange basis.

1. Prover sends p in the Lagrange basis. [changes test vectors] Issue #574
2. Verifier uses our polynomial evaluation algorithm. [no breaking changes] Issue #573

Improving Prio



At the Prio higher level:

Prover Sharding phase is 35% faster.

Verifier PrepInit phase runs twice as fast.

Both Client and Server execution get improved.

Polynomials

Polynomials as Vector Spaces

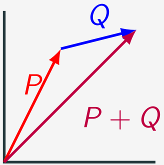
$$\{P : \deg(P) \leq N\} \equiv \mathbb{F}[x]_{[N]}$$

Polynomials of degree at most $N \equiv N$-dimensional vector space over $\mathbb{F}$

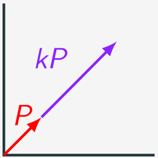
Example:

$P(x) = 2x^3 + 6x - 1$

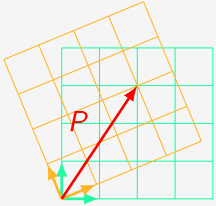
$$P(x) = \underbrace{\begin{bmatrix} -1, 6, 0, 2 \end{bmatrix}}_{\text{Coordinates}} \underbrace{\begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix}}_{\substack{\text{Polynomial} \\ \text{Basis}}}$$



Add Vectors



Scale Vectors



Change of basis

Polynomials

Polynomial Basis

Polynomial Basis

Monomial Basis $\mathcal{M}$

$$\mathcal{M} = \{1, x, x^2, \dots, x^{n-1}\}$$

Lagrange Basis $\mathcal{L}$

Let $\{x_0, \dots, x_n\}$ be distinct values called nodes, then

$$\mathcal{L} = \{\ell_0(x), \ell_1(x), \dots, \ell_n(x)\}$$

where

$$\ell_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Lagrange Basis $\mathcal{L}\mu$

Set the nodes as the Nth roots of unity

$$\{{\omega_N}^0, \dots, {\omega_N}^{N-1}\}$$

s.t. ${\omega_N}^N = 1$ and N is a power of 2.

Polynomials

Polynomial Evaluation

Polynomial Evaluation

Monomial Basis

Horner's algorithm

$$P(x) = x(\dots(x(c_n)+c_{n-1})+\dots)+c_0$$

- Runs in $O(n)$ time.
- No precomputation.
- No inverses.

Lagrange Basis

Barycentric Formula

$$P(x) = \frac{\sum_{i=0}^{n} \frac{w_i}{x - x_i} y_i}{\sum_{i=0}^{n} \frac{w_i}{x - x_i}}$$

where $w_i = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{1}{(x_i - x_j)}$.

- Calculating all w_i takes $O(n^2)$ operations.
- $P(x)$ takes $O(n)$ operations, only if all the w_i are precomputed.
- Requires field inverses.

Polynomials

Polynomial Evaluation in the Pólya Basis

The Pólya Basis

Pólya Basis $\mathcal{P}$

Let $t_1, \ldots, t_{2n}$ be scalar values, then

$$\mathcal{P} = \{d_0(x), d_1(x), \ldots, d_n(x)\}$$

where $d_i(x) = \prod_{j=1}^{n} t_{i+j} - x$.

Pólya Basis $\mathcal{P}\mu$

Choose t_i as the Nth roots of unity.

Key Fact: Pólya basis generalizes the monomial and the Lagrange polynomial basis.

Polynomial Evaluation in the Lagrange Basis $\mathcal{L}\mu$

Barycentric Formula

$$P(x) = \frac{\sum_{i=0}^{n} \frac{w_i}{x - x_i} y_i}{\sum_{i=0}^{n} \frac{w_i}{x - x_i}},$$

s.t. $w_i = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{1}{(x_i - x_j)}$.

- Calculating all w_i takes $O(n^2)$ operations.
- $P(x)$ takes $O(n)$ operations, only if all the w_i are precomputed.
- Requires field inverses.

Our Algorithm (ia.cr/2025/1727)

$$P(x) = \frac{1}{N} \sum_{i=0}^{N-1} p_i d_i(x),$$

s.t. $p_i = (-1)^{N-1} {\omega_N}^i y_i$.

- Runs in $O(n)$ time.
- Similar to Horner's algorithm.
- No inverses since $1/N$ is easy.
- Admits faster batched evaluation.

Our Algorithm for Polynomial Evaluation in the Lagrange Basis $\mathcal{L}\mu$

Input: $[P]_{\mathcal{L}\mu} = (y_0, \dots, y_{N-1})$ and $x \in \mathbb{F}$.

Constant: ${\omega_N}^0, \dots, {\omega_N}^{N-1}$.

Output: $P(x) \in \mathbb{F}$.

$N' = 1/N$ // Using $(\log_2(N) - 1)$ squares.

$l = 1$

$u = y_0$

$d = {\omega_N}^0 - x$

for $i = 1$ **to** $N - 1$ **do**

  $l = ld$

  $d = {\omega_N}^i - x$

  $u = ud + l{\omega_N}^i y_i$

end for

return $-uN'$

Our Algorithm for Batched Polynomial Evaluation in the Lagrange Basis $\mathcal{L}\mu$

Input: $[P_j]_{\mathcal{L}\mu} = \big((y_0)_j, \ldots, (y_{N-1})_j\big)$, for $0 < j \leq k$, and $x \in \mathbb{F}$.

Constant: ${\omega_N}^0, \ldots, {\omega_N}^{N-1}$.

Output: $P_j(x) \in \mathbb{F}$, for $0 < j \leq k$.

$N' = 1/N$ // Using $(\log_2(N) - 1)$ squares.

$l = 1$

$u_j = (y_0)_j$ for $0 < j \leq k$.

$d = {\omega_N}^0 - x$

for $i = 1$ **to** $N - 1$ **do**

  $l = ld$

  $d = {\omega_N}^i - x$

  $t = l{\omega_N}^i$

  for $j = 1$ **to** k **do**

    $u_j = u_j d + t(y_i)_j$

  end for

end for

return $-u_j N'$ for $0 < j \leq k$.

Polynomials

Polynomial Multiplication: Stolon vs Rhizome

Polynomial Multiplication

For n-degree polynomials, polynomial multiplication takes $O(n \log n)$ time.

Monomial Basis $\mathcal{M}$

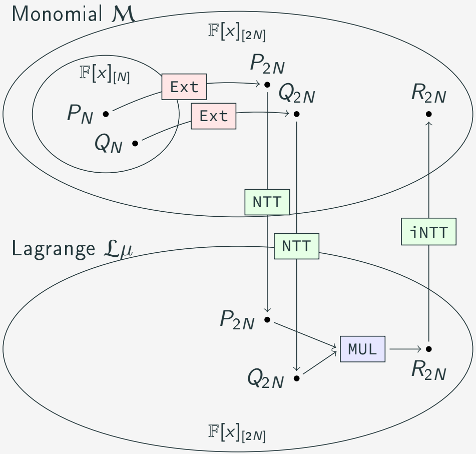
- Textbook method.

Lagrange Basis $\mathcal{L}\mu$

- Stolon
- Rhizome

Polynomial Multiplication in the Monomial Basis

Task: Calculate $R = P \times Q$, where P, Q, R are in the monomial basis $\mathcal{M}$.



$$
\begin{aligned}
\mathrm{Cost}_{\mathrm{PolyMul}}(\mathcal{M}) &= 2\,\mathbf{NTT}_{2N} + \mathbf{iNTT}_{2N} + 2N\boldsymbol{M} \\
&= 6\,\mathbf{NTT}_{N} + 7N\boldsymbol{M}
\end{aligned}
$$

Polynomial Multiplication in the Lagrange Basis $\mathfrak{L}\mu$

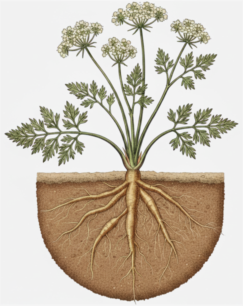
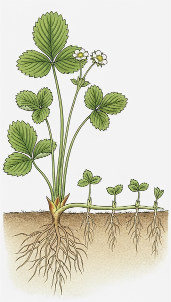
Task: Calculate $R = P \times Q \in \mathbb{F}[x]_{[2N]}$ where P, Q, R are in the Lagrange basis $\mathfrak{L}\mu$.

Two cases:

1) $P, Q \in \mathbb{F}[x]_{[2N]}$ and have degree at most N.
   - N.B. degree $\neq$ dimension.
   - Use the $O(n)$ point-wise multiplication.

2) $P, Q \in \mathbb{F}[x]_{[N]}$.
   - Perform basis extension on each polynomial, so $P, Q \in \mathbb{F}[x]_{[2N]}$.
   - Two $O(n \log n)$ strategies: Stolon & Rhizome.
   - Rhizome is faster.

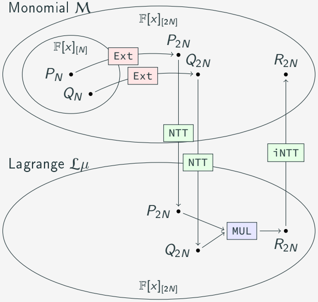
Even All are $O(n \log n)$ Time, Rhizome is Faster

Monomial $\mathcal{M}$	Lagrange $\mathcal{L}\mu$	
Textbook	Stolon	Rhizome
		
$2\,\mathbf{NTT}_{2N} + \mathbf{iNTT}_{2N} + 2N\boldsymbol{M}$	$2\,\mathbf{iNTT}_N + 2\,\mathbf{NTT}_{2N} + 2N\boldsymbol{M}$	$2\,(\mathbf{NTT}_N + \mathbf{iNTT}_N) + 2N\boldsymbol{M}$
$= 6\,\mathbf{NTT}_N + 7N\boldsymbol{M}$	$= 6\,\mathbf{NTT}_N + 6N\boldsymbol{M}$	$= 4\,\mathbf{NTT}_N + 4N\boldsymbol{M}$

Even All are $O(n \log n)$ Time, Rhizome is Faster

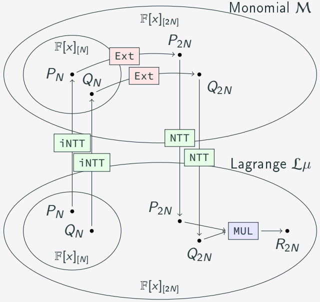
Monomial $\mathcal{M}$ | Lagrange $\mathcal{L}\mu$

Textbook



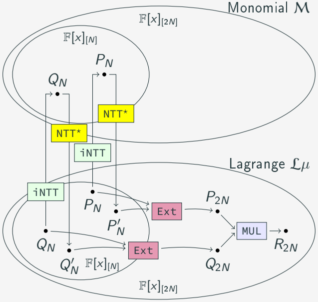
$$2\,\mathbf{NTT}_{2N} + \mathbf{iNTT}_{2N} + 2N\boldsymbol{M}$$
$$= 6\,\mathbf{NTT}_N + 7N\boldsymbol{M}$$

Stolon



$$2\,\mathbf{iNTT}_N + 2\,\mathbf{NTT}_{2N} + 2N\boldsymbol{M}$$
$$= 6\,\mathbf{NTT}_N + 6N\boldsymbol{M}$$

Rhizome



$$2\,(\mathbf{NTT}_N + \mathbf{iNTT}_N) + 2N\boldsymbol{M}$$
$$= 4\,\mathbf{NTT}_N + 4N\boldsymbol{M}$$

Remarks

One Vector Space, Many Polynomial Bases

Monomial Basis

$\mathcal{M} = \{1, x, \ldots, x^n\}$

✓ Easy to understand.

✓ **Evaluation**

$O(n)$ with Horner's alg.

✗ **Multiplication**

$O(n \log n)$ with textbook method.

Lagrange Basis

$\mathcal{L}\mu = \{\ell_0(x), \ldots, \ell_n(x)\}$

✓ Roots of unity as nodes.

✓ **Evaluation**

$O(n)$ with **our algorithm**

✓ **Multiplication**

$O(n)$ with point-wise multiplication.

$O(n \log n)$ with **rhizome algorithm**.

Pólya Basis

$\mathcal{P}\mu = \{d_0(x), \ldots, d_n(x)\}$

✓ Generalizes monomial and Lagrange.

✓ **Evaluation**

$O(n)$ with Warren's alg.

[ ] **Multiplication**

Open Problem.

Takeaways

- Use the Lagrange basis together with:
    - Our inversion-free $O(n)$ algorithm for polynomial evaluation.
    - Rhizomes algorithm for polynomial multiplication.
- Powerful tools from numerical analysis: e.g., the Pólya basis.
- Speed up of privacy-preserving protocols relying on FLPCPs.
- Other applications: polynomial commitments, STARKs, ...

Code: https://github.com/armfazh/rhizomes/
ePrint: https://ia.cr/2025/1727
PR: https://github.com/divviup/libprio-rs/pull/1340

Improving Prio

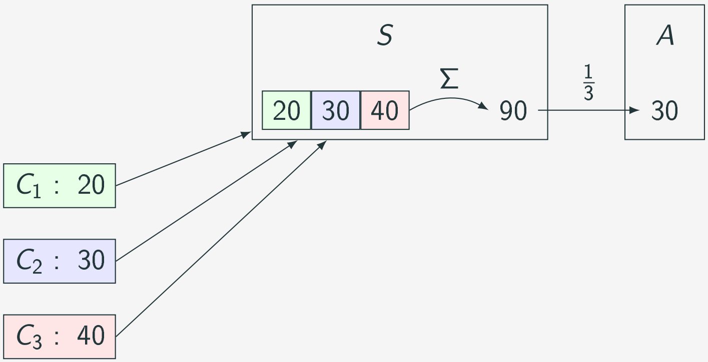
Improving Prio

Prio for Privacy-Preserving Measurement Aggregation

Prio for Privacy-Preserving Measurement Aggregation

Problem: How fast is traffic on a street?

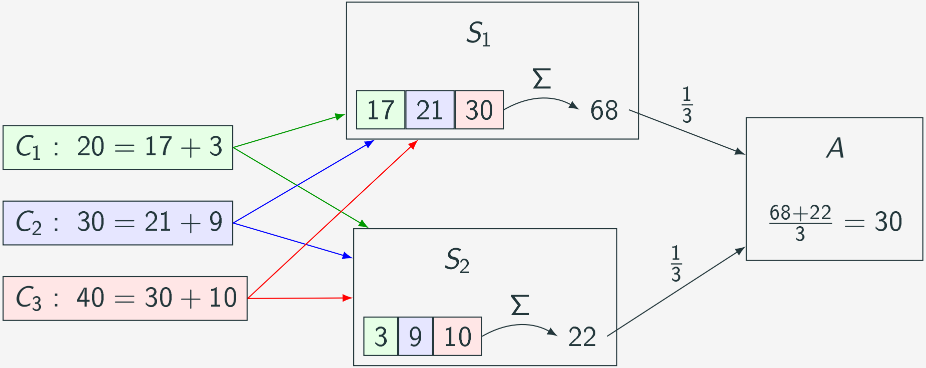
Solution: Each car sends measurements to a central server that aggregates speed values.



Issue. The Server knows the individual values.

Prio for Privacy-Preserving Measurement Aggregation

Instead, each car sends additive shares of measurements to two servers for aggregation.



Private. Each Server only knows a random share of the actual value.

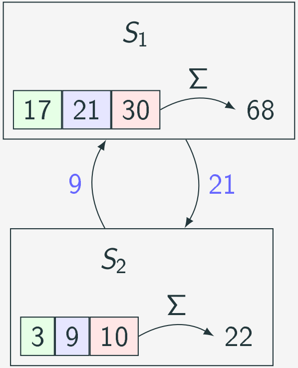
Prio for Privacy-Preserving Measurement Aggregation

Instead, each car sends additive shares of measurements to two servers for aggregation.

$C_1 : \; 20 = 17 + 3$

$C_2 : \; 30 = 21 + 9$

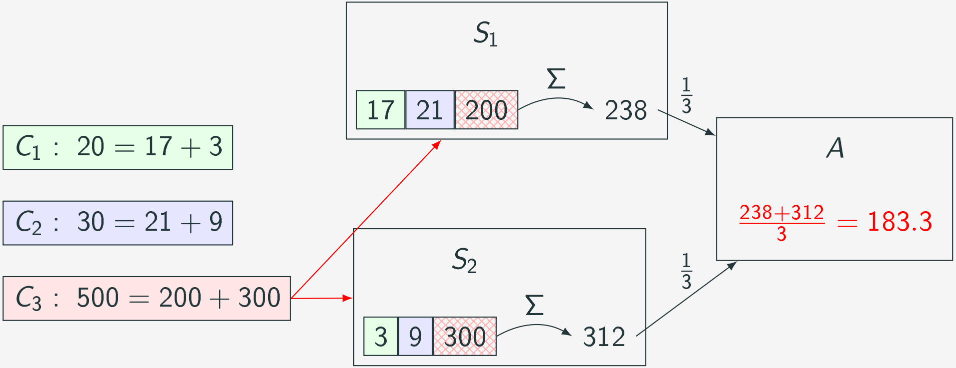
$C_3 : \; 40 = 30 + 10$



Non-colluding Servers. Otherwise, colluded servers can recover individual values.

Prio for Privacy-Preserving Measurement Aggregation

Instead, each car sends additive shares of measurements to two servers for aggregation.



Robust. Client cannot send invalid measurements that affect the aggregate.

Improving Prio

Fully-linear PCP

Probabilistically-Checkable Proofs (PCP)

Fully-linear PCP

Verifier makes linear queries to the statement and the proof.

A fully-linear PCP for the relation $(x, w) \in R$ consists of three algorithms:

Prove: Generates a proof π by evaluting $C(x, w)$.

Query: Make some linear queries $q_i = \langle (x \parallel \pi), r_i \rangle$ to the input *and* the proof.

Decision: Accepts/rejects based on the queries $\{q_i\}$ and the random challenges $\{r_i\}$.

Arithmetic Circuit

C is an arithmetic circuit with M instances of a G-gate.

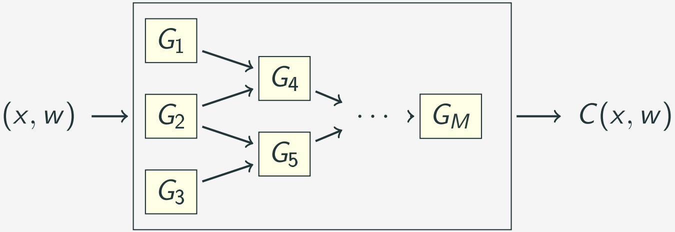
$$(x, w) \longrightarrow \boxed{\quad C \quad} \longrightarrow C(x, w)$$

A G-gate is a circuit with L inputs composed of affine and multiplication gates.

Boneh et al. [BBCGGI19] showed a fully-linear PCP for $R = \{(x, w) \colon C(x, w) = 0\}$

Arithmetic Circuit

C is an arithmetic circuit with M instances of a G-gate.



A G-gate is a circuit with L inputs composed of affine and multiplication gates.

Boneh et al. [BBCGGI19] showed a fully-linear PCP for $R = \{(x, w) \colon C(x, w) = 0\}$

FLPCP.Prove$(x, w) \to \pi$

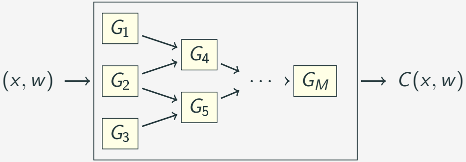
Set polynomials $f_1, \ldots, f_L$ such that

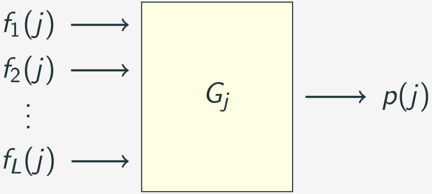
- $f_i(0) \leftarrow_\$ \mathbb{F}$.
- $f_i(j)$ is the ith input wire of the jth gate.

Set the polynomial $p = G(f_1, \ldots, f_L)$ such that

- $p(j)$ is the output wire of the jth gate.

The proof is $\pi = (w,\ f_1(0), \ldots, f_L(0),\ p)$.





FLPCP Verification

Given a proof:

$$\pi = (w,\ f_1'(0), \dots, f_L'(0),\ p')$$

Verifier's goal:

- Check the circuit outputs zero:

$$p'(M) = 0$$

- Check the circuit was correctly evaluated:

$$p' = G(f_1', \dots, f_L')$$

FLPCP Verification: Query & Decision

Verifier's Goal: Check $p' = G(f_1', \ldots, f_L')$ and $p'(M) = 0$

Query

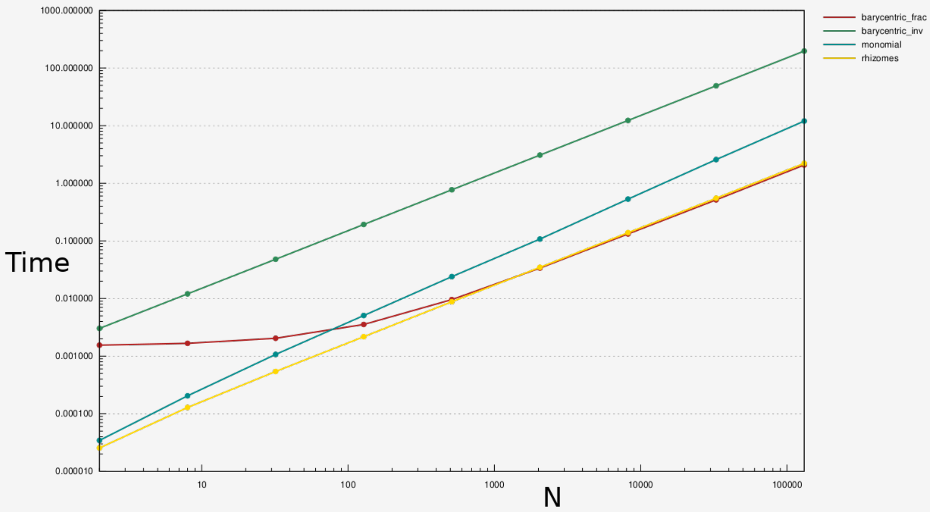
Samples $r \leftarrow_{\$} F \setminus \{1, \ldots, M\}$

$$V = (p'(r),\ f_1'(r), \ldots, f_L'(r),\ p'(M))$$

Decision

Accepts if $p'(r) = G(f_1'(r), \ldots, f_L'(r))$ and $p'(M) = 0$.

Impact of Inversion



Rust Implementation

Code: https://github.com/armfazh/rhizomes/

```
// Copyright (c) 2025 Armando Faz Hernandez.
// SPDX-License-Identifier: MPL-2.0
fn batched_eval<F: FieldElement>(p: &[Vec<F>], roots: &[F], x: F) -> Vec<F> {
    let mut l = F::one();
    let mut u = Vec::with_capacity(p.len());
    u.extend(p.iter().map(|poly| poly.get(0).copied().unwrap_or_else(F::zero)));
    let mut d = roots[0] - x;
    for (i, wn_i) in (1..).zip(&roots[1..]) {
        l *= d;
        d = *wn_i - x;
        let t = l * *wn_i;
        for (u_j, poly) in u.iter_mut().zip(p) {
            *u_j *= d;
            if let Some(yi) = poly.get(i) {
                *u_j += t * *yi;
            }
        }
    }
    if roots.len() > 1 {
        let num_roots_inv = -inv_pow2::<F>(roots.len());
        u.iter_mut().for_each(|u_j| *u_j *= num_roots_inv);
    }
    return u
}
```