Special thanks to Vitalik Buterin and Dan Boneh for rediscovering the barycentric formula, Alessandro Chiesa for discussions on the asymptotics of Pippenger, as well as Dankrad Feist and Dmitry Khovratovich for discussions on the Lagrange SRS FFT. Also thanks to various Ethereum researchers and members of the ZK Study Club for helpful discussions.

TLDR

: We show how to commit, evaluate and open polynomials in the Lagrange basis without FFTs.

Notation

Let:

- $\mathbb{F}$

be a prime field

- $\lambda = \log(|\mathbb{F}|)$
- ω

be a primitive root of unity of order n

in $\mathbb{F}$

- L_i

be the Lagrange polynomial for ω^i

, i.e. the minimal polynomial such that $L_i(w^i) = 1$

and $L_i(w^j) = 0$

if $i \ne j$

- e

be a pairing over pairing-friendly groups $\mathbb{G}_1, \mathbb{G}_2$

with base field $\mathbb{F}$

- $\mathbf{g}$

be a generator of $\mathbb{G}_1$

or $\mathbb{G}_2$

- $\tau^j\textbf{g}$

(in the additive notation) for $j=0, ..., n-1$

be an SRS in the monomial basis (a “powers of tau”) where $\tau\in\mathbb{F}$

is secret

Construction

preprocessing

—During a one-time preprocessing step the prover transforms the SRS from the monomial basis to the Lagrange basis. That is, he computes and caches $L_i(\tau)\mathbf{g}$

for $i=0, ..., n-1$

. This transformation can be done efficiently using a single offline FFT, as presented in Appendix 1.

commitments

—The prover has a polynomial f

with coefficients in the Lagrange basis, i.e. the prover has evaluations $f(\omega^i)$

for $i = 0, ..., n-1$

. The Kate commitment to f is the linear combination (also known as a multiexponentiation in the multiplicative notation) $\sum_{i=0}^{n-1}f(\omega^i)L_i(\tau)\mathbf{g}$. The commitment is readily computable from the evaluations $f(\omega^i)$ and the Lagrange SRS elements $L_i(\tau)\mathbf{g}$ without any FFT. When using Pippenger's algorithm (see discussion in Appendix 3) the cost is $O\big(\frac{n\lambda}{\log(n\lambda)}\big)$ group operations.

evaluations

—Let z be an evaluation point which is not a root of unity, e.g. a random evaluation point. We show how to compute $f(z)$ using a linear number of field operations. When working with roots of unity the barycentric formula yields $f(z) = \frac{1 - z^n}{n}\sum_{i=0}^{n-1}\frac{f(\omega^i)\omega^i}{\omega^i - z}$. Computing the z^n term in the leading factor $\frac{1 - z^n}{n}$ is negligeable (it costs roughly $\log(n)$ multiplications). It therefore suffices to cheaply compute the inverses of $\omega^i - z$. We show in Appendix 2 how to do this using 1 inversion and $3(n - 1)$ multiplications using Montgomery batch inversion.

openings

—We conclude by showing how to compute opening proofs given the evaluation $f(z)$. The opening proof is the linear combination $\sum_{i=0}^{n-1}\frac{f(\omega^i) - f(z)}{\omega^i - z}L_i(\tau)\mathbf{g}$ which can be computed using Pippenger. Notice that the inverses of $\omega^i - z$ were computed in linear time for the evaluation of $f(z)$ above.

Appendix 1—SRS transformation

The goal is to efficiently compute $L_i(\tau)\mathbf{g}$ for $i = 0, ..., n-1$ from the powers of tau $\tau^j\mathbf{g}$ for $j = 0, ..., n-1$. Notice that $L_i(X) = \frac{1}{n}\sum_{j=0}^{n-1}(\omega^{-i}X)^j$ so that $L_i(\tau)\mathbf{g} = \frac{1}{n}\sum_{j=0}^{n-1}\tau^j\mathbf{g}Y^j\rvert_{Y=\omega^{-i}}$. To get the Lagrange SRS it therefore suffices to evaluate the polynomial $\frac{1}{n}\sum_{j=0}^{n-1}\tau^j\mathbf{g}Y^j$ at every root of unity using a single FFT.

Appendix 2—batch inversion

Montgomery's batch inversion trick inverts k field elements $a_1, ..., a_{k}$

in four steps:

1. compute $\prod_{i = 1}^j a_i$

for $j = 2, ..., k$

1. compute $1\big/\prod_{i = 1}^k a_i$
2. compute $1\big/\prod_{i = 1}^{j-1} a_i$

for $j = k, ..., 2$

1. compute $a_j^{-1} = \big(\prod_{i = 1}^j a_i\big)\big(1\big/ \prod_{i = 1}^{j-1} a_i\big)$

for $j = 2, ..., k$

Steps 1, 3, 4 each cost $k - 1$

field multiplications and step 2 costs 1

field inversion.

Appendix 3—asymptotics of Pippenger

Pippenger proved that his algorithm is assymptotically optimal. In the context of SNARKs we have $\lambda > \log(n)$

and (except for tiny circuits) $n > \lambda$

so that $\frac{n\lambda}{\log(n\lambda)} > \frac{n\log(n)}{\log(n\lambda)} > \frac{n\log(n)}{\log(n^2)} = \frac{n}{2}$

. Therefore, contrary to folklore, linear combinations cost at least a linear number of group operations, even with Pippenger.

Technically n

is at most polynomial in λ

since provers are polynomially bounded, and therefore linear combinations with Pippenger are asymptotically superlinear. In practice, after setting λ

to roughly 128

bits, a linear combination with n

terms costs roughly $10n$

group operations.