Privacy Enhancing Technologies
Lecture 4 – Building a Modern SNARG

Florian Tramèr

These lecture notes are inspired by notes of Dan Boneh[1], and Vitalik Buterin[2].

AGENDA

1. Recap on SNARGs
2. Polynomial Magic
3. Polynomial Commitments
4. PLONK

1 Recap on SNARGs

In the last lecture we introduced interactive proofs and zero-knowledge, and discussed a number of applications. As wee saw, in many cases it is particularly nice to have proof systems that are *non-interactive* and *succinct*. We call these *SNARGs*.

We saw a blueprint for SNARG constructions which first builds an information-theoretic proof system where the prover and verifier interact in some constrained way, aided by a special *proof oracle*. We saw one concrete example, the PCP theorem, that allows a verifier to check the correctness of a proof while only reading a constant number of bits from it. You'll show how to build a SNARG from the PCP theorem in your homework. But no one uses this approach in practice, because the PCP theorem is incredibly inefficient: even though the PCP proof is polynomial-sized, existing constructions have huge blowup factors.

In this lecture, we will look at another type of information-theoretic proof system, *Polynomial IOPs*, where the prover and verifier interact via *polynomials*. We'll then present one famous and recent Polynomial IOP (called PLONK [GWC19]) and discuss how to turn it into a SNARG using polynomial commitments.

2 The Magic of Polynomials

The majority of modern SNARGs start from an information-theoretic protocol where the prover and verifier interact by exchanging *polynomials*. But why polynomials?

Essentially, you can think of a polynomial as a succinct way to represent many different equations. Looking forward, this will enable us to represent the computation of an entire arithmetic circuit as just one big polynomial equation.

[1] https://cs251.stanford.edu/lectures/lecture15.pdf
[2] https://vitalik.eth.limo/general/2021/01/26/snarks.html

Let's work through a dummy example, due to (I think) Vitalik Buterin. Suppose I want to convince you that the 100-th Fibonacci number is

$$354{,}224{,}848{,}179{,}261{,}915{,}075$$

Of course you could do this computation yourself (e.g., by using the closed-form expression for the n-th Fibonacci number). But let's assume you can't (or don't want to) do this.

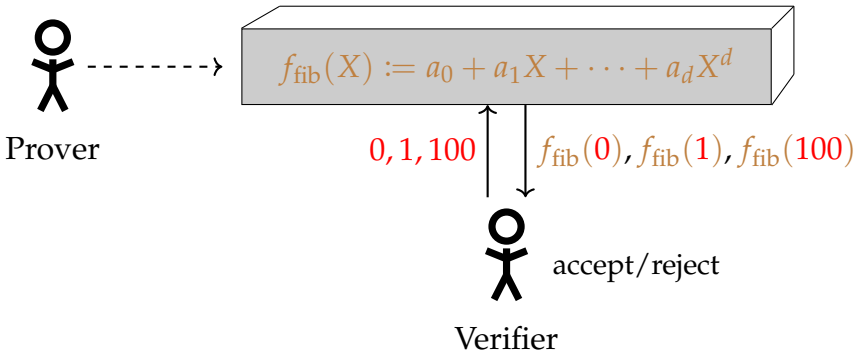
To convince you, I create a polynomial $f_{\text{fib}}(X)$ over some (large) field $\mathbb{F}_p$ and claim that:

1. $f_{\text{fib}}(0) = f_{\text{fib}}(1) = 1$.
2. $f_{\text{fib}}(100) = 354{,}224{,}848{,}179{,}261{,}915{,}075$.
3. $f_{\text{fib}}(x+2) = f_{\text{fib}}(x) + f_{\text{fib}}(x+1)$ for all $x \in \{0, 1, \ldots, 98\}$.

You should convince yourself that this is equivalent to my original claim. Note that creating the polynomial f_{fib} requires a lot of work, but this is work done by the prover.

I put the polynomial f_{fib} in a "box" that lets you evaluate it at any point (we will later replace the box by a polynomial commitment scheme). So what does the verifier need to do?

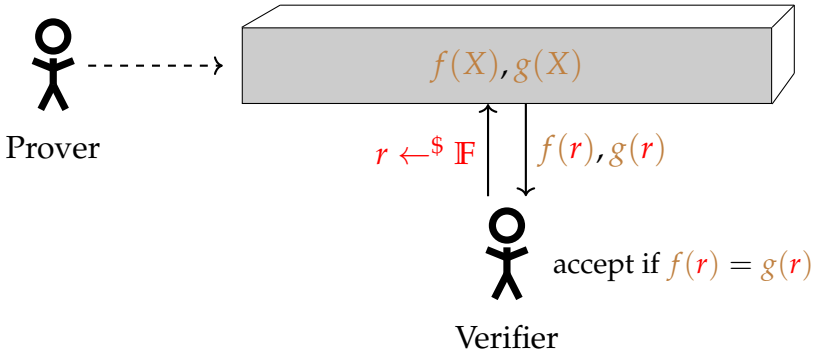
Testing that $f_{\text{fib}}(0) = f_{\text{fib}}(1) = 1$ and that $f_{\text{fib}}(100) = 354{,}224{,}848{,}179{,}261{,}915{,}075$ is easy. You just ask the box for an evaluation of f_{fib} at 0, 1, and 100.



Testing the third property is a bit more complicated, and will require using some nice properties of polynomials.

Equality Test. Let's first see how to prove that two polynomials f and g in the box are equal. This is a very useful building block for proving more interesting things.

The prover puts both polynomials f and g in a box that the verifier can evaluate at any point. A naive solution would be for the verifier to evaluate both polynomials at $d+1$ points and check if they are all equal. But if we allow for some small error, we can do a lot better! What the verifier will do is simply to evaluate both polynomials at a *single* random point $r \in \mathbb{F}$. If $f(r) = g(r)$, then the verifier accepts that the two polynomials are equal.



Why is this sound? This follows from the single most important fact about polynomials that you should remember: they can't have too many roots!

> Any non-zero polynomial over $\mathbb{F}$ of degree d has at most d roots over $\mathbb{F}$.

A fact so fundamental deserves a fundamental name: *the fundamental theorem of algebra*.

Suppose the prover cheats and $f \neq g$. Then $f(X) - g(X)$ is a non-zero polynomial of degree at most d, and so it can have at most d roots over $\mathbb{F}$. Then, the probability that the verifier picks a "bad" r such that $f(r) = g(r)$ is at most $\frac{d}{|\mathbb{F}|}$. If we pick a large enough field $\mathbb{F}$, then this probability can be made negligibly small.

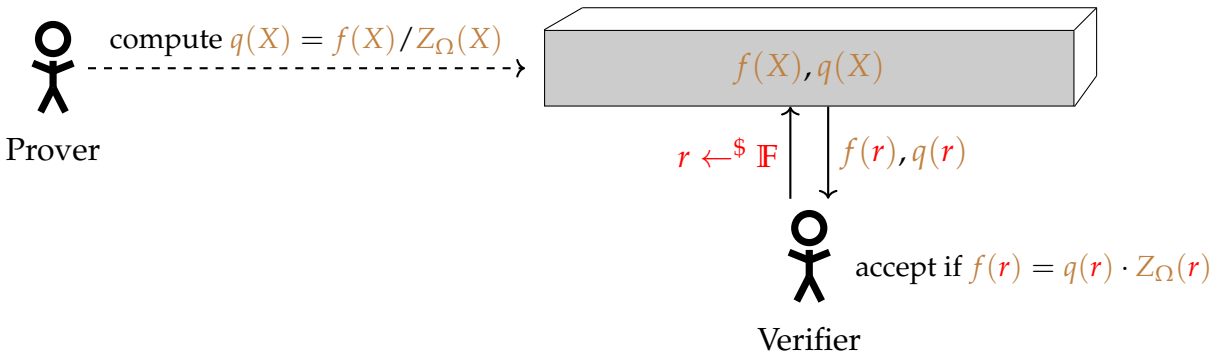
ZeroTest. Recall that in our running Fibonacci example, we want to show that $f_{\text{fib}}(x+2) = f_{\text{fib}}(x) + f_{\text{fib}}(x+1)$ for all $x \in \{0, 1, \ldots, 98\}$. This is not quite the same as an equality test, since we only need to check the equality on a subset of points, rather than all points (i.e., for $x > 98$ we don't care if the equality holds or not). We can recast this problem as showing that the polynomial $f_{\text{fib}}(X+2) - f_{\text{fib}}(X) - f_{\text{fib}}(X+1)$ is zero on the set $\{0, 1, \ldots, 98\}$.

We will abstract this slightly as this problem will re-appear in PLONK. Suppose the prover wants to convince the verifier that a polynomial $f \in \mathbb{F}[X]^{\leq d}$ is zero on all points in some set $\Omega \subseteq \mathbb{F}$ of size $|\Omega| = k \leq d$. We now use another useful fact. Let $Z_\Omega(X) := \prod_{a \in \Omega}(X - a)$ be the *vanishing polynomial* of Ω. Then,

> f is zero on Ω if and only if $f(X)$ is divisible by $Z_\Omega(X)$.

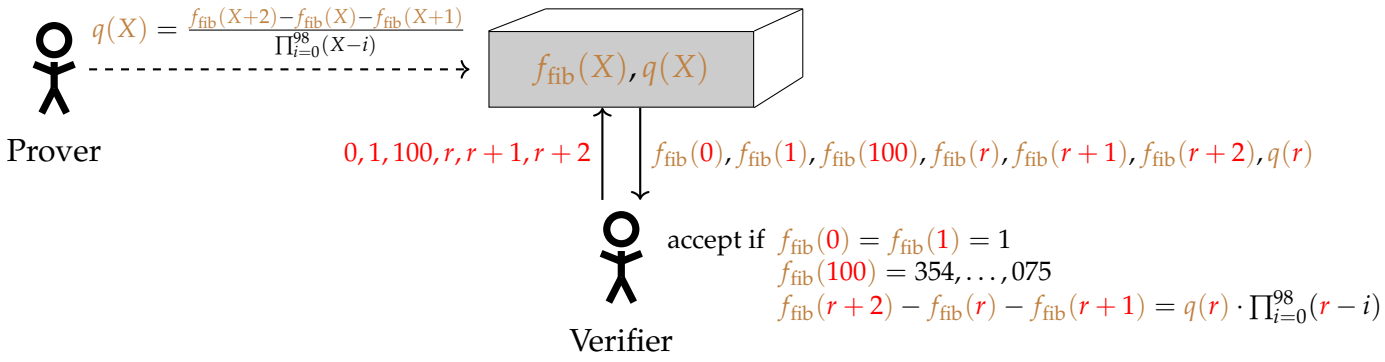
As an example, let $f(X) = X^3 - 7X + 6$ and $\Omega = \{1, 2\}$. We have $f(1) = f(2) = 0$, so f is zero on Ω. And indeed, we can write $f(X) = \underbrace{(X-1)(X-2)}_{Z_\Omega(X)} \cdot (X+3)$.

So if f is indeed zero on Ω, then we can write $f(X) = Z_\Omega(X) \cdot q(X)$ for some quotient polynomial q of degree less than d. If f is not zero on Ω, then no such polynomial q exists. We can now set up our proof system in the "box" world:



To argue soundness, assume f is not zero on Ω. Then $f(X)/Z_\Omega(X)$ is not a polynomial over $\mathbb{F}$, and so the prover has to send the box another polynomial $q(X)$ of degree d. Now consider the polynomial $f(X) - q(X) \cdot Z_\Omega(X)$. This polynomial is non-zero, and has degree at most $2d$. So the probability that this polynomial vanishes at a random point r is at most $\frac{2d}{|\mathbb{F}|}$.

Putting it all together. We now have all the building blocks we need for our Poly-IOP for proving the value of the 100-th Fibonacci number. Our final protocol looks like this:



There is one small issue with this protocol that we glossed over: the verifier has to compute $\prod_{i=0}^{98}(r-i)$. This is not a succinct computation (compared to calculating $f_{\text{fib}}(100)$ directly). When we design the PLONK proof system later, we will show how to get around this by either having this vanishing polynomial be *pre-computed*, or by having the vanishing polynomial have some special structure that allows $Z_\Omega(r)$ to be computed efficiently.

3 Polynomial Commitments

The protocols we defined so far were in this special "box" world where the prover can put a bunch of polynomials in a locked box and the verifier can query them at random points. This is more formally called a *Polynomial IOP* (or Poly-IOP) [CHM+20, BFS20].

To turn a Poly-IOPs into a SNARG, we just need a suitable commitment scheme for polynomials: a *Polynomial Commitment Scheme* [KZG10]. This lets a prover provide a *short* commitment to a polynomial f and provide short proofs that $f(z) = y$ for arbitrary points z.

Definition 1 (Polynomial Commitment Scheme)**.** A polynomial commitment scheme for polynomials in a field $\mathbb{F}$ is a tuple of algorithms:

- $\texttt{Setup}(d, \lambda) \to \texttt{pp}$, outputs public parameters $\texttt{pp}$ given a degree bound d and a security parameter λ.
- $\texttt{Commit}(\texttt{pp}, f) \to c$, outputs a commitment c to a polynomial $f \in \mathbb{F}[X]^{\leq d}$.
- $\texttt{Open}(\texttt{pp}, f, z) \to \pi$, outputs a proof π for the evaluation $y = f(z)$.
- $\texttt{Verify}(\texttt{pp}, c, z, y, \pi) \to \{0, 1\}$, checks the proof π for the evaluation point (z, y) with respect to the commitment c.

The scheme should satisfy the following properties:

Correctness. For all $d \in \mathbb{N}$, all polynomials $f \in \mathbb{F}[X]^{\leq d}$ and all points $z \in \mathbb{F}$, we have:

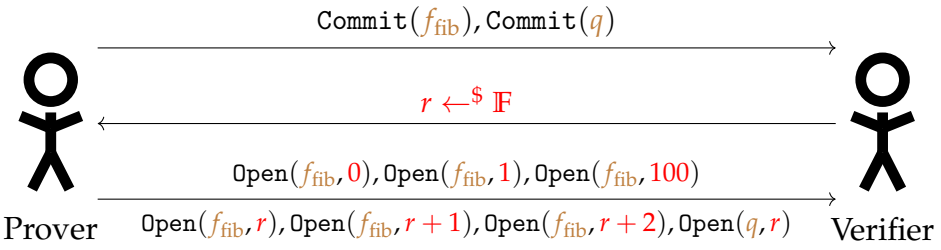
$$\Pr\left[\texttt{Verify}(\texttt{pp}, c, z, f(z), \pi) = 1 \;:\; \begin{array}{r} \texttt{pp} \leftarrow \texttt{Setup}(d) \\ c \leftarrow \texttt{Commit}(\texttt{pp}, f) \\ \pi \leftarrow \texttt{Open}(\texttt{pp}, f, z) \end{array}\right] = 1 \,.$$

Evaluation Binding. For all $d \in \mathbb{N}$ and poly-time adversaries $\mathcal{A}$ we have:

$$\Pr\left[\begin{array}{c} \texttt{Verify}(\texttt{pp}, c, z, y, \pi) = 1, \\ \texttt{Verify}(\texttt{pp}, c, z, y', \pi') = 1, \\ y \neq y' \end{array} \;:\; \begin{array}{r} \texttt{pp} \leftarrow \texttt{Setup}(d) \\ (c, z, y, \pi, y', \pi') \leftarrow \mathcal{A}(\texttt{pp}) \end{array}\right] \leq \texttt{negl}(\lambda) \,.$$

We won't describe any concrete construction here, but you will familiarize yourself with one in the homework.

From a Poly-IOP to a SNARG. So coming back to our simple Fibonacci example, our final SNARG would have the prover send commitments to $f_{\text{fib}}(X)$ and $q(X)$, and then provide opening proofs for all the verifier's queries.



Since the verifier only makes one random query, we can turn the entire protocol non-interactive using the Fiat-Shamir heuristic, by deriving the challenge r from a random oracle.

4 The PLONK IOP

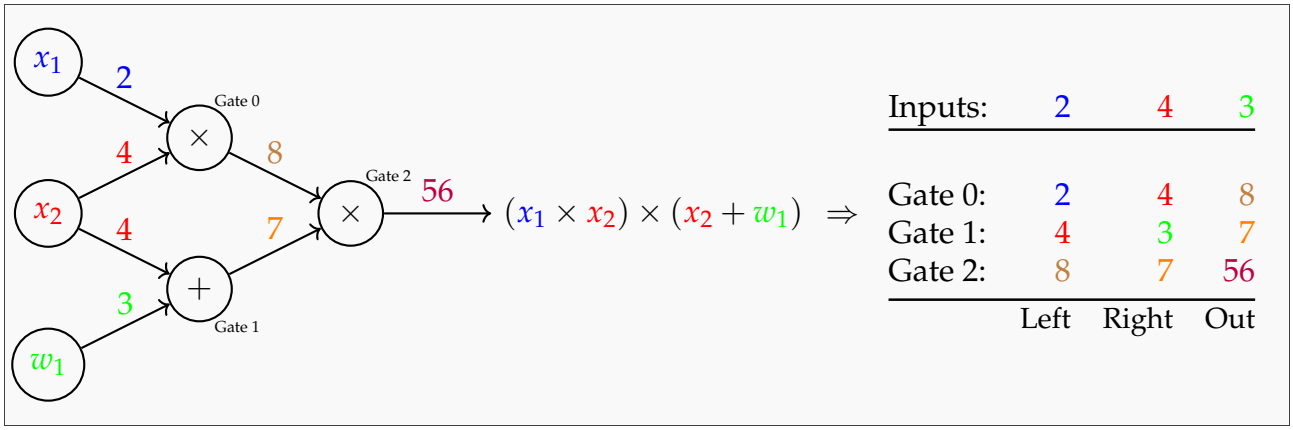
We've seen a flavor of how to prove statements using polynomials. But for now we looked at a very specific (and not very interesting) example of computing Fibonacci numbers.

We are now going to describe a popular Poly-IOP that applies to *arbitrary* arithmetic circuits: PLONK [GWC19]. When combined with a Polynomial Commitment Scheme and the Fiat Shamir heuristic, the PLONK IOP becomes a SNARG (it is also easy to turn it zero-knowledge, although we won't describe this part). Depending on the choice of commitment scheme, we get SNARGs with different properties (e.g., with or without the need for a "trusted setup", with post-quantum security or not, etc.)

Recall from the last lecture that we are interested in proving statements of the form $\mathcal{C}(x, w) = 0$ where $\mathcal{C}$ is an arithmetic circuit, $x \in \mathbb{F}^m$ is a a *public* input (known to the verifier) and $w \in \mathbb{F}^n$ is the prover's witness. Our goal is to build a *succinct non-interactive* proof system for such circuits. Specifically, the proof string should be of length $O(\lambda, \text{polylog}\,(|\mathcal{C}|))$ and the verifier should run in time $O(\lambda, |x|, \text{polylog}\,(|\mathcal{C}|))$.

4.1 From Circuits to Polynomials

Circuit traces. Given an arithmetic circuit, we can represent its execution as a *circuit trace* which represents the value of each wire in the circuit at each step.



Representing circuit traces as polynomials. Let $|\mathcal{C}|$ be the total number of gates in the circuit $\mathcal{C}$, and let $|I| = |I_x| + |I_w|$ be the number of inputs to $\mathcal{C}$. Let $d = 3|\mathcal{C}| + |I|$ be the polynomial degree (in our example above, we have $d = 12$). Define a set $\Omega = \{1, \omega, \omega^2, \ldots, \omega^{d-1}\}$ where $\omega \in \mathbb{F}$ is a *root of unity* in $\mathbb{F}$ (i.e., $\omega^d = 1$).

The prover then interpolates a polynomial $f_{\text{trace}} \in \mathbb{F}[X]^{\leq d}$ that encodes all inputs and wires:

1. f_{trace} **encodes all inputs**: $f_{\text{trace}}(\omega^{-j})$ = input #j for $j = 1, \ldots, |I|$.
2. f_{trace} **encodes all wires**: For all $l = 0, \ldots, |\mathcal{C}| - 1$:
   - $f_{\text{trace}}(\omega^{3l})$ = left input to gate #l
   - $f_{\text{trace}}(\omega^{3l+1})$ = right input to gate #l
   - $f_{\text{trace}}(\omega^{3l+2})$ = output of gate #l

In our example above, f_{trace} has degree 11:

Inputs:	$f_{\text{trace}}(\omega^{-1}) = 2$	$f_{\text{trace}}(\omega^{-2}) = 4$	$f_{\text{trace}}(\omega^{-3}) = 3$
Gate 0:	$f_{\text{trace}}(\omega^{0}) = 2$	$f_{\text{trace}}(\omega^{1}) = 4$	$f_{\text{trace}}(\omega^{2}) = 8$
Gate 1:	$f_{\text{trace}}(\omega^{3}) = 4$	$f_{\text{trace}}(\omega^{4}) = 3$	$f_{\text{trace}}(\omega^{5}) = 7$
Gate 2:	$f_{\text{trace}}(\omega^{6}) = 8$	$f_{\text{trace}}(\omega^{7}) = 7$	$f_{\text{trace}}(\omega^{8}) = 56$

4.2 Verifying The Circuit Trace

So now the prover puts the trace polynomial f_{trace} in a box and lets the verifier evaluate f_{trace} on arbitrary points. What should the verifier check to ensure that the proof is correct?

1. The output of the last gate is 0
2. f_{trace} encodes the correct public inputs
3. Every gate is evaluated correctly
4. The wiring is implemented correctly

We won't cover all of these in detail, but the idea is that the verifier will check all of these properties by querying the polynomial on appropriate points.

(1) Checking the output. The verifier simply checks that $f_{\text{trace}}(\omega^{3|\mathcal{C}|-1}) = 0$.

(2) Checking the inputs. The verifier builds a polynomial $f_{\text{in}}(X) \in \mathbb{F}[X]^{\leq |I_x|}$ that encodes the public inputs x to the circuit: for $j = 1, \ldots, |I_x|$: $f_{\text{in}}(\omega^{-j}) = x_j$.

The prover then proves to the verifier that $f_{\text{trace}}(x) - f_{\text{in}}(x) = 0$, $\forall x \in \Omega_I = \{\omega^{-j} \mid j = 1, \ldots, |I_x|\}$. This is an instance of the ZeroTest problem! So the prover puts f_{trace} and $f_{\text{trace}} - f_{\text{in}}$ in the box. The verifier can first check that these oracles are consistent (i.e., they correspond to the same trace f_{trace}) by querying both polynomials on a random point r, and testing that the results differ by $f_{\text{in}}(r)$ (which the verifier can calculate locally). Then, we do the ZeroTest IOP on $f_{\text{trace}} - f_{\text{in}}$ over Ω_I.

(3) Checking the evaluation. We need to check that the addition and multiplication gates are actually evaluated correctly. That is, we need to check that $f_{\text{trace}}(\omega^2) = f_{\text{trace}}(\omega^0) \times f_{\text{trace}}(\omega^1)$, $f_{\text{trace}}(\omega^5) = f_{\text{trace}}(\omega^3) + f_{\text{trace}}(\omega^4)$, etc.

The idea here is to build a *selector polynomial* f_{sel} that encodes the type ($\times$ or $+$) of each gate. Formally:

$$\text{Define } f_{\text{sel}}(X) \in \mathbb{F}[X]^{\leq d} \text{ such that } \forall l = 0, \ldots, |\mathcal{C}| - 1 :$$
$$f_{\text{sel}}(\omega^{3l}) = \begin{cases} 0 & \text{if gate } l \text{ is a multiplication gate} \\ 1 & \text{if gate } l \text{ is an addition gate} \end{cases}$$

In our running example, we would have:

				$f_{\text{sel}}(X)$	
Gate 0 (ω^0):	2	4	8	0	($\times$)
Gate 1 (ω^3):	4	3	7	1	($+$)
Gate 2 (ω^6):	8	7	56	0	($\times$)

Then, we want to check that:

$$\forall x \in \Omega_{\text{gates}} := \{\omega^0, \omega^3, \omega^6, \omega^9, \ldots, \omega^{3(|\mathcal{C}|-1)}\} :$$
$$f_{\text{sel}}(x) \times [\underbrace{f_{\text{trace}}(x)}_{\text{Left input}} + \underbrace{f_{\text{trace}}(\omega x)}_{\text{Right input}}] + (1 - f_{\text{sel}}(x)) \times [\underbrace{f_{\text{trace}}(x)}_{\text{Left input}} \times \underbrace{f_{\text{trace}}(\omega x)}_{\text{Right input}}] = \underbrace{f_{\text{trace}}(\omega^2 x)}_{\text{Output}} .$$

We can do this with a ZeroTest over Ω_{gates} for the polynomial:

$$f_{\text{gates}}(X) := f_{\text{sel}}(X) \cdot [f_{\text{trace}}(X) + f_{\text{trace}}(\omega X)] + (1 - f_{\text{sel}}(X)) \cdot f_{\text{trace}}(X) \cdot f_{\text{trace}}(\omega X) - f_{\text{trace}}(\omega^2 X)$$

One important thing to note though, is that f_{sel} has degree $O(|\mathcal{C}|)$ and so the verifier cannot even build this polynomial if we want the proof to be succinct. The nice observation here is that f_{sel} only depends on the circuit $\mathcal{C}$, and not on the specific instance x or witness w. So if we are going to re-use the circuit $\mathcal{C}$ many times, we can do a one-time expensive *pre-processing* step to compute f_{sel} and commit to it. This commitment then becomes part of the public parameters of the scheme.

(4) Checking the wiring. Note that the polynomial f_{trace} contains some redundancy. Specifically, the inputs (which are encoded in the points Ω_I) are also left or right inputs of some gates. Then, the output of each gate (except the last) is also an input to another gate. The verifier thus has to check these consistency constraints.

In our running example, this corresponds to checking:

- Inputs: $f_{\text{trace}}(\omega^{-1}) = f_{\text{trace}}(\omega^0)$, $f_{\text{trace}}(\omega^{-2}) = f_{\text{trace}}(\omega^1) = f_{\text{trace}}(\omega^3)$
- Gate outputs: $f_{\text{trace}}(\omega^2) = f_{\text{trace}}(\omega^6)$, $f_{\text{trace}}(\omega^5) = f_{\text{trace}}(\omega^7)$

We won't describe the Poly-IOP that checks this property here. The idea is basically to prove that $f_{\text{trace}}(x) = f_{\text{trace}}(f_{\text{rot}}(x)), \forall x \in \Omega$ where $f_{\text{rot}} : \Omega \to \Omega$ is a *rotation polynomial* that depends only on the circuit's wiring. This polynomial is also committed to in a preprocessing phase.

5 A PLONK SNARG

So what does our final SNARG look like, when we combine the PLONK IOP with a polynomial commitment scheme?

(Universal) setup. We run the setup of the polynomial commitment scheme to get pp. This setup is *universal*, in that it does not depend on the specific circuit $\mathcal{C}$ being used (as long as we have a bound on the size of the circuits we may want to make proofs for).

Circuit-specific preprocessing. We compute the polynomials f_{sel} and f_{rot}, commit to them, and add the commitments to the public parameters. This preprocessing takes time $O(|\mathcal{C}|)$.

Proof. The prover runs the circuit $\mathcal{C}(x, w)$ on the public input x and witness w. Then:

1. The prover computes the trace polynomial f_{trace} and commits to it. The verifier queries it at $\omega^{3|\mathcal{C}|-1}$ to check the circuit is satisfied.
2. The prover and verifier compute the input polynomial f_{in}. The prover commits to $f_{\text{trace}} - f_{\text{in}}$ and to the quotient polynomial for the ZeroTest. The verifier queries these polynomials on random points.
3. The prover computes the quotient polynomial of $f_{\text{gates}}(X)$ to perform a ZeroTest over Ω_{gates}. The verifier queries the committed polynomials (f_{sel}, f_{trace} and the quotient) to do the ZeroTest.
4. The prover and the verifier do another Poly-IOP (not covered here) to check that f_{trace} is consistent with the committed rotation polynomial f_{rot} (this ultimately also reduces to a ZeroTest).

In this protocol, all the verifier's queries are randomly sampled from $\mathbb{F}$, and so we can apply Fiat-Shamir to turn the whole thing non-interactive.

References

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 677–706. Springer, 2020.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 738–768. Springer, 2020.

[GWC19] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.