Privacy Enhancing Technologies FS2025
Lecture 9-10-11-12 – Building a Modern SNARG

Florian Tramèr

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

AGENDA

1. Blueprint of a SNARG
2. Polynomial Magic
3. Polynomial Commitments
4. PLONK

1 SNARGs

In many applications, a standard ZK proof system is impractical and we actually want further properties from our proof system.

For example, suppose you want a server to compute the function $\text{SHA}^{1'000'000}(x)$ for you (i.e., applying SHA-256 a million times to an input x). One way for the server to "prove" that the answer is y is just to send you all $1'000'000$ hashes and let the verifier check each one (or just send only y and let the verifier re-run the entire computation...) So we need a proof system where the proof is *succinct*, i.e., much faster to verify than to generate.

In some settings (e.g., cryptocurrencies), we also do not want the prover to have to *interact* with every other party in the system to convince them their proof is valid. Here, we want a *non-interactive proof* [BFM88], i.e., a proof string that can just be sent to anyone.

Proofs that satisfy these two properties are called *SNARGs* (Succinct Non-Interactive Arguments) [BCCT12]. (notice that for now, we've left zero-knowledge out of the picture. It turns out that for modern SNARGs, zero-knowledge is typically the easy part. Once we have a proof system that is non-interactive and succinct, getting zero-knowledge as well can often be done by appropriately randomizing the prover).

There are different ways to define succinctness. The one we'll consider here is as follows:

- *Short proofs:* The proof size $|\pi|$ is *poly-logarithmic* in the size of the input and witness, i.e., at most $O(\lambda, \text{polylog}(|x|, |w|))$, where λ is the security parameter.
- *Efficient verification:* The time required to check the proof is $O(\lambda, |x|, \text{polylog}(|w|))$, i.e., linear in the length of the statement x and poly-logarithmic in the size of the witness w (note that the linear dependence on $|x|$ is unavoidable as the verifier has to be able to read the statement to be proven).

[1] https://cs251.stanford.edu/lectures/lecture15.pdf

[2] https://vitalik.eth.limo/general/2021/01/26/snarks.html

In contrast to non-interactivity, succinctness is a highly non-trivial property even without considering zero-knowledge. Indeed, a standard "one-shot" proof for an NP statement is *not* succinct as the proof is as long as the witness. But it turns out that this is possible by using cryptography! (under some strong assumptions)

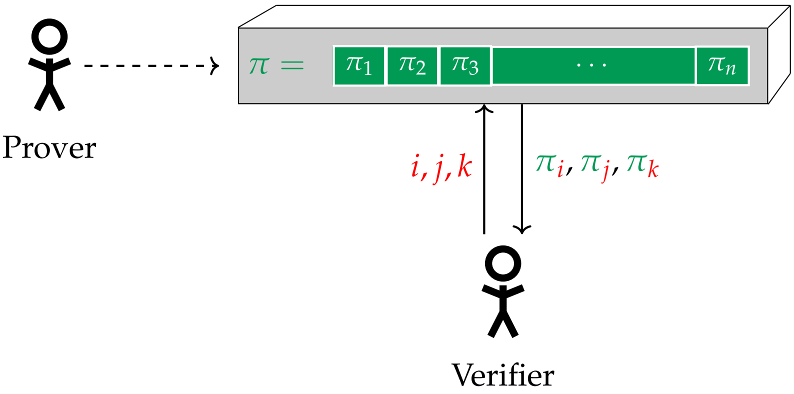
2 Blueprint for a SNARG

So how do we construct SNARGs? Modern constructions that aim to maximize efficiency are quite complicated. We will give an overview of a popular approach, but we won't be able to cover all the details. You'll build an older and conceptually simpler (but incredibly inefficient) scheme in your homework!

All these schemes actually have a common *blueprint*. We start by building a succinct proof system where the security is *information theoretic* (i.e., we don't need any cryptographic assumptions). Because standard NP proofs are not succinct, this requires working in some weird model that constrains how the verifier and prover interact (you can think of this a bit like the random oracle model, where we assume access to some imaginary object that helps us prove security). So we first build a succinct proof system that is unconditionally secure in this weird computational model.

Then, cryptography comes into mix. We use cryptographic tools to force the prover to adhere to the constraints, without having the ability to cheat. Then we further apply Fiat-Shamir to make the whole thing non-interactive. And we're done! Sounds easy, right...

The box game. Let's take a little detour first, and consider the following problem: The verifier wants to test whether some statement x is in an NP language $\mathcal{L}$. The prover puts their proof π in a special "box", that lets the verifier see the value of π at only three randomly chosen bits. The verifier can then decide whether to accept or reject the proof.



Is this possible?

At first glance, this seems impossible. You're only allowed to see three bits of the proof! That's a bit like saying that I can grade your exam by only reading three random lines you wrote. And yet, it turns out that this is possible, if we encode the proof in a special way!

This follows from one of the most celebrated results in theoretical computer science, the PCP theorem [ALM+98] (PCP stands for Probabilistically Checkable Proofs). The PCP theorem says, informally, that you can create a proof π for any NP statement (where the length of π is polynomial in the size of the instance), so that you can verify the proof by reading only 3 random bits of it [Hås01]! The proof has perfect completeness, and soundness error of at most $1/2$ (i.e., incorrect proofs are accepted with probability at most $1/2$).

There's a flavor of succinctness that is apparent in the PCP theorem. For a proof of size n, the verifier only needs to sample $O(\log n)$ random bits, and receives $O(1)$ bits in return, from which they can decide whether to accept the proof.

So we now have an information-theoretic proof system that has some succinctness properties, but where the prover and verifier interact through some special "box". The second step is to use cryptography to actually instantiate this "box".

From PCPs to SNARGs. The prover could just send the verifier the entire PCP proof, and have the verifier sample three random bits from it. But this defeats the point of succinctness, as the PCP proof is as long (actually much longer) than the original witness.

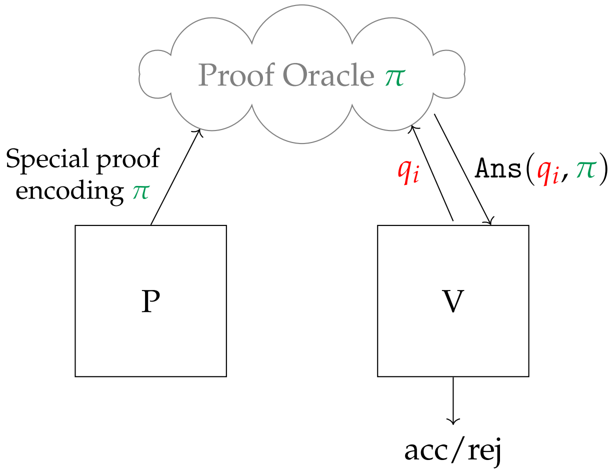
Another way would be for the verifier to send the prover the three random indices, but then we somehow need to ensure that the prover doesn't cheat by returning arbitrary values. This is where cryptography comes in.

How do we instantiate a "box" in cryptography? This should remind you of *commitment schemes*! But to get succinctness as in the PCP theorem, we'll need a special form of commitment scheme called a *vector commitment scheme*. This allows the prover to send a *short* commitment to an entire vector of values $x = (x_1, x_2, \ldots, x_n)$, and later let the verifier *open* the commitment at a specific index i to obtain x_i. But wait, didn't you construct such a scheme in homework 1? What a great coincidence! In this week's homework, you'll show how to use this commitment scheme to create a (very inefficient) SNARG.

2.1 Abstracting the Box Game: Interactive Oracle Proofs

Formally, we refer to the "box" in the above example as an *oracle*, which you can think of as a trusted third party that takes as input a proof string from the prover, and allows the verifier to ask only specific queries to the proof. Such proof systems are called *Interactive Oracle Proofs* (IOPs) [BSCS16].

Again, note that IOPs are information-theoretic proofs. That is, we first show that in a special model where this oracle exists, the proof system is sound and complete. Then, we use cryptography (typically a special form of commitment scheme) to instantiate the oracle.



Beyond PCPs, other types of commonly used IOPs include:

- *Linear IOPs*, where the proof is encoded as a *vector* π in some field $\mathbb{F}$, and the verifier asks for a constant number of *linear combinations* of the proof values (i.e., $< q, \pi >= \sum_{i \in [n]} q_i \cdot \pi_i$ for some vector $q \in \mathbb{F}^n$).

We will get back to linear IOPs in a later lecture, when we discuss protocols for aggregating private statistics.

- *Polynomial IOPs*, where the proof is encoded as a *polynomial* $\pi(X)$ over some field $\mathbb{F}$, and the verifier asks for *polnomial evaluations* $\pi(z)$ at a constant number of points $z \in \mathbb{F}$.

  We will discuss Polynomial IOPs in more detail below, to describe PLONK, a modern practical SNARG.

3 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.

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 do this (or don't want to).

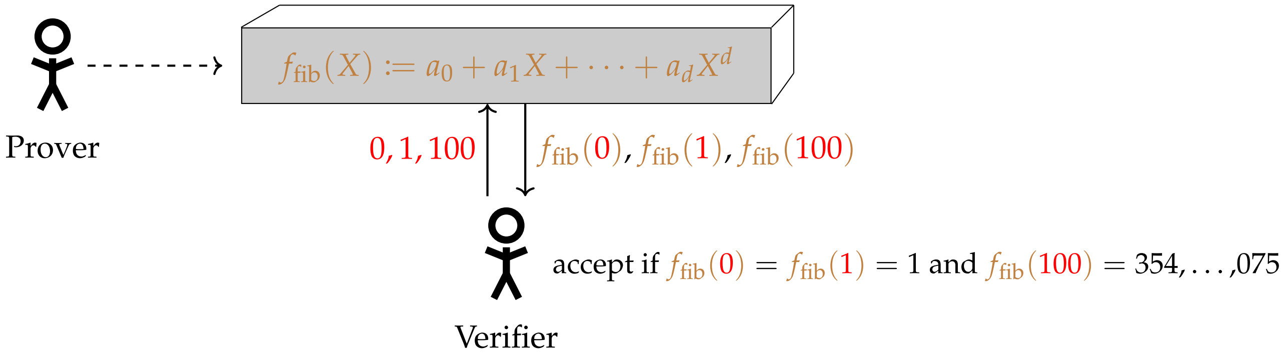
To convince you, I create a polynomial $f_{\text{fib}}(X)$ over some (large) field $\mathbb{F}_p$ and claim that:

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?

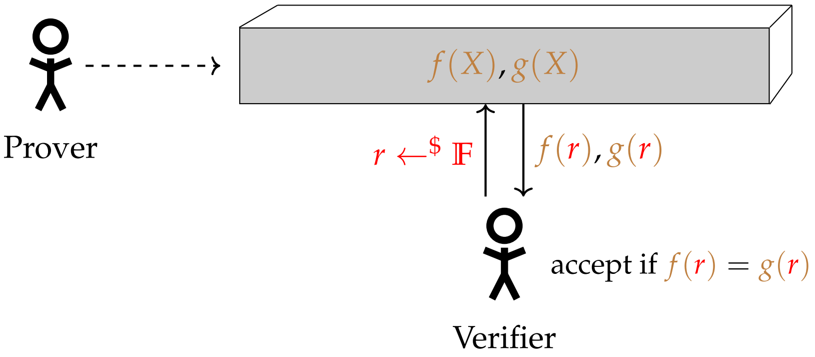
Testing that $f_{\text{fib}}(0) = f_{\text{fib}}(1) = 1$ and that $f_{\text{fib}}(100) = 354{,}224{,}848{,}179{,}261{,}915{,}075$ is easy. You just ask the box for an evaluation of f_{fib} at 0, 1, and 100.


Prover
$f_{\text{fib}}(X) := a_0 + a_1 X + \cdots + a_d X^d$
$0, 1, 100$
$f_{\text{fib}}(0), f_{\text{fib}}(1), f_{\text{fib}}(100)$
accept if $f_{\text{fib}}(0) = f_{\text{fib}}(1) = 1$ and $f_{\text{fib}}(100) = 354, \ldots, 075$
Verifier

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 d-degree polynomials $f, g \in \mathbb{F}[X]^{\leq d}$ 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.



Lemma 1. The Equality Test protocol has perfect completeness and soundness error $d/|\mathbb{F}|$.

Proof of Lemma 1. Completeness is trivial. For soundness, we use the single most important fact about polynomials that you should remember: they can't have too many roots!

Theorem 1 (Fundamental theorem of algebra)**.** Any non-zero polynomial over $\mathbb{F}$ of degree d has at most d roots over $\mathbb{F}$.

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. □

Note that Theorem 1 can also be extended to multivariate polynomials, and is often used in this form in cryptographic applications.

Lemma 2 (Schwartz-Zippel lemma)**.** Let $f(X_1, \ldots, X_n) \in \mathbb{F}[X_1, \ldots, X_n]$ be a non-zero polynomial of total degree[a] d. Let Ω be a finite subset of $\mathbb{F}$ and let $r_1, \ldots, r_n$ be uniformly random elements of Ω. Then,

$$\Pr\left[f(r_1, \ldots, r_n) = 0\right] \leq \frac{d}{|\Omega|} .$$

[a] The total degree of a polynomial is the maximum of the degrees of the monomials in the polynomial. The degree of a monomial is the sum of the degrees of the variables in the monomial. (e.g., the polynomial $3X^2Y^3Z + 2XY^2$ has total degree $2 + 3 + 1 = 6$.)

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

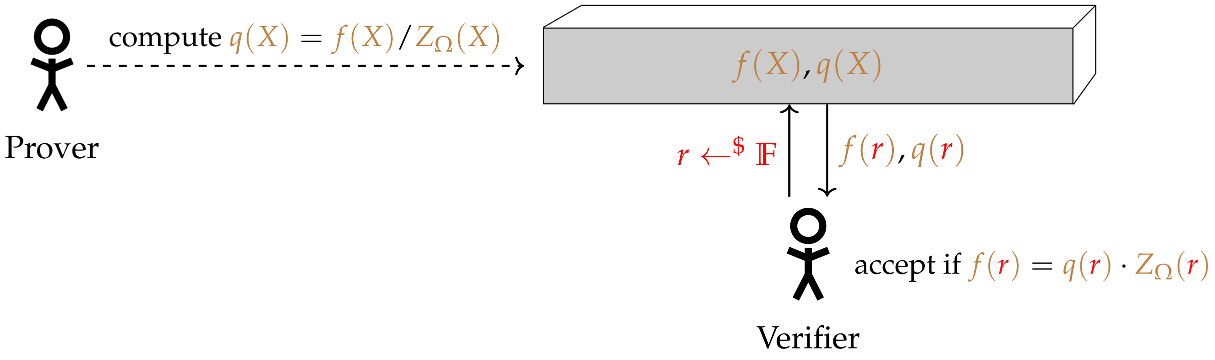
$x > 98$ we don't care if the equality holds or not). We can recast this problem as showing that the polynomial $f_{\text{fib}}(X+2) - f_{\text{fib}}(X) - f_{\text{fib}}(X+1)$ is zero on the set $\{0, 1, \ldots, 98\}$.

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,

Theorem 2. 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 $f(X) = \underbrace{(X-1) \cdot (X-2)}_{Z_\Omega(X)} \cdot (X+3)$.

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:

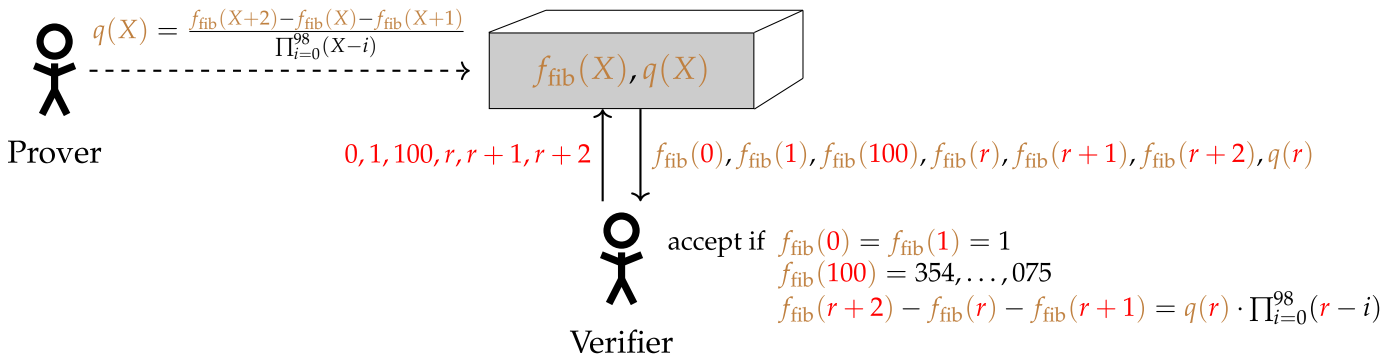

compute $q(X) = f(X)/Z_\Omega(X)$
Prover
$f(X), q(X)$
$r \xleftarrow{\$} \mathbb{F}$
$f(r), q(r)$
accept if $f(r) = q(r) \cdot Z_\Omega(r)$
Verifier

Lemma 3. The ZeroTest protocol has perfect completeness and soundness error $2d/|\mathbb{F}|$.

Proof of Lemma 2. Completeness is again trivial.

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:


$q(X) = \frac{f_{\text{fib}}(X+2) - f_{\text{fib}}(X) - f_{\text{fib}}(X+1)}{\prod_{i=0}^{98}(X-i)}$
Prover
$f_{\text{fib}}(X), q(X)$
$0, 1, 100, r, r+1, r+2$
$f_{\text{fib}}(0), f_{\text{fib}}(1), f_{\text{fib}}(100), f_{\text{fib}}(r), f_{\text{fib}}(r+1), f_{\text{fib}}(r+2), q(r)$
accept if $f_{\text{fib}}(0) = f_{\text{fib}}(1) = 1$
$f_{\text{fib}}(100) = 354, \ldots, 075$
$f_{\text{fib}}(r+2) - f_{\text{fib}}(r) - f_{\text{fib}}(r+1) = q(r) \cdot \prod_{i=0}^{98}(r-i)$
Verifier

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.

4 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 $f(x) = y$ for arbitrary points x.

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 $f \in \mathbb{F}[X]^{\leq d}$.
- $\texttt{Open}(\texttt{pp}, f, x) \to \pi$, outputs a proof π for the evaluation $y = f(x)$.
- $\texttt{Verify}(\texttt{pp}, c, x, y, \pi) \to \{0,1\}$, checks the proof π for the evaluation point (x, 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 $x \in \mathbb{F}$, we have:

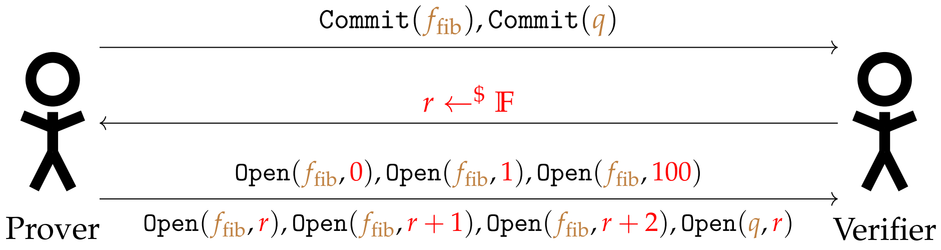
$$\Pr\left[\texttt{Verify}(\texttt{pp}, c, x, f(x), \pi) = 1 \ : \ \begin{array}{r} \texttt{pp} \leftarrow \texttt{Setup}(d) \\ c \leftarrow \texttt{Commit}(\texttt{pp}, f) \\ \pi \leftarrow \texttt{Open}(\texttt{pp}, f, x) \end{array}\right] = 1\,.$$

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

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

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.


$\texttt{Commit}(f_{\text{fib}}), \texttt{Commit}(q)$
$r \xleftarrow{\$} \mathbb{F}$
$\texttt{Open}(f_{\text{fib}}, 0), \texttt{Open}(f_{\text{fib}}, 1), \texttt{Open}(f_{\text{fib}}, 100)$
$\texttt{Open}(f_{\text{fib}}, r), \texttt{Open}(f_{\text{fib}}, r+1), \texttt{Open}(f_{\text{fib}}, r+2), \texttt{Open}(q, r)$
Prover
Verifier

Doesn't this look like a Sigma protocol? (hint: it does!) So we can turn the entire protocol into a non-interactive proof in the random oracle model by using the Fiat-Shamir heuristic.

5 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* $\texttt{NP}$ statements: PLONK [GWC19]. When combined with a Polynomial Commitment Scheme and the Fiat Shamir heuristic, the PLONK IOP becomes a SNARG for $\texttt{NP}$. (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.)

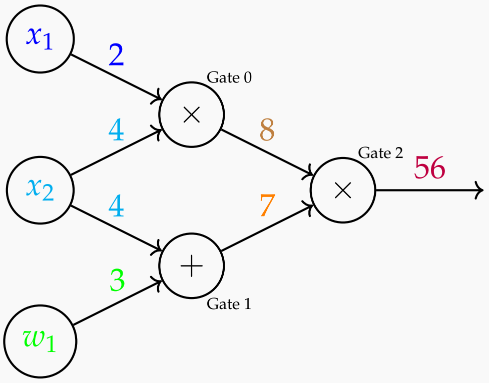
Arithmetic Circuits. The type of statements we'll consider will be represented as arithmetic circuits.

Definition 2 (Arithmetic Circuit)**.** An arithmetic circuit $\mathcal{C} : \mathbb{F}^l \to \mathbb{F}$ over a field $\mathbb{F}$ is a circuit consisting of binary gates applying the operations $\times, +$ over $\mathbb{F}$. (a standard Boolean circuit is a special case where $\mathbb{F} = \mathbb{Z}_2$, and the gates correspond to AND and XOR operations).

For clarity, we will view our arithmetic circuit as taking two different inputs: a *public* input $x \in \mathbb{F}^m$ (the statement, known to the verifier) and a *private* witness $w \in \mathbb{F}^n$. The goal of the prover is to convince the verifier that $\mathcal{C}(x, w) = 0$, i.e., that x is in the language $\mathcal{L} = \{x \in \mathbb{F}^m \mid \exists w \in \mathbb{F}^n : \mathcal{C}(x, w) = 0\}$. Our goal is to build a *succinct non-interactive* proof system for this language. Specifically, the proof string should be of length $O(\lambda, \text{polylog}(|\mathcal{C}|))$ and the verifier should run in time $O(\lambda, |x|, \text{polylog}(|\mathcal{C}|))$.

5.1 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.



$(x_1 \times x_2) \times (x_2 + w_1) \Rightarrow$

Inputs:	2	4	3
Gate 0:	2	4	8
Gate 1:	4	3	7
Gate 2:	8	7	56
	Left	Right	Out

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. $f_{\textbf{trace}}$ **encodes all inputs**: $f_{\text{trace}}(\omega^{-j})$ = input #j for $j = 1, \ldots, |I|$.
2. $f_{\textbf{trace}}$ **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:	$f_{\text{trace}}(\omega^{6}) = 8$	$f_{\text{trace}}(\omega^{7}) = 7$	$f_{\text{trace}}(\omega^{8}) = 56$

5.2 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![3] 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. Here, the special structure of the set Ω becomes crucial: it turns out that $Z_\Omega(X) := X^d - 1$, which can be evaluated in time $O(\log d)$ using repeated squaring (the polynomial Z_{Ω_I} has a similar structure). As a result, computing the vanishing polynomial is not a bottleneck for the verifier.

(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	$\boxed{56}$	0	($\times$)

Then, we want to check that:

$$\forall x \in \Omega_{\text{gates}} := \{\omega^0, \omega^3, \omega^6, \omega^9, ..., \omega^{3(|\mathcal{C}|-1)}\} :$$

$$f_{\text{sel}}(x) \times [\underbrace{f_{\text{trace}}(x)}_{\text{Left input}} + \underbrace{f_{\text{trace}}(\omega x)}_{\text{Right input}}] + (1 - f_{\text{sel}}(x)) \times [\underbrace{f_{\text{trace}}(x)}_{\text{Left input}} \times \underbrace{f_{\text{trace}}(\omega x)}_{\text{Right input}}] = \underbrace{f_{\text{trace}}(\omega^2 x)}_{\text{Output}} .$$

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.

[3] Why doesn't the verifier just query f_{trace} on all points in Ω_I? While this would be sound, it would ultimately result in a SNARG that doesn't satisfy our definition of succinctness, as the proof would grow linearly with the size of the public input x. But we want the proof size to be $O(\lambda, \text{polylog}|\mathcal{C}|)$, *independent* of x. The verifier's *running time* is allowed to grow with x, but not the proof size.

So if we are going to re-use the circuit $\mathcal{C}$ many times, we can do a one-time expensive *preprocessing* 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: $f_{\mathrm{trace}}(\omega^{-1}) = f_{\mathrm{trace}}(\omega^0)$, $f_{\mathrm{trace}}(\omega^{-2}) = f_{\mathrm{trace}}(\omega^1) = f_{\mathrm{trace}}(\omega^3)$
- Gate outputs: $f_{\mathrm{trace}}(\omega^2) = f_{\mathrm{trace}}(\omega^6)$, $f_{\mathrm{trace}}(\omega^5) = f_{\mathrm{trace}}(\omega^7)$

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

6 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 $f_{\mathrm{trace}} - f_{\mathrm{in}}$ and to the quotient polynomial for the ZeroTest. The verifier queries these polynomials on random points.
3. The prover computes the quotient polynomial of $f_{\mathrm{gates}}(X)$ 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

[ALM$^+$98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 326–349. Association for Computing Machinery (ACM), 2012.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112. Association for Computing Machinery (ACM), 1988.

[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.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II 14*, pages 31–60. Springer, 2016.

[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.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[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.