CS251 Fall 2023

(cs251.stanford.edu)

Building a SNARK

Dan Boneh

Recap: zk-SNARK applications

Private Tx on a public blockchain: Zcash, IronFish

Compliance:

- Proving that a private Tx are in compliance with banking laws
- Proving solvency in zero-knowledge

Scalability: privacy in a zk-SNARK Rollup (next week)

Bridging between blockchains: zkBridge

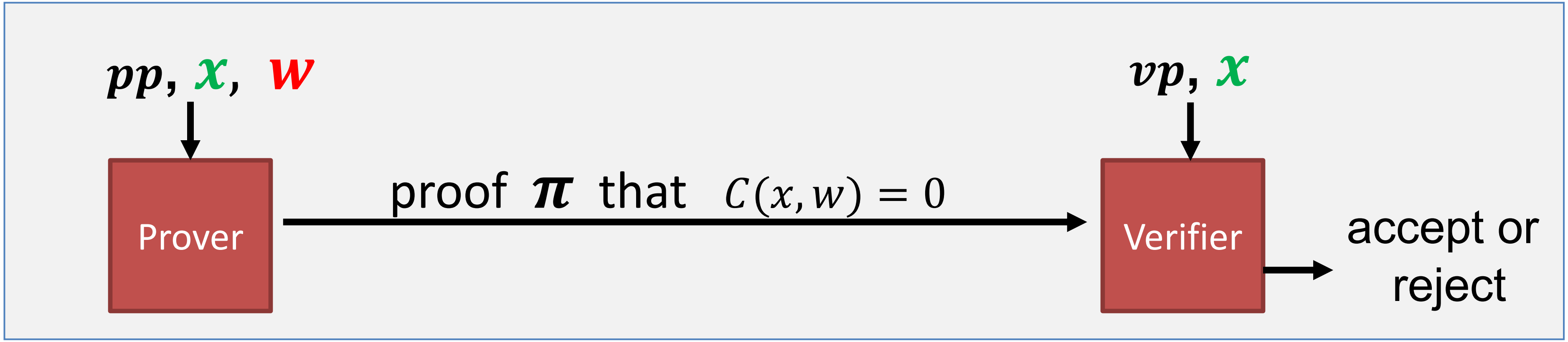
(preprocessing) NARK: Non-interactive ARgument of Knowledge

Public arithmetic circuit: $C(\boldsymbol{x}, \boldsymbol{w}) \rightarrow \mathbb{F}$

public statement in $\mathbb{F}^n$

secret witness in $\mathbb{F}^m$

Preprocessing (setup): $\mathbf{S}(C) \rightarrow$ public parameters $(\boldsymbol{pp}, \boldsymbol{vp})$



$\boldsymbol{pp}, \boldsymbol{x}, \boldsymbol{w}$

Prover

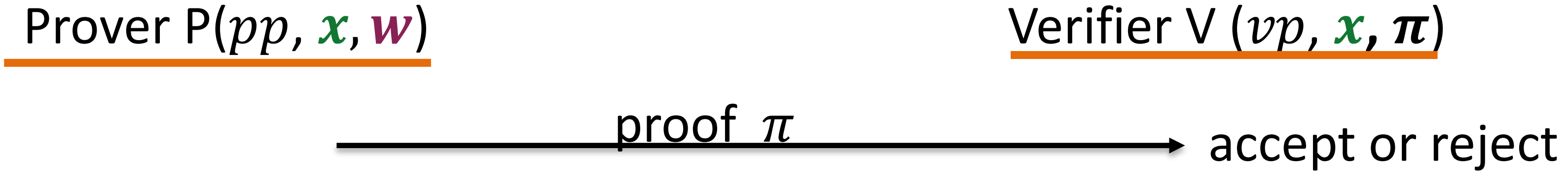
proof $\boldsymbol{\pi}$ that $C(x, w) = 0$

$\boldsymbol{vp}, \boldsymbol{x}$

Verifier

accept or reject

NARK: requirements (informal)


Prover P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$)

Verifier V (vp, $\boldsymbol{x}$, $\boldsymbol{\pi}$)

proof π

accept or reject

Complete: $\forall x, w$: $C(\boldsymbol{x}, \boldsymbol{w}) = 0$ ⇒ Pr[V(vp, x, P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$)) = accept] = 1

Adaptively **knowledge sound**: V accepts ⇒ P “knows” $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

(an extractor E can extract a valid $\boldsymbol{w}$ from P)

Optional: **Zero knowledge**: $(C, pp, vp, \boldsymbol{x}, \pi)$ “reveal nothing new” about $\boldsymbol{w}$

(witness exists ⇒ can simulate the proof)

SNARK: a Succinct ARgument of Knowledge

A **succinct preprocessing NARK** is a triple (S, P, V):

- **S**(C) → public parameters (pp, vp) for prover and verifier
- **P**$(pp, \boldsymbol{x}, \boldsymbol{w})$ → **short** proof π ; $\boxed{\text{len}(\pi) = O_\lambda(\,\textbf{polylog}(|\boldsymbol{C}|))}$
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi})$ **fast to verify** ; $\boxed{\text{time(V)} = O_\lambda(|x|,\ \textbf{polylog}(|\boldsymbol{C}|))}$

short “summary” of circuit

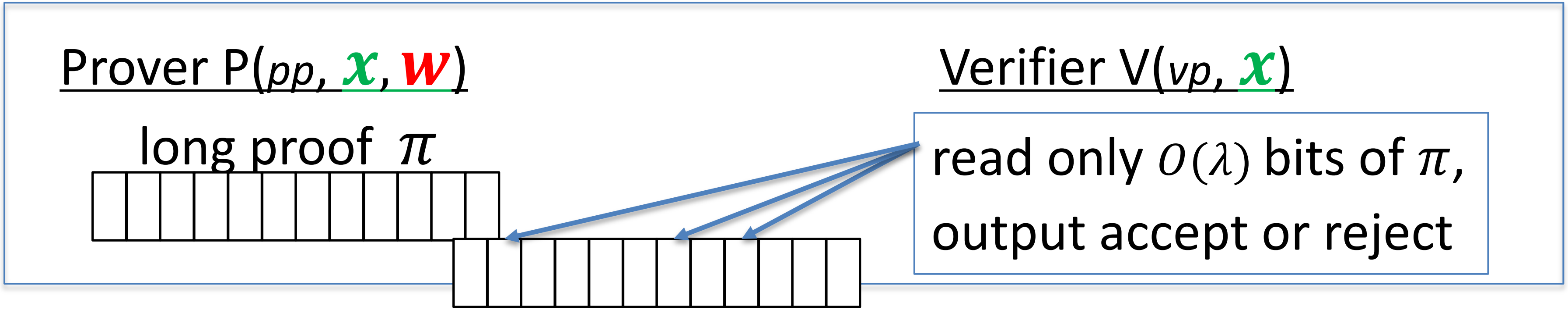
A simple PCP-based SNARK

[Kilian’92, Micali’94]

A simple construction: PCP-based SNARK

The PCP theorem: Let $C(x, w)$ be an arithmetic circuit.

there is a proof system that for every x proves $\exists w: C(x, w) = 0$ as follows:



Prover P(*pp*, $\boldsymbol{x}$, $\boldsymbol{w}$)

long proof π

Verifier V(*vp*, $\boldsymbol{x}$)

read only $O(\lambda)$ bits of π, output accept or reject

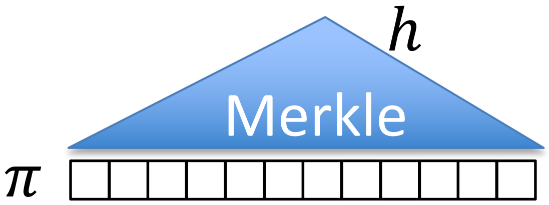
V always accepts valid proof. If no w, then V rejects with high prob.

size of proof π is $poly(|C|)$. (not succinct)

Converting a PCP proof to a SNARK

Prover P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$)

Verifier V(vp, $\boldsymbol{x}$)



Merkle root h → 1 hash

← open k positions of π $(k = O(\lambda))$

→ $O(k \log |C|)$ hashes

k opening and Merkle proofs

output accept or reject

Verifier sees $O(\lambda \log |C|)$ data $\Rightarrow$ succinct proof.

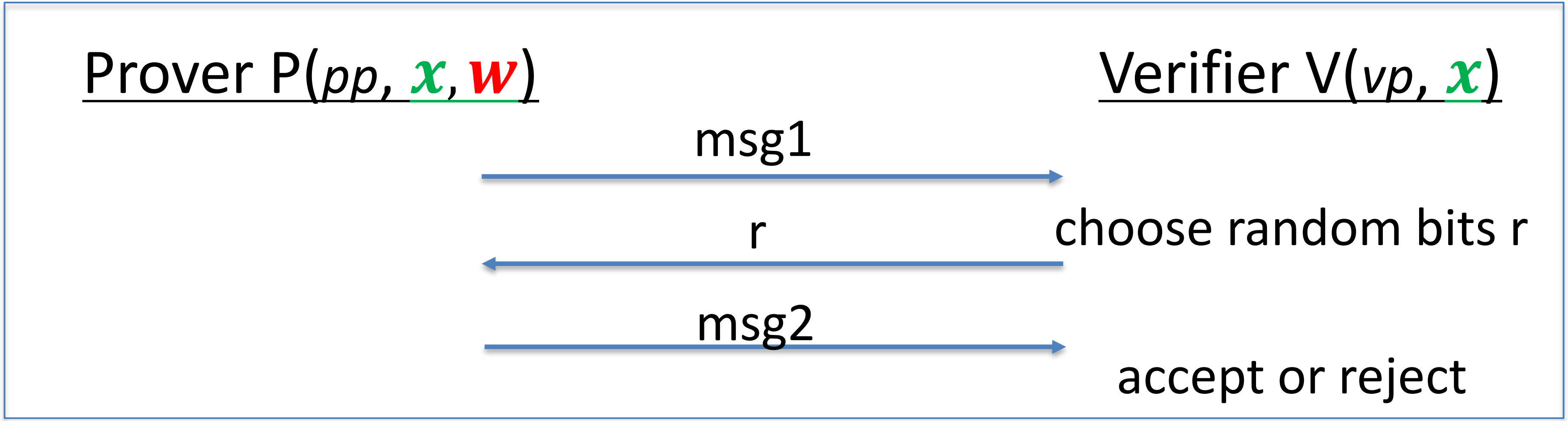
Problem: **interactive**

Making the proof non-interactive

The **Fiat-Shamir transform**:

- public-coin interactive protocol ⇒ non-interactive protocol

  public coin: all verifier randomness is public (no secrets)



Prover P(pp, x, w)

Verifier V(vp, x)

msg1

r

choose random bits r

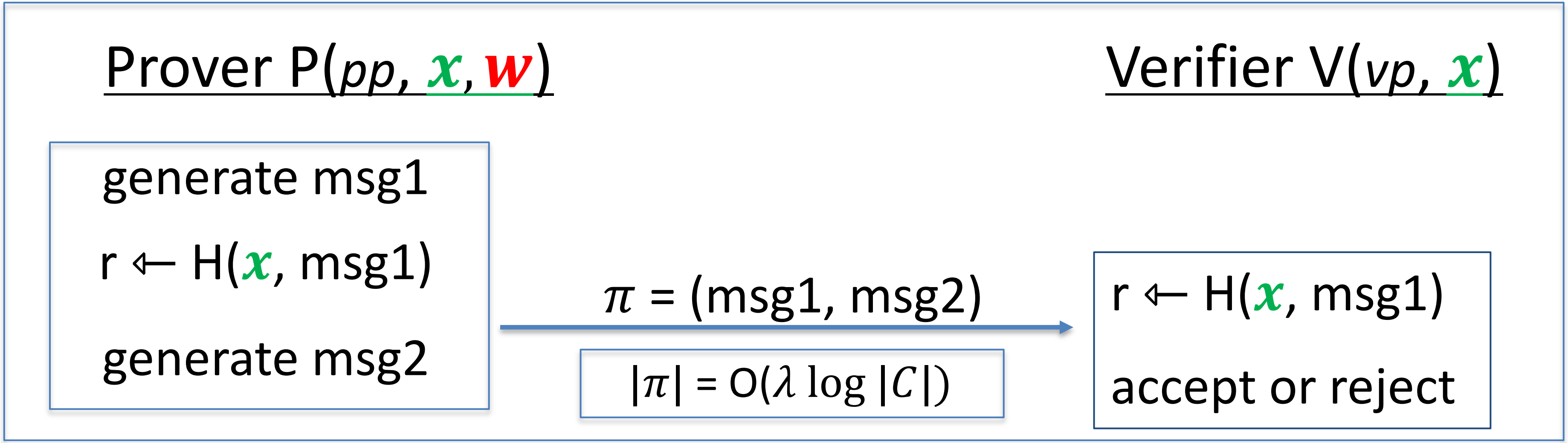
msg2

accept or reject

Making the proof non-interactive

Fiat-Shamir transform: $H : M \to R$ a cryptographic hash function

- idea: prover generates random bits on its own (!)



Prover P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$)

Verifier V(vp, $\boldsymbol{x}$)

generate msg1

r ⟵ H($\boldsymbol{x}$, msg1)

generate msg2

π = (msg1, msg2)

$|\pi|$ = O($\lambda \log |C|$)

r ⟵ H($\boldsymbol{x}$, msg1)

accept or reject

Fiat-Shamir: certain secure interactive protocols ⟹ non-interactive

Are we done?

Simple transparent SNARK from the PCP theorem

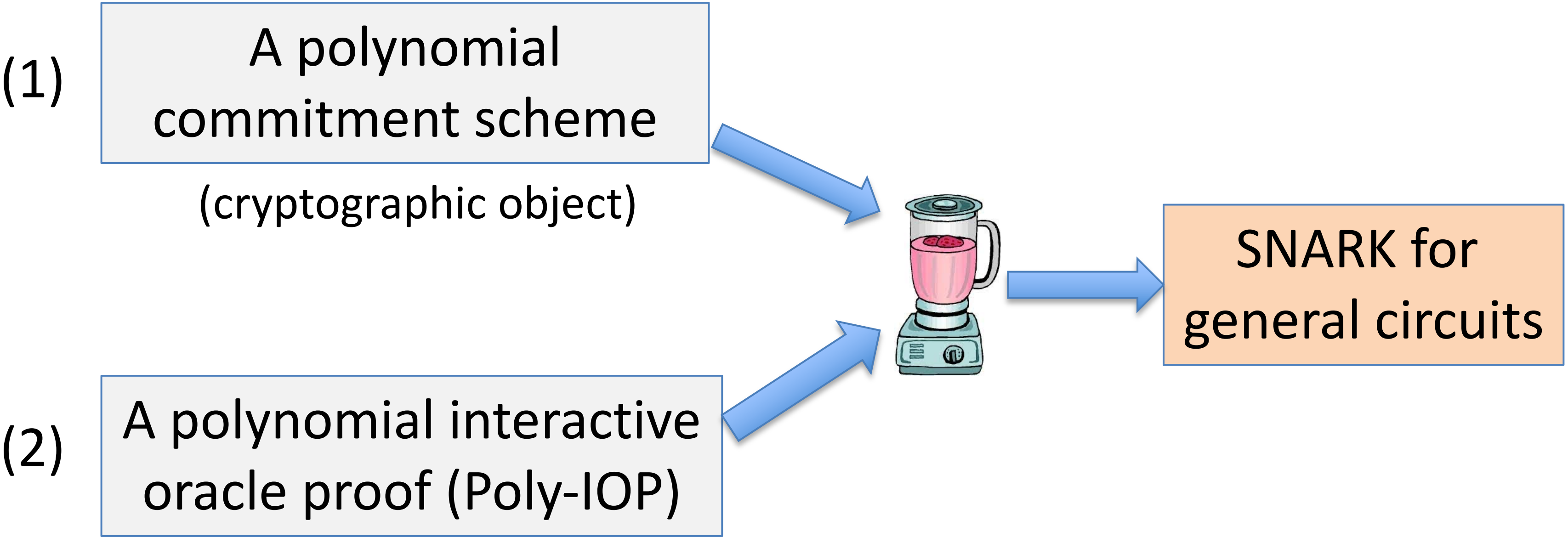
- Use Fiat-Shamir transform to make non-interactive
- We will apply Fiat-Shamir in many other settings

The bad news: an impractical SNARK --- Prover time too high

Better SNARKs: Goal: Time(Prover) = $\tilde{O}(|C|)$

Building an efficient SNARK

General paradigm: two steps



(1) A polynomial commitment scheme

(cryptographic object)

(2) A polynomial interactive oracle proof (Poly-IOP)

(info. theoretic object)

SNARK for general circuits

Let’s explain each concept …

Recall: commitments

Two algorithms:

- *commit*(m, r) → ***com*** (r chose at random)
- *verify*$(m,$ ***com***$, r)$ → accept or reject

Properties:

- **binding**: cannot produce two valid openings for ***com***
- **hiding**: ***com*** reveals nothing about committed data

(1) Polynomial commitment scheme (PCS)

Notation: $\mathbb{F}_p^{(\leq d)}[X]$ is all polynomials in $\mathbb{F}_p[X]$ of degree ≤ d.

Prover commits to a polynomial $f(X)$ in $\mathbb{F}_p^{(\leq d)}[X]$ (univariate)

- ***eval***: for public $u, v \in \mathbb{F}_p$, prover can convince the verifier that committed poly satisfies

$$\boxed{f(u) = v \quad \text{and} \quad \deg(f) \leq d.}$$

verifier has (d, com_f, u, v)

- Eval proof size and verifier time should be $O_\lambda(\log \boldsymbol{d})$

$\boxed{f}$

(1) Polynomial commitment scheme (PCS)

- *setup*(d) → *pp*,   public parameters for polynomials of degree ≤ d

- *commit*(*pp*, f, r) → $\boldsymbol{com_f}$ commitment to f ∈ $\mathbb{F}_p^{(\leq d)}[X]$

- *eval*:   goal: for a given $\boldsymbol{com_f}$ and x, y ∈ $\mathbb{F}_p$, prove that f(x) = y.

Formally: *eval* = (S, P, V) is a SNARK for:

statement *st* = (*pp*, $\boldsymbol{com_f}$, x, y) with witness = w = (f, r)

where $C(st, w) = 0$ iff

$$\left[\ f(x) = y \quad \textit{and} \quad f \in \mathbb{F}_p^{(\leq d)}[X] \quad \textit{and} \quad \textit{commit}(pp, f, r) = \boldsymbol{com}_{\mathrm{f}}\ \right]$$

(1) Polynomial commitment scheme (PCS)

Properties:

- Binding: cannot produce two valid openings $(f_{1,} r_1)$, $(f_{2,} r_2)$ for $\boldsymbol{com_f}$.
- eval is knowledge sound (can extract (f, r) from a successful prover)
- optional:
  - commitment is hiding
  - eval is zero knowledge

Note: poly. commitments have many applications beyond SNARKs

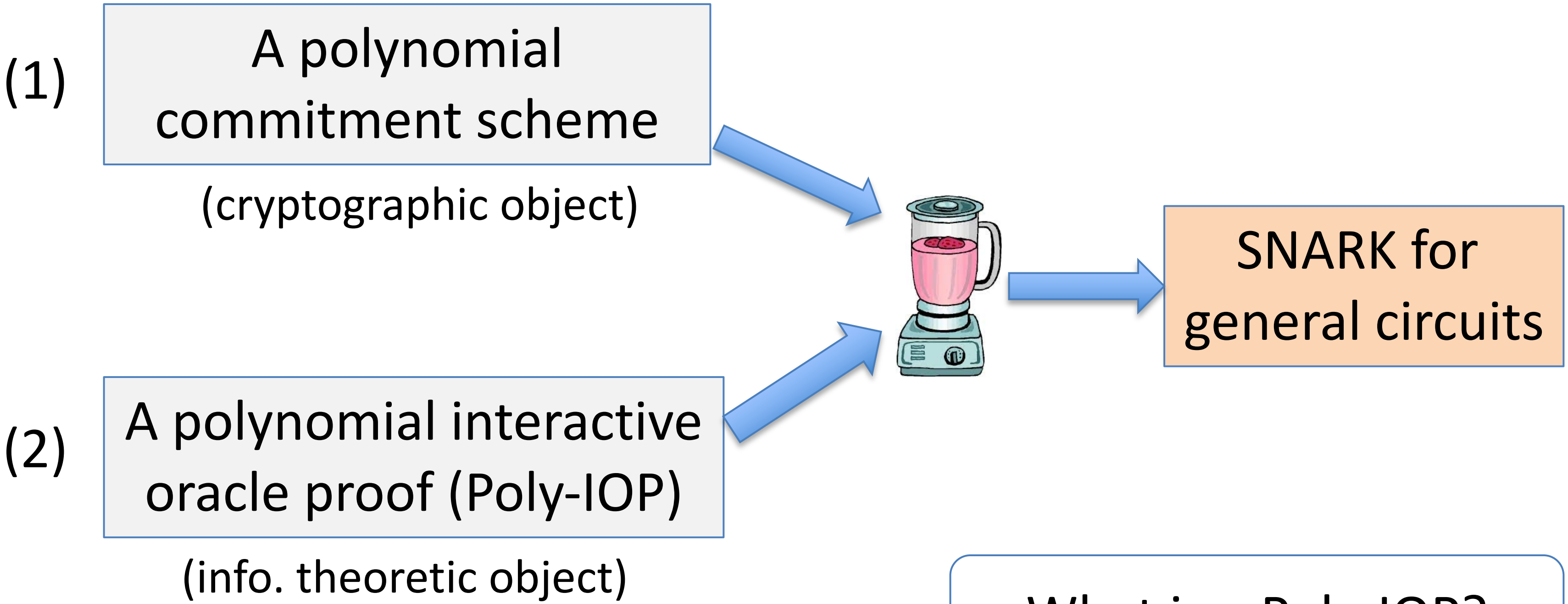
Constructing a PCS

Not today … (see readings or CS355)

Properties of the most widely used in practice (called KZG) :

- trusted setup: secret randomness in setup. $\boxed{|pp| = O_\lambda(d)}$
- ***com***$_f$: constant size (one group element)
- eval proof size: constant size (one group element)
- eval verify time: constant time. Prover time: $\boxed{O_\lambda(d)}$

General paradigm: two steps



(1) A polynomial commitment scheme

(cryptographic object)

(2) A polynomial interactive oracle proof (Poly-IOP)

(info. theoretic object)

SNARK for general circuits

What is a Poly-IOP?

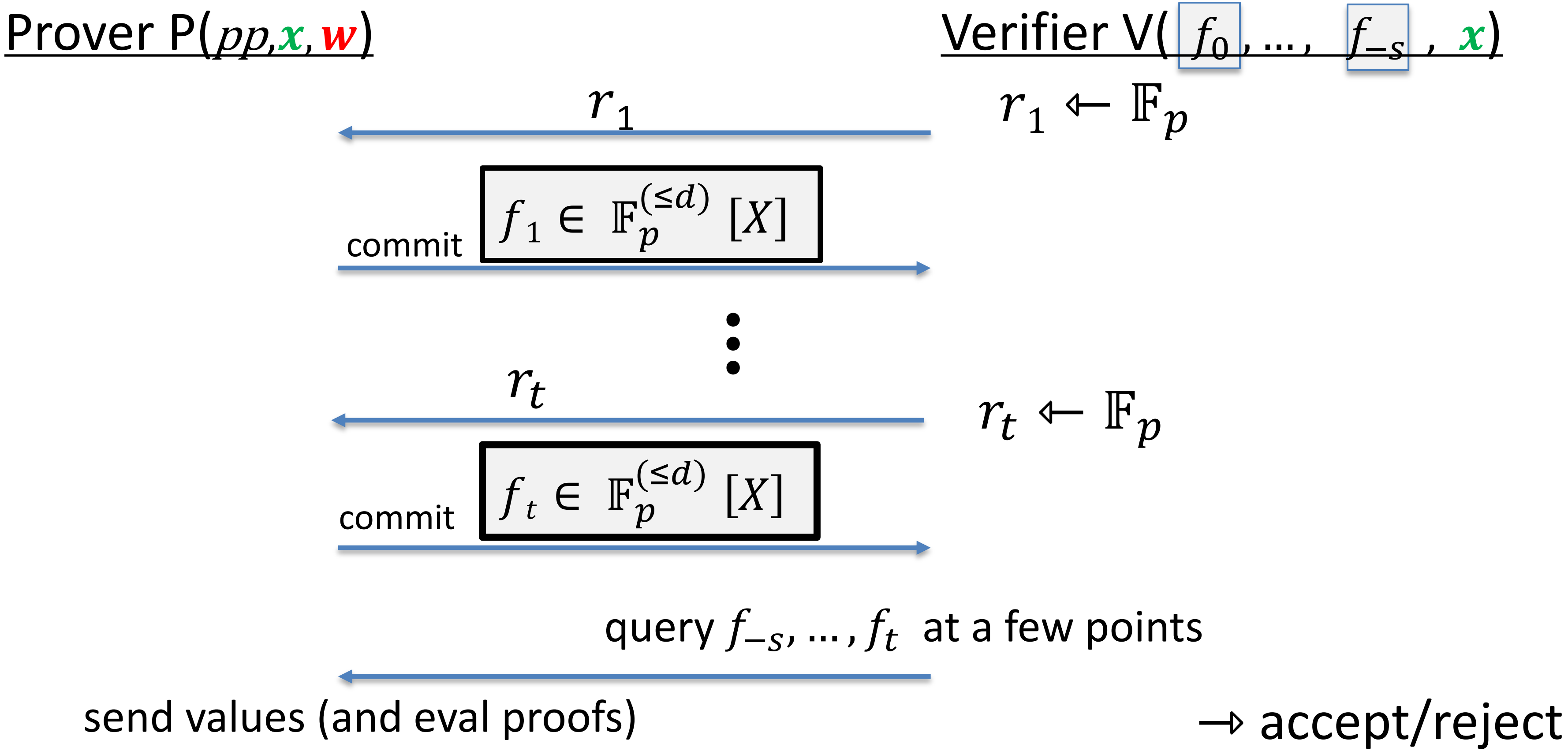
Component 2: Polynomial IOP

Let $C(x, w)$ be some arithmetic circuit. Let $x \in \mathbb{F}_p^n$.

Poly-IOP: a proof system that proves $\exists w: C(x, w) = 0$ as follows:

Setup(C) → public parameters $\boldsymbol{pp}$ and $\boldsymbol{vp}$ = ($\boxed{f_0}$, $\boxed{f_{-1}}$, ... , $\boxed{f_{-s}}$)

Polynomial IOP



Prover P(pp,$\boldsymbol{x}$,$\boldsymbol{w}$)

Verifier V($\boxed{f_0}$, … , $\boxed{f_{-s}}$, $\boldsymbol{x}$)

r_1

$r_1 \leftarrow \mathbb{F}_p$

commit $\boxed{f_1 \in \mathbb{F}_p^{(\le d)}[X]}$

$\vdots$

r_t

$r_t \leftarrow \mathbb{F}_p$

commit $\boxed{f_t \in \mathbb{F}_p^{(\le d)}[X]}$

query $f_{-s}, \dots, f_t$ at a few points

send values (and eval proofs)

→ accept/reject

The Plonk poly-IOP

Goal: construct a poly-IOP called ***Plonk*** (eprint/2019/953)

[Gabizon – Williamson – Ciobotaru]

Plonk + PCS ⇒ SNARK

(and also a zk-SNARK)

[PCS = Polynomial Commitment Scheme]

Proving properties of committed polynomials

Goal: succinct proofs

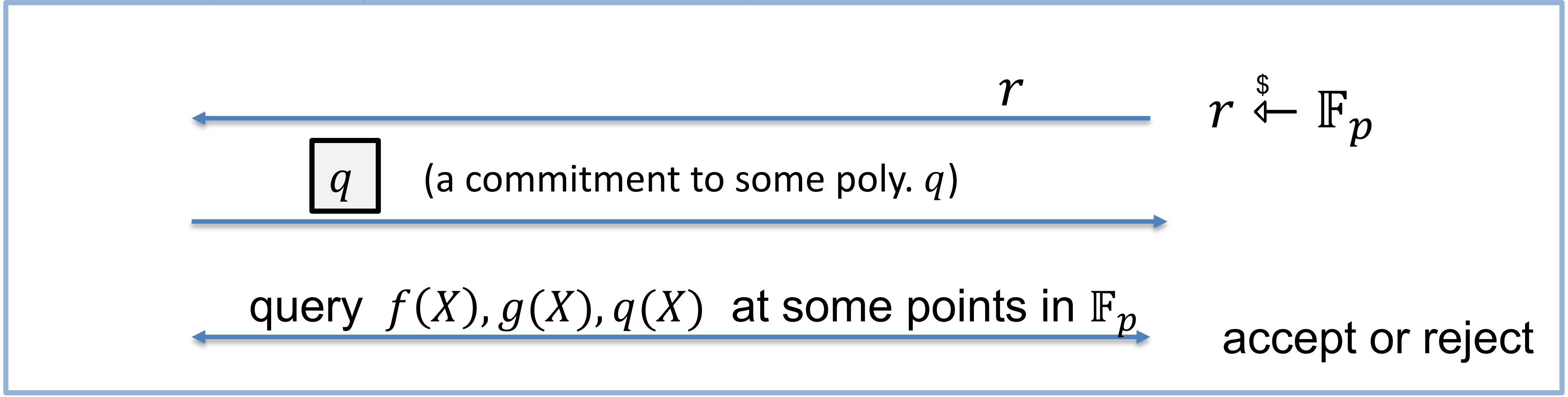
Proving properties of committed polynomials

Prover P(f, g)

Verifier V($\boxed{f}$, $\boxed{g}$)

Goal: convince verifier that $f, g \in \mathbb{F}_p^{(\leq d)}[X]$ satisfy some properties

Proof systems presented as a Poly-IOP:



r

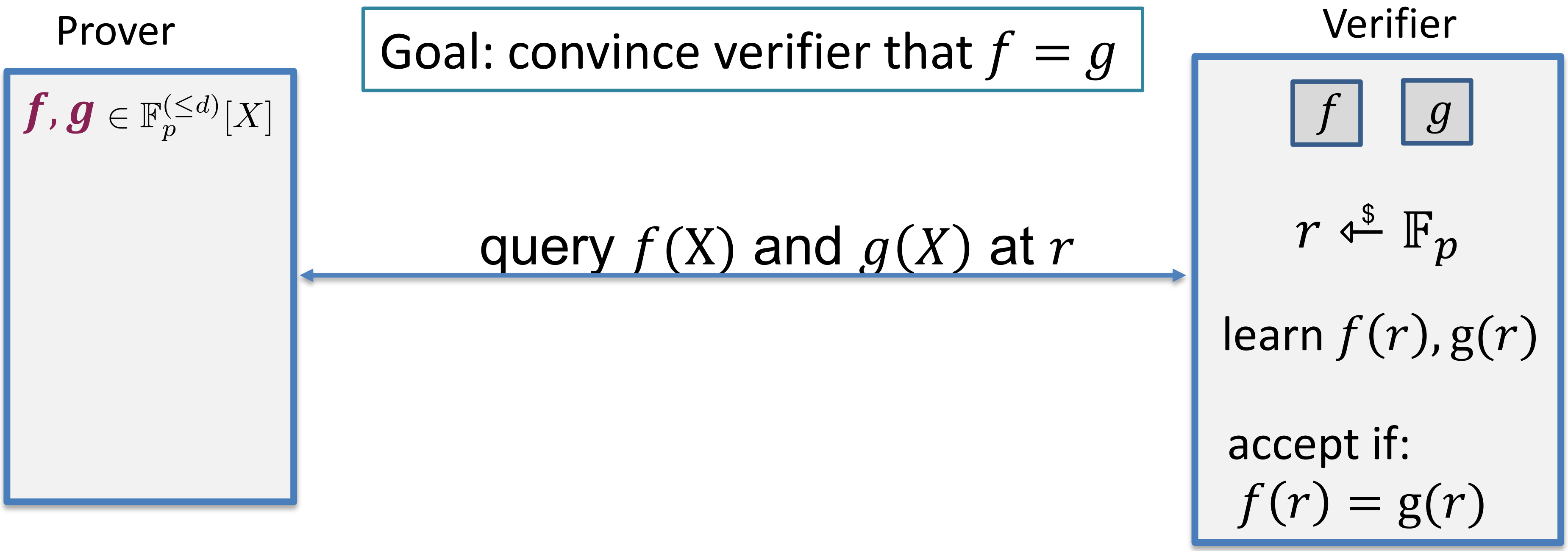
$r \xleftarrow{\$} \mathbb{F}_p$

$\boxed{q}$ (a commitment to some poly. q)

query $f(X), g(X), q(X)$ at some points in $\mathbb{F}_p$

accept or reject

A simple example: polynomial equality testing



Prover

$f, g \in \mathbb{F}_p^{(\leq d)}[X]$

Goal: convince verifier that $f = g$

query $f(X)$ and $g(X)$ at r

Verifier

$\boxed{f}$ $\boxed{g}$

$r \xleftarrow{\$} \mathbb{F}_p$

learn $f(r), g(r)$

accept if:
$f(r) = g(r)$

Why is this sound?

Why is this sound?

A key fact: for non-zero $f \in \mathbb{F}_p^{(\leq d)}[X]$

$$\boxed{\text{for } r \leftarrow \mathbb{F}_p : \quad \Pr[\, f(r) = 0 \,] \leq d/p} \qquad (*)$$

$\Rightarrow$ suppose $p \approx 2^{256}$ and $d \leq 2^{40}$ then d/p is negligible

$\Rightarrow$ for $r \leftarrow \mathbb{F}_p$: if $f(r) = 0$ then f is identically zero w.h.p

$\Rightarrow$ a simple test if a committed poly. is the zero poly.

SZDL lemma: $(*)$ also holds for **multivariate** polynomials (where d is total degree of f)

Why is this sound?

Suppose $p \approx 2^{256}$ and $d \leq 2^{40}$ so that d/p is negligible

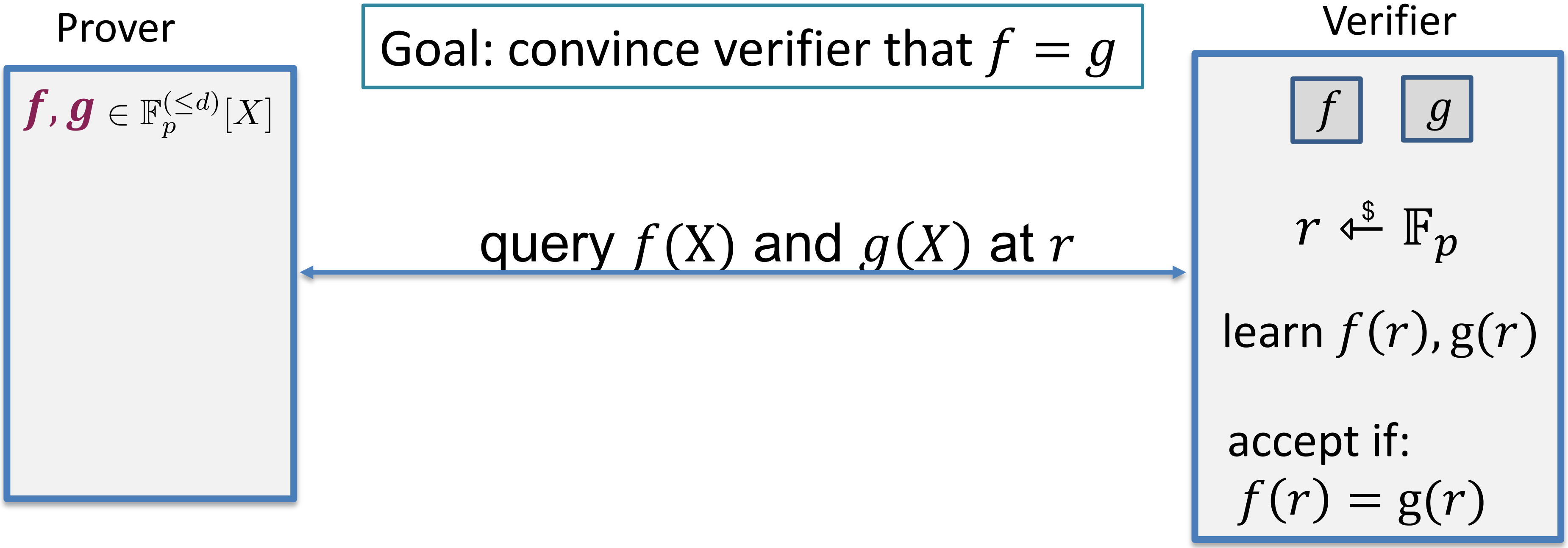
Let $f,\ g \in \mathbb{F}_p^{(\leq d)}[X]$.

For $r \leftarrow \mathbb{F}_p$, if $f(r) = g(r)$ then $f = g$ w.h.p

$f(r) - g(r) = 0 \quad \Rightarrow \quad f - g = 0$ w.h.p

$\Rightarrow$ a simple equality test for two committed polynomials

The polynomial equality testing protocol



Prover

Goal: convince verifier that $f = g$

$\boldsymbol{f}, \boldsymbol{g} \in \mathbb{F}_p^{(\leq d)}[X]$

query $f(X)$ and $g(X)$ at r

Verifier

$\boxed{f}$ $\boxed{g}$

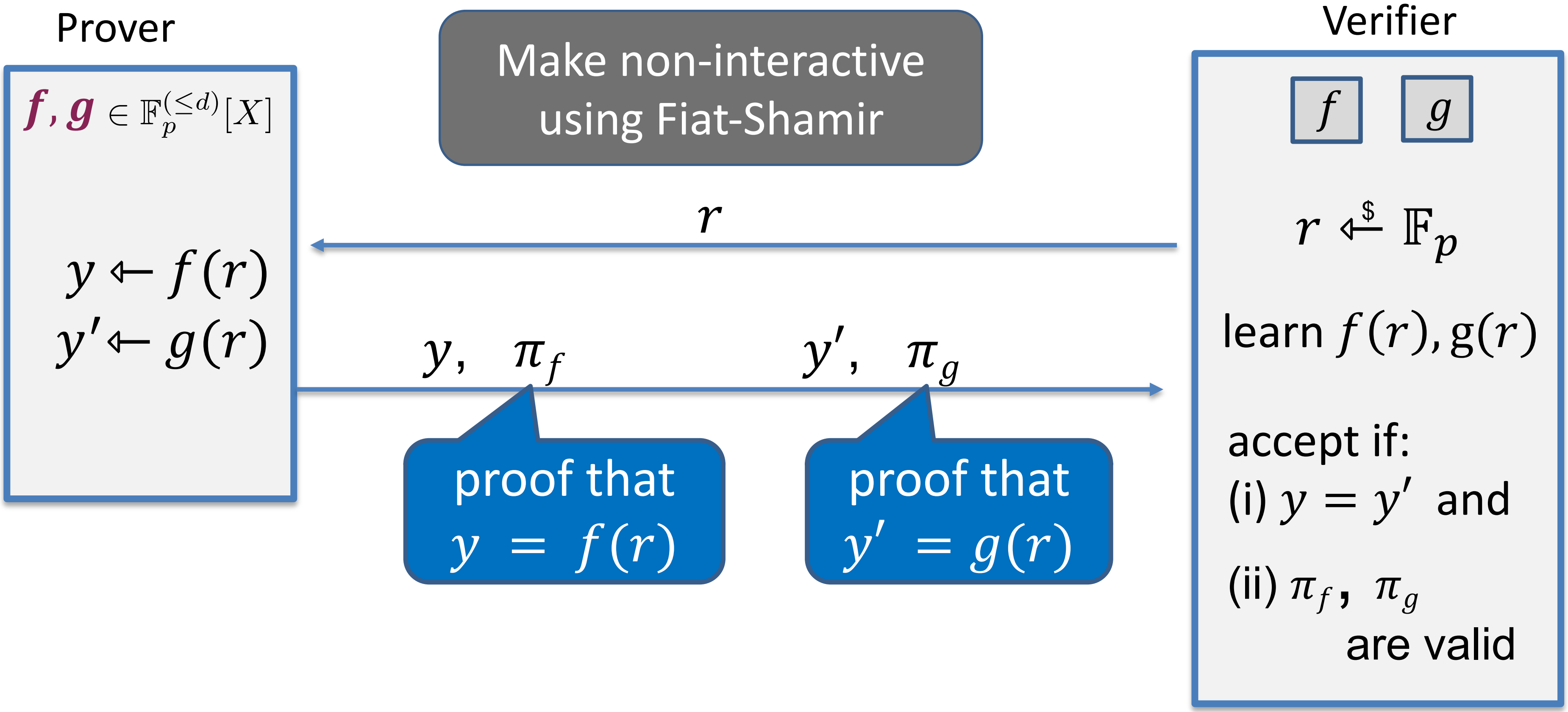
$r \xleftarrow{\$} \mathbb{F}_p$

learn $f(r), \mathrm{g}(r)$

accept if:
$f(r) = \mathrm{g}(r)$

Lemma: complete and sound assuming d/p is negligible

Review: the compiled proof system



Prover

$f, g \in \mathbb{F}_p^{(\leq d)}[X]$

$y \leftarrow f(r)$

$y' \leftarrow g(r)$

Make non-interactive using Fiat-Shamir

r

$y, \quad \pi_f$

$y', \quad \pi_g$

proof that $y = f(r)$

proof that $y' = g(r)$

Verifier

$\boxed{f}$ $\boxed{g}$

$r \xleftarrow{\$} \mathbb{F}_p$

learn $f(r), g(r)$

accept if:

(i) $y = y'$ and

(ii) $\pi_f, \ \pi_g$ are valid

Important proof gadgets for univariates

Let Ω be some subset of $\mathbb{F}_p$ of size k.

Let $f \in \mathbb{F}_p^{(\leq d)}[X]$ $(d \geq k)$ Verifier has $\boxed{f}$

Let us construct efficient Poly-IOPs for the following tasks:

Task 1 (**ZeroTest**): prove that f is identically zero on Ω

Task 2 (**SumCheck**): prove that $\sum_{a \in \Omega} f(a) = 0$

Task 3 (**ProdCheck**): prove that $\prod_{a \in \Omega} f(a) = 1$

The vanishing polynomial

Let Ω be some subset of $\mathbb{F}_p$ of size k.

Def: the **vanishing polynomial** of Ω is $Z_\Omega(X) \coloneqq \prod_{a\in\Omega}(X-a)$

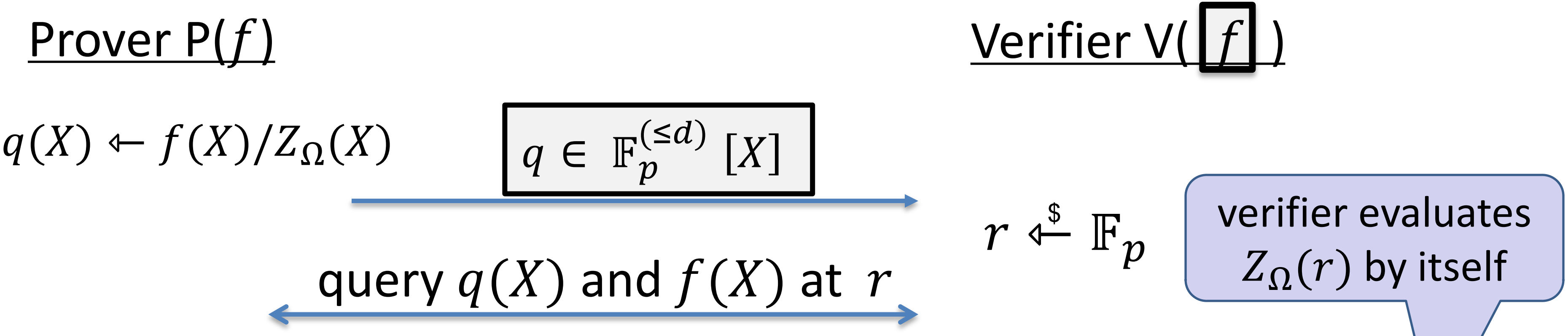
$\deg(Z_\Omega) = k$

Let $\omega \in \mathbb{F}_p$ be a primitive k-th root of unity (so that $\omega^k = 1$).

- if $\Omega = \{1, \omega, \omega^2, \ldots, \omega^{k\text{-}1}\} \subseteq \mathbb{F}_p$ then $Z_\Omega(X) = X^k - 1$

$\Rightarrow$ for $r \in \mathbb{F}_p$, evaluating $Z_\Omega(r)$ takes $2\log_2 k$ field operations

(1) ZeroTest on Ω $\quad$ ($\Omega = \{ 1, \omega, \omega^2, \ldots, \omega^{k\text{-}1} \}$)



Prover P(f)

Verifier V($\boxed{f}$)

$q(X) \leftarrow f(X)/Z_\Omega(X)$

$\boxed{q \in \mathbb{F}_p^{(\leq d)}[X]}$

$r \xleftarrow{\$} \mathbb{F}_p$

verifier evaluates $Z_\Omega(r)$ by itself

query $q(X)$ and $f(X)$ at r

learn $q(r)$, $f(r)$

accept if $f(r) \overset{?}{=} q(r) \cdot Z_\Omega(r)$

(implies that $f(X) = q(X) \cdot Z_\Omega(X)$ w.h.p)

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

Thm: this protocol is complete and sound, assuming d/p is negligible.

(1) ZeroTest on Ω ($\Omega = \{ 1, \omega, \omega^2, \ldots, \omega^{k\text{-}1} \}$)

Prover P(f)

Verifier V($\boxed{f}$)

$q(X) \leftarrow f(X)/Z_\Omega(X)$

$\boxed{q \in \mathbb{F}_p^{(\leq d)}[X]}$ →

$r \xleftarrow{\$} \mathbb{F}_p$

verifier evaluates $Z_\Omega(r)$ by itself

query $q(X)$ and $f(X)$ at r ↔

learn $q(r)$, $f(r)$

accept if $f(r) \stackrel{?}{=} q(r) \cdot Z_\Omega(r)$

(implies that $f(X) = q(X) \cdot Z_\Omega(X)$ w.h.p)

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

Verifier time: O(log k) and two poly queries (but can be batched)

Prover time: dominated by time to compute $q(X)$ and commit to $q(X)$

(4) Another useful gadget: permutation check

Let f, g polynomials in $\mathbb{F}_p^{(\leq d)}[X]$. Verifier has $\boxed{f}$, $\boxed{g}$.

Prover wants to prove that $\left(f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{k-1}) \right) \in \mathbb{F}_p^k$

is a permutation of $\left(g(1), g(\omega), g(\omega^2), \ldots, g(\omega^{k-1}) \right) \in \mathbb{F}_p^k$

$\Rightarrow$ Proves that $g(\Omega)$ is the same as $f(\Omega)$, just permuted

(4) Another useful gadget: permutation check

Prover P(f, g) Verifier V($\boxed{f}$, $\boxed{g}$)

Let $\hat{f}(X) = \prod_{a \in \Omega}(X - f(a))$ and $\hat{g}(X) = \prod_{a \in \Omega}(X - g(a))$

Then: $\hat{f}(X) = \hat{g}(X) \iff g(\Omega)$ is a permutation of $f(\Omega)$

$$\xleftarrow{\quad r \quad} \qquad r \overset{\$}{\leftarrow} \mathbb{F}_p$$

prove that $\hat{f}(r) = \hat{g}(r)$

prod-check: $\frac{\hat{f}(r)}{\hat{g}(r)} = \prod_{a \in \Omega} \left(\frac{r - f(a)}{r - g(a)} \right) = 1$

$$\longleftrightarrow$$

implies $\hat{f}(X) = \hat{g}(X)$ w.h.p

accept or reject

[Lipton's trick, 1989]

(5) final gadget: prescribed permutation check

$W: \Omega \rightarrow \Omega$ is a **permutation of** Ω if $\forall i \in [k]$: $W(\omega^i) = \omega^j$ a bijection

example $(k = 3)$: $W(\omega^0) = \omega^2$, $W(\omega^1) = \omega^0$, $W(\omega^2) = \omega^1$

Let f, g polynomials in $\mathbb{F}_p^{(\leq d)}[X]$. Verifier has $\boxed{f}$, $\boxed{g}$, $\boxed{W}$.

<u>Goal</u>: prover wants to prove that $f(y) = g(W(y))$ for all $y \in \Omega$

$\Rightarrow$ Proves that $g(\Omega)$ is the same as $f(\Omega)$, permuted by the prescribed W

Prescribed permutation check

How? Use a zero-test to prove $\boxed{f(y) - g\big(W(y)\big) = 0 \quad \text{on } \Omega}$

The problem: the polynomial $f(y) - g\big(W(y)\big)$ has degree k^2

⇒ prover would need to manipulate polynomials of degree k^2

⇒ quadratic time prover !! (goal: linear time prover)

Can reduce this to a prod-check on a poly of degree $2k$ (not k^2)

Summary of proof gadgets

polynomial equality testing

zero test on Ω

product check, sum check

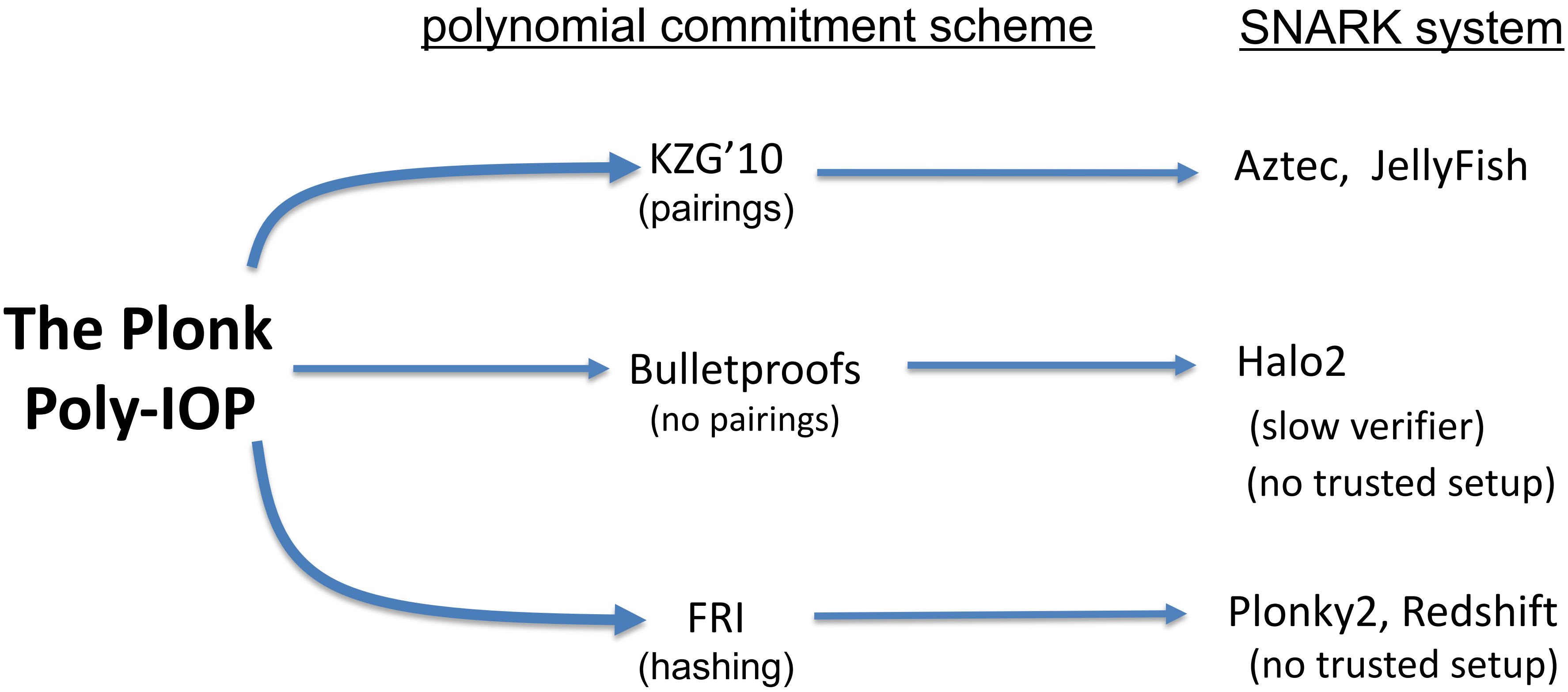
permutation check

prescribed permutation check

The PLONK Poly-IOP for general circuits

eprint/2019/953

PLONK: widely used in practice



polynomial commitment scheme

SNARK system

The Plonk Poly-IOP

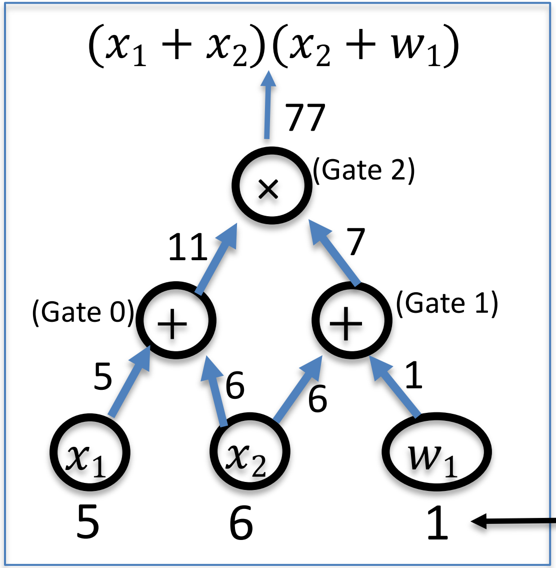
KZG'10 (pairings) → Aztec, JellyFish

Bulletproofs (no pairings) → Halo2 (slow verifier) (no trusted setup)

FRI (hashing) → Plonky2, Redshift (no trusted setup)

PLONK: a poly-IOP for a general circuit $C(x, w)$

Step 1: compile circuit to a computation trace (gate fan-in = 2)



example input

The computation trace (arithmetization):

inputs:	5,	6,	1
Gate 0:	5 ,	6 ,	11
Gate 1:	6 ,	1 ,	7
Gate 2:	11,	7,	77
	left inputs	right inputs	outputs

Encoding the trace as a polynomial

$|C|$:= total # of gates in C , $|I| := |I_x| + |I_w| =$ # inputs to C

let $d := 3\,|C| + |I|$ (in example, $d = 12$) and $\Omega := \{\, 1, \omega, \omega^2, \ldots, \omega^{d-1} \,\}$

The plan:

prover interpolates a poly. $T \in \mathbb{F}_p^{(\leq d)}[X]$

that encodes the entire trace.

Let's see how …

inputs:	5,	6,	1
Gate 0:	5 ,	6 ,	11
Gate 1:	6 ,	1 ,	7
Gate 2:	11,	7,	77

Encoding the trace as a polynomial

The plan: Prover interpolates $T \in \mathbb{F}_p^{(\leq d)}$[X] such that

(1) ***T* encodes all inputs**: T$(\omega^{-j}) =$ input #j for j = 1, …, $|I|$

(2) ***T* encodes all wires**: $\forall\, l = 0, \ldots, |C| - 1$:

- T(ω^{3l}): left input to gate #l
- T(ω^{3l+1}): right input to gate #l
- T(ω^{3l+2}): output of gate #l

inputs:	5,	6,	1
Gate 0:	5 ,	6 ,	11
Gate 1:	6 ,	1 ,	7
Gate 2:	11,	7,	77

Encoding the trace as a polynomial

In our example, Prover interpolates $T(X)$ such that:

inputs:	$\mathrm{T}(\omega^{-1}) = 5,$	$\mathrm{T}(\omega^{-2}) = 6,$	$\mathrm{T}(\omega^{-3}) = 1,$
gate 0:	$\mathrm{T}(\omega^{0}) = 5,$	$\mathrm{T}(\omega^{1}) = 6,$	$\mathrm{T}(\omega^{2}) = 11,$
gate 1:	$\mathrm{T}(\omega^{3}) = 6,$	$\mathrm{T}(\omega^{4}) = 1,$	$\mathrm{T}(\omega^{5}) = 7,$
gate 2:	$\mathrm{T}(\omega^{6}) = 11,$	$\mathrm{T}(\omega^{7}) = 7,$	$\mathrm{T}(\omega^{8}) = 77$

degree(T) = 11

inputs:	5,	6,	1
Gate 0:	5 ,	6 ,	11
Gate 1:	6 ,	1 ,	7
Gate 2:	11,	7,	77

Prover can use FFT to compute the coefficients of T in time $O(d \log d)$

Step 2: proving validity of T

Prover P(S_p, $\boldsymbol{x}$, $\mathbf{w}$)

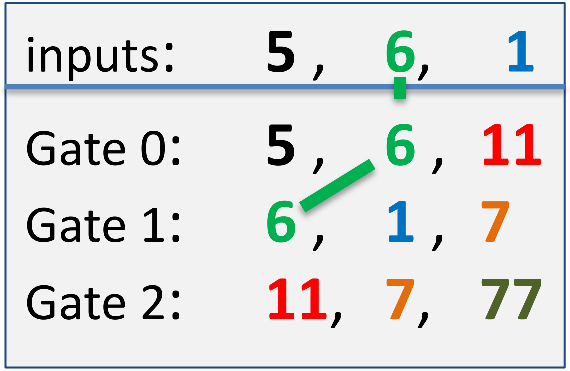
Verifier V(S_v, $\boldsymbol{x}$)

build T(X) ∈ $\mathbb{F}_p^{(\leq d)}$[X] $\xrightarrow{\boxed{T}}$

Prover needs to prove that T is a correct computation trace:

1. T encodes the correct inputs,
2. every gate is evaluated correctly,
3. the wiring is implemented correctly,
4. the output of last gate is 0

(wiring constraints)



inputs:	5 ,	6,	1
Gate 0:	5 ,	6 ,	11
Gate 1:	6 ,	1 ,	7
Gate 2:	11,	7,	77

Proving (4) is easy: prove $T(\omega^{3|C|-1}) = 0$

Proving (1): T encodes the correct inputs

Both prover and verifier interpolate a polynomial $v(X) \in \mathbb{F}_p^{(\leq|I_x|)}$[X] that encodes the x-inputs to the circuit:

for $j = 1, \ldots, |I_x|$: $v(\omega^{-j})$ = input #j

In our example: $v(\omega^{-1}) = 5$, $v(\omega^{-2}) = 6$. (v is linear)

constructing $v(X)$ takes time proportional to the size of input x

⇒ verifier has time do this

Proving (1): T encodes the correct inputs

Both prover and verifier interpolate a polynomial $v(X) \in \mathbb{F}_p^{(\leq|I_x|)}[X]$ that encodes the x-inputs to the circuit:

$$\text{for } j = 1, \ldots, |I_x|: \qquad v(\omega^{-j}) = \text{input \#j}$$

Let $\Omega_{\text{inp}} := \{\, \omega^{-1}, \omega^{-2}, \ldots, \omega^{-|I_x|} \,\} \subseteq \Omega$ (points encoding the input)

Prover proves (1) by using a ZeroTest on Ω_{inp} to prove that

$$\boxed{\mathrm{T}(y) - v(y) = 0 \qquad \forall\, y \in \Omega_{\text{inp}}}$$

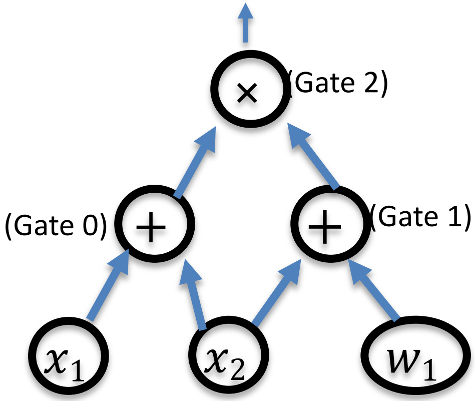
Proving (2): every gate is evaluated correctly

Idea: encode gate types using a *selector* polynomial S(X)

define S(X) ∈ $\mathbb{F}_p^{(\le d)}$[X] such that $\forall\, l = 0, \dots, |C| - 1$:

S(ω^{3l}) = 1 if gate #l is an addition gate

S(ω^{3l}) = 0 if gate #l is a multiplication gate



inputs:	5 ,	6,	1	$S(X)$	
Gate 0 (ω^0):	5 ,	6 ,	11	1	(+)
Gate 1 (ω^3):	6 ,	1 ,	7	1	(+)
Gate 2 (ω^6):	11,	7,	77	0	(×)

Proving (2): every gate is evaluated correctly

Idea: encode gate types using a *selector* polynomial S(X)

define S(X) $\in \mathbb{F}_p^{(\leq d)}[X]$ such that $\forall\, l = 0, \dots, |C| - 1$:

S(ω^{3l}) = 1 if gate #l is an addition gate

S(ω^{3l}) = 0 if gate #l is a multiplication gate

Then $\forall$ y $\in \Omega_{\text{gates}} := \{\, 1, \omega^3, \omega^6, \omega^9, \dots, \omega^{3(|C|-1)} \,\}$:

$$S(y)\cdot[\underbrace{T(y)}_{\text{left input}} + \underbrace{T(\omega y)}_{\text{right input}}] \;+\; (1 - S(y))\cdot\underbrace{T(y)}_{\text{left input}}\cdot\underbrace{T(\omega y)}_{\text{right input}} \;=\; \underbrace{T(\omega^2 y)}_{\text{output}}$$

Proving (2): every gate is evaluated correctly

$$\text{Setup}(C) \;\to\; pp{:=}\text{S} \quad \text{and} \quad vp{:=}\,(\,\boxed{\text{S}}\,)$$

Prover $\text{P}(pp, \boldsymbol{x}, \mathbf{w})$

Verifier $\text{V}(vp, \boldsymbol{x})$

build $\text{T}(X) \in \mathbb{F}_p^{(\leq d)}[\text{X}]$ $\xrightarrow{\quad\boxed{T}\quad}$

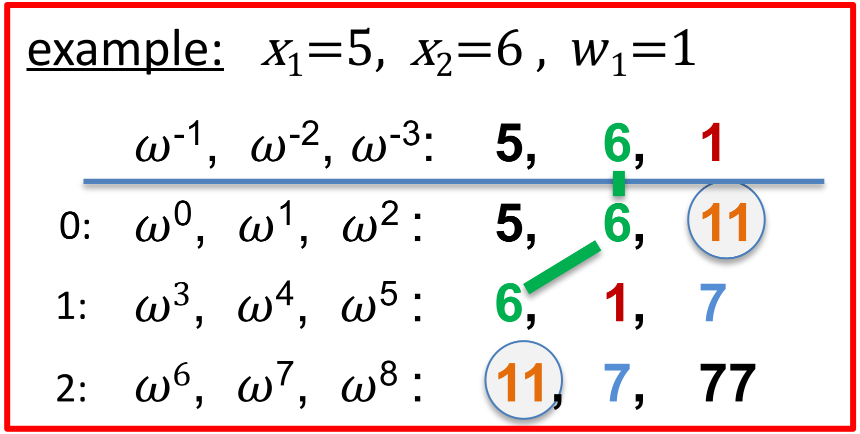
Prover uses ZeroTest to prove that for all $\forall\, y \in \Omega_{gates}$:

$$\text{S(y)}\cdot[\text{T(y)} + \text{T}(\omega\text{y})] \;+\; (1 - \text{S(y)})\cdot\text{T(y)}\cdot\text{T}(\omega\text{y}) \;-\; \text{T}(\omega^2\text{y}) \;=\; 0$$

Proving (3): the wiring is correct

Step 4: encode the wires of C:

$$\begin{cases} T(\omega^{-2}) = T(\omega^{1}) = T(\omega^{3}) \\ T(\omega^{-1}) = T(\omega^{0}) \\ T(\omega^{2}) = T(\omega^{6}) \\ T(\omega^{-3}) = T(\omega^{4}) \end{cases}$$



<u>example:</u> $x_1{=}5,\ x_2{=}6\ ,\ w_1{=}1$

	$\omega^{-1},\ \omega^{-2},\ \omega^{-3}$:	5,	6,	1
0:	$\omega^{0},\ \omega^{1},\ \omega^{2}$:	5,	6,	11
1:	$\omega^{3},\ \omega^{4},\ \omega^{5}$:	6,	1,	7
2:	$\omega^{6},\ \omega^{7},\ \omega^{8}$:	11,	7,	77

Define a polynomial W: $\Omega \to \Omega$ that implements a rotation:

$$W(\omega^{-2}, \omega^{1}, \omega^{3}) = (\omega^{1}, \omega^{3}, \omega^{-2}) \ , \quad W(\omega^{-1}, \omega^{0}) = (\omega^{0}, \omega^{-1}) \ , \ \ldots$$

<u>Lemma</u>: $\forall\, y \in \Omega$: $T(y) = T(W(y))$ $\Rightarrow$ wire constraints are satisfied

Proving (3): the wiring is correct

Step 4: encode the wires of C:

$$\begin{cases} T(\omega^{-2}) = T(\omega^{1}) = T(\omega^{3}) \\ T(\omega^{-1}) = T(\omega^{0}) \\ T(\omega^{2}) = T(\omega^{6}) \end{cases}$$

example: $x_1=5,\ x_2=6,\ w_1=1$

	$\omega^{-1}, \omega^{-2}, \omega^{-3}$:	5,	6,	1
0:	$\omega^{0}, \omega^{1}, \omega^{2}$:	5,	6,	11
1:	$\omega^{3}, \omega^{4}, \omega^{5}$:	6,	1,	7
	$\omega^{6}, \omega^{7}, \omega^{8}$	[illegible]	[illegible]	77

Proved using a prescribed permutation check

Define a polynon … $\to \Omega$ that implements a rotation:

$W(\omega^{-2}, \omega^{1}, \omega^{3})$ … $\omega^{3}, \omega^{-2})$, $W(\omega^{-1}, \omega^{0}) = (\omega^{0}, \omega^{-1})$, …

Lemma: $\forall\, y \in \Omega$: $T(y) = T(W(y)) \Rightarrow$ wire constraints are satisfied

The complete Plonk Poly-IOP (and SNARK)

Setup(C) → $pp := (S, W)$ and $vp := (\boxed{S}$ and $\boxed{W})$ (untrusted)

<u>Prover P($pp, \boldsymbol{x}, \mathbf{w}$)</u>

build T(X) ∈ $\mathbb{F}_p^{(\leq d)}$[X] $\xrightarrow{\boxed{T}}$

<u>Verifier V($vp, \boldsymbol{x}$)</u>

build $v(X) \in \mathbb{F}_p^{(\leq |I_x|)}$ [X]

Prover proves:

gates: (1) S(y)·[T(y) + T(ωy)] + (1 – S(y))·T(y)·T(ωy) $-$ T(ω^2y) $= 0$; ∀ y ∈ Ω_{gates}

inputs: (2) T(y) $- v$(y) $= 0$ ∀ y ∈ Ω_{inp}

wires: (3) T(y) $-$ T(W(y)) $= 0$ (using prescribed perm. check) ∀ y ∈ Ω

output: (4) T($\omega^{3|C|-1}$) $= 0$ (output of last gate = 0)

$\longrightarrow$

The complete Plonk Poly-IOP (and SNARK)

Setup(C) → $pp := (S, W)$ and $vp := (\boxed{S}$ and $\boxed{W})$ (untrusted)

<u>Prover P($pp, \boldsymbol{x}, \mathbf{w}$)</u>

build T(X) ∈ $\mathbb{F}_p^{(\leq d)}$[X] $\xrightarrow{\boxed{T}}$

<u>Verifier V($vp, \boldsymbol{x}$)</u>

build $v(X) \in \mathbb{F}_p^{(\leq |I_x|)}$ [X]

<u>Thm</u>: The Plonk Poly-IOP is complete and knowledge sound, assuming $7|C|/p$ is negligible

(eprint/2019/953)

Many extensions …

- Plonk proof: a short proof (O(1) commitments), fast verifier
- The SNARK can be made into a zk-SNARK

Main challenge: reduce prover time

- **Hyperplonk**: replace Ω with $\{0,1\}^t$ (where $t = \log_2|\Omega|$)
  - The polynomial T is now a multilinear polynomial in t variables
  - ZeroTest is replaced by a multilinear SumCheck (linear time)

END OF LECTURE

Next lecture: scaling the blockchain