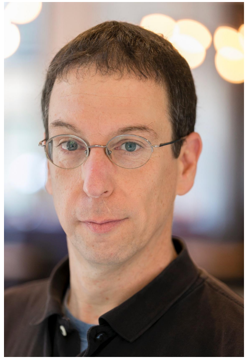
Zero Knowledge Proofs

Introduction to Modern SNARKs

Instructors: **Dan Boneh**, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang





Stanford University

Berkeley UNIVERSITY OF CALIFORNIA

GEORGETOWN UNIVERSITY

TEXAS A&M UNIVERSITY

What is a zk-SNARK ? (intuition)

- **SNARK:** a <u>succinct</u> proof that a certain statement is true

Example statement: "I know an m such that $\text{SHA256}(m) = 0$"

- **SNARK:** the proof is **"short"** and **"fast"** to verify
  [if m is 1GB then the trivial proof (the message m) is neither]

- **zk-SNARK:** the proof "reveals nothing" about m (privacy for m)

Commercial interest in SNARKs

Many more building applications that use SNARKs

Why so much commercial interest?

Babai-Fortnow-Levin-Szegedy 1991:

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with unreliable software.

"Checking Computations in Polylogarithmic Time"

Why so much commercial interest?

Babai-Fortnow-Levin-Szegedy 1991:

a slow and expensive computer

In this setup, a si~~ngle reliable PC can~~ monitor the operation of a herd of sup~~ercomputers~~ ***GPUs*** working with unreliable software.

"Checking Computations in Polylogarithmic Time"

Why so much commercial interest?

Babai-Fortnow-Levin-Szegedy 1991:

L1 blockchain

In this setup, a si~~ngle reliable PC can~~ monitor the operation of a herd of sup~~ercomputers~~ working with unreliable software.

GPUs

"Checking Computations in Polylogarithmic Time"

Blockchain Applications I

Outsourcing computation: (no need for zero knowledge)

L1 chain quickly verifies the work of an off-chain service

Examples:

- **Scalability:** proof-based Rollups (zkRollup)
  off-chain service processes a batch of Tx;
  L1 chain verifies a succinct proof that Tx were processed correctly

- **Bridging blockchains:** proof of consensus (zkBridge)
  enables transfer of assets from one chain to another

Blockchain Applications II

Some applications require zero knowledge (privacy):

- **Private Tx on a public blockchain**:
  - zk proof that a private Tx is valid (Tornado cash, Zcash, IronFish, Aleo)
- **Compliance:**
  - Proof that a private Tx is compliant with banking laws (Espresso)
  - Proof that an exchange is solvent in zero-knowledge (Raposa)

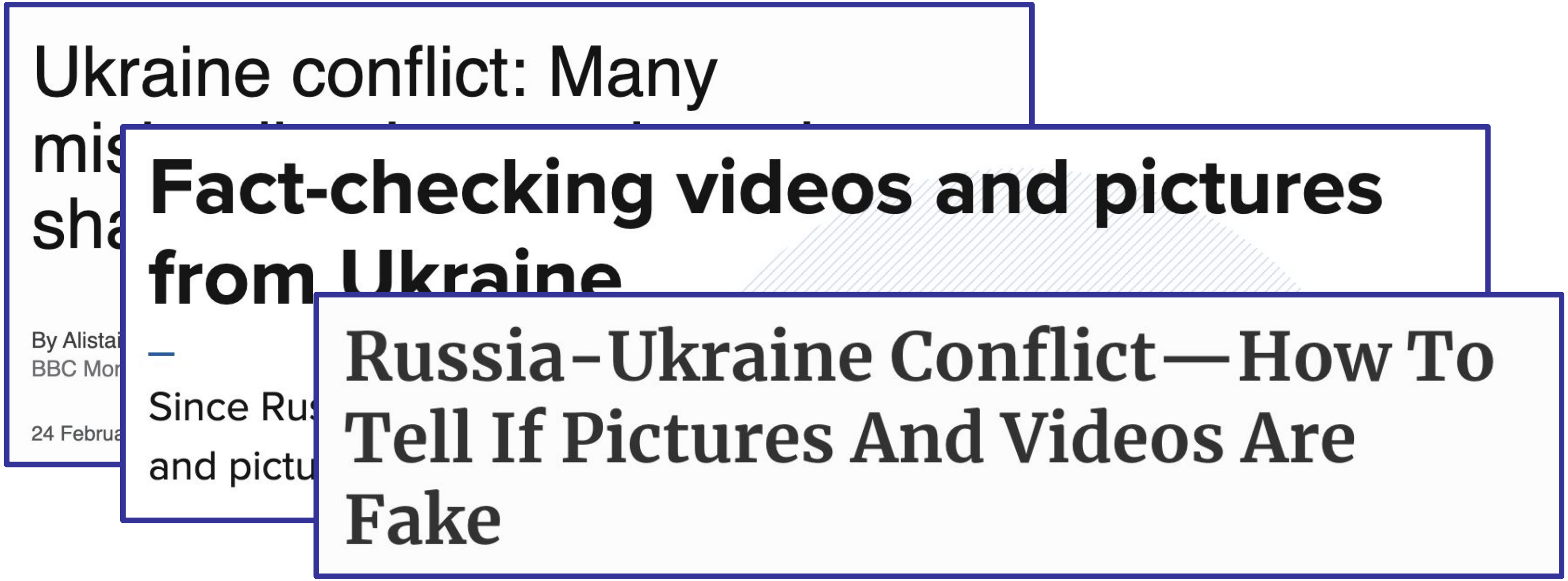
Many non-blockchain applications

More on these blockchain applications later in course

Blockchains drive the development of SNARKs

… but **many** non-blockchain applications

Using ZK to fight disinformation


Ukraine conflict: Many
mis
sha
By Alistai
BBC Mon
24 Februa

Fact-checking videos and pictures
from Ukraine
Since Rus
and pictu

Russia-Ukraine Conflict—How To
Tell If Pictures And Videos Are
Fake

C2PA: a standard for content provenance

Sony Unlocks In-Camera Forgery-Proof Technology

04 Aug, 2022

embedded certified
signing key *sk*

location
timestamp

signature

C2PA

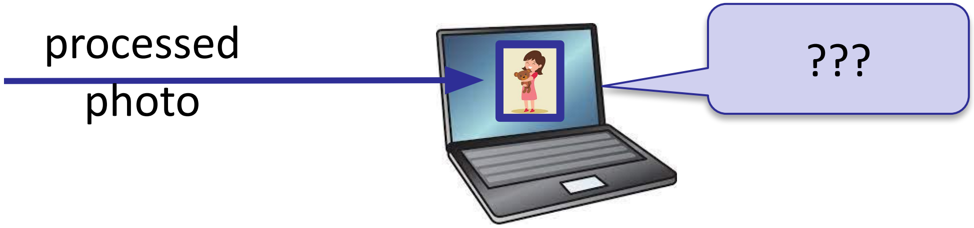
verify metadata
by checking sig

A problem: post-processing

Newspapers often process the photos before publishing:

- Resize (1500×1000), Crop, Grayscale (AP lists allowed ops)

The problem: laptop cannot verify signature on processed photo



A solution using a zk-SNARK

(with T. Datta, 2022)

Laptop has (***Photo***, ***Ops***). Editing software attaches a proof π that:

I know a pair (***Orig***, ***Sig***) such that

1. ***Sig*** is a valid C2PA signature on ***Orig***
2. ***Photo*** is the result of applying ***Ops*** to ***Orig***
3. metadata(***Photo***) = metadata(***Orig***)

⇒ Laptop verifies π and shows metadata to user

Performance

- Proof size: ≤1KB. Verification time: ≤10 ms.

(in browser)

Time to generate proof π: (by newspaper, one time per image)

- for images that are about 6000×4000 pixels:
  resize, crop, grayscale ⇒ a few minutes with sufficient HW

See also: PhotoProof by Naveh & Tromer (2016)

Why are all these applications possible now?

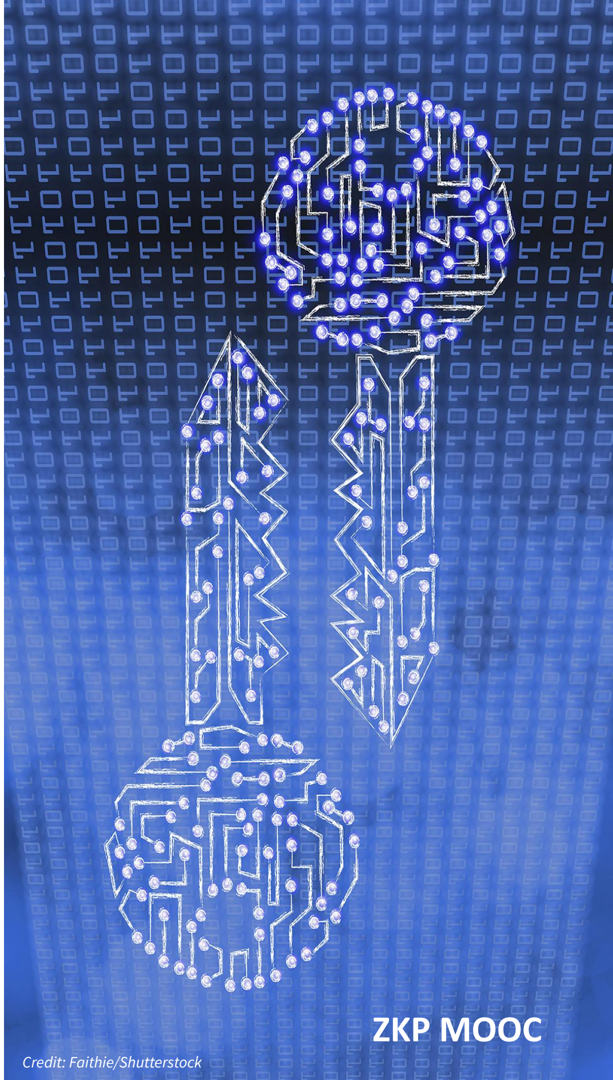
The breakthrough: new fast SNARK provers

- Proof generation time is linear (or quasilinear) in computation size
- **Many** beautiful ideas … will cover in lectures

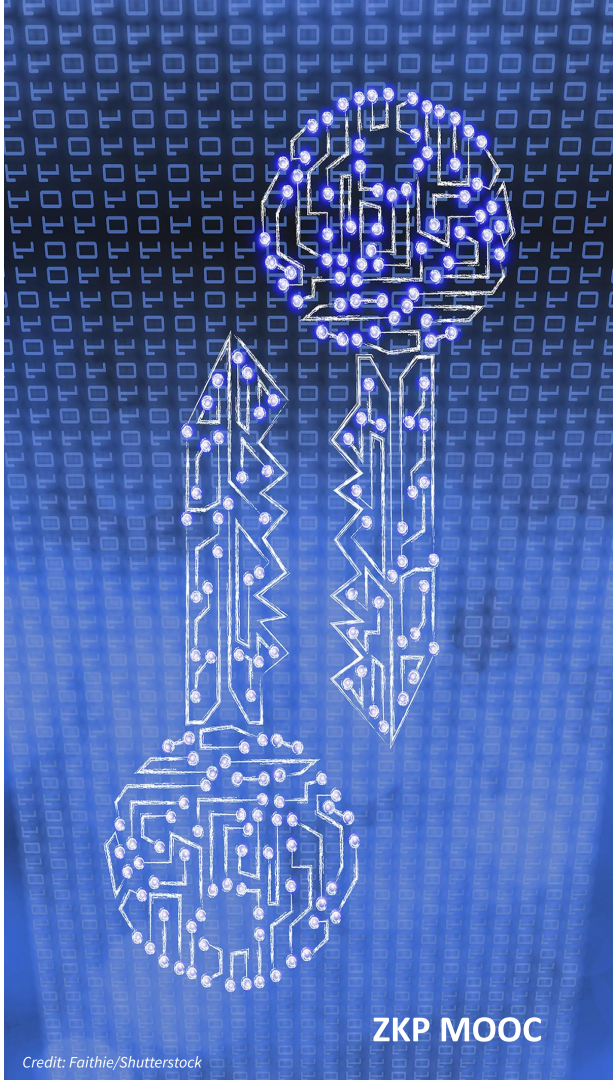
a large bibliography: a16zcrypto.com/zero-knowledge-canon

Next segment:

What is a SNARK?



What is a SNARK?



Credit: Faithie/Shutterstock

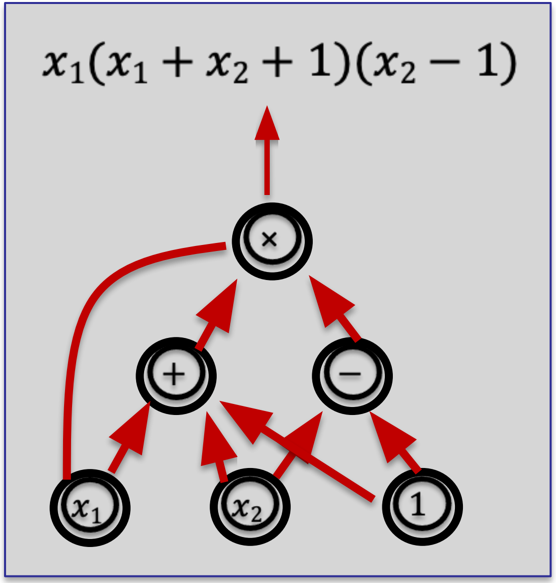
Review: arithmetic circuits

Fix a finite field $\mathbb{F} = \{0, \ldots, p-1\}$ for some prime p>2.

Arithmetic circuit: $C\colon \mathbb{F}^n \to \mathbb{F}$

- directed acyclic graph (DAG) where
  internal nodes are labeled $+, -,$ or $\times$
  inputs are labeled $1, x_1, \ldots, x_n$
- defines an n-variate polynomial
  with an evaluation recipe

$|C|$ = # gates in C



Interesting arithmetic circuits

Examples:

- $C_{SHA}(h, \mathbf{m})$: outputs 0 if $SHA256(\mathbf{m}) = h$, and $\neq 0$ otherwise

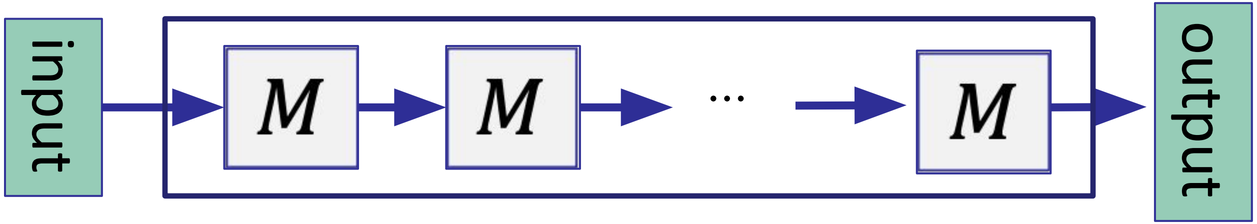
  $C_{hSHA}(h, \mathbf{m}) = \big(h - SHA256(\mathbf{m})\big)$, $|C_{SHA}| \approx$ 20K gates

- $C_{sig}(pk, m, \sigma)$: outputs 0 if σ is a valid ECDSA signature on m with respect to pk

Structured vs. unstructured circuits

An **unstructured circuit**: a circuit with arbitrary wires

A **structured** circuit:



M is often called a virtual machine (VM) -- one step of a processor

Some SNARK techniques only apply to structured circuits

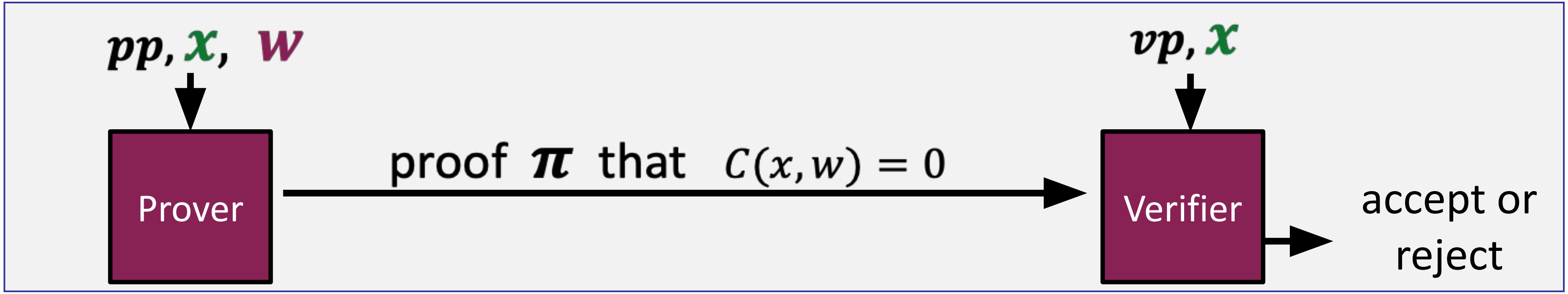
(preprocessing) NARK: Non-interactive ARgument of Knowledge

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

public statement in $\mathbb{F}^n$ — $\boldsymbol{x}$

$\boldsymbol{w}$ — secret witness in $\mathbb{F}^m$

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



(preprocessing) NARK: Non-interactive ARgument of Knowledge

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

- **S**(C) → public parameters (pp, vp) for prover and verifier
- **P**$(pp, \boldsymbol{x}, \boldsymbol{w})$ → proof π
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi})$ → accept or reject

all algs. and adversary have access to a random oracle

NARK: requirements (informal)

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

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

proof π $\longrightarrow$ accept or reject

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

Adaptively **knowledge sound:** V accepts $\Rightarrow$ 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 $\boldsymbol{w}$

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 ; $\boxed{\text{len}(\pi) = \text{sublinear}(|w|)}$
- **V**(vp, $\boldsymbol{x}$, $\boldsymbol{\pi}$) **fast to verify** ; $\boxed{\text{time(V)} = O_\lambda(|x|, \text{sublinear}(|C|))}$

example sublinear function: $f(n) = \sqrt{n}$

SNARK: a Succinct ARgument of Knowledge

A **strongly 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 ; $\boxed{\text{len}(\pi) = O_\lambda(\log(|C|))}$
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi})$ **fast to verify** ; $\boxed{\text{time(V)} = O_\lambda(|x|,\ \log(|C|))}$

short "summary" of circuit

V has no time to read C !!

SNARK: a <u>Succinct</u> AR**gument of Knowledge**

SNARK: a NARC (complete and knowledge sound) that is **<u>succinct</u>**

zk-SNARK: a SNARK that is also **zero knowledge**

The trivial SNARK is not a SNARK

(a) Prover sends w to verifier,
(b) Verifier checks if $C(x, w) = 0$ and accepts if so.

Problems with this:

(1) w might be long: we want a "short" proof

(2) computing $C(x, w)$ may be hard: we want a "fast" verifier

(3) w might be secret: prover might not want to reveal w to verifier

Types of preprocessing Setup

Setup for circuit C: $\mathbf{S}(C; r) \to$ public parameters (pp, vp)

r: random bits

<u>Types of setup</u>:

better

trusted setup per circuit: $\mathbf{S}(C; r)$ random r must be kept secret from prover

prover learns $r \Rightarrow$ can prove false statements

trusted but universal (updatable) setup: secret r is independent of C

$$\boldsymbol{S} = (S_{init}, S_{index}): \quad \underbrace{S_{init}(\lambda; r) \to gp,}_{\text{one-time setup, secret } r} \quad \underbrace{S_{index}(gp, C) \to (pp, vp)}_{\text{deterministic algorithm}}$$

<u>transparent setup</u>: $\mathbf{S}(C)$ does not use secret data (no trusted setup)

ZKP MOOC

Significant progress in recent years (partial list)

		verifier time	setup	post-quantum?
Groth'16			trusted per circuit	no
Plonk / Marlin			universal trusted setup	no

Significant progress in recent years (partial list)

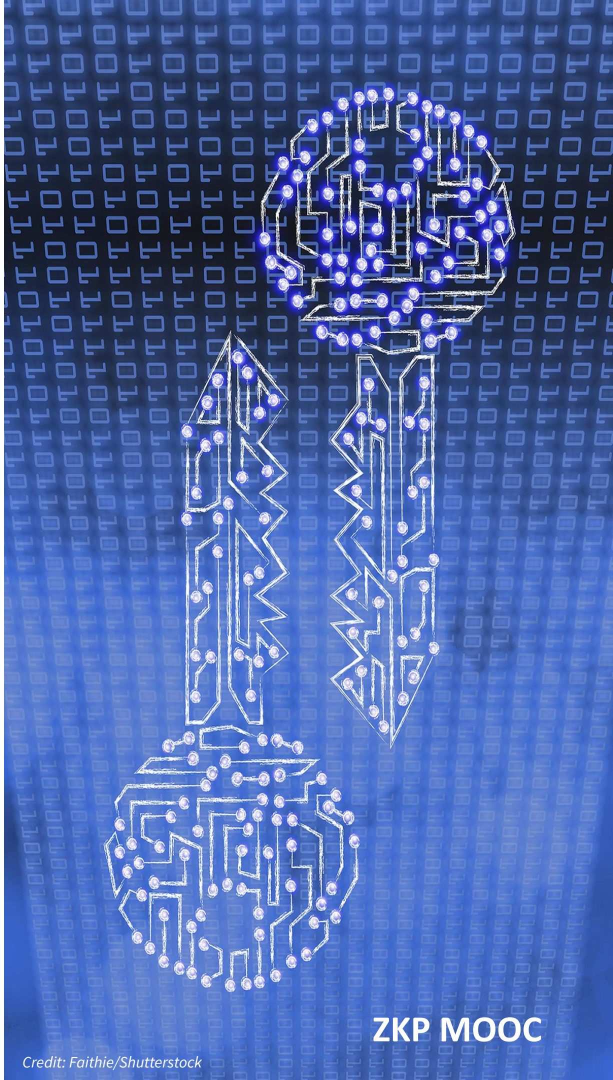
		verifier time	setup	post-quantu m?
Groth'16		(for a circuit with ≈2^{20} gates)		no
Plonk / Marlin			universal trusted setup	no
Bulletproofs			transparent	no
STARK			transparent	yes
⋮				

Significant progress in recent years (partial list)

		verifier time	setup	post-quantum?
Groth'16				
Plonk / Marlin				
Bulletproofs				
STARK				
⋮				⋮

Prover time is almost linear in $|C|$

How to define
“knowledge soundness”?



Definitions: knowledge soundness

Goal: if V accepts then P “knows” $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

What does it mean to ”know” $\boldsymbol{w}$??

informal def: P knows $\boldsymbol{w}$, if $\boldsymbol{w}$ can be “extracted” from P

Definitions: knowledge soundness

Formally: (S, P, V) is (adaptively) **knowledge sound** for a circuit C if

for every poly. time adversary $A = (A_0, A_1)$ such that

$gp \leftarrow S_{init}(\,)$, $(C, x, st) \leftarrow A_0(gp)$, $(pp, vp) \leftarrow S_{index}(C)$, $\pi \leftarrow A_1(pp, x, st)$:

$\Pr[\, V(vp, x, \pi) = \text{accept} \,] > 1/10^6$ (non-negligible)

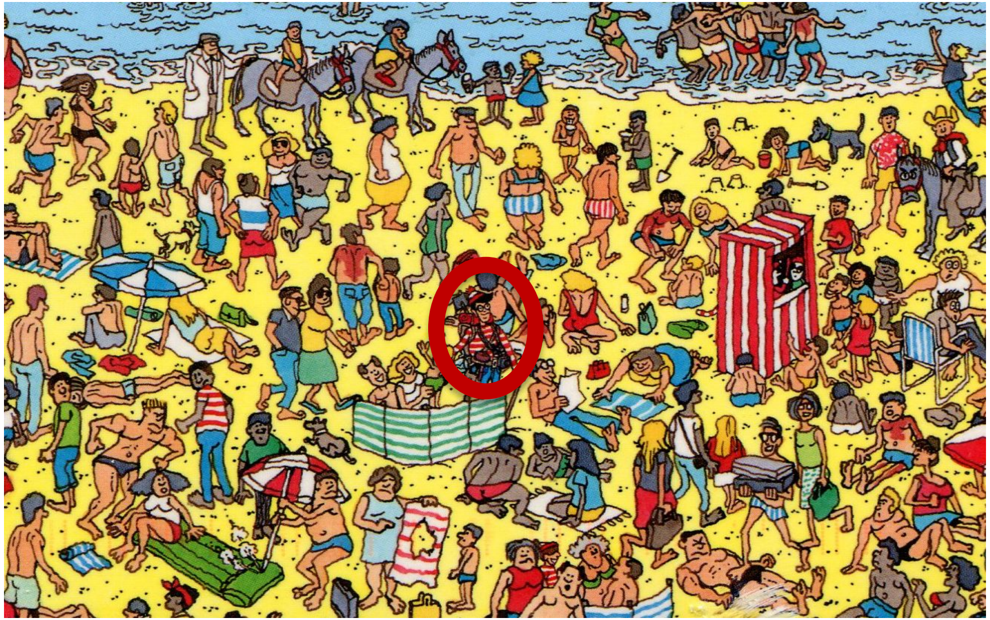
there is an efficient **extractor** E (that uses A) s.t.

$gp \leftarrow S_{init}(\,)$, $(C, x, st) \leftarrow A_0(gp)$, $\boxed{w \leftarrow E(gp, C, x):}$

$\Pr[\, C(x, w) = 0 \,] > 1/10^6 - \epsilon$ (for a negligible ϵ)

Zero knowledge

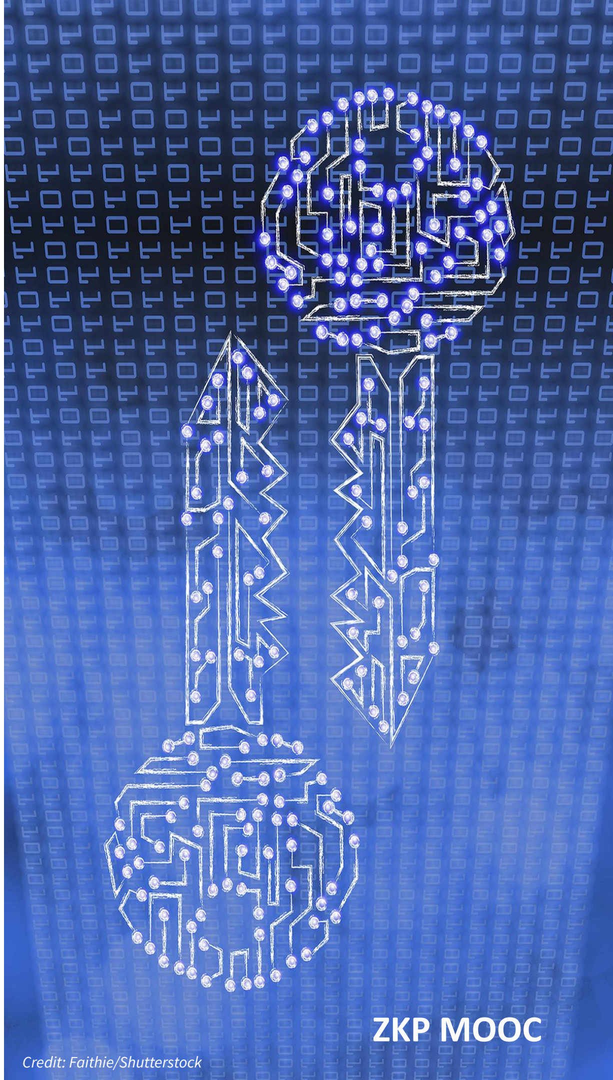
Where is Waldo?



Defining Zero knowledge

See previous lecture

Building an efficient SNARK



Credit: Faithie/Shutterstock

Recall: SNARK (Non-interactive ARgument of Knowledge)

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

- **S**(C) → public parameters (pp, vp) for prover and verifier
- **P**$(pp, \boldsymbol{x}, \boldsymbol{w})$ → proof π
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi})$ → accept or reject

proof size is $O_\lambda(\mathbf{log}(|\boldsymbol{C}|))$

verifier time is $O_\lambda(|x|,\ \mathbf{log}(|\boldsymbol{C}|))$

General paradigm: two steps

(1) A functional commitment scheme

(cryptographic object)

(2) A compatible interactive oracle proof (IOP)

(info. theoretic object)

SNARK for general circuits

Let’s explain each concept …

Review: commitments

Two algorithms:

- *commit*$(m, r) \to$ ***com*** (r chosen at random)
- *verify*$(m,$ ***com***$, r) \to$ accept or reject

Properties: (informal)

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

A standard construction

Fix a hash function $H: \mathcal{M} \times \mathcal{R} \rightarrow T$

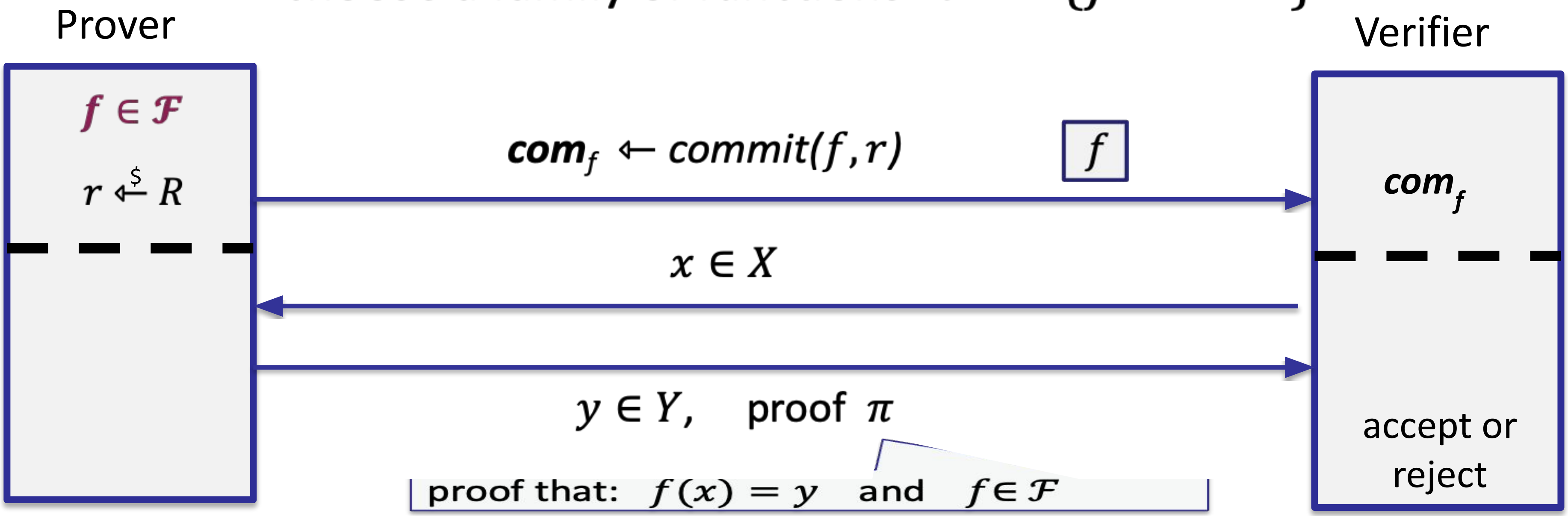
commit(m, r): **com** $:= H(m, r)$

verify$(m,$ ***com***$, r)$: accept if ***com*** $= H(m, r)$

Hiding and Binding for a suitable function H

Committing to a function

choose a family of functions $\mathcal{F} = \{f : X \to Y\}$



Committing to a function: syntax

A **functional commitment** scheme for $\mathcal{F}$:

- $\underline{setup}(1^\lambda) \to gp$, outputs public parameters gp
- $\underline{commit}(gp, f, r) \to \mathbf{com}_f$ commitment to $f \in \mathcal{F}$ with $r \in \mathcal{R}$

  a **binding** (and optionally **hiding**) commitment scheme for $\mathcal{F}$

- $\underline{eval}$(Prover P, verifier V): for a given $\mathbf{com}_f$ and $x \in X$, $y \in Y$:

  P(gp, f, x, y, r) $\to$ short proof π

  V($gp, \mathbf{com}_f, x, y, \pi$) $\to$ accept/reject

a (zk)SNAR**K** for the relation:

$\boxed{f(x) = y \text{ and } f \in \mathcal{F} \text{ and } commit(gp, f, r) = \mathbf{com}_f}$

Four important functional commitments

Polynomial commitments: commit to a <u>univariate</u> $f(X)$ in $\mathbb{F}_p^{(\leq d)}[X]$

Multilinear commitments: commit to multilinear f in $\mathbb{F}_p^{(\leq 1)}[X_1, \ldots, X_k]$

e.g., $f(x_1, \ldots, x_k) = x_1 x_3 + x_1 x_4 x_5 + x_7$

Vector commitments (e.g., Merkle trees):

- Commit to $\vec{u} = (u_1, \ldots, u_d) \in \mathbb{F}_p^d$. Open cells: $f_{\vec{u}}(i) = u_i$

Inner product commitments (inner product arguments – IPA):

- Commit to $\vec{u} \in \mathbb{F}_p^d$. Open an inner product: $f_{\vec{u}}(\vec{v}) = (\vec{u}, \vec{v})$

Let's look at polynomial commitments

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

- ***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 $O_\lambda(\log \boldsymbol{d})$

Let's look at polynomial commitments

We will see several constructions in the coming lectures

<u>A few examples</u>:

- Using bilinear groups: KZG'10 (trusted setup), Dory'20, …
- Using hash functions only: based on FRI (long eval proofs)
- Using elliptic curves: Bulletproofs (short proof, but verifier time is $O(d)$)
- Using groups of unknown order: Dark'20

The trivial commitment scheme is not a polynomial commitment

- commit($f = \sum_{i=0}^{d} a_i X^i, r$): output $\text{com}_f \leftarrow H\big((a_0, \dots, a_d), r\big)$
- eval: prover sends $\pi = ((a_0, \dots, a_d),\ r)$ to verifier;
  verifier accepts if $f(u) = v$ and $H\big((a_0, \dots, a_d), r\big) = \text{com}_f$

The problem: the proof π is not succinct.

Proof size and verification time are **linear** in d

A useful observation

For a non-zero $f \in \mathbb{F}_p^{(\leq d)}[X]$

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

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

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

$\Rightarrow$ a simple zero test for a committed polynomial

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

A useful observation

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

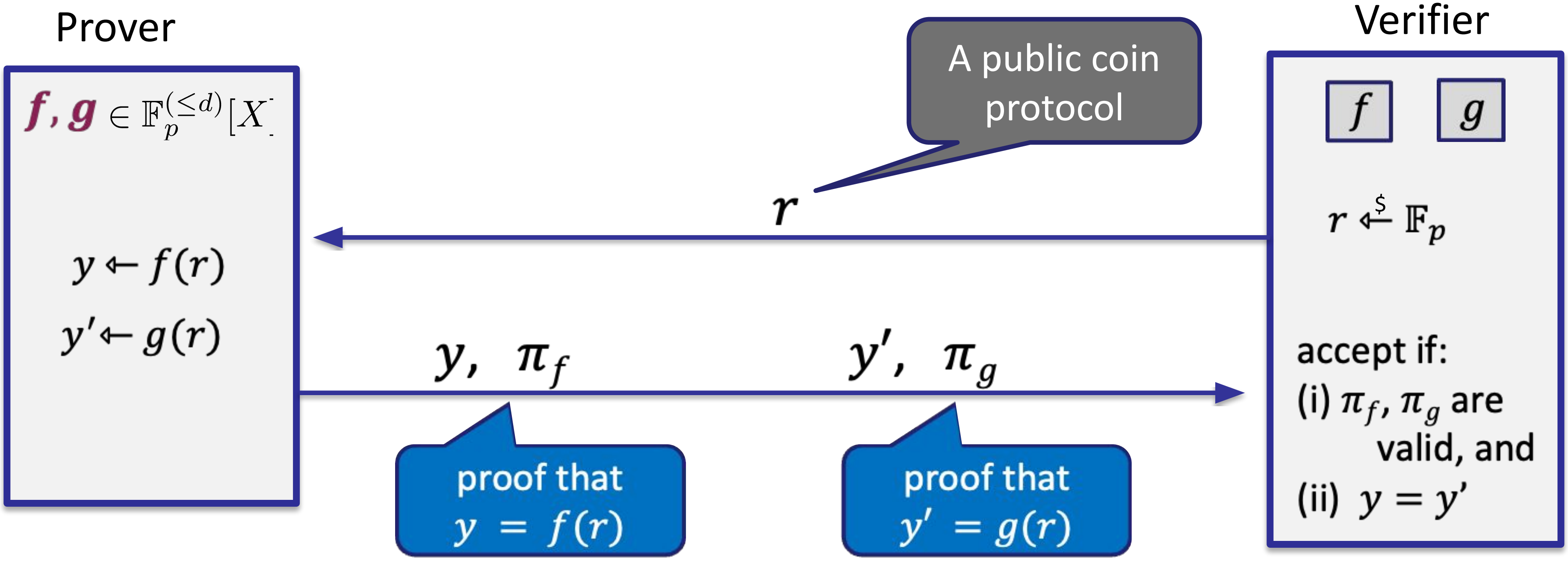
Let $f, g \in \mathbb{F}_p^{(\leq d)}[X]$.

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

Let’s look at the equality test protocol



Making it a SNARK (non-interactive)

The **Fiat-Shamir transform**:

- public-coin interactive protocol $\Rightarrow$ non-interactive protocol

  [public coin: all verifier randomness is public]

Prover P(pp, $\mathbf{x}$, $\mathbf{w}$) — Verifier V(vp, $\mathbf{x}$)

$\xleftarrow{\quad r \quad}$ choose random r

$\xrightarrow{\quad y, y', \pi_f, \pi_g \quad}$ accept or reject

A SNARK for polynomial equality testing

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

- idea: prover generates verifier's random bits on its own using H

com_f, com_g $\quad f, g$

Prover P(pp, x, w)

$r \leftarrow H(x)$

$y \leftarrow f(r), \quad y' \leftarrow g(r)$

generate π_f, π_g

y, y', π_f, π_g

com_f, com_g

Verifier V(vp, x)

accept or reject

A SNARK for polynomial equality testing

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

- Thm: this is a SNARK if (i) d/p is negligible (where $f, g \in \mathbb{F}_p^{(\leq d)}[X]$), and (ii) H is modeled as a random oracle.

Prover P(*pp*, ***x***, ***w***)

$r \leftarrow H(x)$

$y \leftarrow f(r), \quad y' \leftarrow g(r)$

generate π_f, π_g

in practice, use SHA256

com_f, com_g

y, y', π_f, π_g

Verifier V(*vp*, ***x***)

accept or reject

Back to the paradigm

(1) A functional commitment scheme

(2) A compatible interactive oracle proof (IOP)

SNARK for general circuits

Component 2: $\mathcal{F}$ – IOP

Goal: boost functional commitment $\Rightarrow$ SNARK for general circuits

Example: polynomial commitment scheme for $\mathbb{F}_p^{(\leq d)}[X]$

$\Downarrow$ Poly-IOP

SNARK for any circuit C where $|C| < d$

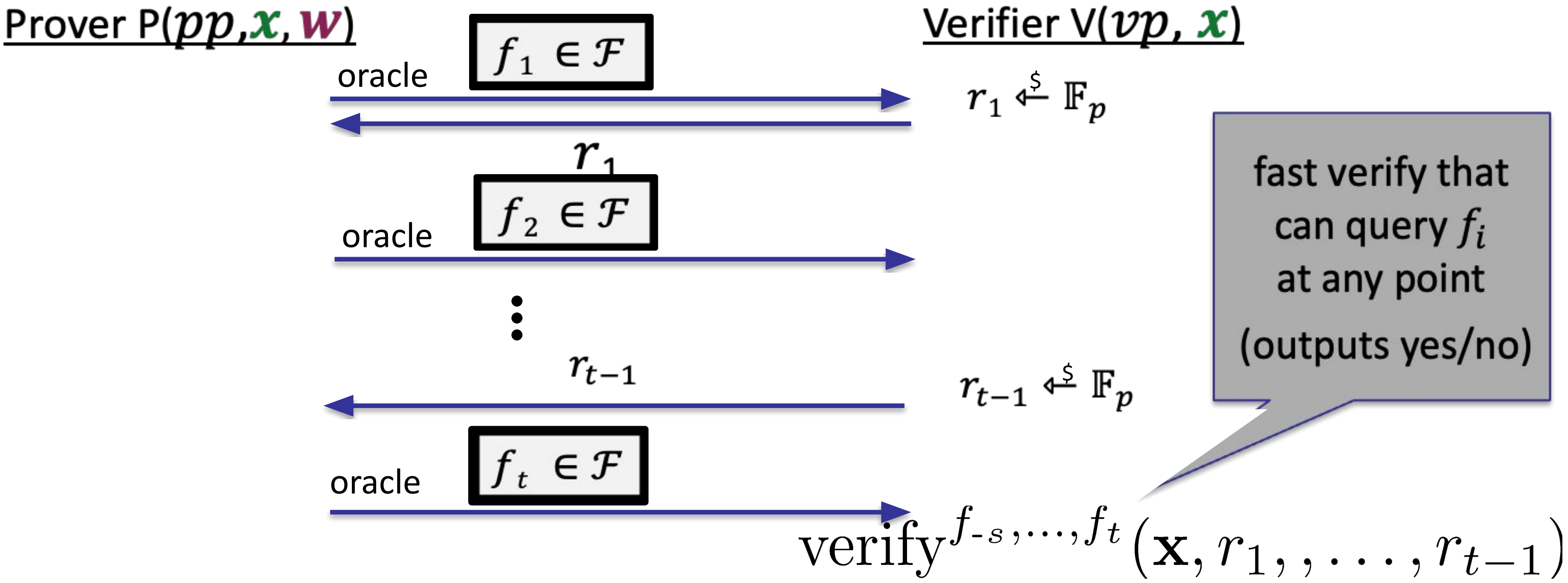
Component 2: $\mathcal{F}$ – IOP

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

$\underline{\mathcal{F}}$ **–IOP**: a proof system that proves $\underline{\exists w: C(x, w) = 0}$ as follows:

Setup(C) $\rightarrow$ public parameters $\boldsymbol{pp}$ and $\boldsymbol{vp} = \underbrace{(\boxed{f_0}, \boxed{f_{-1}}, \ldots, \boxed{f_{-s}})}_{\text{oracles for functions in } \mathcal{F}}$

$\mathcal{F}$ – IOP: proving that $C(x, w) = 0$



Properties

- **Complete**: if $\exists w: C(x, w) = 0$ then Pr[verifier accepts] = 1
- (Unconditional) **knowledge sound** (as an IOP)

  [extractor is given $(x, f_1, r_1, \ldots, r_{t-1}, f_t)$ and outputs w]
- Optional: **zero knowledge** (for a zk-SNARK)

An example Poly-IOP (a bit contrived)

Prover P(pp,X,W)

$C(X,W) = 0 \iff X \subseteq W \subseteq \mathbb{F}_p$

Verifier V(vp,X)

$f(Z) := \prod_{w \in W}(Z - w)$

$g(Z) := \prod_{x \in X}(Z - x)$

$q(Z) := f/g \in \mathbb{F}_p^{(\le d)}[X]$

$\boxed{f}$ $\boxed{q}$ $\longrightarrow$

$\longleftarrow$ r

$g(Z) := \prod_{x \in X}(Z - x)$

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

(i) query $w \leftarrow f(r)$

(ii) query $q' \leftarrow q(r)$

(iii) compute $x \leftarrow g(r)$

(iv) accept if $x \cdot q' = w$

$\Rightarrow$ $g(Z)$ is a factor of $f(Z)$

Knowledge soundness: V accepts $\Rightarrow$ $f = g \cdot q$ w.h.p $\Rightarrow$ $X \subseteq W$

Extractor(X,f,q,r): output witness W by computing all roots of $f(Z)$

The IOP Zoo (⇒ SNARKs for general circuits)

Poly-IOP
Sonic,
Marlin,
Plonk, …

Poly-Commit

Multilinear-IOP
Spartan,
Clover,
Hyperplonk, …

Multilinear-Commit

Vector-IOP
STARK,
Breakdown,
Orion, …

Merkle

non-interactive
via Fiat-Shamir

SNARK

SNARKs in practice

DSL
program

Circom,
ZoKrates,
Leo,
Zinc,
Cairo,
Noir,
…

domain specific language

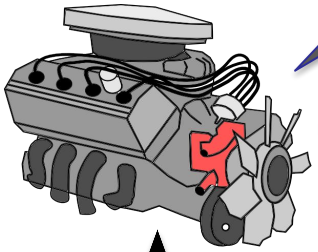
compiler

SNARK friendly format

circuit,
R1CS,
…
EVM byte code

SNARK backend prover

heavy computation



π

x, witness

END OF LECTURE

Coming up:

(i) SNARK DSLs

(ii) an efficient multilinear IOP