CS251 Fall 2020

(cs251.stanford.edu)

Proof Systems and SNARKs

Dan Boneh

Where we are in the course

- Basics: Consensus protocols and Bitcoin
- Composable decentralized applications (e.g., on Ethereum)

  ⇒ Decentralized Finance (DeFi)

  ⇒ Scaling the blockchain:
  payment channels,
  Rollup (Proof-based or Optimistic),
  faster consensus

Last core topic: **privacy** -- private transactions on a public blockchain

Managing assets on a blockchain: key principles

- **Universal verifiability** of blockchain rules
  - ⇒ all data written to the blockchain is public; everyone can verify
  - ⇒ added benefit: interoperability between chains

- Assets are **controlled by signature keys**
  - ⇒ assets <u>cannot</u> be transferred without a valid signature

    (of course, users can choose to custody their keys)

Privacy?

Naïve reasoning:

universal verifiability $\Rightarrow$ blockchain data is public

$\Rightarrow$ all transactions data is public

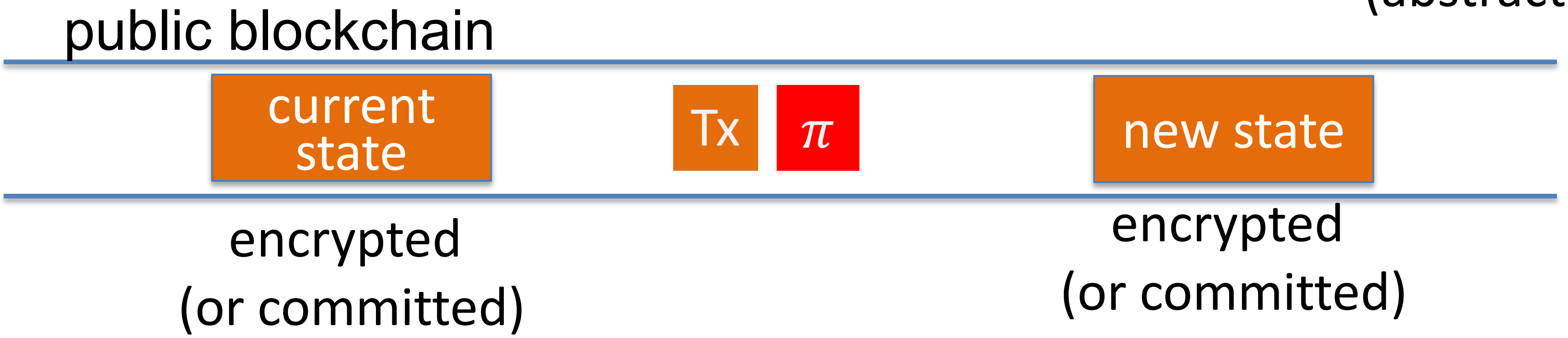
otherwise, how we can verify Tx?

not quite …

crypto magic $\Rightarrow$ private Tx on a publicly verifiable blockchain

Public blockchain & universal verifiability

(abstractly)



- **Tx data**: encrypted (or committed)
- **Proof π**: *zero-knowledge proof* that (reveals nothing about Tx data)
  (1) plaintext Tx data is consistent with plaintext current state
  (2) plaintext new state is correct

Public blockchain & universal verifiability

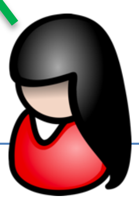
(abstractly)

public blockchain

current state | Tx | π | new state

encrypted (or committed)

anyone can verify π



encrypted (or committed)

- **Tx data**: encrypted (or committed)
- **Proof π**: *zero-knowledge proof* that (reveals nothing about Tx data)

  (1) plaintext Tx data is consistent with plaintext current state

  (2) plaintext new state is correct

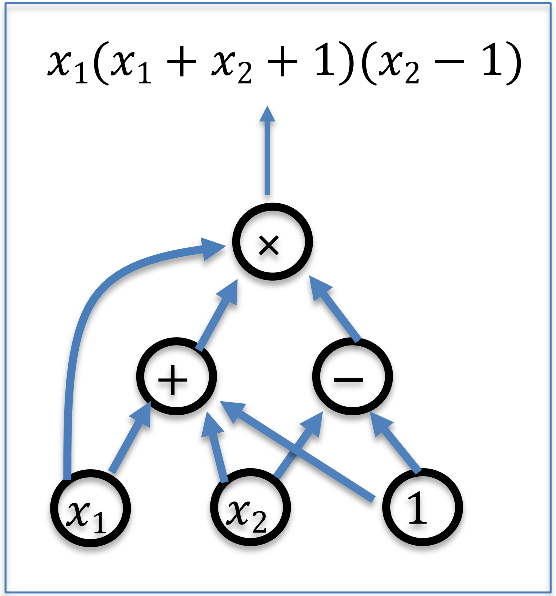
Zero Knowledge Proof Systems

(1) arithmetic circuits

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

- **Arithmetic circuit:** $C\colon \mathbb{F}^n \rightarrow \mathbb{F}$
  - directed acyclic graph (DAG) where
    - internal nodes are labeled +, −, or ×
    - inputs are labeled $1, x_1, \ldots, x_n$
  - defines an n-variate polynomial with an evaluation recipe

- $|C| =$ # multiplication gates in C



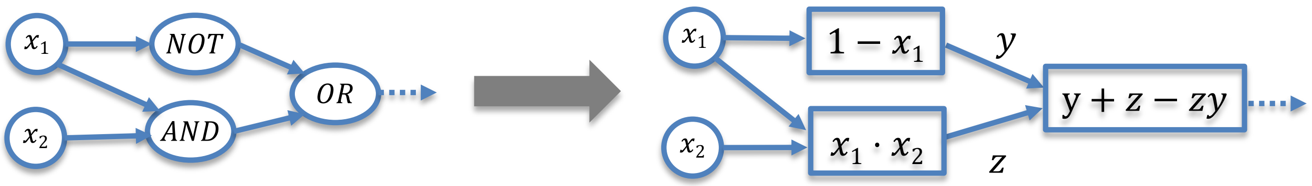
Boolean circuits as arithmetic circuits

Boolean circuits: circuits with AND, OR, NOT gates

Encoding a boolean circuit as an arithmetic circuit over $\mathbb{F}_p$:

- $\text{AND}(x, y)$ encoded as $x \cdot y$
- $\text{OR}(x, y)$ encoded as $x + y - x \cdot y$
- $\text{NOT}(x)$ encoded as $1 - x$

x	y	$\text{OR}(x, y)$
0	0	0
0	1	1
1	0	1
1	1	1



Interesting arithmetic circuits

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

  $$C_{hash}(h, \mathbf{m}) = \big(h - \mathrm{SHA256}(\mathbf{m})\big) \;, \qquad |\, C_{hash} \,| \approx 20\text{K gates}$$

- $C_{sig}((pk, m), \sigma)$: output 0 if σ is
  a valid ECDSA signature of m under pk

(2) non-interactive proof systems (for NP)

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

public statement in $\mathbb{F}_p^n$ secret witness in $\mathbb{F}_p^m$

Let $x \in \mathbb{F}_p^n$. Two standard goals for prover P:

(1) <u>Soundness</u>: convince Verifier that $\exists w$ s.t. $C(x, w) = 0$

(e.g., $\exists w$ such that $[\,H(w) = x$ and $0 < w < 2^{60}\,]$)

(2) <u>Knowledge</u>: convince Verifier that P "knows" w s.t. $C(x, w) = 0$

(e.g., P knows a w such that $H(w) = x$)

The trivial proof system

Why can't prover simply send $\boldsymbol{w}$ to verifier?

- Verifier checks if $C(\boldsymbol{x}, \boldsymbol{w}) = 0$ and accepts if so.

<u>Problems with this</u>:

(1) $\boldsymbol{w}$ might be secret: prover cannot reveal $\boldsymbol{w}$ to verifier

(2) $\boldsymbol{w}$ might be long: we want a "short" proof

(3) computing $C(\boldsymbol{x}, \boldsymbol{w})$ may be hard: want to minimize Verifier's work

Non-interactive Proof Systems (for NP)

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

public input in $\mathbb{F}_p^n$

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

setup: **S**(C) → public parameters (***S**$_p$*, ***S**$_v$*)

Prover P(***S**$_p$*, x, w)

Verifier V(***S**$_v$*, x, π)

proof π

output accept or reject

Non-interactive Proof Systems (for NP)

A **non-interactive proof system** is a triple (S, P, V):

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

proof systems: properties (informal)

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

proof π

accept or reject

Complete: $\forall x, w$: $C(\boldsymbol{x}, \boldsymbol{w}) = 0 \Rightarrow$ V(S_v, x, P(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$)) = accept

Proof of knowledge: V accepts $\Rightarrow$ P "knows" $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

in some cases, **soundness** is sufficient: $\exists \boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

Zero knowledge (optional): $(\boldsymbol{x}, \pi)$ "reveals nothing" about $\boldsymbol{w}$

(a) Proof/argument of knowledge

Goal: V accepts $\Rightarrow$ P "knows" w s.t. $C(x, w) = 0$

What does it mean to "know" w ??

informal def: P knows w, if w can be "extracted" from P

P

(a) Proof/argument of knowledge

Formally: (S, P, V) is a **proof of knowledge** for a circuit C if
for every adversary $A = (A_0, A_1)$ such that

$$S(C) \to (S_p, S_v), \qquad (x, \mathrm{st}) \leftarrow A_0(S_p), \qquad \pi \leftarrow A_1(S_p, x, \mathrm{st}):$$

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

there is an efficient extractor E (that uses A_1 as a black box) s.t.

$$S(C) \to (S_p, S_v), \qquad (x, \mathrm{st}) \leftarrow A_0(S_p), \qquad \boxed{w \leftarrow E(S_p, x, \mathrm{st}):}$$

$$\Pr[C(x, w) = 0] > 1/10^6 \quad \text{(non-negligible)}$$

If only for poly. time A $\Rightarrow$ (S, P, V) is only an **argument of knowledge**.

(a) Proof/argument of knowledge

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

<u>**Proof**</u>: secure against unbounded cheating provers

.t.

<u>**Argument**</u>: secure against polynomial-time cheating provers

If only for poly. time A $\Rightarrow$ (S, P, V) is only an **argument of knowledge**.

(b) Zero knowledge

(S, P, V) is **zero knowledge** if proof π "reveals nothing" about $\boldsymbol{w}$

Formally: (S, P, V) is **zero knowledge** for a circuit C

if there is an efficient simulator ***Sim***,

such that for all $x \in \mathbb{F}_p^n$ s.t. $\exists w: C(x, w) = 0$ the distribution:

(S$_p$, S$_v$, x, π) where (S$_p$, S$_v$) ← S(C), π ← P(x, $\boldsymbol{w}$)

is indistinguishable from the distribution:

(S$_p$, S$_v$, x, π) where (S$_p$, S$_v$, π) ← ***Sim***(x)

key point: ***Sim***(x) simulates proof π without knowledge of $\boldsymbol{w}$

(3) Succinct arguments: SNARKs

Goal: P wants to show that it knows $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

<u>Succinct</u>:

- Proof π should be **short** [i.e., $|\pi| = O(\boxed{\mathbf{log}(|\boldsymbol{C}|)}, \lambda)$]
- Verifying π should be **fast** [i.e., time(V) = $O(|x|, \boxed{\mathbf{log}(|\boldsymbol{C}|)}, \lambda)$]

note: if SNARK is zero-knowledge, then called a **zkSNARK**

(3) Succinct arguments: SNARKs

Goal: P wants to show that it knows $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 1$

verifier cannot read C !! Instead,
V relies on setup(C) to pre-process (summarize) C in S_v

Succinct:

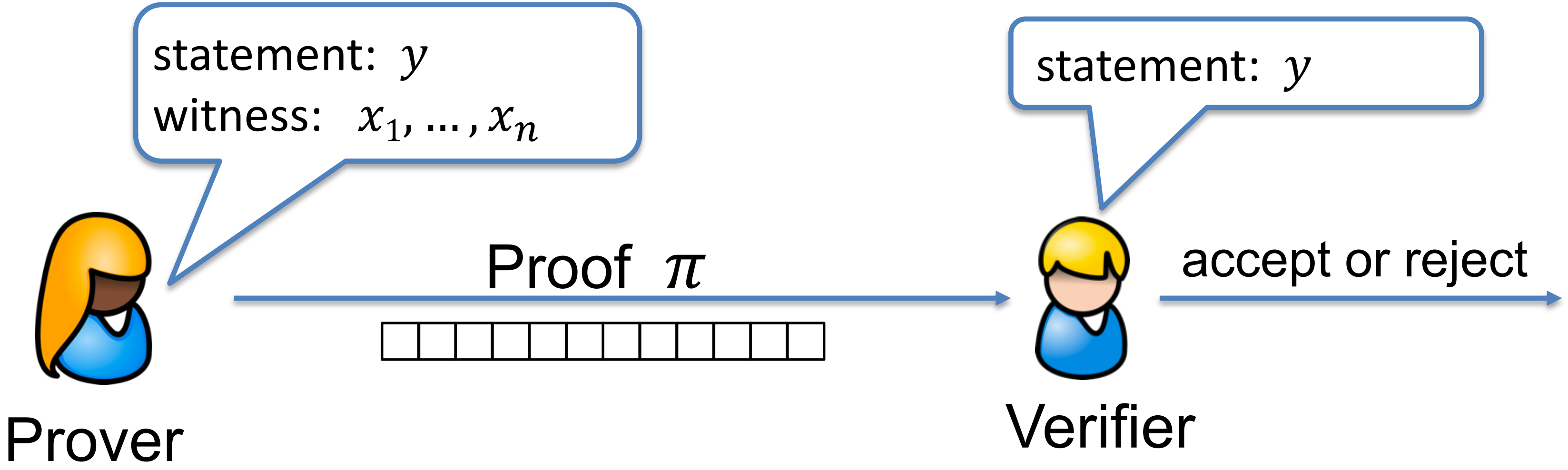
- Proof π should be **short** [i.e., $|\pi| = O(\boxed{\mathbf{log}(\quad}\lambda)$]
- Verifying π should be **fast** [i.e., time(V) = $O(|x|, \boxed{\mathbf{log}(|\boldsymbol{C}|)}, \lambda)$]

note: if SNARK is zero-knowledge, then called a **zkSNARK**

An example

Prover says: I know $(x_1, \dots, x_n) \in X$ such that $H(x_1, \dots, x_n) = y$

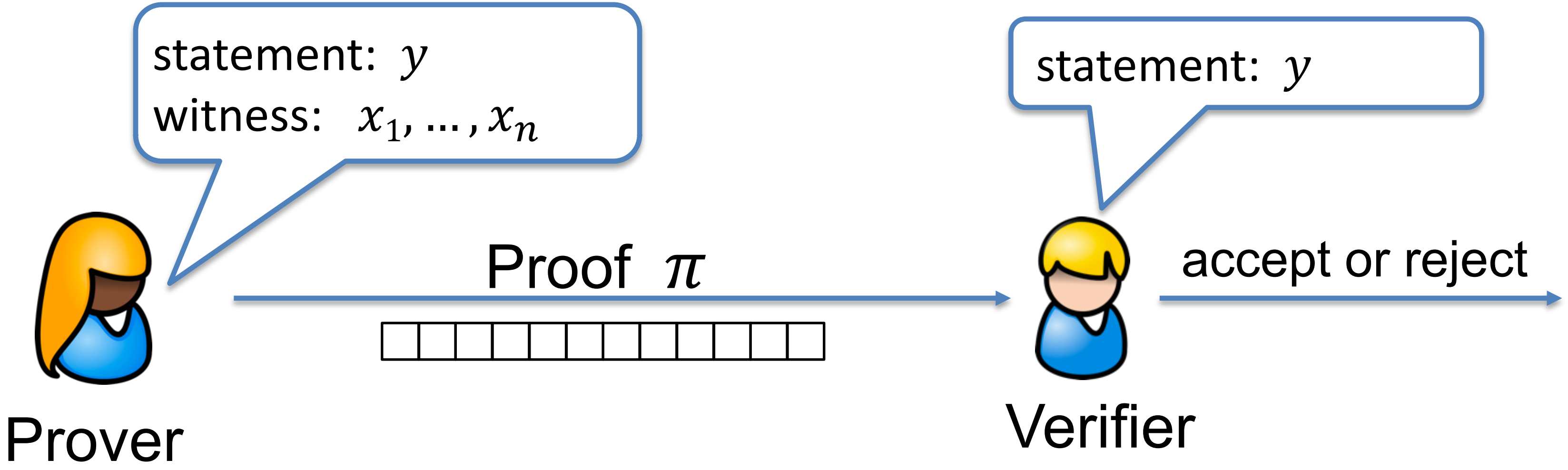
SNARK: $\text{size}(\pi)$ and $\text{VerifyTime}(\pi)$ should be $O(\log n)$!!



An example

How is this possible ???

SNARK: $\text{size}(\pi)$ and $\text{VerifyTime}(\pi)$ should be $O(\log n)$!!



Types of pre-processing Setup

Recall setup for circuit C: **S**(C) → public parameters (S_p, S_v)

Types of setup:

trusted setup per circuit: **S**(C) uses data that must be kept secret

compromised trusted setup ⇒ can prove false statements

updatable universal trusted setup: (S_p, S_v) can be updated by anyone

<u>transparent</u>: **S**() does not use secret data (no trusted setup)

Significant progress in recent years

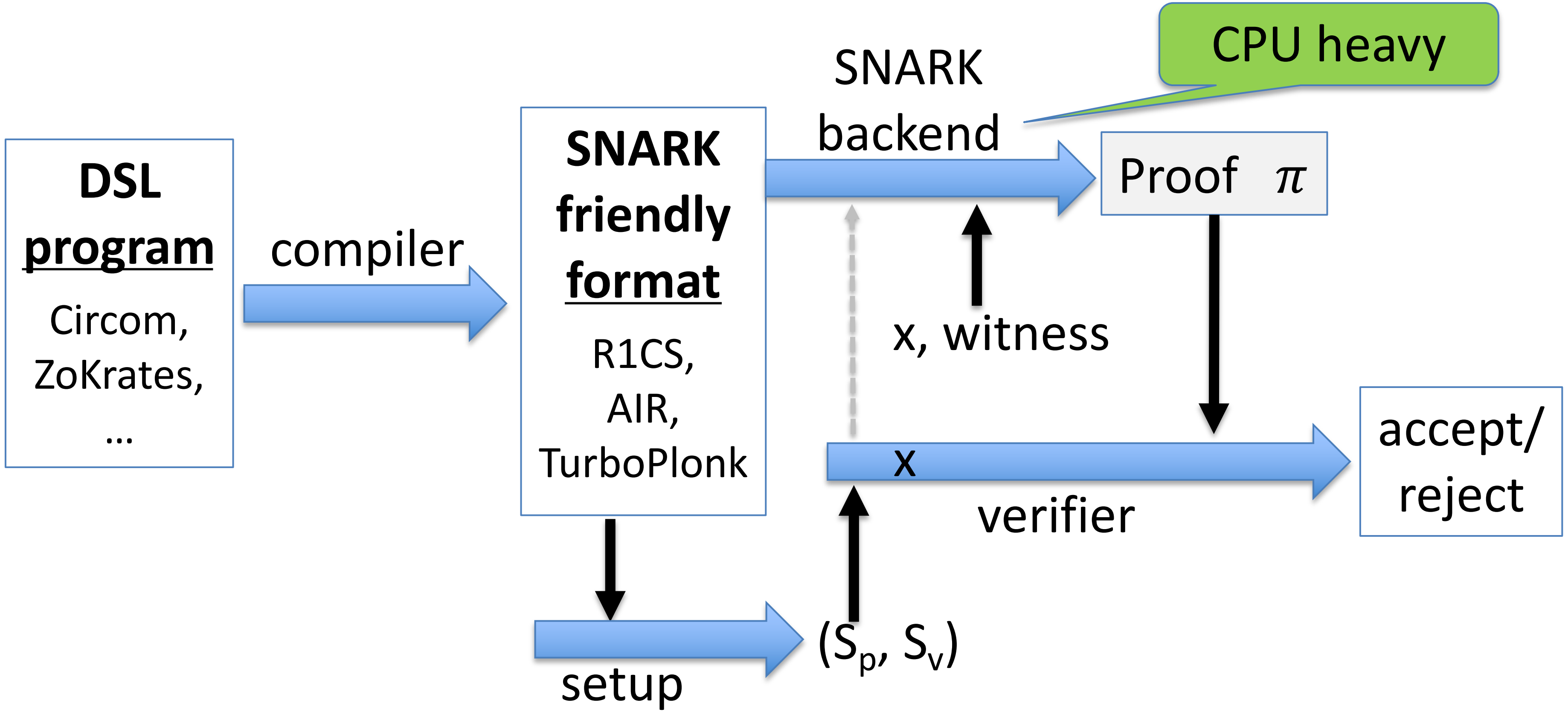
- **Kilian'92, Micali'94**: succinct transparent arguments from PCP
    - impractical prover time
- **GGPR'13**, **Groth'16, ...**: linear prover time, **constant size proof** ($O_\lambda(1)$)
    - **trusted setup per circuit** (setup alg. uses secret randomness)
    - compromised setup $\Rightarrow$ proofs of false statements
- **Sonic'19, Marlin'19, Plonk'19**, ... : universal trusted setup
- **DARK'19, Halo'19, STARK**, ... : no trusted setup (transparent)

Types of SNARKs (partial list)

	size of $\lvert\pi\rvert$	size of $\lvert S_p \rvert$	verifier time	trusted setup?
Groth'16	O(1)	$O(\lvert C \rvert)$	O(1)	yes/per circuit
PLONK/MARLIN	O(1)	$O(\lvert C \rvert)$	O(1)	yes/updatable
Bulletproofs	$O(\log \lvert C \rvert)$	O(1)	$O(\lvert C \rvert)$	no
STARK	$O(\log \lvert C \rvert)$	O(1)	$O(\log \lvert C \rvert)$	no
DARK	$O(\log \lvert C \rvert)$	O(1)	$O(\log \lvert C \rvert)$	no

⋮ ⋮ ⋮

A typical SNARK software system



ZoKrates Example

Goal: prove knowledge of a hash (SHA256) preimage of $x \in \{0,1\}^{256}$

- For a public x, prover knows $w \in \mathbb{F}_p$
- $\mathbb{F}_p$ is a 254-bit prime field

Compiled into an arithmetic circuits (R1CS) over $\mathbb{F}_p$

```
def main(field x[2],   private field w) -> (field):
   h = sha256packed( w )
   h[0] == x[0]      //  check top 128 bits
   h[1] == x[1]      //  check bottom 128 bits
   return 1
```

zkSNARK applications

Blockchain Applications

Scalability:

- SNARK Rollup (zkSNARK for privacy from public)

Privacy: Private Tx on a public blockchain

- Confidential transactions
- Zcash

Compliance:

- Proving solvency in zero-knowledge
- Zero-knowledge taxes

A simple PCP-based SNARK

[Kilian'92, Micali'94]

A simple construction: PCP-based SNARK

<u>The PCP theorem</u>: Let $C(x, w)$ be a circuit where $x \in \mathbb{F}_p^n$.

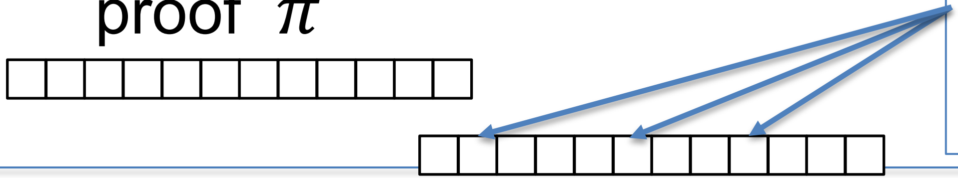
there is a proof system that for every x proves $\exists w: C(x, w) = 0$ as follows:

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

proof π

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

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

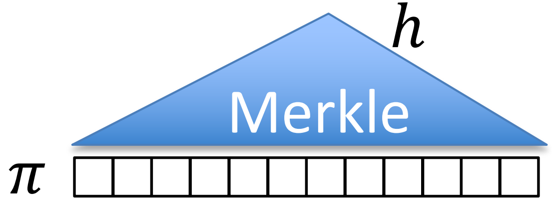


V always accepts valid proof. If no w, then V rejects with high prob.

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

Converting a PCP proof to a SNARK

Prover P(S_p, x, w) Verifier V(S_v, x)



Merkle root h → 1 hash

← open $O(\lambda)$ positions of π

→ $O(\lambda \log |C|)$ hashes

$O(\lambda)$ 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 heuristic:**

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

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

Prover P($\boldsymbol{S_p}$, $\boldsymbol{x}$, $\boldsymbol{w}$) Verifier V($\boldsymbol{S_v}$, $\boldsymbol{x}$)

$\xrightarrow{\text{msg1}}$

$\xleftarrow{\text{r}}$ choose random bits r

$\xrightarrow{\text{msg2}}$

accept or reject

Making the proof non-interactive

Fiat-Shamir heuristic: $H: M \rightarrow R$ a cryptographic hash function

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

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

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

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

Thm: this is a secure SNARK assuming H is a random oracle

Are we done?

Simple transparent SNARK from the PCP theorem

- Use Fiat-Shamir heuristic 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: next lecture! Goal: Time(Prover) = O(|C|)

END OF LECTURE

Next lecture: zkSNARK applications