Verifiable Delay Functions

Dan Boneh, Joe Bonneau,

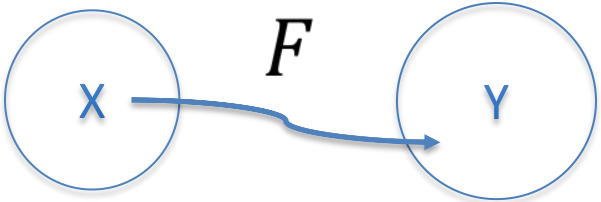
Benedikt Bünz, Ben Fisch

Crypto 2018

What is a VDF?

- **Function** – unique output for every input
- **Delay –** can be evaluated in time T

  cannot be evaluated in time (1-ϵ)T on parallel machine
- **Verifiable** – correctness of output can be verified efficiently



Verifier

What is a VDF?

- **Setup(λ, T)** $\longrightarrow$ public parameters ***pp***
  - *pp* specify domain **X** and range **Y**
- **Eval(*pp*, *x*)** $\longrightarrow$ output ***y***, proof $\boldsymbol{\pi}$
  - PRAM runtime T with polylog(T) processors
- **Verify(*pp*, *x*, *y*, $\boldsymbol{\pi}$)** $\longrightarrow$ { *yes*, *no* }
  - Time complexity at most polylog(T)

Security Properties (Informal)

- Setup(λ, T) $\longrightarrow$ public parameters pp
- Eval(pp, $\boldsymbol{x}$) $\longrightarrow$ output $\boldsymbol{y}$, proof $\boldsymbol{\pi}$ (requires T steps)
- Verify(pp, $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{\pi}$) $\longrightarrow$ { *yes*, *no* }

"**Soundness**": if Verify(pp, x, $\boldsymbol{y}$, $\boldsymbol{\pi}$) = Verify(pp, x, $\boldsymbol{y'}$, $\boldsymbol{\pi'}$) = yes
then $\boldsymbol{y} = \boldsymbol{y'}$

"σ-**Sequentiality**": if A is a PRAM algorithm, time(A) $\leq \sigma(T)$,

e.g. $\sigma(T) = (1 - \epsilon)T$ then Pr[$\boldsymbol{A(pp, x) = y}$] < negligible(λ)

Related Crypto Primitives

- Time-lock puzzles [RSW’96, BN’00, BGJPVW’16]
  - Trapdoor (secret key) setup per puzzle
  - Not ``publicly verifiable”
- Proof-of-sequential-work [MMV’13, CP’18]
  - Publicly verifiable
  - Not a function (output isn’t unique)

VDF minus any property is “easy”

- Not **Verifiable** – chained one-way function
- No **Delay** – Many *moderately hard* functions with efficient verification, e.g. discrete log $g^y = x$
- Not a **Function** – Proofs of sequential work

Modular square roots [DN'92, LW'15]

Assumption: No O(log(T)) time algorithm can compute (with non-negligible probability) $x^T \mathrm{mod}\ \mathrm{p}$ faster than log(T) sequential multiplications (repeated squaring) for T $\in [1, p)$

Modular square roots [DN'92, LW'15]

- Setup: pick prime p, p = 3 mod 4

- Eval(x): Compute a square root of x mod p ⇔ y = $x^{\frac{p+1}{4}}$

log(p) squarings

- Verify(x, y): $y^2 = x$

1 squaring

proof size = log(p)

Modular square roots

- A “proto-VDF”
  - Eval time: log(p) * M(p)
  - Verify time: M(p)
  - **Problem:** Verify time not polylogarithmic in Eval time

M(p) = time complexity of multiplication mod p

Security Properties (Informal)

- Setup(λ, T) $\longrightarrow$ public parameters *pp*
- Eval(*pp*, $\boldsymbol{x}$) $\longrightarrow$ output $\boldsymbol{y}$, proof $\boldsymbol{\pi}$ (requires T steps)
- Verify(*pp*, $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{\pi}$) $\longrightarrow$ { *yes, no* }

"**Soundness**": if Verify(*pp*, *x*, $\boldsymbol{y}$, $\boldsymbol{\pi}$) = Verify(*pp*, *x*, $\boldsymbol{y'}$, $\boldsymbol{\pi'}$) = yes

then $\boldsymbol{y = y'}$

"σ-**Sequentiality**": if A is a PRAM algorithm, time(A) < $\sigma(T)$,

e.g. $\sigma(T) = (1-\epsilon)T$ then Pr[$\boldsymbol{A(pp, x) = y}$] < negligible(λ)

VDF security more formally…

Sequentiality Game

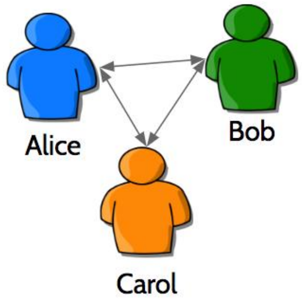
$pp \leftarrow Setup(\lambda, T)$ //sample setup params
$L \leftarrow A_0(pp, T)$ //adversary preprocesses params
$x \leftarrow X$ //choose a random challenge input x
$y_A \leftarrow A_1(L, pp, x)$ //adversary computes output y

$A = (A_0, A_1)$ **"wins"** the game if $y_A = y$ s.t. $Eval(pp, x) = (y, \pi)$

Def: VDF is (p, σ)-sequential if no (A_0, A_1) with A_0 runtime poly(λ) and A_1 PRAM runtime $\sigma(T)$ on $p(T)$ processors wins the game with prob. > negl(λ)

Part I: Applications of VDFs

Randomness beacons



Multiparty randomness

Timestamping

Proof-of-replication

Permissionless consensus

Randomness beacon

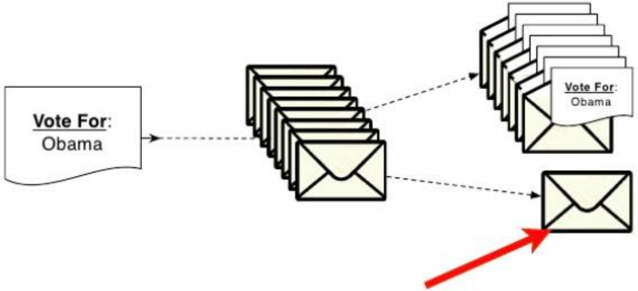
- Rabin '83

An ideal service that regularly publishes random value which no party can predict or manipulate

Many uses for random beacons

Games

Lotteries



Cryptographic proofs

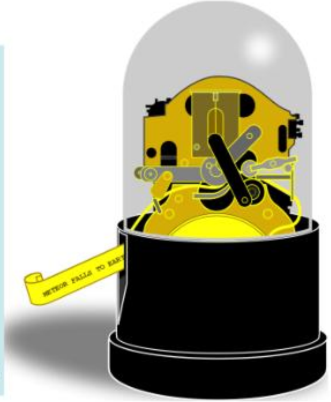
Leader election

Randomness beacon

``Public displays"
are easily corrupted

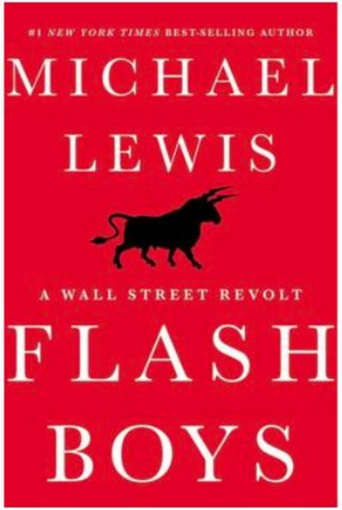
Public entropy source

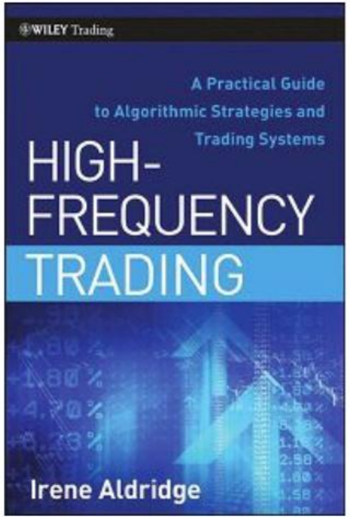
Stock prices [Clark, Hengartner 2010]

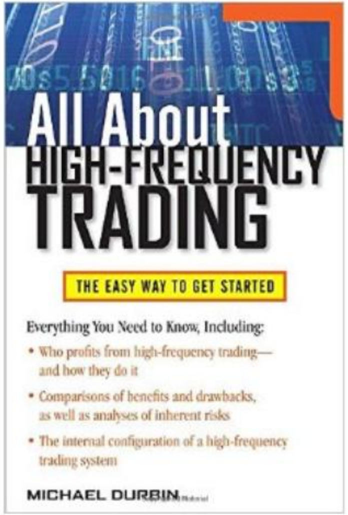


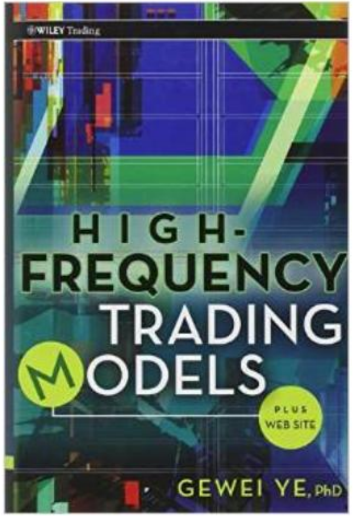
Assumption: (1) unpredictable, (2) adversary cannot fix stock prices

Stock price manipulation







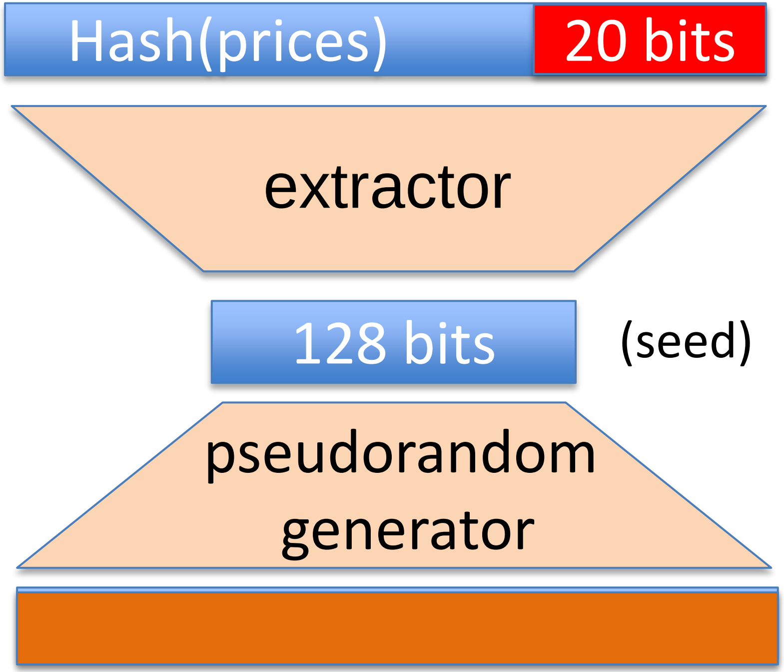


Stock price randomness beacon

Closing prices of 100 stocks:

<u>The problem</u>:

- Once prices settle a minute before closing, attacker executes 20 last-minute trades to influence seed.
- Attacker can **predict** outcome of trades and choose favorable trades to **bias** result

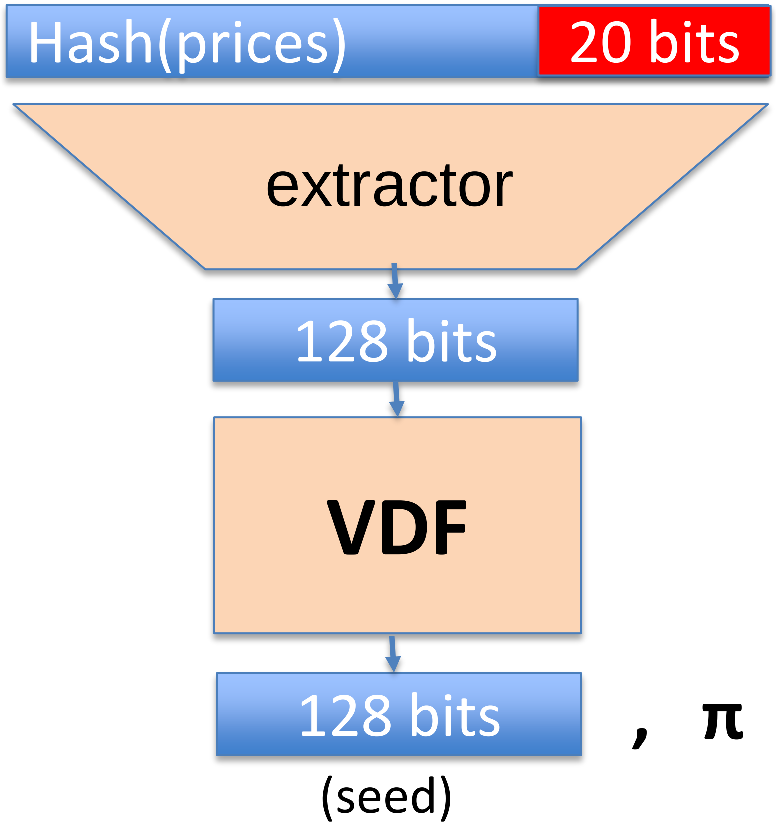


Solution: slow things down with a VDF

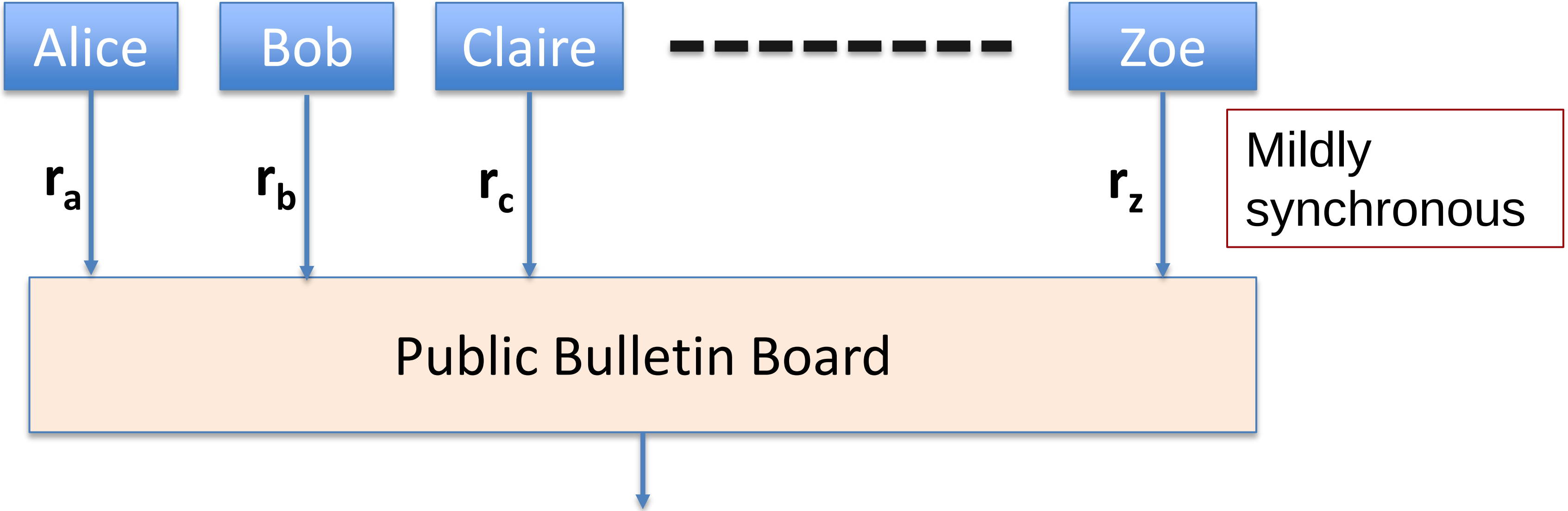
A solution: **one hour VDF**

- Attacker cannot tell what trades to execute before market closes

Uniqueness: ensures no ambiguity about output



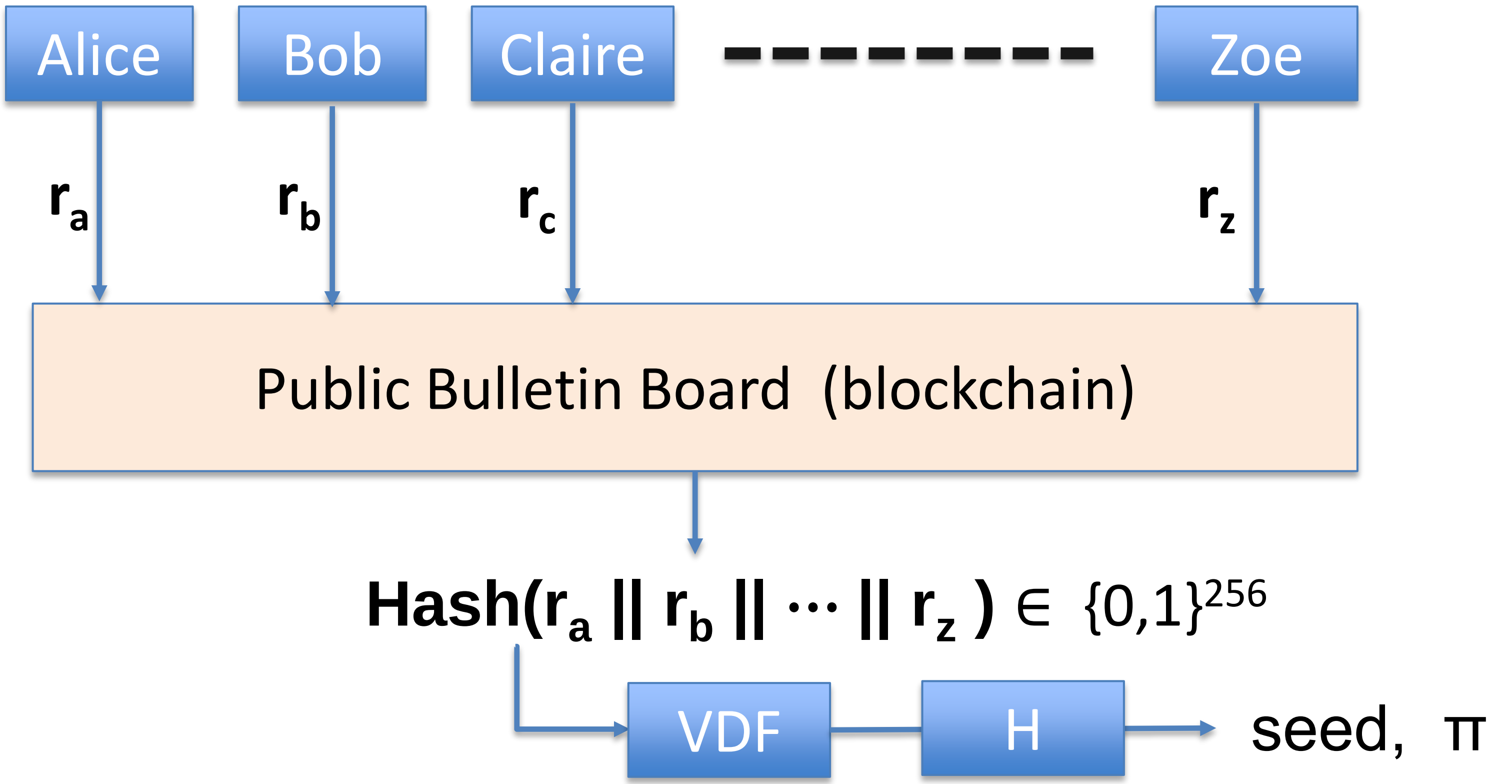
Simple Bulletin Board



output $\mathbf{seed = Hash(r_a \,||\, r_b \,||\, \cdots \,||\, r_z\,)} \in \{0,1\}^{256}$

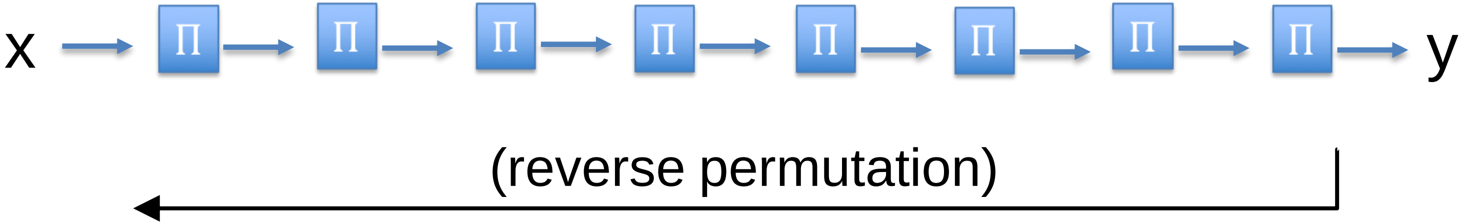
Problem: Zoe controls the final seed !!

Solution: slow things down with a VDF [LW'15]



Part II: Constructions

I.



SNARK/STARK proof π

This work

II.

$$y = g^{2^{2^t}} \in G$$

π = {proof of correct exponentiation}

Assumption: the group G has <u>unknown size</u>

Followup:
Pietrzak'18, Wesolowski'18

Hash Chain w/ Verifiable Computation

$$x \to \underbrace{H(x) \to H(H(x)) \to \cdots \to H^{(t)}(x)}_{\pi} = y$$

- SNARK = "succinct non-interactive argument of knowledge" [G'10,GGPR'13, BCIOP'13, BCCT'13]
- STARK = "succinct transparent non-interactive argument of knowledge" [M'00, BBHR'18]

Hash Chain w/ Verifiable Computation

$$\text{x} \rightarrow H(x) \rightarrow H\big(H(x)\big) \rightarrow \cdots \rightarrow \underbrace{H^{(t)}(x)}_{\pi} = y$$

Problem

- Proof generation slower than hash chain, without massive parallelism

Incrementally Verifiable Computation

- Incrementally verifiable computation, proof carrying data [Val08, BCCT12]

- A σ-sequential VDF with $\sigma(t) = (1 - \epsilon)t$ for small ϵ

$$\underbrace{x \to H(x) \to H(H(x))}_{\pi_1} \to \underbrace{H^{(3)}(x) \to H^{(4)}(x)}_{\pi_2} \to \underbrace{\cdots \to H^{(t)}(x)}_{\pi_{final}}$$

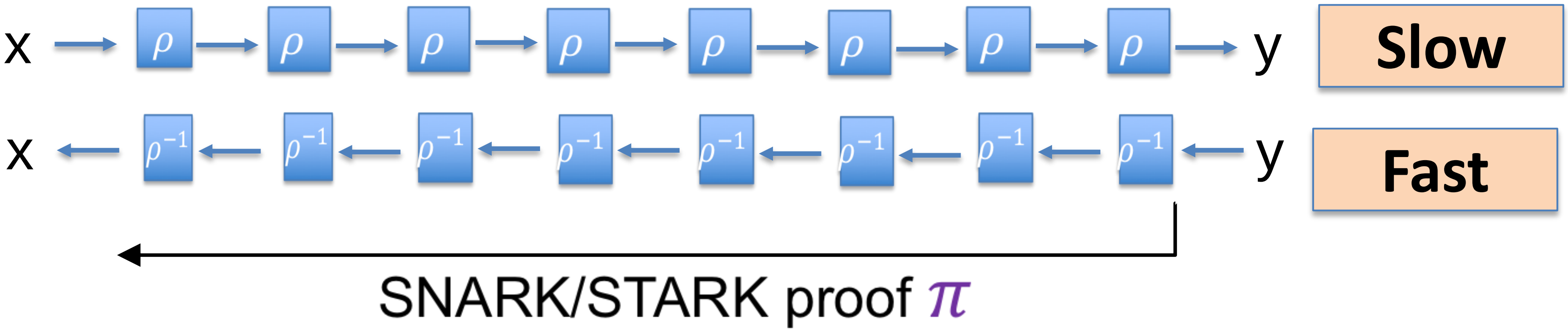
IVC SNARK Optimizations

$$x \to H(x) \to H(H(x)) \to H^{(3)}(x) \to H^{(4)}(x) \to \cdots \to H^{(t)}(x)$$

$$\pi_1 \longrightarrow \pi_2 \longrightarrow \cdots \longrightarrow \pi_{final}$$

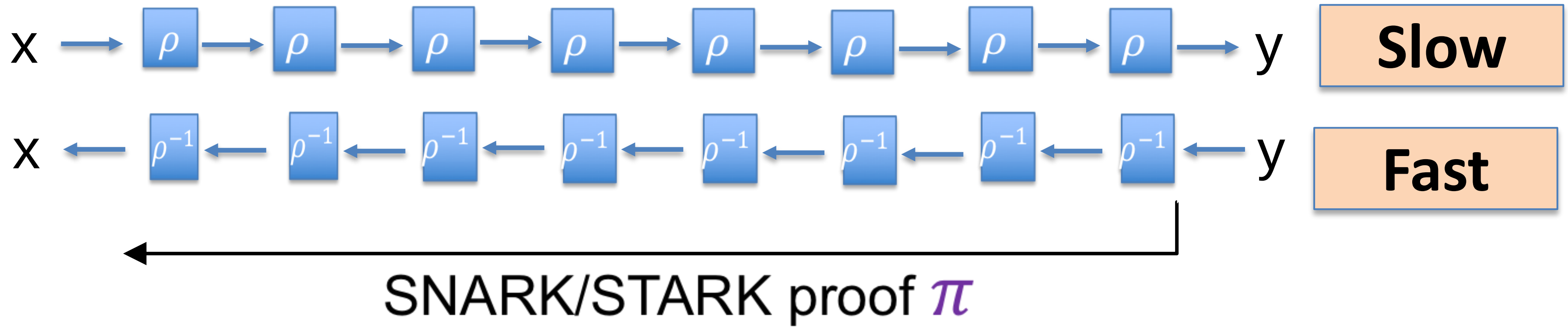
1. Replace H with "SNARK friendly" hash function
   - Low mult. complexity over F_q
   - E.g. MiMC (round function $x \mapsto x^3$) [AGRRT'16]
   - LowMC [ARTTZ'16]

IVC SNARK Optimizations



2. Replace H with permutation ρ that is *slow* in forward direction, but fast / *low complexity* in reverse
   - ➢ SNARK/STARK for the low complexity direction

IVC SNARK Optimizations

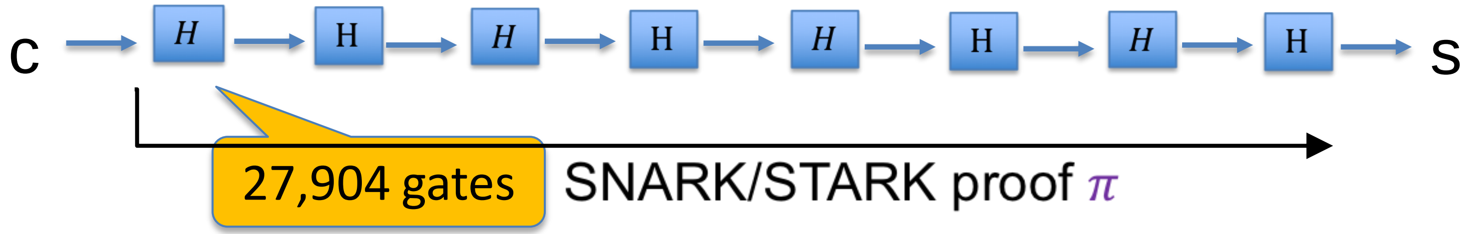


What have we gained?

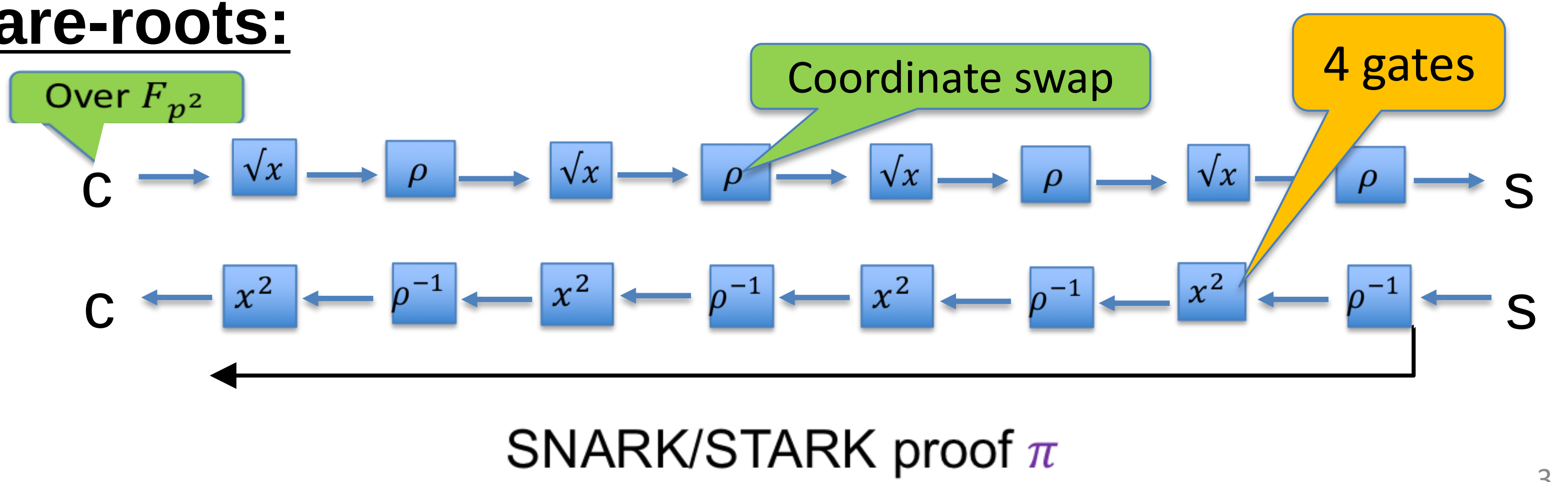
- H can be ``weaker" than VDF, i.e. "proto-VDF" but still asymmetric
  - E.g. square roots mod p, factor 256 asymmetry

Square-roots vs SHA256

SHA256:



Square-roots:



Better asymmetric permutations?

	Slow	Fast
- Square roots / Cube roots - Invert $f(x) = x^3$ over $\mathbb{F}_p$	$y^{1/3}$	x^3
- More general: *injective* polynomial inversion - Find *unique* x such that $f(x) = y$	$f^{-1}(y)$	$f(x)$
- Even more general: *injective rational maps* on algebraic sets		

Permutation polynomials

f is a degree *d* polynomial and $f\colon \mathbb{F}_q \to \mathbb{F}_q$ is a permutation on the field $\mathbb{F}_q$

- Inversion ⇔ find root of $f(x)$ – c
  - Find inverse of c by computing: GCD($x^q - x, f(x)$
  - **Euclidean GCD algorithm: *d* sequential steps**
    - ➢ Each step takes *d* parallel arithmetic operations
  - NC algorithm: O($d^{3.85}$) parallel processors [CDDL'97]
    - ➢ Parallel advantage kicks in at $d^{2.85}$ processors

Eval requires d parallelism

$d^{2.85}$ parallel. infeasible for Adv.

Permutation Polynomials Holy Grail

Exponentially large

- Tunable degree, independent of field size $|\mathbb{F}_q|$
- Fast to evaluate (e.g. sparse polynomial)
- No faster way to invert than computing GCD
  (Assuming fewer than $O(d^{2.85})$ parallel processors)

Eval: O(d) PRAM steps

Verify: O(log(d))

Exponential gap!

Permutation polynomials

f is a degree d polynomial and $f: \mathbb{F}_q \to \mathbb{F}_q$ is a permutation of the field $\mathbb{F}_q$

$\boldsymbol{a}$ is not p-1st power

$$\boldsymbol{x^3}$$

$$\boldsymbol{x^{2^{t+1}+1} + x^3 + x} \in \mathbb{F}_{2^{2t+1}}$$

$$\boldsymbol{x^{p^i} - ax} \in \mathbb{F}_{p^m}$$

$\boldsymbol{a}$ is not s-1st power

Guralnick, Müler ’97

$$\left(\frac{\boldsymbol{1}}{\boldsymbol{2x^s}}\right)(\boldsymbol{x^s - ax - a})(\boldsymbol{x^s - ax + a})^{\boldsymbol{s}} + \left((\boldsymbol{x^s - ax + a})^2 + \boldsymbol{4a^2x}\right)^{\frac{\boldsymbol{s+1}}{\boldsymbol{2}}} \in \mathbb{F}_{p^m}$$

Permutation polynomials

f is a degree d polynomial and $f: \mathbb{F}_q \to \mathbb{F}_q$ is a permutation of the field $\mathbb{F}_q$

x^3

$x^{2^{t+1}+1} + x^3 + x \in \mathbb{F}_{2^{2t+1}}$

a is not p-1st power

$x^{p^i} - ax \in \mathbb{F}_{p^m}$

a is not s-1st power

Guralnick, Müler ’97

$$\left(\frac{1}{2x^s}\right)(x^s - ax - a)(x^s - ax + a)^s + \left((x^s - ax + a)^2 + 4a^2x\right)^{\frac{s+1}{2}} \in \mathbb{F}_{p^m}$$

Construction Summary

Verification	O(log(T)) SNARKs
Proof size	O(log(T))
Assumption	SNARK/STARK + Sqr. rts. or ideal perm. polynomial
Trusted setup	None w/ STARKs or using “slower” verification, sequentiality not broken
Quantum resistant	Possibly with STARKs
Simple	No

Newer VDFs [P'18, W'18]

- Let G be a finite cyclic group with generator $g \in G$

$$G = \{1, g, g^2, g^3, \ldots\}$$

- **Assumption**: the group G has <u>unknown size</u>

$$pp = (G, \quad H: X \longrightarrow G)$$

T squarings

- **Eval(pp, x):** output $y = H(x)^{(2^T)} \in G$

  proof **π = (proof of correct exponentiation)** [P'18, W'18]

THE END

https://eprint.iacr.org/2018/601

Survey of VDFs

https://eprint.iacr.org/2018/712.pdf