Blockchains from Proofs of Space and Time: from Spacemint to Chia

Krzysztof Pietrzak

Guest Lecture, Blockchains and Cryptocurrencies (Spring 2019)

Outline

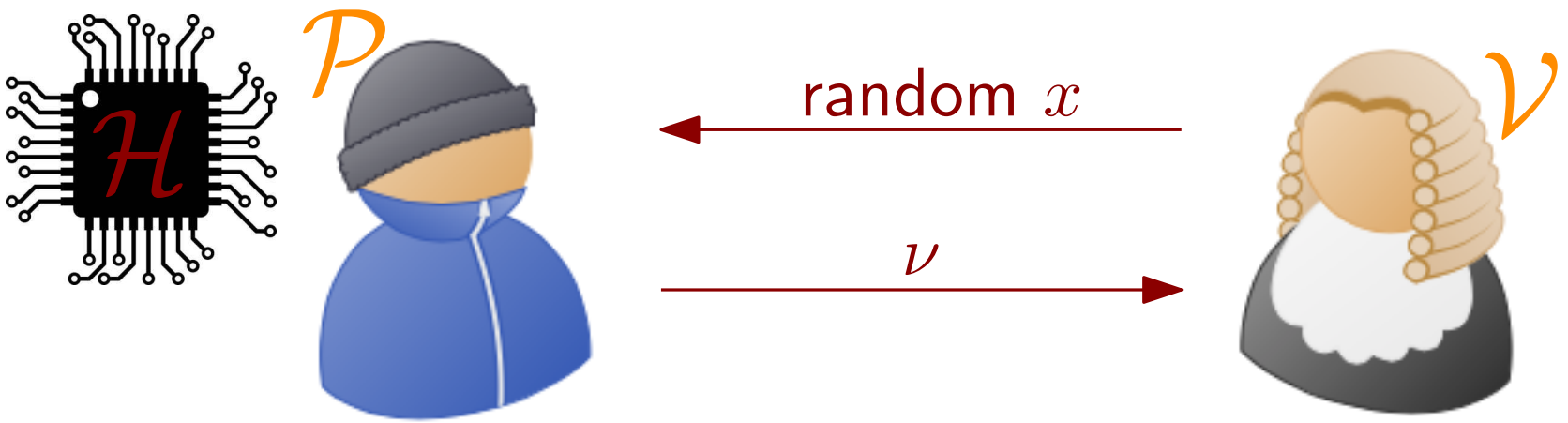
- Bitcoin and Proofs of Work
- Proofs of Stake
- Proofs of Space
- Verifiable Delay Functions
- Putting it all together (Chia)

Proofs of Work

Proof system where prover $\mathcal{P}$ must waste T units of computation to make verifier $\mathcal{V}$ accept.

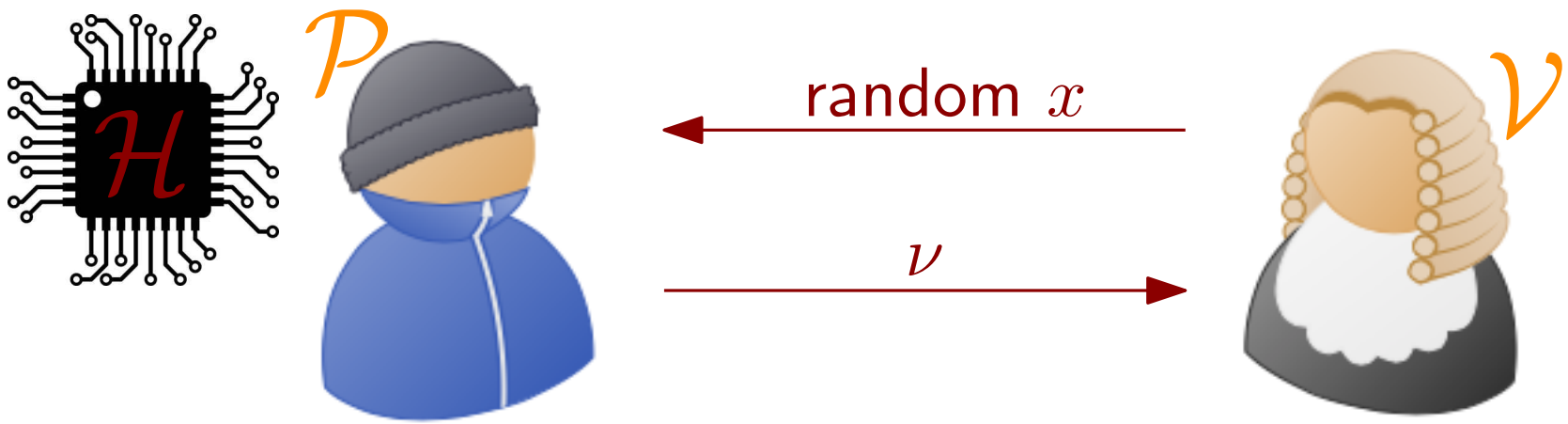
Proofs of Work

Proof system where prover $\mathcal{P}$ must waste T units of computation to make verifier $\mathcal{V}$ accept.



Proofs of Work

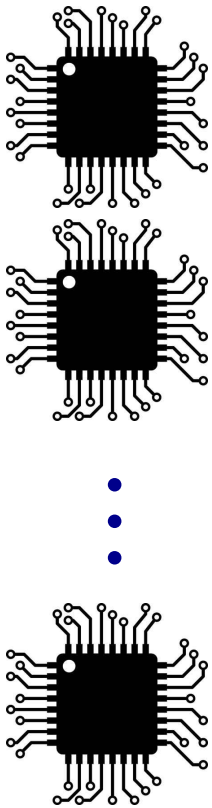
Proof system where prover $\mathcal{P}$ must waste T units of computation to make verifier $\mathcal{V}$ accept.



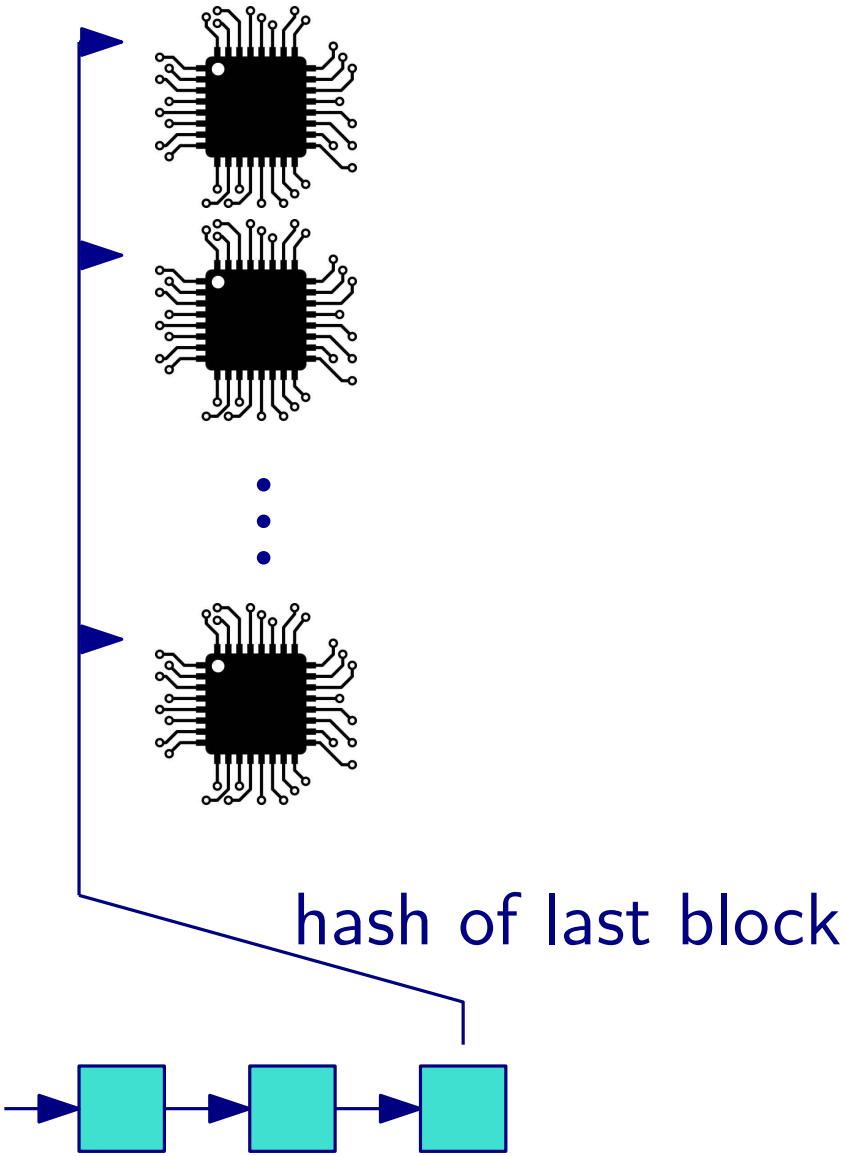
If $\mathcal{H}$ is modelled as a random function, then making an expected D queries is necessary (security) and sufficient (soundness).

$0.\mathcal{H}(\nu, x) \stackrel{?}{\leq} 1/D$

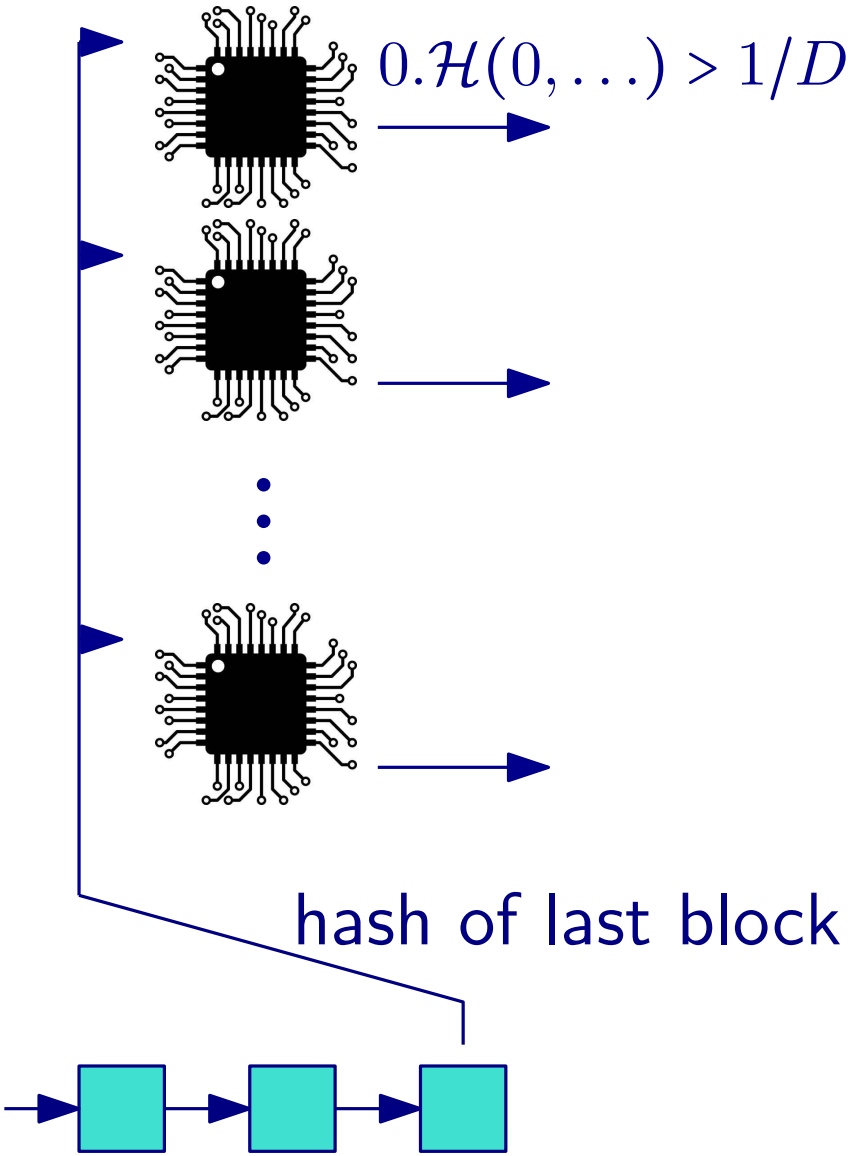
Proofs of Work in Bitcoin



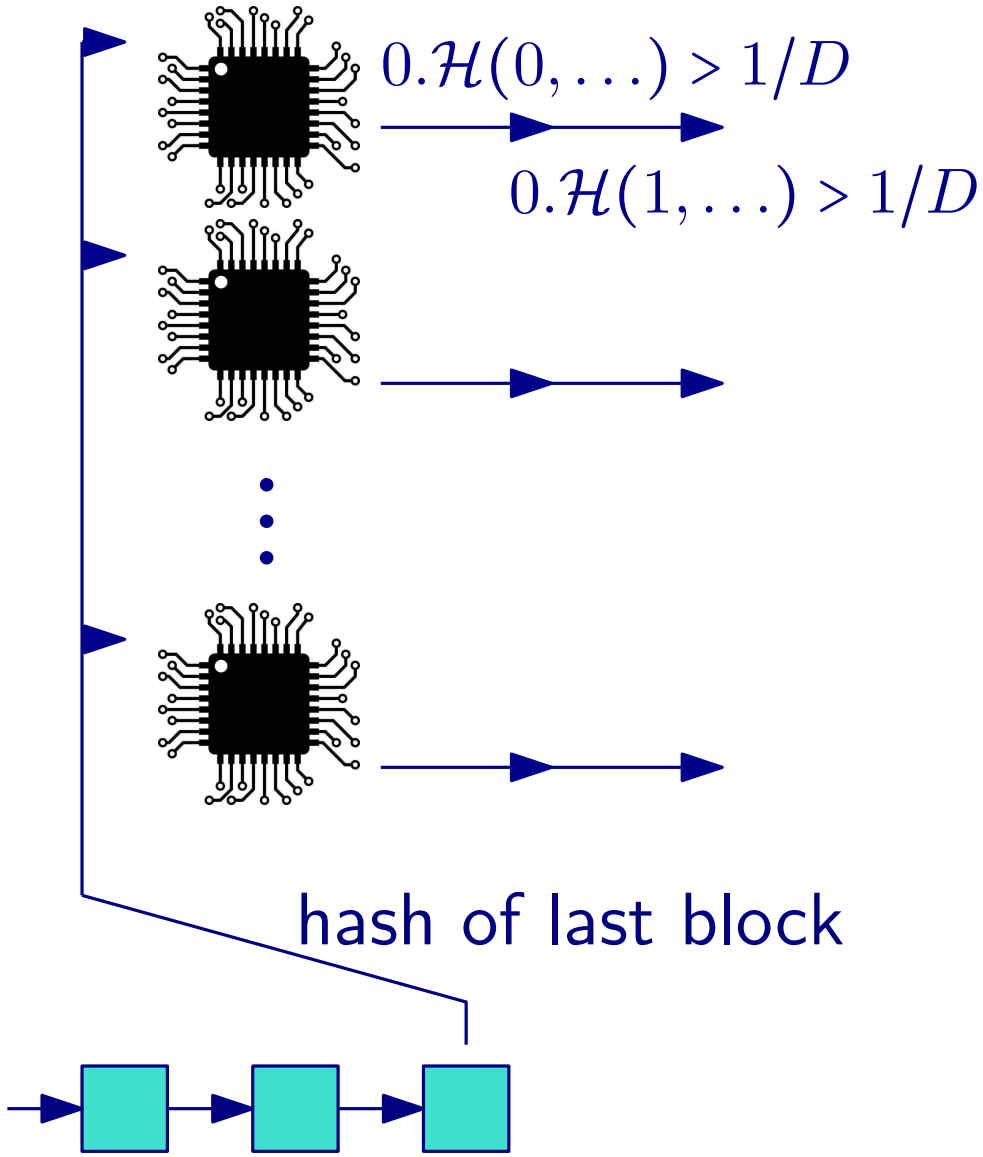
Proofs of Work in Bitcoin



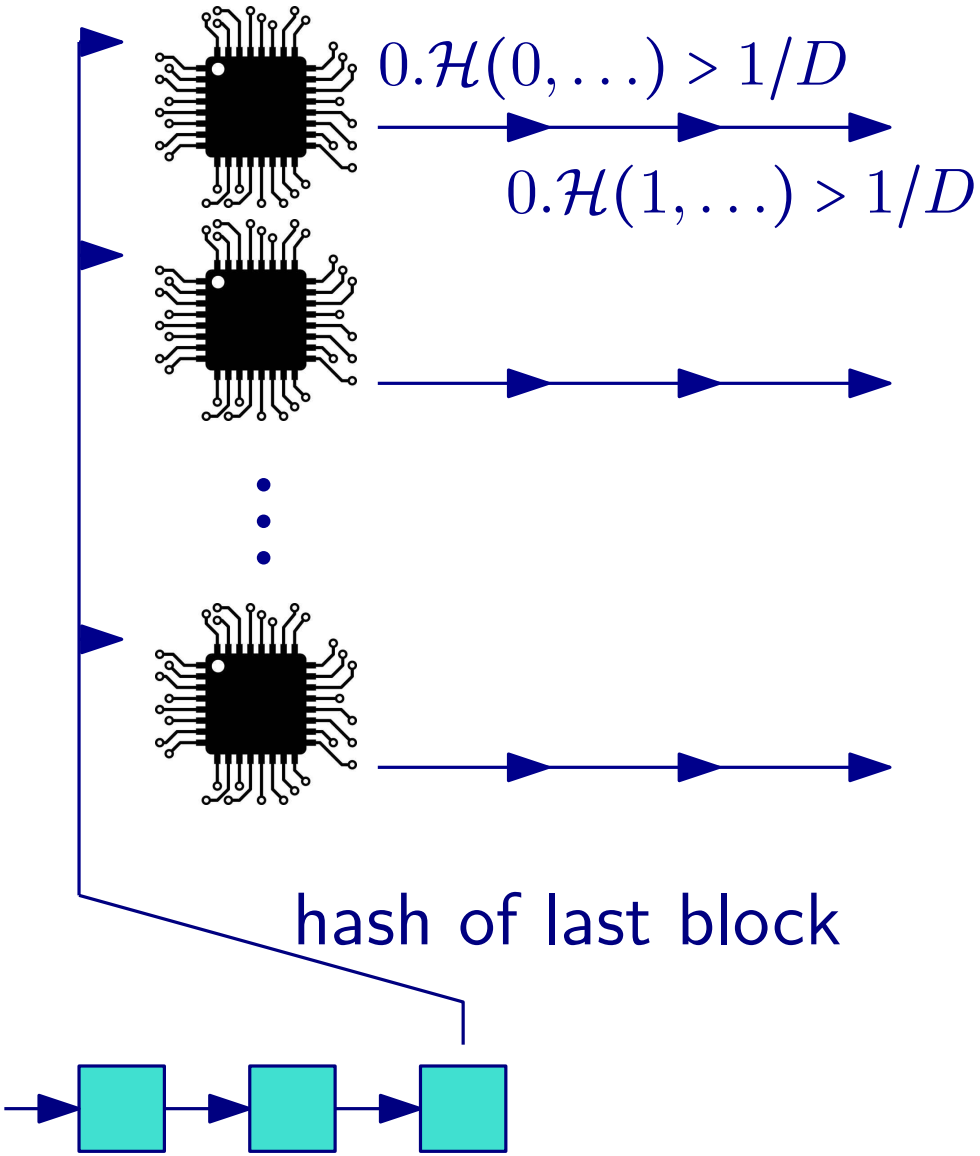
Proofs of Work in Bitcoin



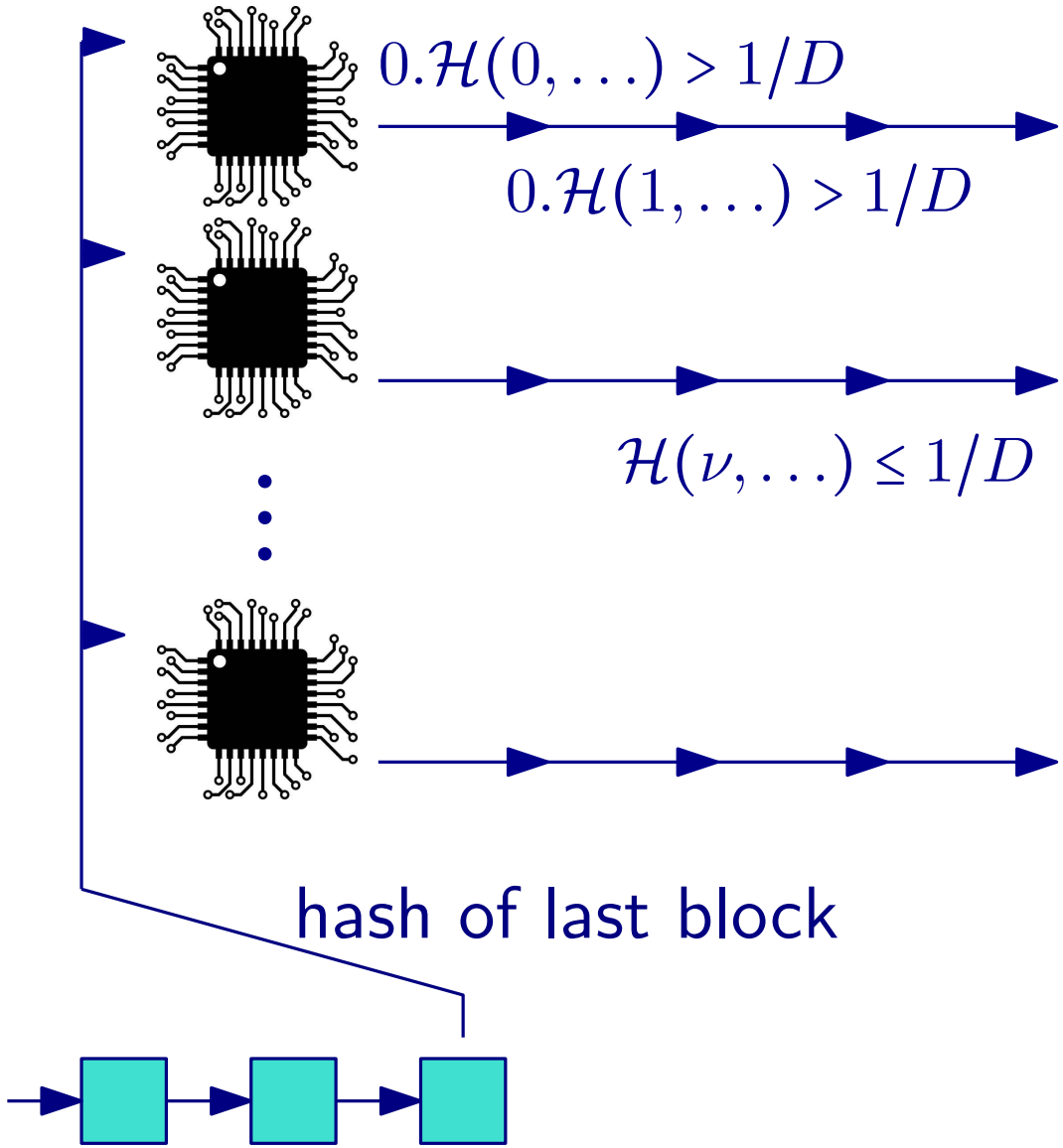
Proofs of Work in Bitcoin



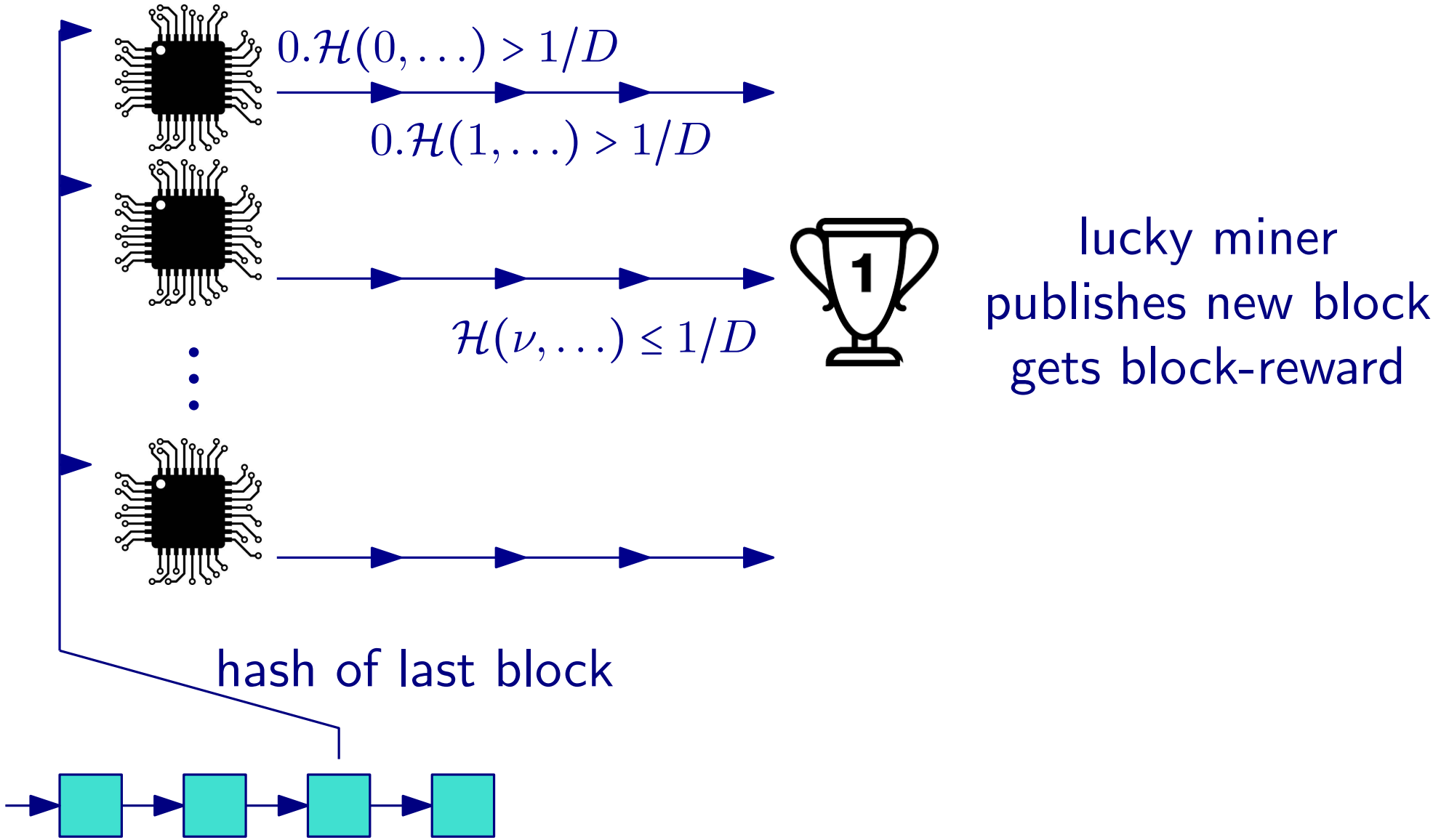
Proofs of Work in Bitcoin



Proofs of Work in Bitcoin



Proofs of Work in Bitcoin



Mining Bitcions

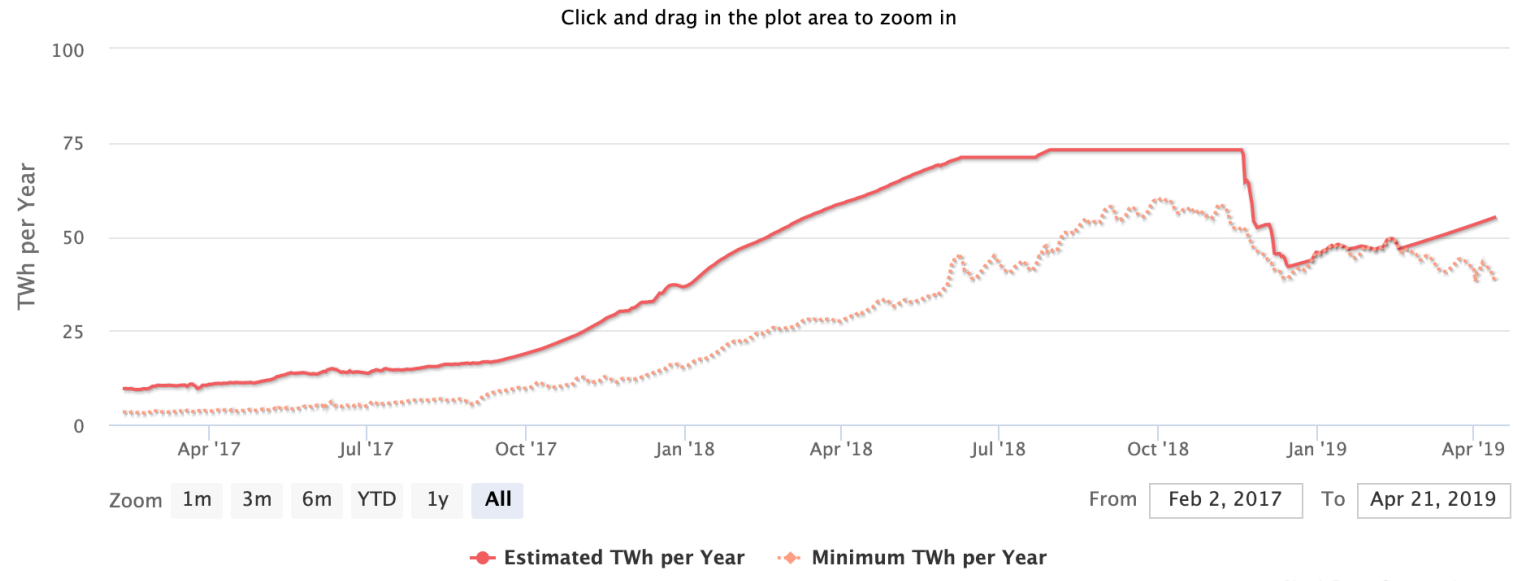
Mining Bitcions

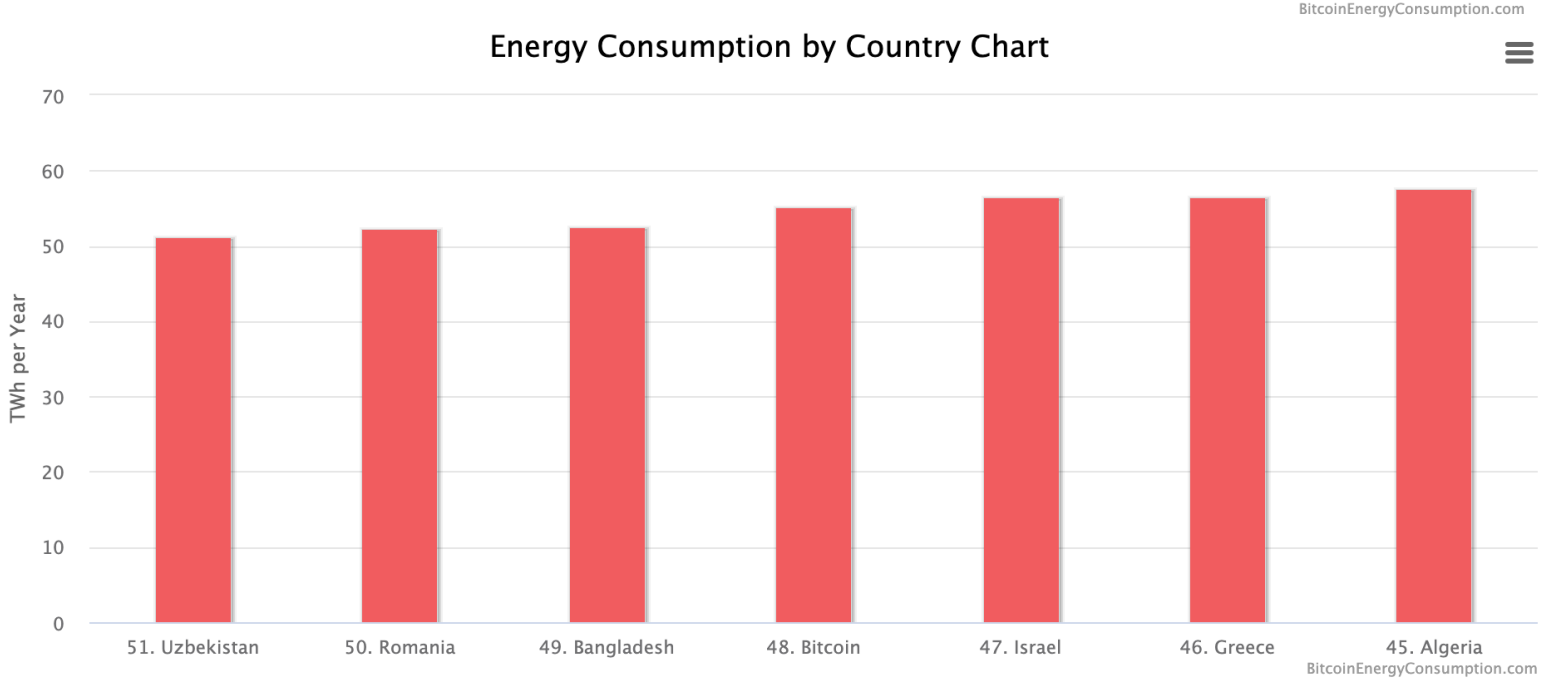
Mining Bitcions

https://digiconomist.net/bitcoin-energy-consumptior

Bitcoin Energy Consumption Index

NEW: Bitcoin Electronic Waste Monitor





Can we have a more "sustainable" Blockchain?

Using Stake instead of Work as Resource?

PoW based blockchain (Bitcoin): Probability a miner can add a block proportional to its hashing power.

Proof of Stake: Probability proportional to the fraction of coins the miner owns.

Using Stake instead of Work as Resource?

PoW based blockchain (Bitcoin): Probability a miner can add a block proportional to its hashing power.

Proof of Stake: Probability proportional to the fraction of coins the miner owns.

Nxt, Algorand, Snow White, Ouroboros,...

Using Stake instead of Work as Resource?

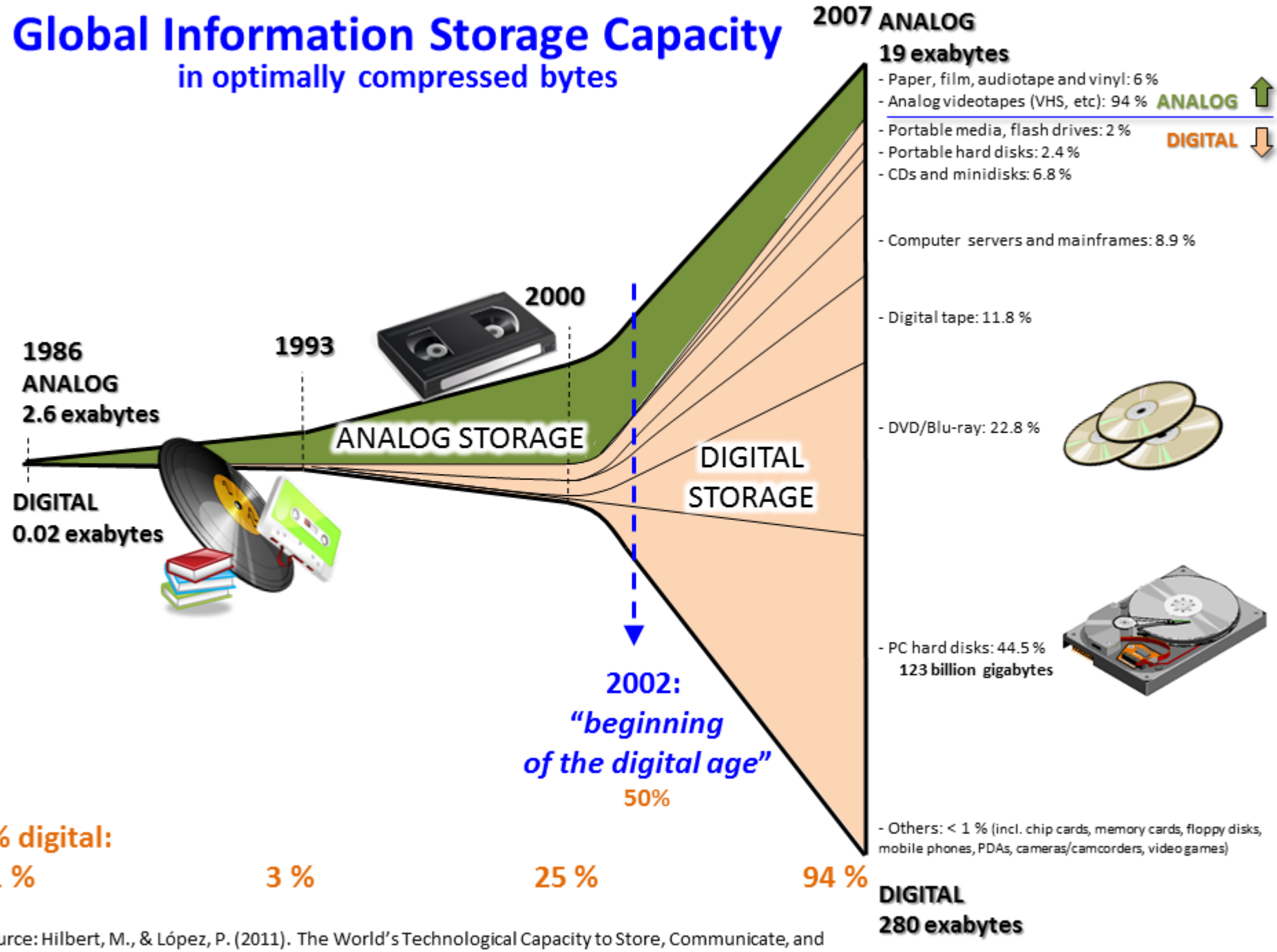
PoW based blockchain (Bitcoin): Probability a miner can add a block proportional to its hashing power.

Proof of Stake: Probability proportional to the fraction of coins the miner owns.

Very difficulty to get (Bitcoin like) blockchain for various reasons. E.g. the following long range attack cannot be prevented unless we assume checkpoints:

- Buy (or borrow) 51% of coins at time T, then sell (or return) everything.
- Later (when you hold 0%) create longest chain by forking at T. This is possible as you have majority of stake at time T!

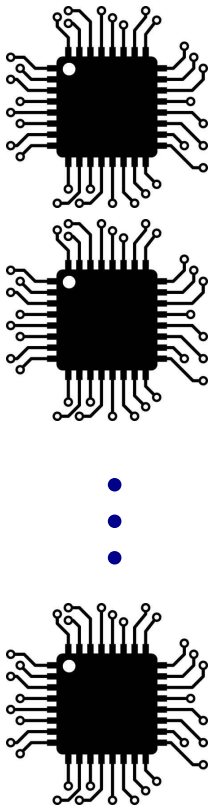
Using (disk)Space instead of Work as Resource?



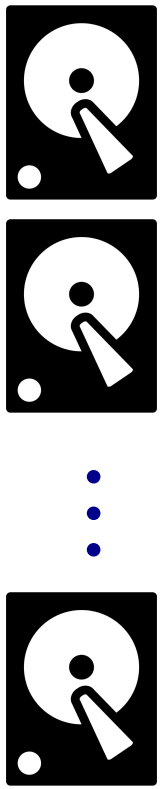
Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. http://www.martinhilbert.net/WorldInfoCapacity.html

https://chia.net/

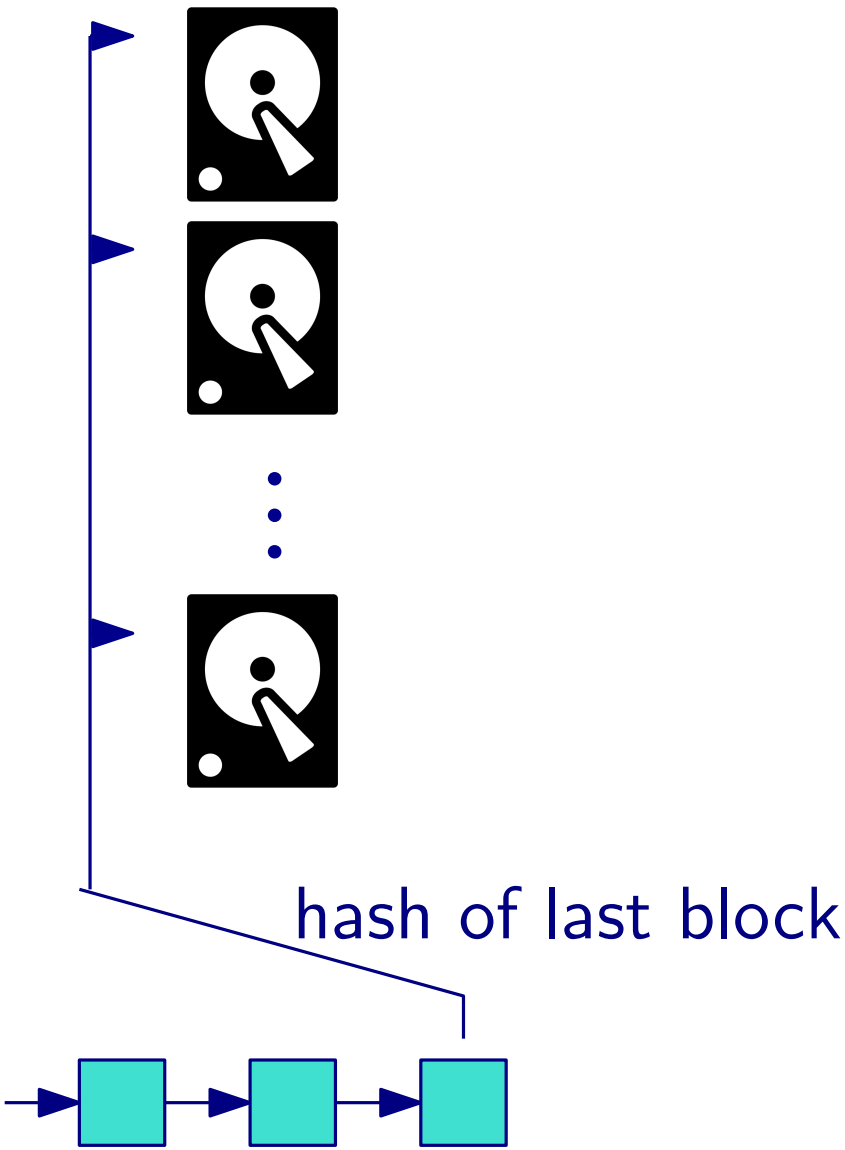
Blockchain by Proofs of Space and VDFs (Chia Network)



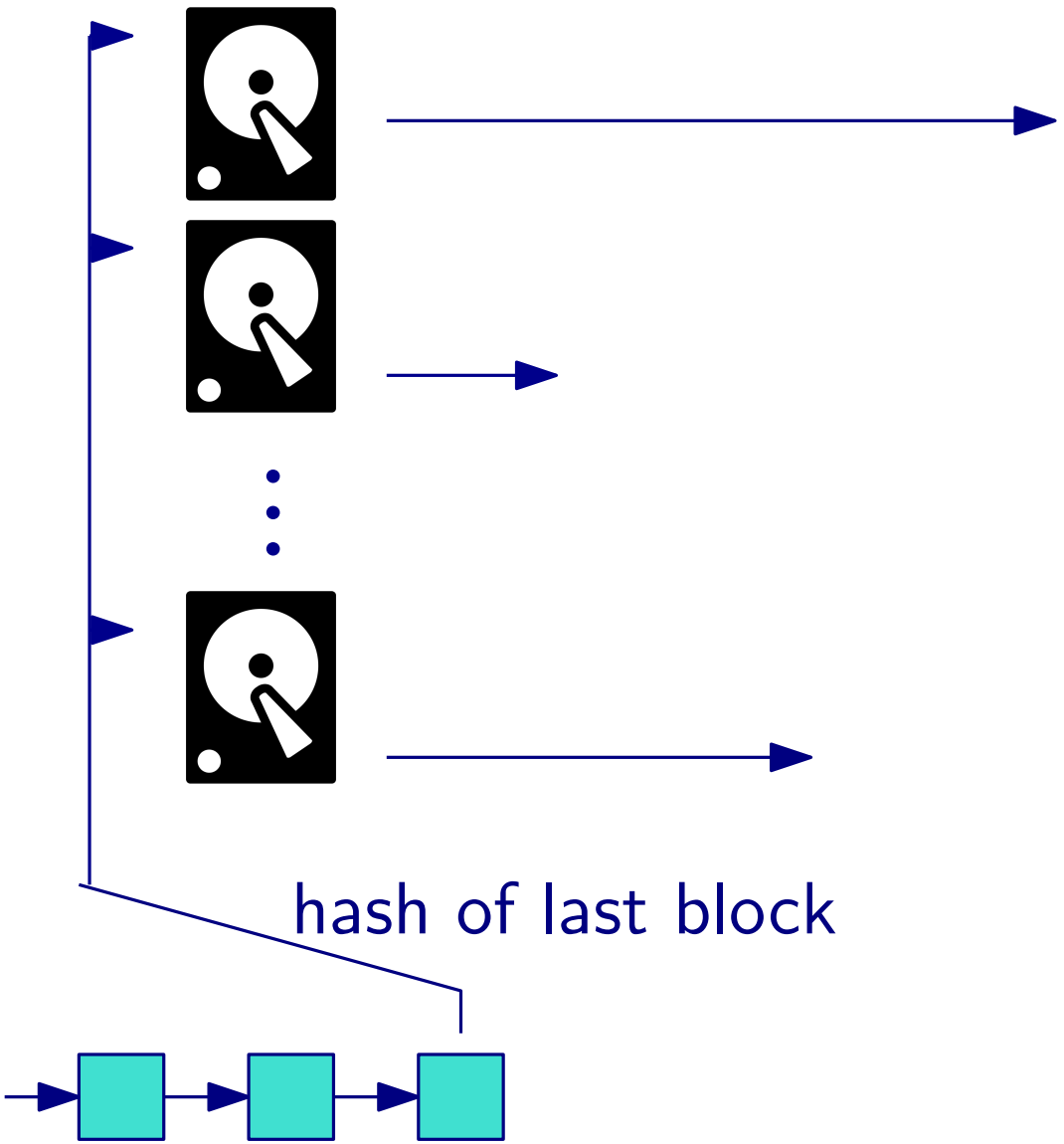
Blockchain by Proofs of Space and VDFs (Chia Network)



Blockchain by Proofs of Space and VDFs (Chia Network)

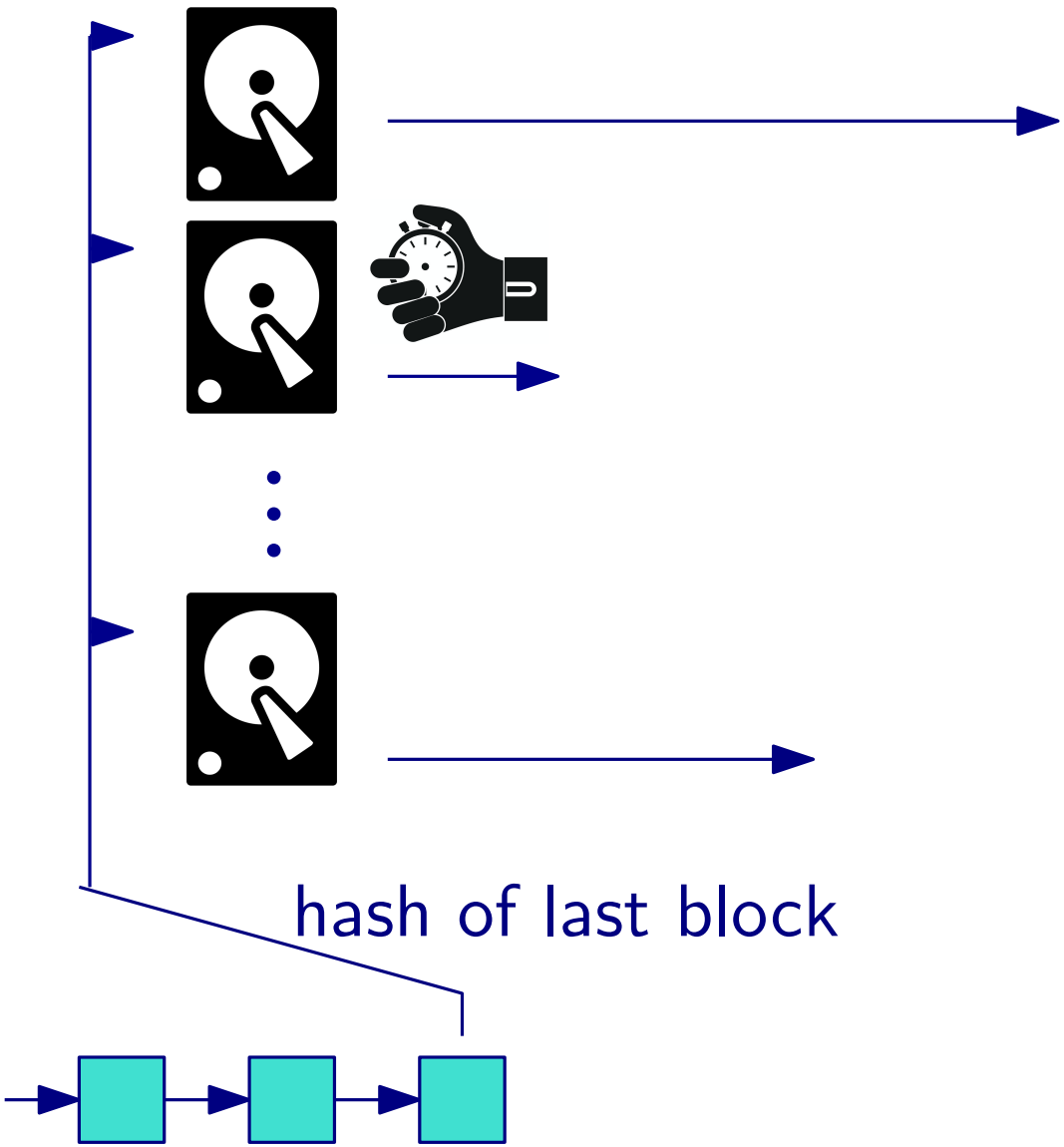


Blockchain by Proofs of Space and VDFs (Chia Network)



each miner gets proof of random quality (indicated by length of arrow)

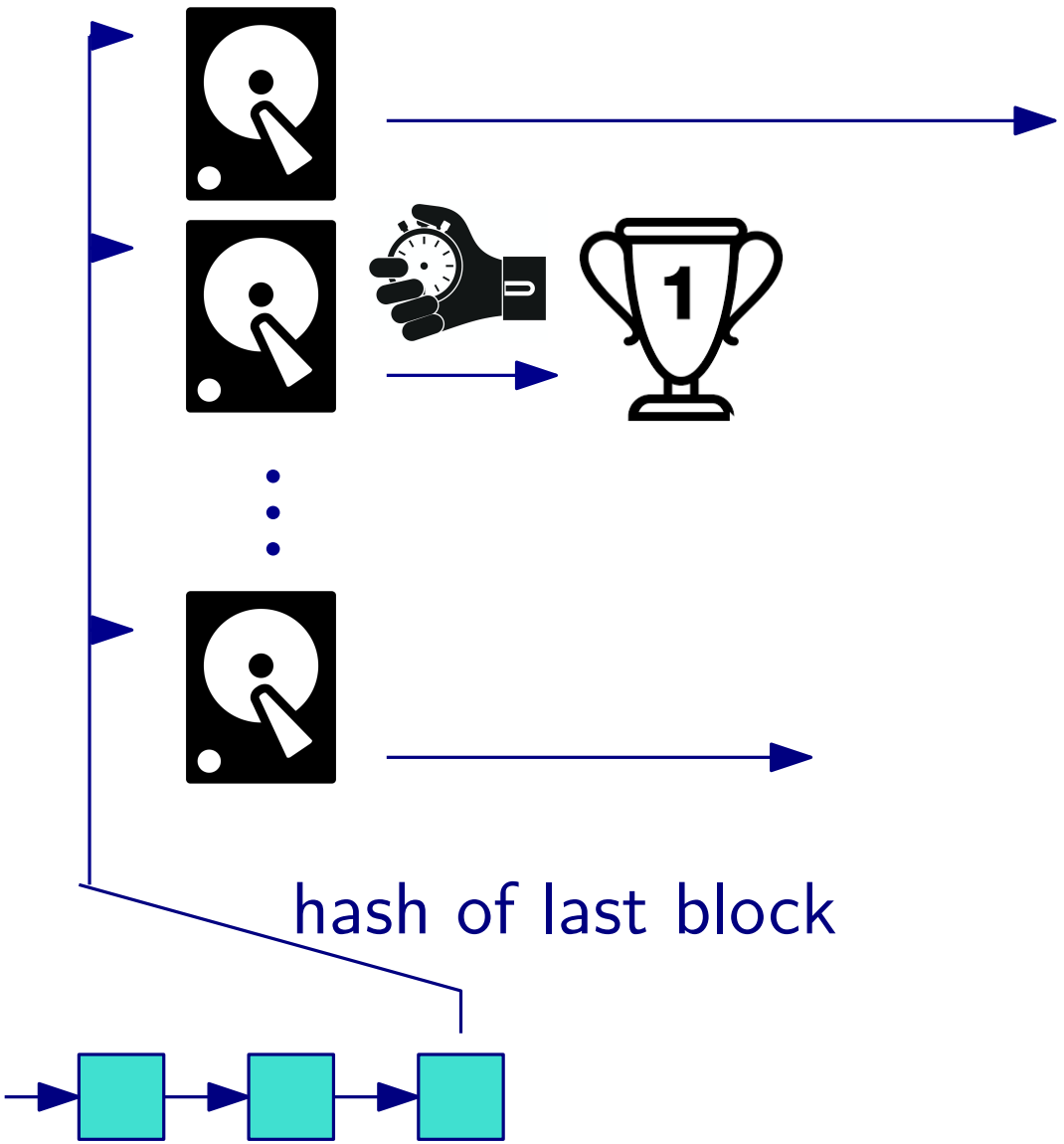
Blockchain by Proofs of Space and VDFs (Chia Network)



each miner gets proof of random quality (indicated by length of arrow)

to complete block run VDF for time=quality

Blockchain by Proofs of Space and VDFs (Chia Network)



each miner gets proof of random quality (indicated by length of arrow)

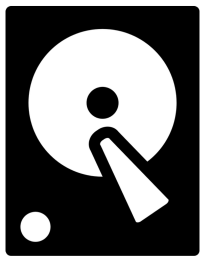
to complete block run VDF for time=quality

First Ingredient

Proofs of Space

Proofs of Space

Proof system where prover $\mathcal{P}$ must waste N units of disk-space to make verifier $\mathcal{V}$ accept.

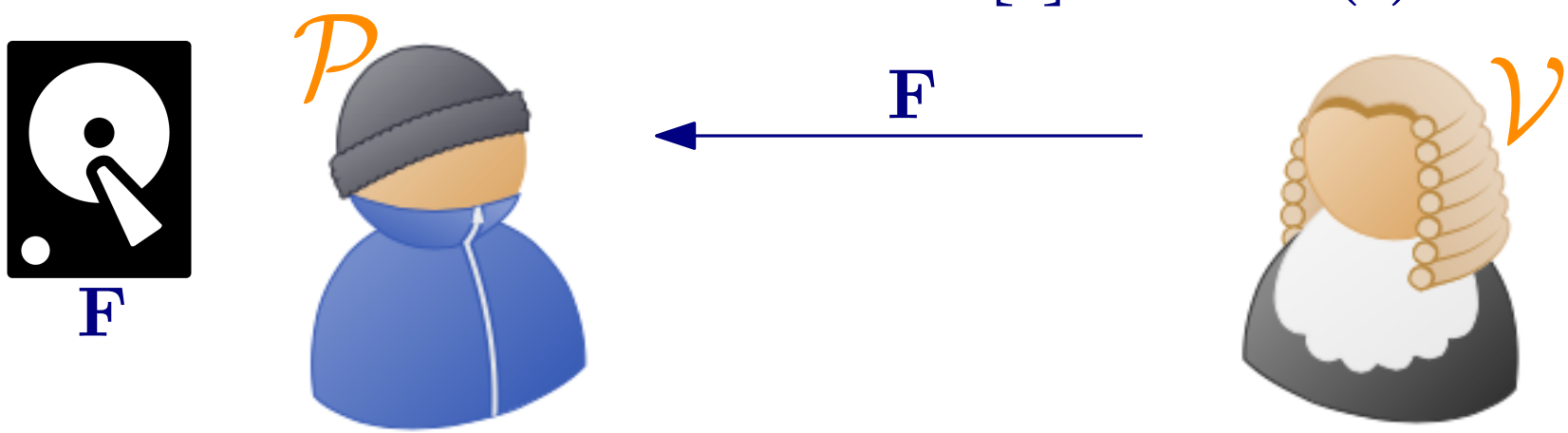


Proofs of Space

Proof system where prover $\mathcal{P}$ must waste N units of disk-space to make verifier $\mathcal{V}$ accept.

- $\mathcal{V}$ samples and sends (pseudo)random file $\mathbf{F}$ of size N to $\mathcal{P}$.
- Later can efficiently audit $\mathcal{P}$ by asking for $\mathbf{F}$ at random positions.

$$\mathbf{F}[i] = PRF_k(i) \text{ for } i = 1 \ldots N$$

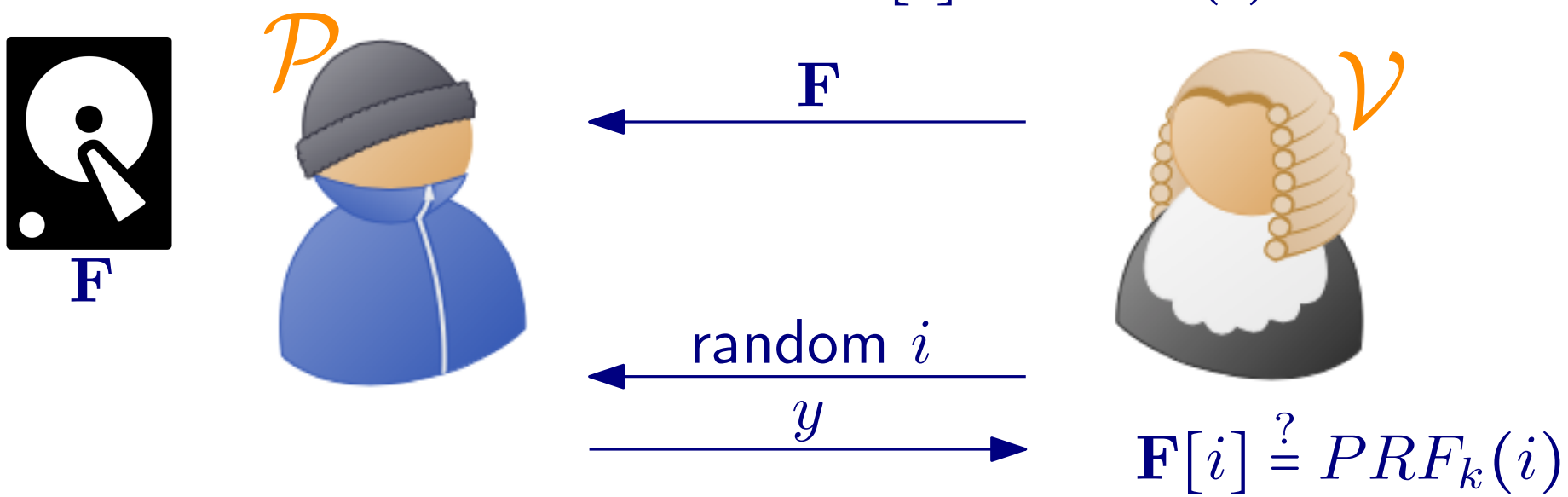


Proofs of Space

Proof system where prover $\mathcal{P}$ must waste N units of disk-space to make verifier $\mathcal{V}$ accept.

- $\mathcal{V}$ samples and sends (pseudo)random file $\mathbf{F}$ of size N to $\mathcal{P}$.
- Later can efficiently audit $\mathcal{P}$ by asking for $\mathbf{F}$ at random positions.

$$\mathbf{F}[i] = PRF_k(i) \text{ for } i = 1 \ldots N$$

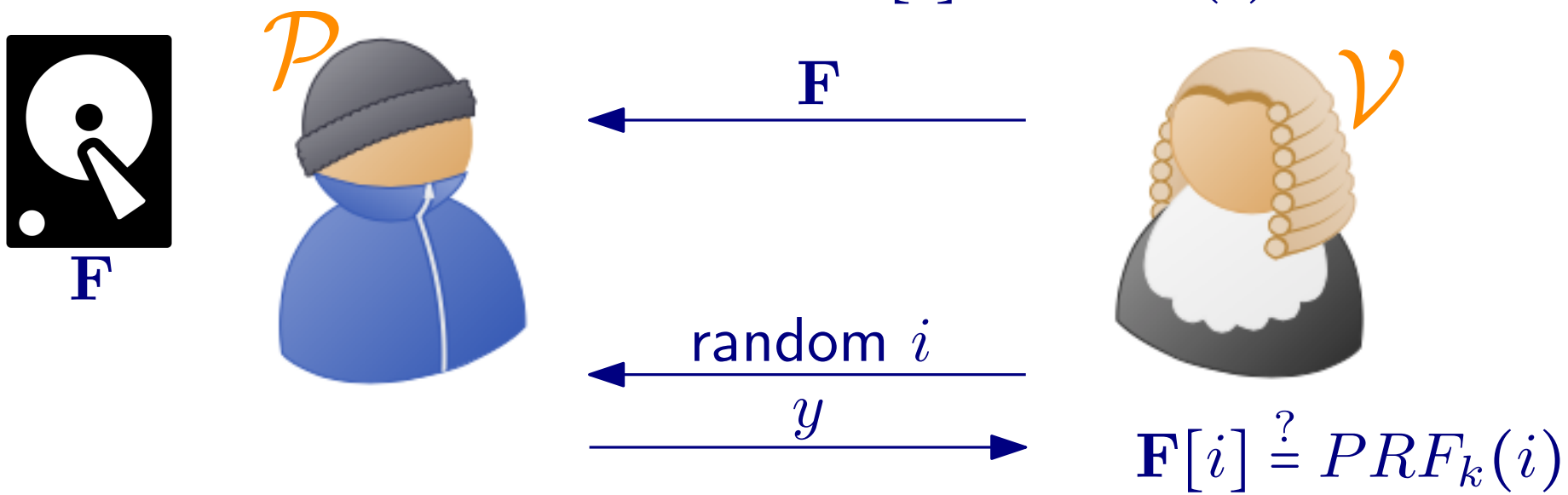


Proofs of Space

Proof system where prover $\mathcal{P}$ must waste N units of disk-space to make verifier $\mathcal{V}$ accept.

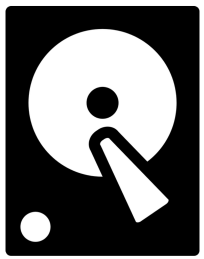
- $\mathcal{V}$ samples and sends (pseudo)random file $\mathbf{F}$ of size N to $\mathcal{P}$.
- Later can efficiently audit $\mathcal{P}$ by asking for $\mathbf{F}$ at random positions.

$$\mathbf{F}[i] = PRF_k(i) \text{ for } i = 1 \ldots N$$



$\mathcal{V}$ has huge (communication)complexity $|\mathbf{F}| = N$.
For blockchain applications $\mathcal{V}$ must be efficient.

Proofs of Space

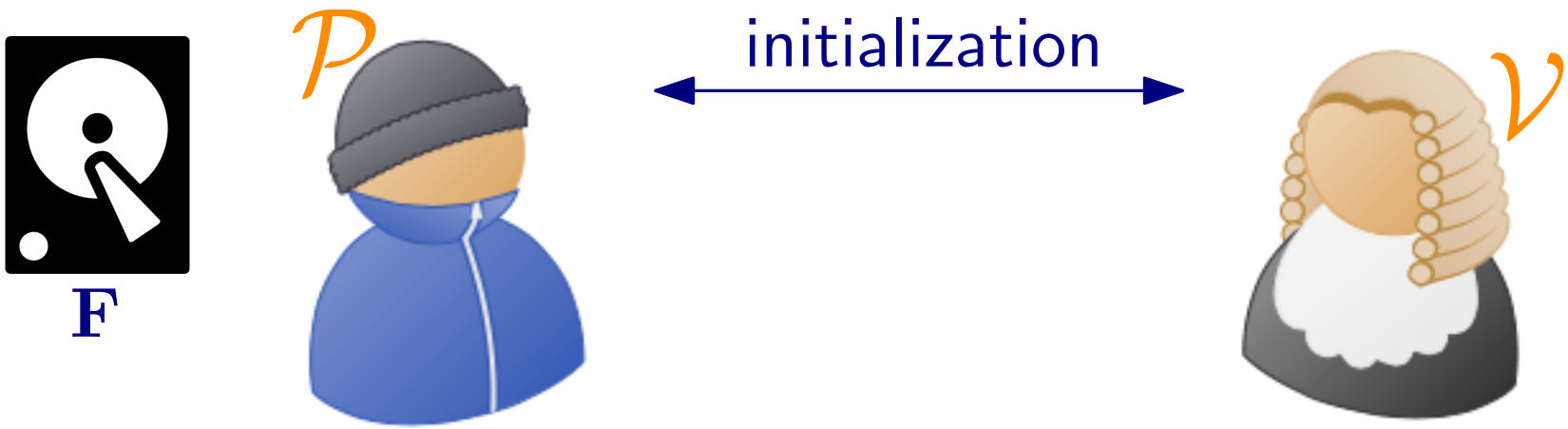


Proofs of Space

initialization phase

Efficient for $\mathcal{V}$

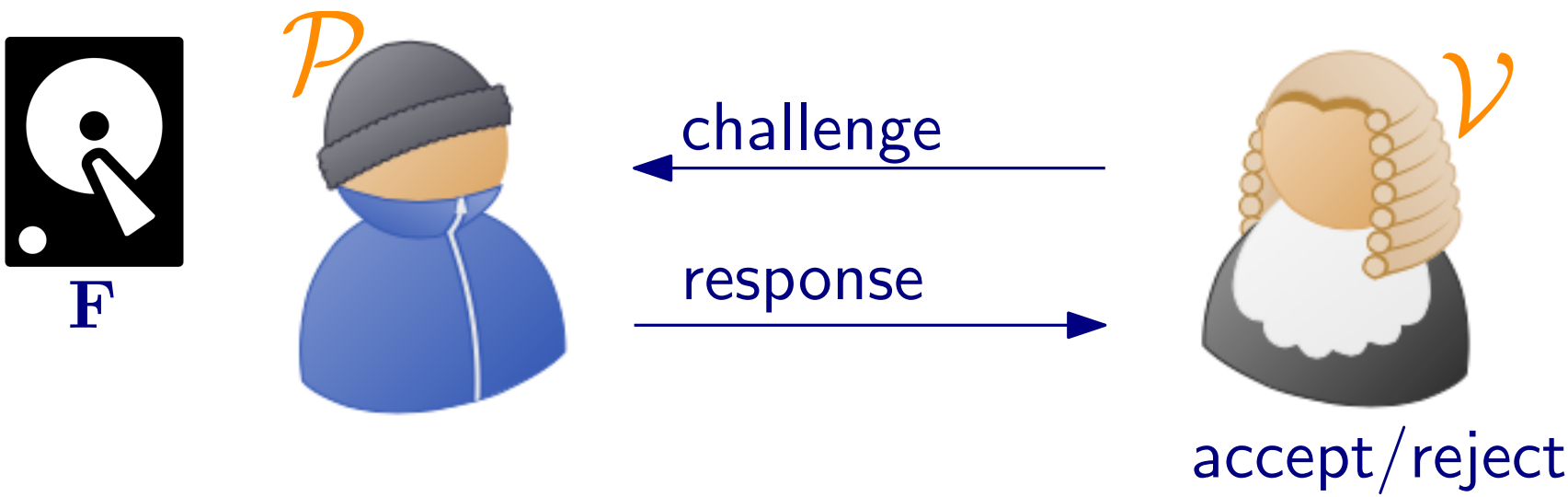
$\mathcal{P}$ runs in time $\tilde{O}(N)$ and at the end stores file $|\mathbf{F}| = N$



Proofs of Space

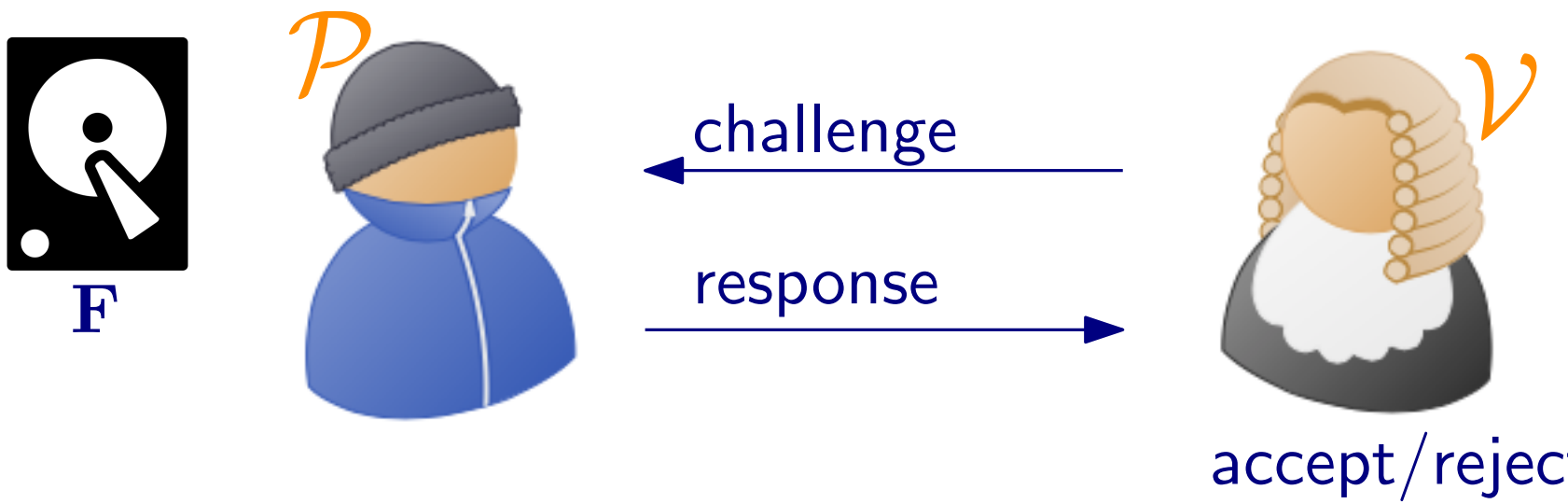
proof execution (or audit)

Efficient for $\mathcal{V}$ and $\mathcal{P}$



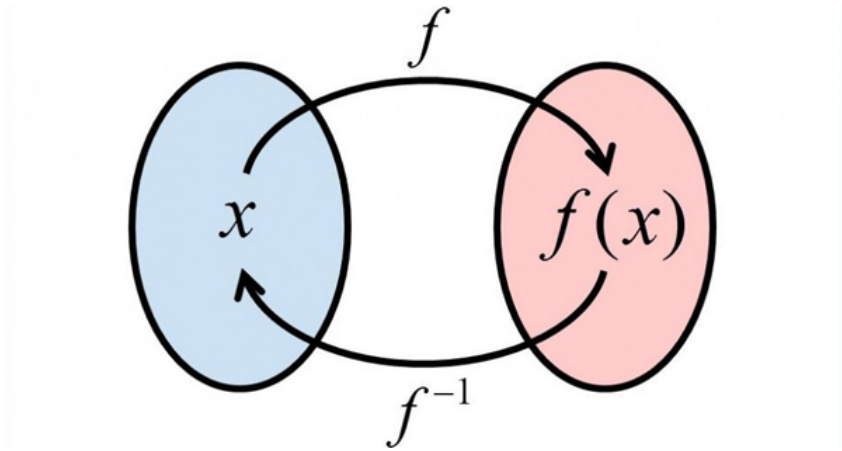
Proofs of Space

proof execution (or audit)
Efficient for $\mathcal{V}$ and $\mathcal{P}$

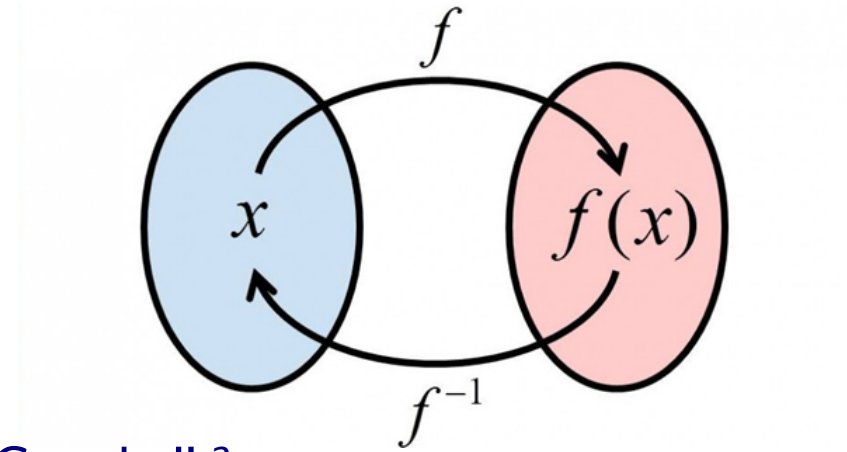


security
Malicious $\tilde{\mathcal{P}}$ who stores $\tilde{\mathbf{F}}$ of size $|\tilde{\mathbf{F}}| \ll |\mathbf{F}| = N$ must run in time $\Omega(N)$ to pass audit.

Two Types of Proofs of Space



Two Types of Proofs of Space

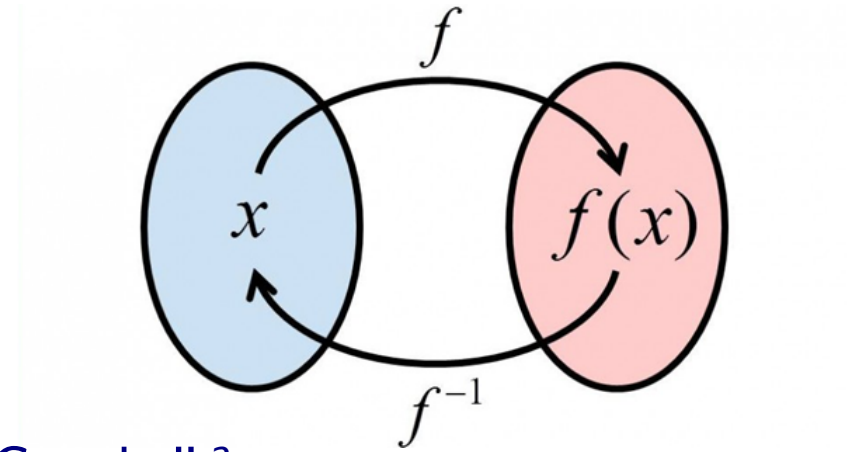


Constructions from "Hard to Pebble Graphs"[a]

- Optimal bounds: either $\Theta(N)$ space or $\Theta(N)$ time
- Non-Interactive Initialization Phase, Complicated

[a]Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, Krzysztof Pietrzak: Proofs of Space. CRYPTO 2015

Two Types of Proofs of Space



Constructions from "Hard to Pebble Graphs"[a]

- Optimal bounds: either $\Theta(N)$ space or $\Theta(N)$ time
- Non-Interactive Initialization Phase, Complicated

[a]Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, Krzysztof Pietrzak: Proofs of Space. CRYPTO 2015

Inverting Random Functions[a]

- Bounds (only) asymptotically optimal: $T \cdot S^k \geq N^k$ for "small" k, e.g. $S = T = N^{k/(1+k)}$ (proof size exponential in k)
- Non-Interactive Initialization Phase, Simple!

[a]H. Abusalah, J. Alwen, B. Cohen, D. Khilko, K. Pietrzak, L. Reyzin: Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space. ASIACRYPT 2017

Spacemint*: A Cryptocurrency Based on Proofs of Space

Sunoo Park*, Krzysztof Pietrzak†, Albert Kwon*, Joël Alwen†, Georg Fuchsbauer†, and Peter Gaži†

* MIT
† IST Austria

Constructions from "Hard to Pebble Graphs"[a]

- Optimal bounds: either $\Theta(N)$ space or $\Theta(N)$ time
- Non-Interactive Initialization Phase, Complicated

[a]Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, Krzysztof Pietrzak: Proofs of Space. CRYPTO 2015

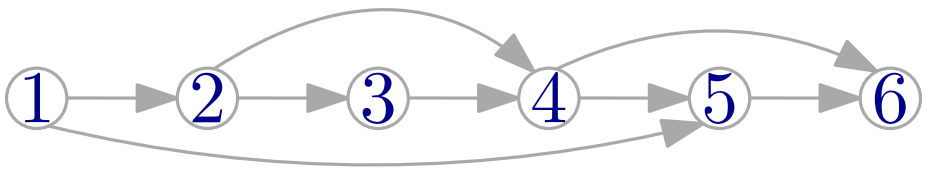
Inverting Random Functions[a]

- Bounds (only) asymptotically optimal: $T \cdot S^k \geq N^k$ for "small" k, e.g. $S = T = N^{k/(1+k)}$ (proof size exponential in k)
- Non-Interactive Initialization Phase, Simple!

[a]H. Abusalah, J. Alwen, B. Cohen, D. Khilko, K. Pietrzak, L. Reyzin: Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space. ASIACRYPT 2017

Two Basic Concepts

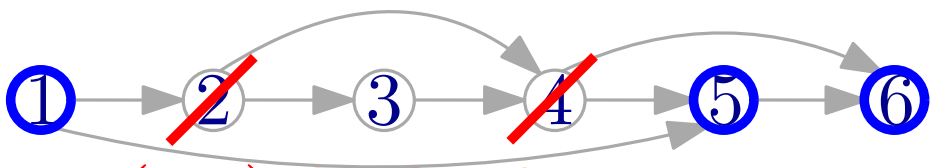
Depth-Robust Graphs



DAG $G = (V, E)$ is (e, d) **depth-robust** if after removing any e nodes a path of length d exists.

Two Basic Concepts

Depth-Robust Graphs

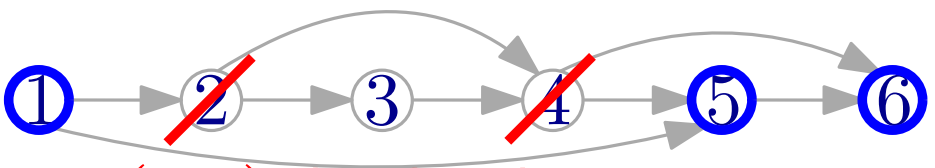


is $(2, 3)$ depth-robust

DAG $G = (V, E)$ is (e, d) **depth-robust** if after removing any e nodes a path of length d exists.

Two Basic Concepts

Depth-Robust Graphs



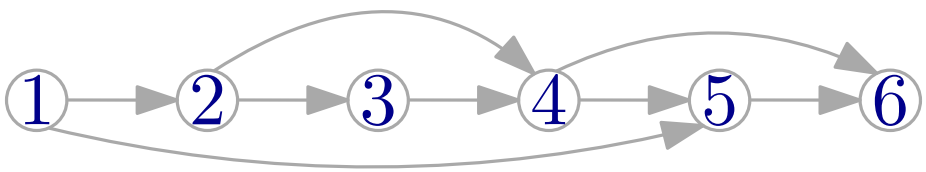
is $(2, 3)$ depth-robust

DAG $G = (V, E)$ is (e, d) **depth-robust** if after removing any e nodes a path of length d exists.

$\exists$ $(\Theta(N), \Theta(N))$ depth-robust graphs on N nodes with $O(\log(N))$ max-indegree [EGS75].

Two Basic Concepts

Depth-Robust Graphs

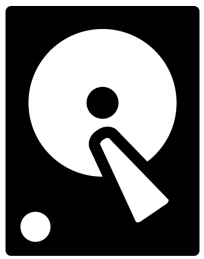


DAG $G = (V, E)$ is (e, d) **depth-robust** if after removing any e nodes a path of length d exists.

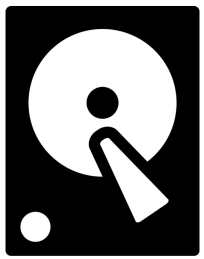
Graph **Labelling**

label $\ell_i = \mathcal{H}(\ell_{parents(i)})$, e.g. $\ell_4 = \mathcal{H}(\ell_3, \ell_4)$

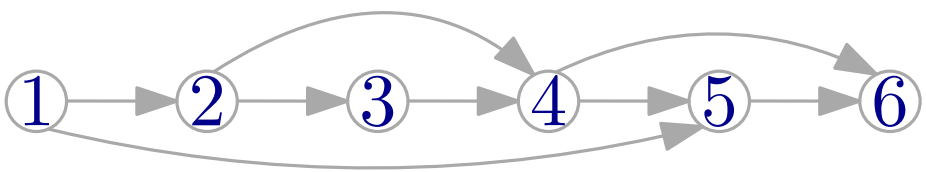
Pebbling Based Proofs of Space [FDPK'15]



Pebbling Based Proofs of Space [FDPK'15]



depth-robust DAG (on $\Theta(N)$ nodes)



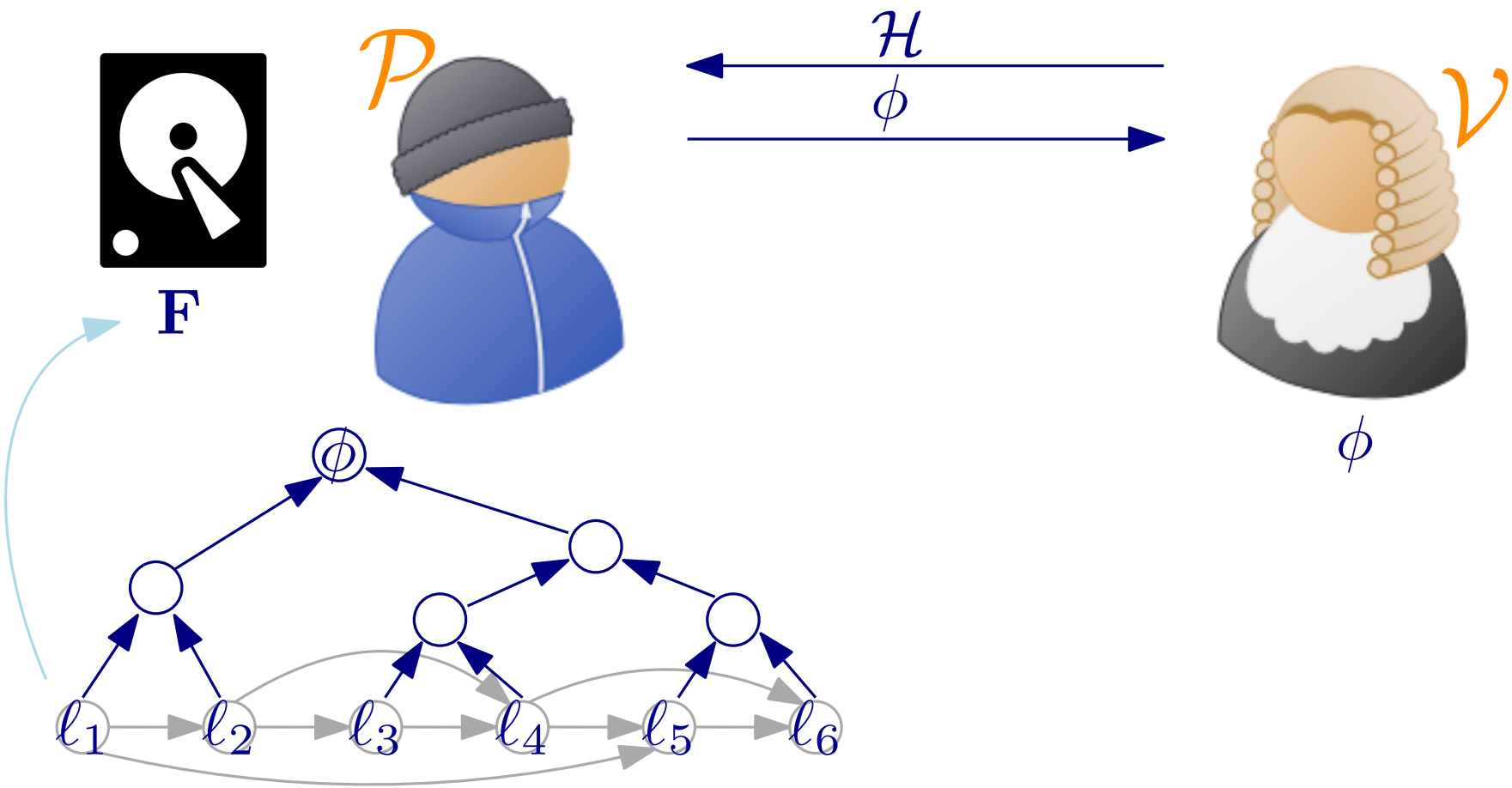
Pebbling Based Proofs of Space [FDPK'15]

initialization

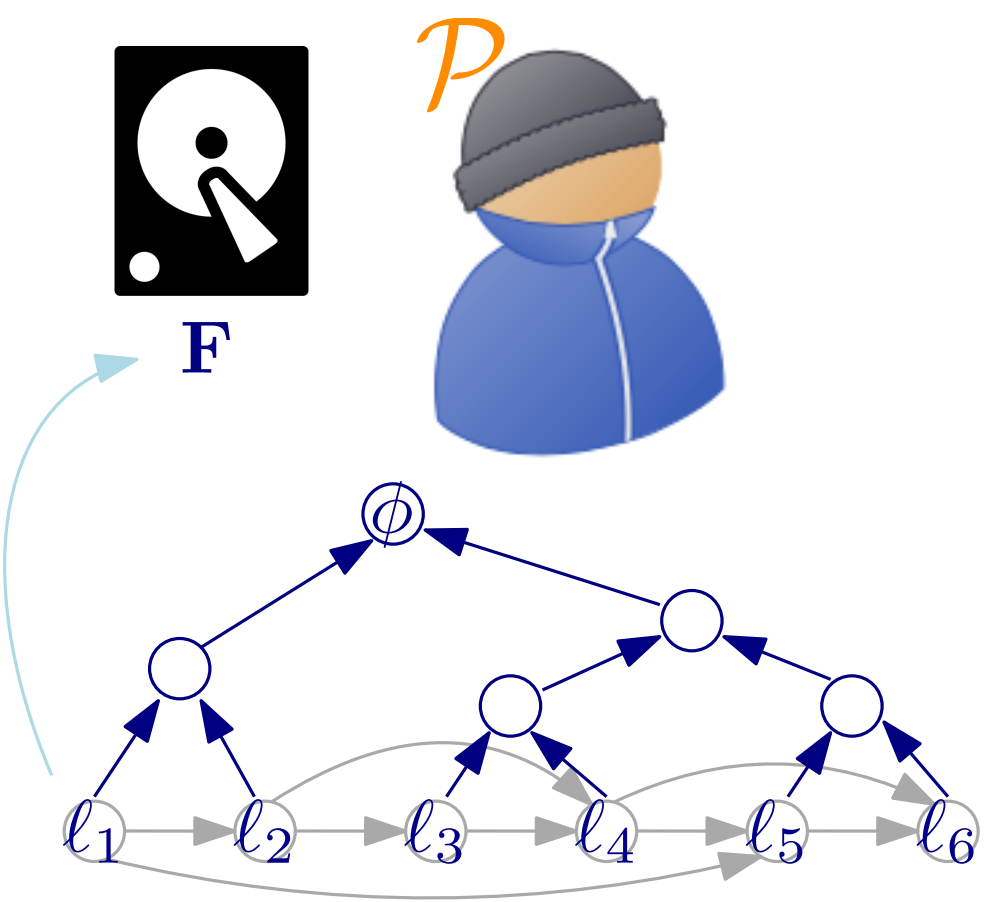
$\mathcal{P}$ computes labelling of DR graph.

Stores labels

Sends Merkle-tree commitment to labels to $\mathcal{V}$.



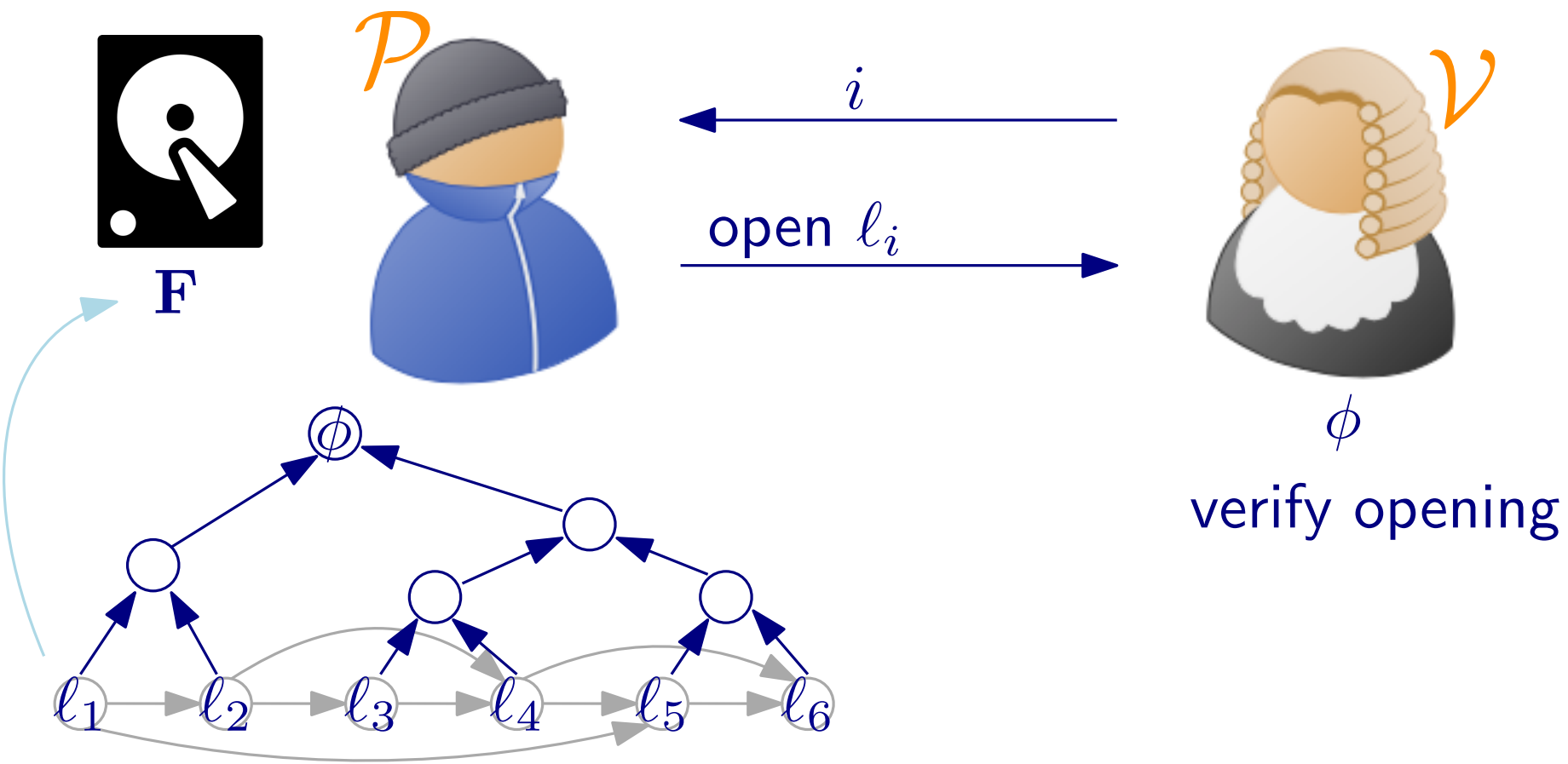
Pebbling Based Proofs of Space [FDPK'15]



Pebbling Based Proofs of Space [FDPK'15]

proof execution

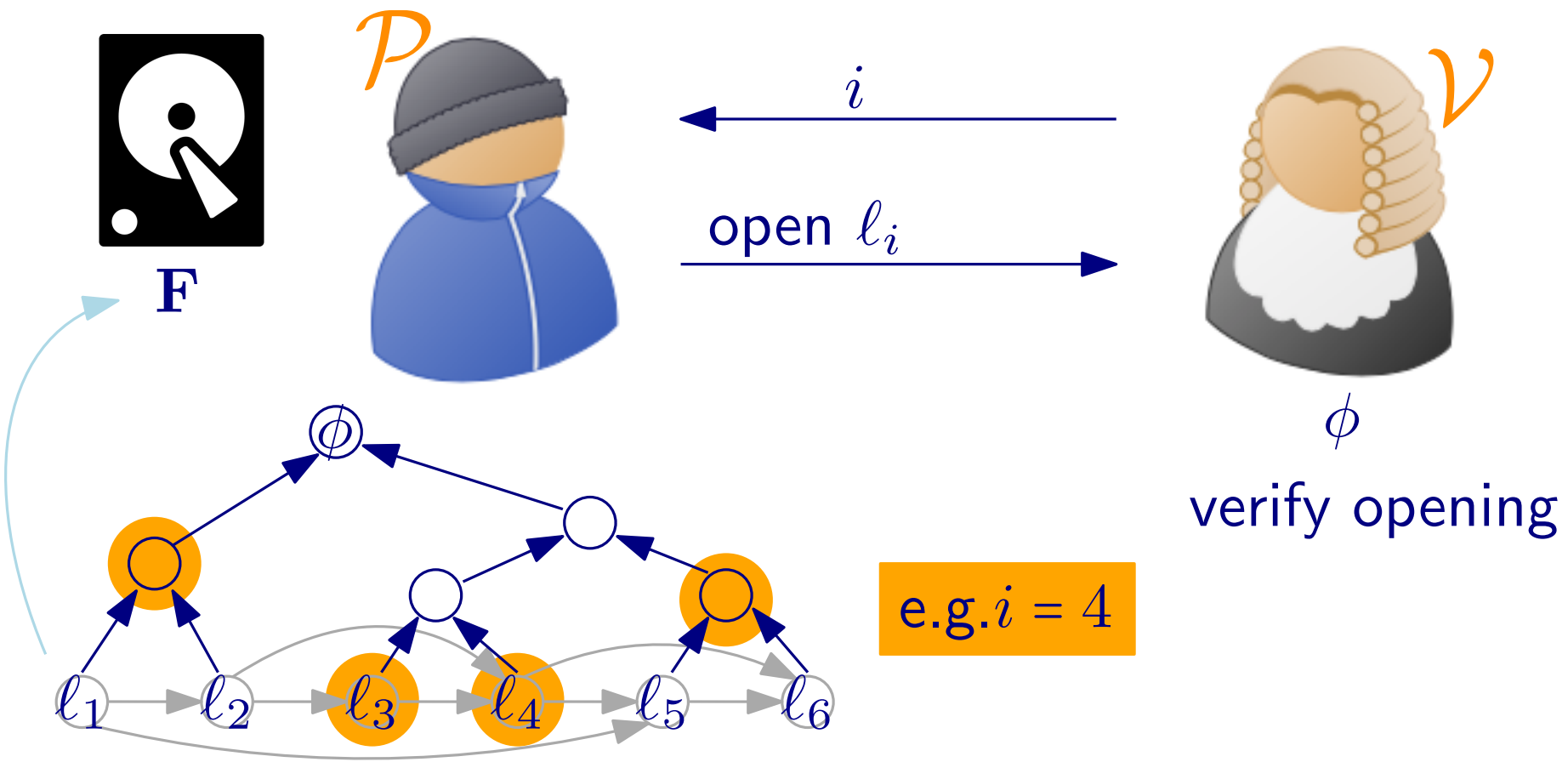
$\mathcal{V}$ hallenges $\mathcal{P}$ to open a few random labels.



Pebbling Based Proofs of Space [FDPK'15]

proof execution

$\mathcal{V}$ hallenges $\mathcal{P}$ to open a few random labels.



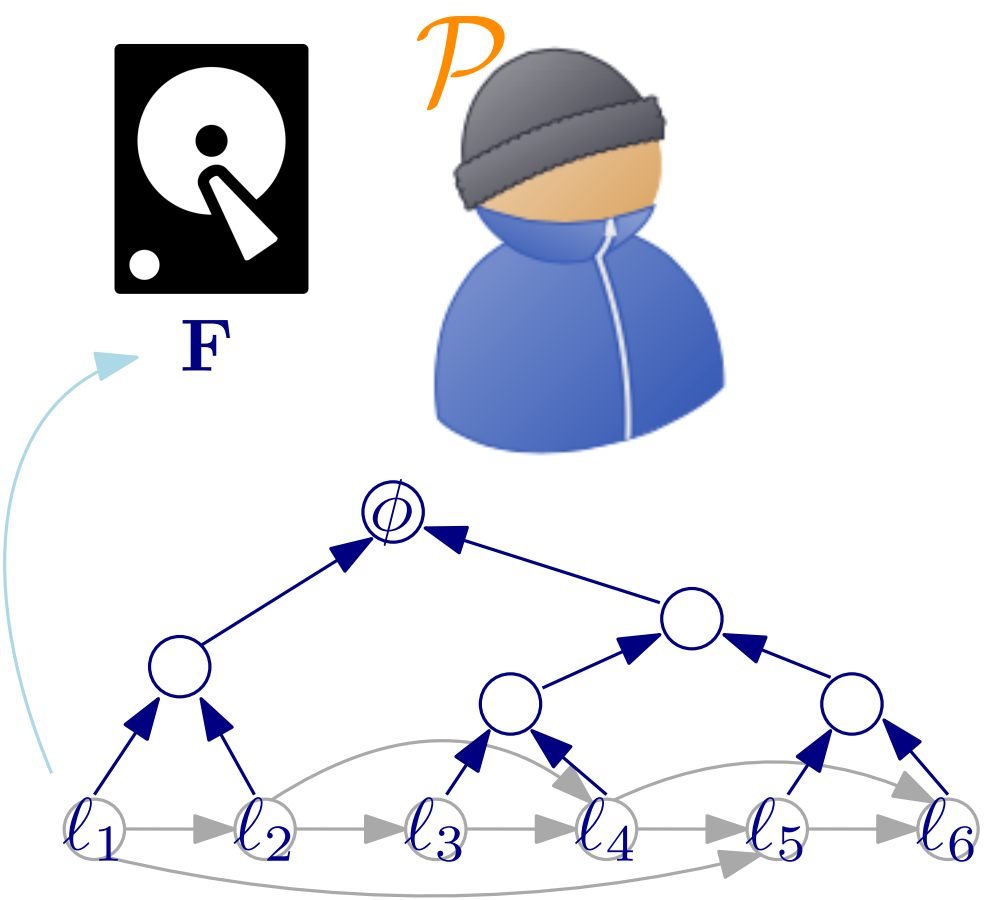
Pebbling Based Proofs of Space [FDPK'15]

security [FDPK'15]

$\tilde{\mathcal{P}}$ only stores $N(1-\epsilon)$ labels $\Rightarrow$

$\tilde{\mathcal{P}}$ needs to make $\Omega(N)$ $\mathcal{H}$ queries to make $\mathcal{V}$ accept

intuition: $\exists$ long path on lables that are not stored



Pebbling Based Proofs of Space [FDPK'15]

~~**security** [FDPK'15]~~

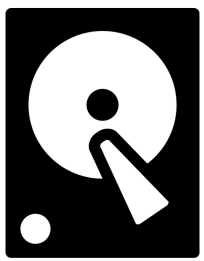
~~$\tilde{\mathcal{P}}$ only stores $N(1-\epsilon)$ labels $\Rightarrow$~~

security [Pie'19] security against general adversaries:

$\tilde{\mathcal{P}}$ stores any file of size $\leq N(1-\epsilon) \quad \Rightarrow$

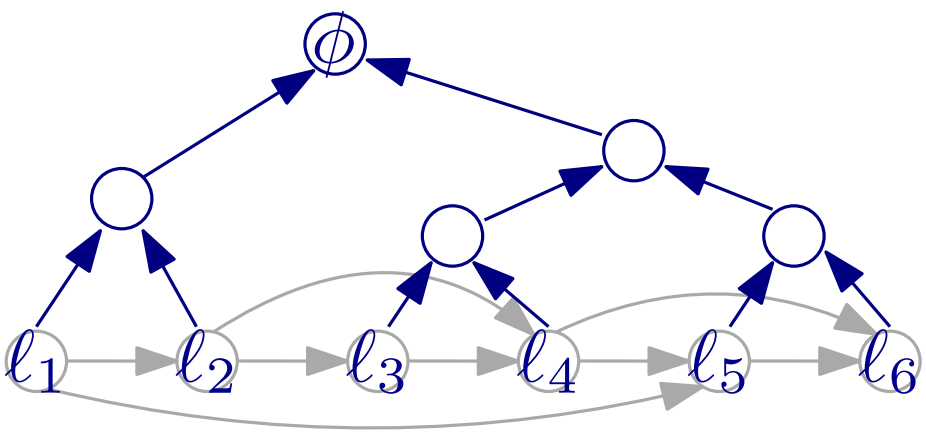
$\tilde{\mathcal{P}}$ needs to make $\Omega(N)$ $\mathcal{H}$ queries to make $\mathcal{V}$ accept

intuition: $\exists$ long path on lables that are not stored



F

ϕ



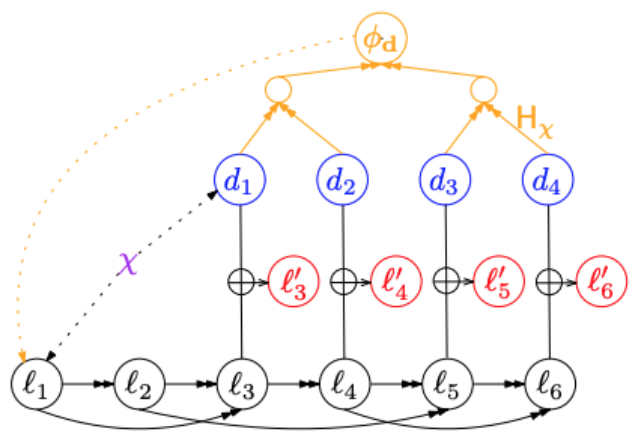
Proofs of "useful" Space

In a proof of space the dedicated space must be "wasted". In Proofs of catalytic space and Proofs of replication it can be used to store useful data.

EPoS_o: The **proof of space from [DFKP15]** instantiated with (a toy example of) a depth-robust graph.

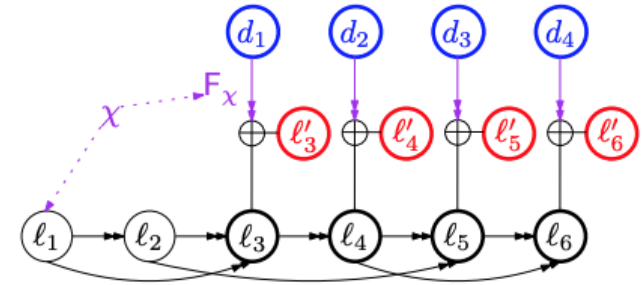
$\ell_i = \mathsf{H}_\chi(i, \ell_{p_1}, \ldots, \ell_{p_{\delta_i}}) \quad V_C = \{3,4,5,6\}$

EPoCS_ϕ: Our **proof of catalytic space** where the data is committed by a standard Merkle tree commitment $\phi_{\mathbf{d}}$. The hash function for this commitment depends on χ, the hash function for the labelling also on $\phi_{\mathbf{d}}$.



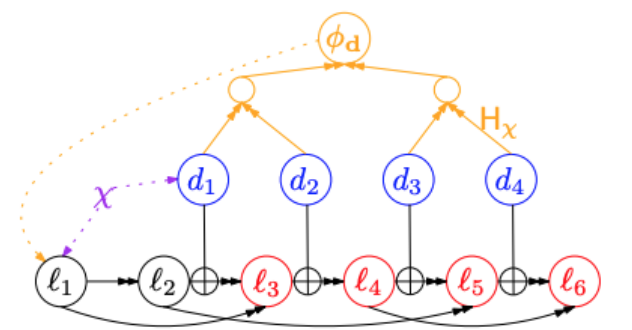
$\ell_i = \mathsf{H}_{\chi,\phi_{\mathbf{d}}}(i, \ell_{p_1}, \ldots, \ell_{p_{\delta_i}})$ for all $i \in V$
$\ell'_i = \ell_i \oplus d_{\tilde{i}}$ for $i \in V_C$

$\mathsf{EPoCS}_\mathsf{F}$: Our **efficiently updatable proof of catalytic space** where the catalytic data committed via random invertible function F.



$\ell_i = \mathsf{H}_\chi(i, \ell_{p_1}, \ldots, \ell_{p_{\delta_i}})$ for all $i \in V$
$\ell'_i = \ell_i \oplus \mathsf{F}_\chi(d_{\tilde{i}})$ for $i \in V_C$

EPoR: Our **proof of replication** is similar to EPoCS_ϕ, but the data is XOR'ed to the lables as the computation goes on, not just at the end.



$$\ell_i = \begin{cases} \mathsf{H}_{\chi,\phi_{\mathbf{d}}}(i, \ell_{p_1}, \ldots, \ell_{p_{\delta_i}}) & \text{if } i \in V \setminus V_C \\ \mathsf{H}_{\chi,,\phi_{\mathbf{d}}}(i, \ell_{p_1}, \ldots, \ell_{p_{\delta_i}}) \oplus d_{\tilde{i}} & \text{if } i \in V_C \end{cases}$$

Proofs of "useful" Space

In a proof of space the dedicated space must be "wasted".
In Proofs of catalytic space and Proofs of replication it can be used to store useful data.

$\mathsf{E}_{\mathsf{PoS}_o}$: The **proof of space from [DFKP15]** instantiated with (a toy example of) a depth-

$\mathsf{E}_{\mathsf{PoCS}_F}$: Our **efficiently updatable proof of catalytic space** where the catalytic data com-

Filecoin

$$\ell_i = \mathsf{H}_{\chi,\phi_{\mathbf{d}}}(i, \ell_{p_1}, \ldots, \ell_{p_{\delta_i}}) \text{ for all } i \in V$$
$$\ell'_i = \ell_i \oplus d_{\tilde{i}} \text{ for } i \in V_C$$

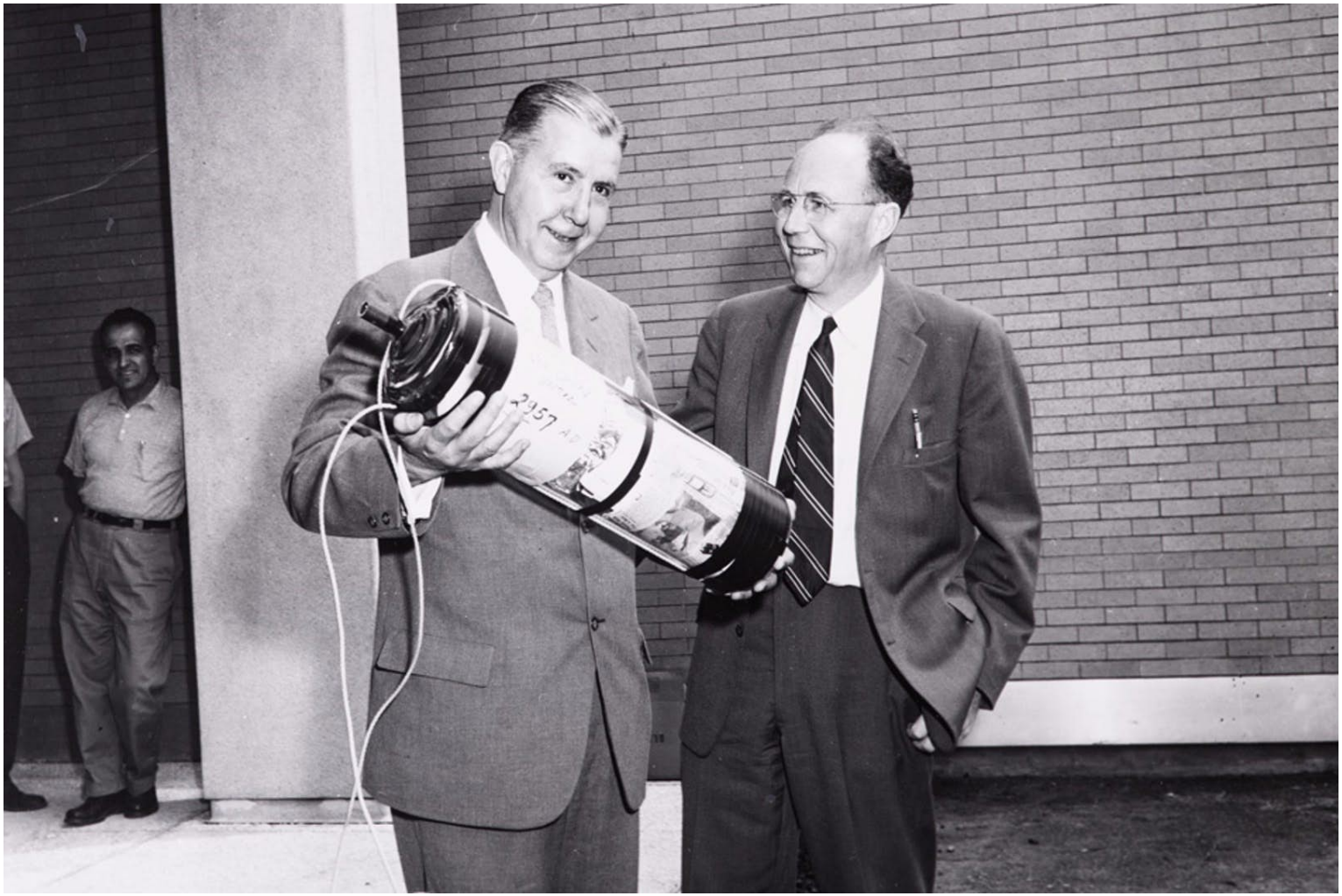
$$\ell_i = \begin{cases} \mathsf{H}_{\chi,\phi_{\mathbf{d}}}(i, \ell_{p_1}, \ldots, \ell_{p_{\delta_i}}) & \text{if } i \in V \setminus V_C \\ \mathsf{H}_{\chi,,\phi_{\mathbf{d}}}(i, \ell_{p_1}, \ldots, \ell_{p_{\delta_i}}) \oplus d_{\tilde{i}} & \text{if } i \in V_C \end{cases}$$

Second Ingredient

Verifiable Delay Functions

aka unique Proofs of Sequential Work

Time-Capsules



Time-Capsules

Time-lock puzzles and timed-release Crypto

Ronald L. Rivest*, Adi Shamir**, and David A. Wagner***

Revised March 10, 1996

Time-lock puzzles and timed-release Crypto

Ronald L. Rivest*, Adi Shamir**, and David A. Wagner***

Revised March 10, 1996

inherently sequential computation ~ computation time

RSW96 Time-Lock Puzzle

Definition

on input a time parameter T sample a puzzle π

Instantiation

$$\pi = (N = p \cdot q, x \in \mathbb{Z}_N^*, T \in \mathbb{Z})$$

RSW96 Time-Lock Puzzle

Definition

on input a time parameter T sample a puzzle π and the solution σ.

Instantiation

$$\pi = (N = p \cdot q, x \in \mathbb{Z}_N^*, T \in \mathbb{Z})$$

solution $\sigma = x^{2^T} \mod N$ can be computed with two exponentiation given p, q:

$$e \leftarrow 2^T \mod \phi(N) \quad , \quad x^{2^T} = x^e \bmod N$$

RSW96 Time-Lock Puzzle

Definition

on input a time parameter T sample a puzzle π and the solution σ.

(completeness) given π the solution σ can be computed in T sequential computational "steps"

(security) but not less, even given parallelism.

Instantiation

$$\pi = (N = p \cdot q, x \in \mathbb{Z}_N^*, T \in \mathbb{Z})$$

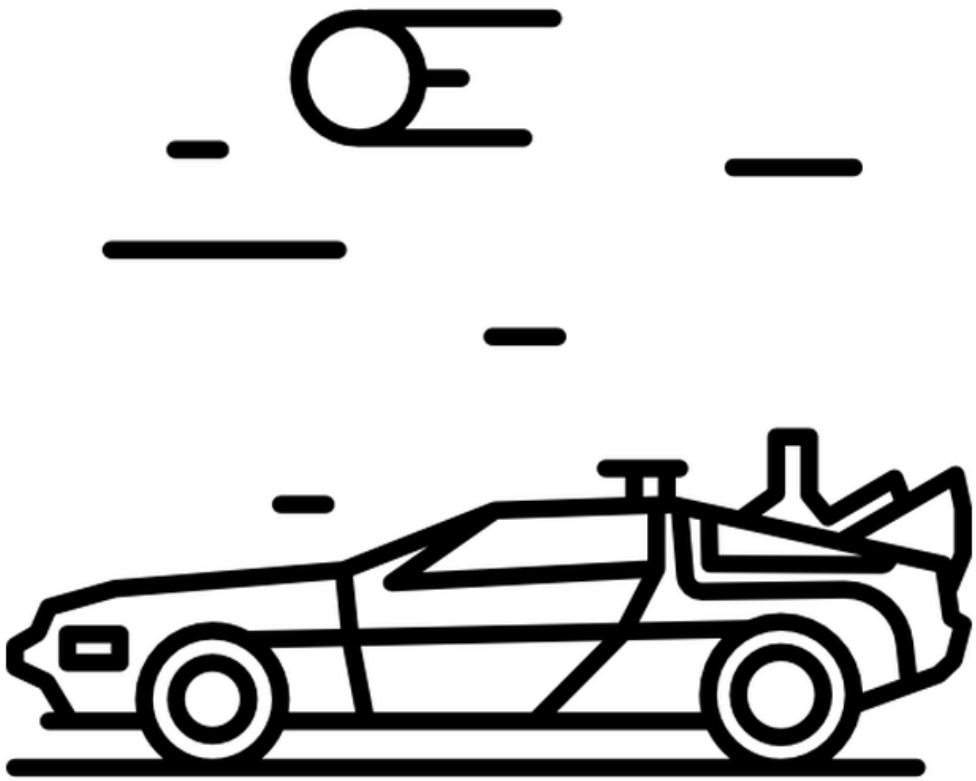
solution $\sigma = x^{2^T} \mod N$ can be computed with two exponentiation given p, q:

$$e \leftarrow 2^T \mod \phi(N) \quad , \quad x^{2^T} = x^e \bmod N$$

requires T sequential squarings given only N

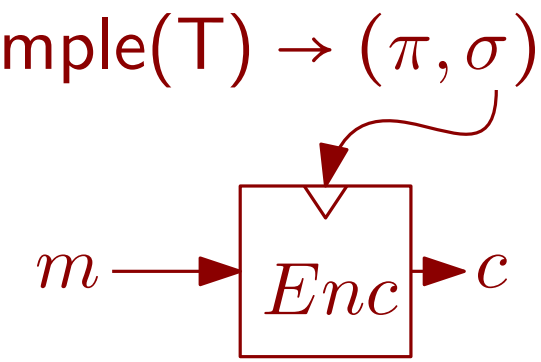
$$x \to x^2 \to x^{2^2} \to \ldots x^{2^T} \bmod N$$

Sending Messages to the Future



Sending Messages to the Future

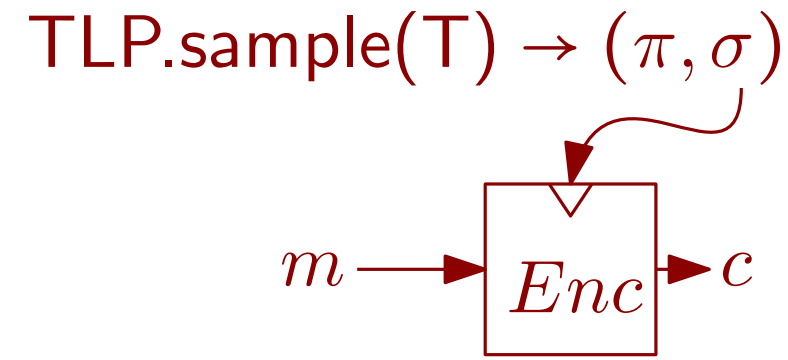
Compute puzzle/solution (π, σ) and ciphertext $c = Enc(\sigma, m)$

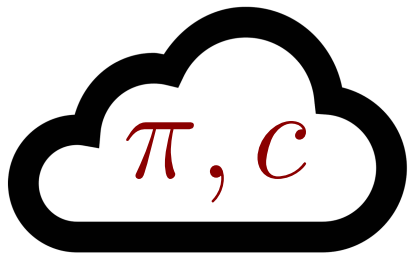


Sending Messages to the Future

Compute puzzle/solution (π, σ) and ciphertext $c = Enc(\sigma, m)$

Publish π, c



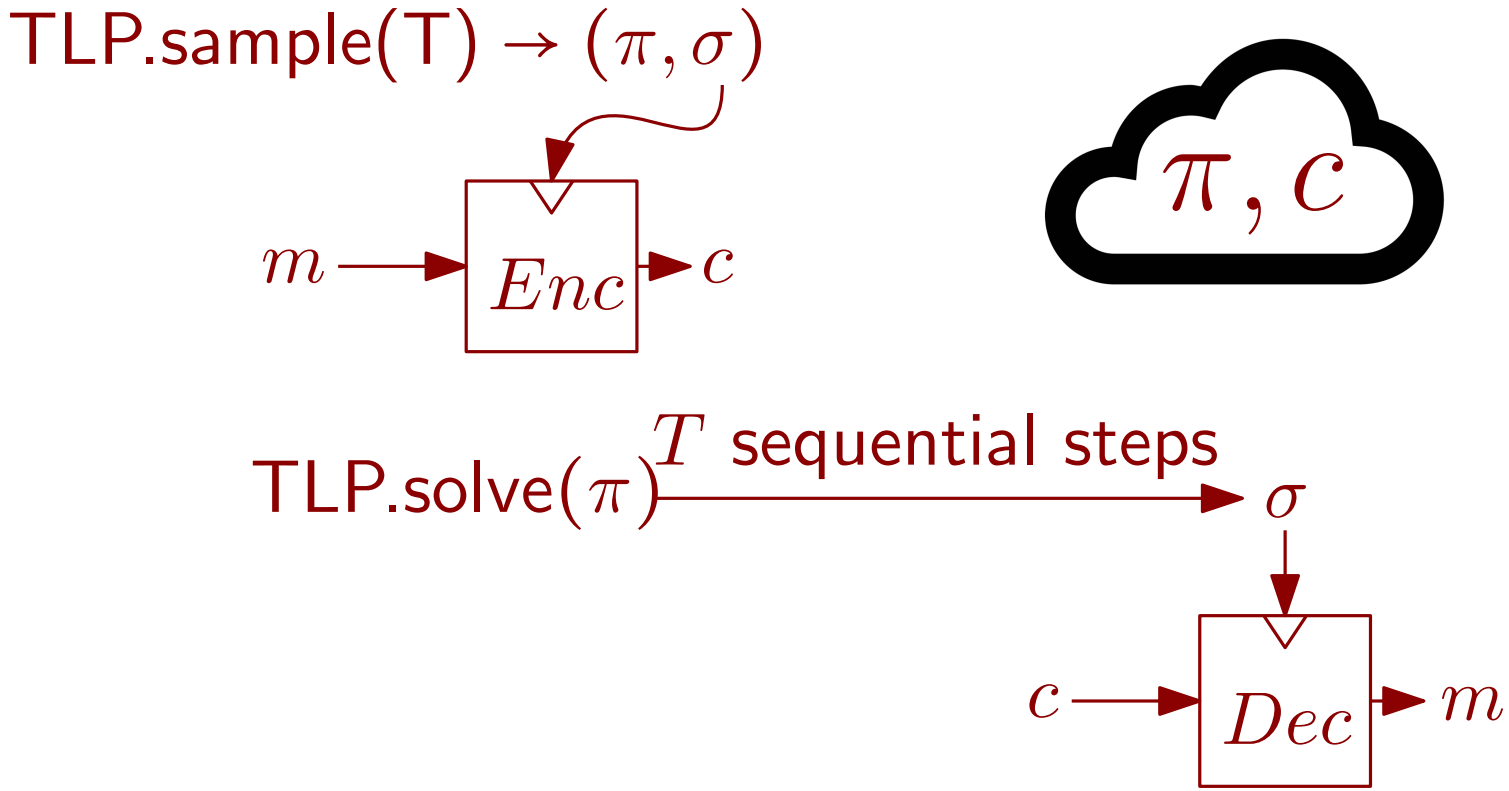


Sending Messages to the Future

Compute puzzle/solution (π, σ) and ciphertext $c = Enc(\sigma, m)$

Publish π, c

Anyone can decrypt after solving the puzzle

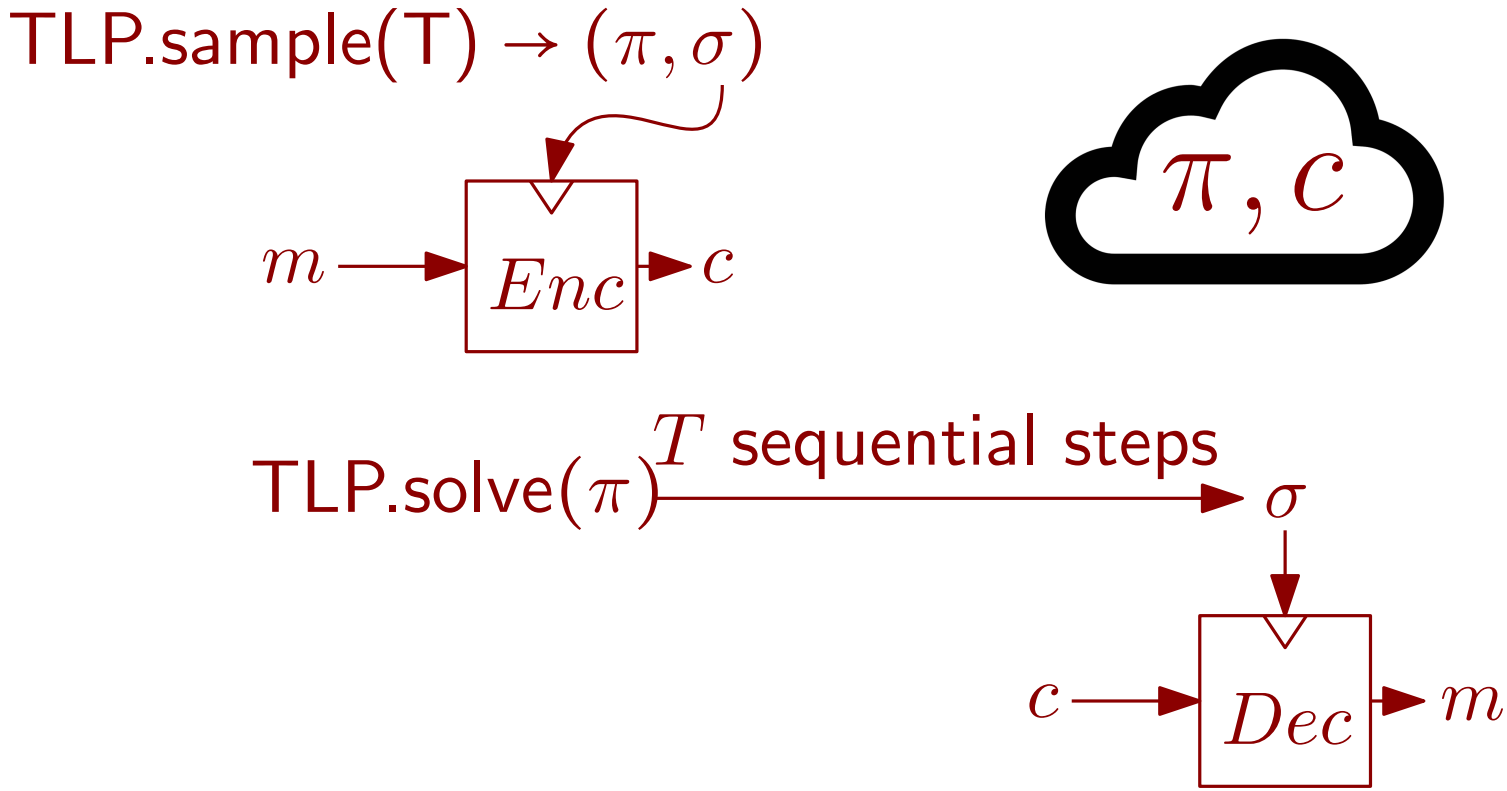


Sending Messages to the Future

Compute puzzle/solution (π, σ) and ciphertext $c = Enc(\sigma, m)$

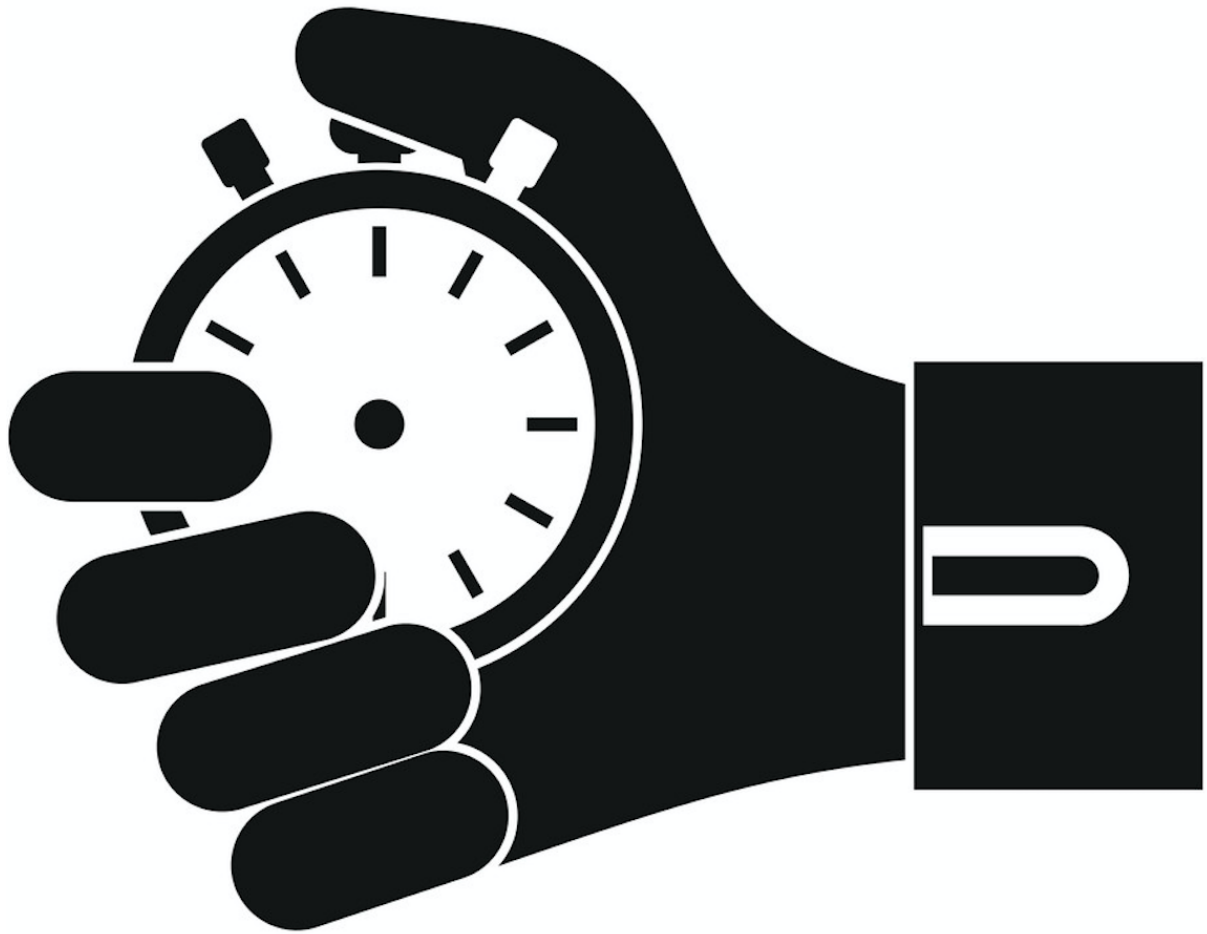
Publish π, c

Anyone can decrypt after solving the puzzle



Description of the LCS35 Time Capsule Crypto-Puzzle
by Ronald L. Rivest
April 4, 1999

Proofs of Sequential Work / Verifiable Delay Function



Proofs of Sequential Work / Verifiable Delay Function

Proof of Sequential Work

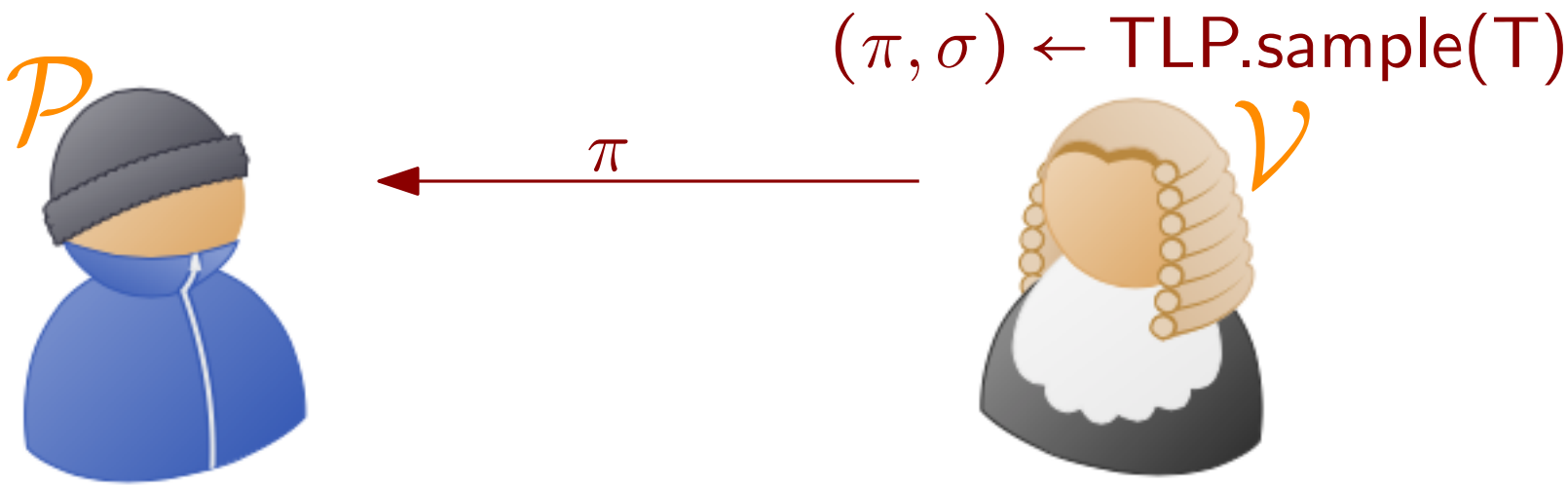
Proof system where prover $\mathcal{P}$ convinces verifier $\mathcal{V}$ it performed a sequential computation of T steps.

Proofs of Sequential Work / Verifiable Delay Function

Proof of Sequential Work

Proof system where prover $\mathcal{P}$ convinces verifier $\mathcal{V}$ it performed a sequential computation of T steps.

PoSW from a time-lock puzzle

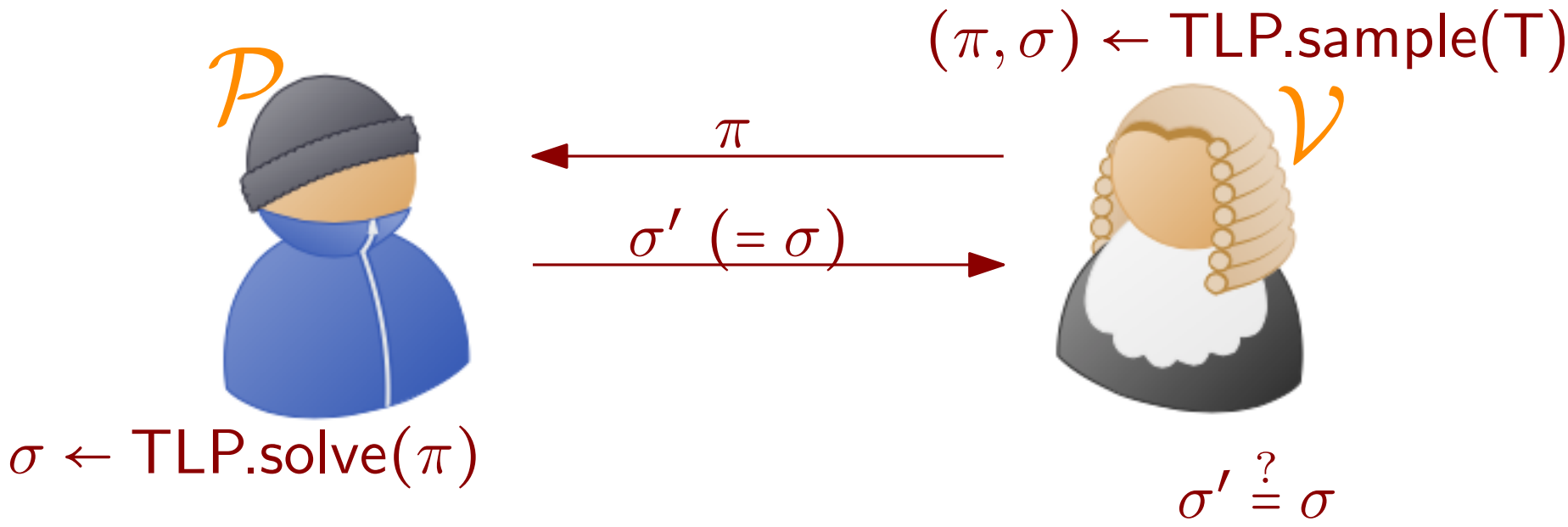


Proofs of Sequential Work / Verifiable Delay Function

Proof of Sequential Work

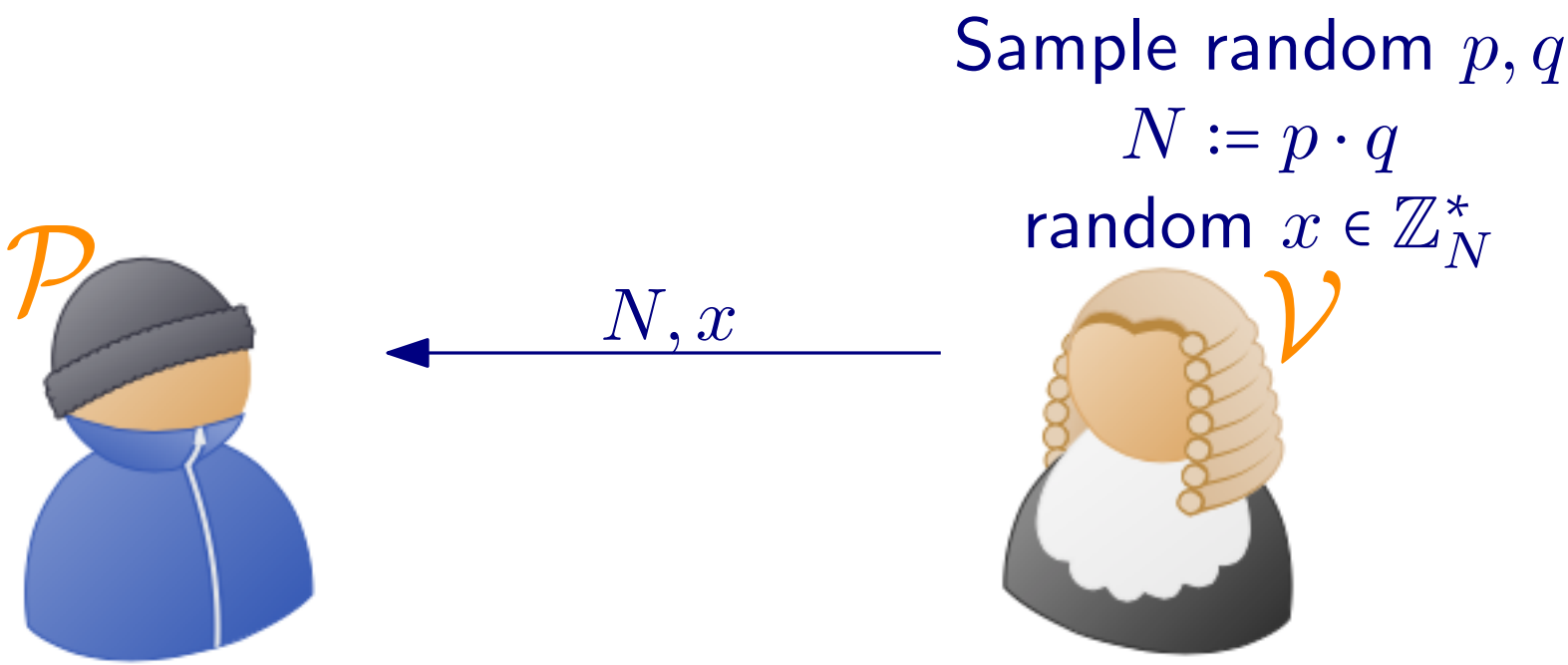
Proof system where prover $\mathcal{P}$ convinces verifier $\mathcal{V}$ it performed a sequential computation of T steps.

PoSW from a time-lock puzzle



Proofs of Sequential Work / Verifiable Delay Function

instantiated with the RSW96 puzzle



Proofs of Sequential Work / Verifiable Delay Function

instantiated with the RSW96 puzzle

Sample random p, q

$N := p \cdot q$

random $x \in \mathbb{Z}_N^*$

N, x

σ

computes $\sigma = x^{2^T} \bmod N$

in T sequential steps

$\sigma \stackrel{?}{=} x^{2^T} \bmod N$

Proofs of Sequential Work / Verifiable Delay Function

SECRET COIN : p, q required for verification, but must be secret otherwise puzzle does not need T sequential work.

instantiated with the RSW96 puzzle

Sample random p, q

$N := p \cdot q$

random $x \in \mathbb{Z}_N^*$

N, x

σ

computes $\sigma = x^{2^T} \bmod N$

in T sequential steps

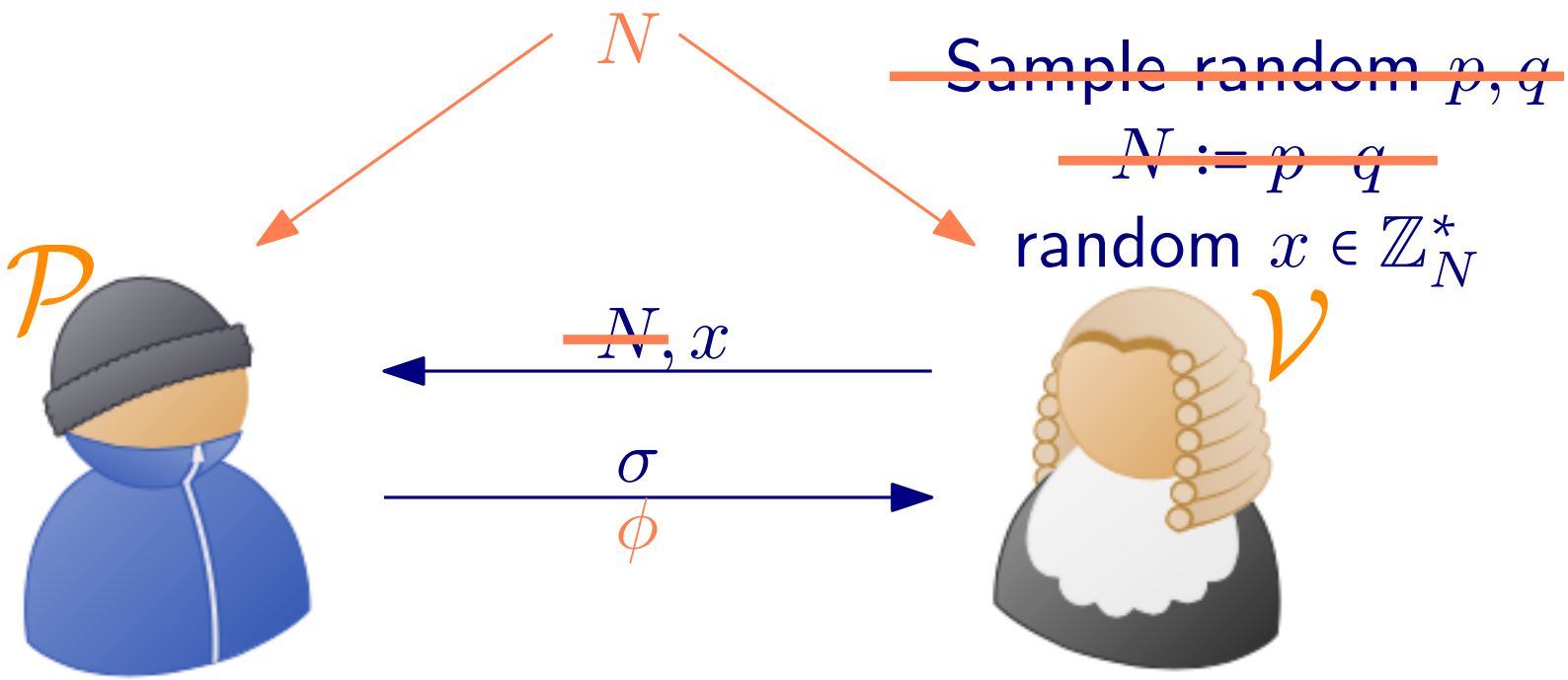
$\sigma \stackrel{?}{=} x^{2^T} \bmod N$

Proofs of Sequential Work / Verifiable Delay Function

SECRET COIN : p, q required for verification, but must be secret otherwise puzzle does not need T sequential work.

This Work : A publicly verifiable version (i.e., a "verifiable delay function") of the RSW96 time lock puzzle.

instantiated with the RSW96 puzzle



computes $\sigma = x^{2^T} \bmod N$

in T sequential steps

and proof ϕ certifying $\sigma = x^{2^T}$

$\sigma \stackrel{?}{=} x^{2^T} \bmod N$

verify$(x, \sigma, \phi) \in \{0, 1\}$

Proofs of Sequential Work / Verifiable Delay Function

SECRET COIN : p, q required for verification, but must be secret otherwise puzzle does not need T sequential work.

This Work : A publicly verifiable version (i.e., a "verifiable delay function") of the RSW96 time lock puzzle.

instantiated with the RSW96 puzzle

NOBODY knows factorization of N (group order of $\mathbb{Z}_N^*$)

N

~~Sample random p, q~~

~~$N := p \cdot q$~~

random $x \in \mathbb{Z}_N^*$

$\mathcal{P}$

~~N, x~~

σ

ϕ

$\mathcal{V}$

computes $\sigma = x^{2^T} \bmod N$

in T sequential steps

and proof ϕ certifying $\sigma = x^{2^T}$

~~$\sigma \stackrel{?}{=} x^{2^T} \bmod N$~~

verify$(x, \sigma, \phi) \in \{0, 1\}$

Proofs of Sequential Work / Verifiable Delay Function

SECRET COIN : p, q required for verification, but must be secret otherwise puzzle does not need T sequential work.

This Work : A publicly verifiable version (i.e., a "verifiable delay function") of the RSW96 time lock puzzle.

instantiated with the RSW96 puzzle

NOBODY knows factorization of N (group order of $\mathbb{Z}_N^*$)

N

~~Sample random p, q~~

~~$N := p \cdot q$~~

random $x \in \mathbb{Z}_N^*$

$\mathcal{P}$

computing ϕ must be cheap compared to computing σ

~~N, x~~

σ

ϕ

$\mathcal{V}$

computes $\sigma = x^{2^T} \bmod N$

in T sequential steps

and proof ϕ certifying $\sigma = x^{2^T}$

$\sigma \stackrel{?}{=} x^{2^T} \bmod N$

verify$(x, \sigma, \phi) \in \{0, 1\}$

History of Time Release Crypto

Time-lock puzzles and timed-release Crypto

Ronald L. Rivest*, Adi Shamir**, and David A. Wagner***

Revised March 10, 1996

History of Time Release Crypto

Time-Lock Puzzles in the Random Oracle Model

[Crypto'11]

Mohammad Mahmoody[1], Tal Moran[2], and Salil Vadhan[⋆2]

☹ No Time-Lock Puzzles from Random Oracles

History of Time Release Crypto

Time-Lock Puzzles in the Random Oracle Model

[Crypto'11]

Mohammad Mahmoody[1], Tal Moran[2], and Salil Vadhan[⋆2]

☹ No Time-Lock Puzzles from Random Oracles

Publicly Verifiable Proofs of Sequential Work

[ITCS'13]

Mohammad Mahmoody* Tal Moran† Salil Vadhan‡

February 18, 2013

☺ Introduce Proofs of Sequential Work and constructs them from Random Oracles.

☹ Not practical as prover need not only T sequential steps, but also T space.

☹ Not unique (finding many proofs at same cost as finding one). Uniqueness required for some applications (blockchains, randomness beacons), but not for "non-interactive time-stamps".

History of Time Release Crypto

Time-Lock Puzzles in the Random Oracle Model
[Crypto'11]

Mohammad Mahmoody[1], Tal Moran[2], and Salil Vadhan[⋆2]

☹ No Time-Lock Puzzles from Random Oracles

Publicly Verifiable Proofs of Sequential Work

Mohammad Mahmoody* Tal Moran† Salil Vadhan‡

February 18, 2013

Simple Proofs of Sequential Work [Eurocrypt'17]

Bram Cohen[1] and Krzysztof Pietrzak[2⋆]

[1] Chia Network, bram@chia.network
[2] IST Austria, pietrzak@ist.ac.at

☺ Simple construction where prover just needs $\log(T)$ space.
☹ Still not unique....

History of Time Release Crypto

Verifiable Delay Functions
[Crypto'18]

Dan Boneh[1], Joseph Bonneau[2], Benedikt Bünz[1], and Ben Fisch[1]

[1]Stanford University
[2]New York University

☺ VDF (morally a **unique** proof of sequential work): on input (x, T) compute (y, π) where $y = f(x)$ needs T sequential steps and π proof for $y = f(x)$.

☹ Use incrementally verifiable computation (Valiant'08).

History of Time Release Crypto

Verifiable Delay Functions
[Crypto'18]

Dan Boneh[1], Joseph Bonneau[2], Benedikt Bünz[1], and Ben Fisch[1]

[1]Stanford University
[2]New York University

☺ VDF (morally a **unique** proof of sequential work): on input (x, T) compute (y, π) where $y = f(x)$ needs T sequential steps and π proof for $y = f(x)$.

☹ Use incrementally verifiable computation (Valiant'08).

Efficient verifiable delay functions

Benjamin Wesolowski

École Polytechnique Fédérale de Lausanne, EPFL IC LACAL, Switzerland

Simple Verifiable Delay Functions
[ITCS'19]

Krzysztof Pietrzak*
IST Austria
pietrzak@ist.ac.at

July 1, 2018

☺ simple/efficient VDFs based on the RSW time-lock puzzle

History of Time Release Crypto

Verifiable Delay Functions
[Crypto'18]

Dan Boneh[1], Joseph Bonneau[2], Benedikt Bünz[1], and Ben Fisch[1]

[1]Stanford University
[2]New York University

☺ VDF (morally a **unique** proof of sequential work): on input (x, T) compute (y, π) where $y = f(x)$ needs T sequential steps and π proof for $y = f(x)$.

☹ Use incrementally verifiable computation (Valiant'08).

Efficient verifiable delay functions

Benjamin Wesolowski

École Polytechnique Fédérale de Lausanne, EPFL IC LACAL, Switzerland

Simple Verifiable Delay Functions
[ITCS'19]

Krzysztof Pietrzak*
IST Austria
pietrzak@ist.ac.at

July 1, 2018

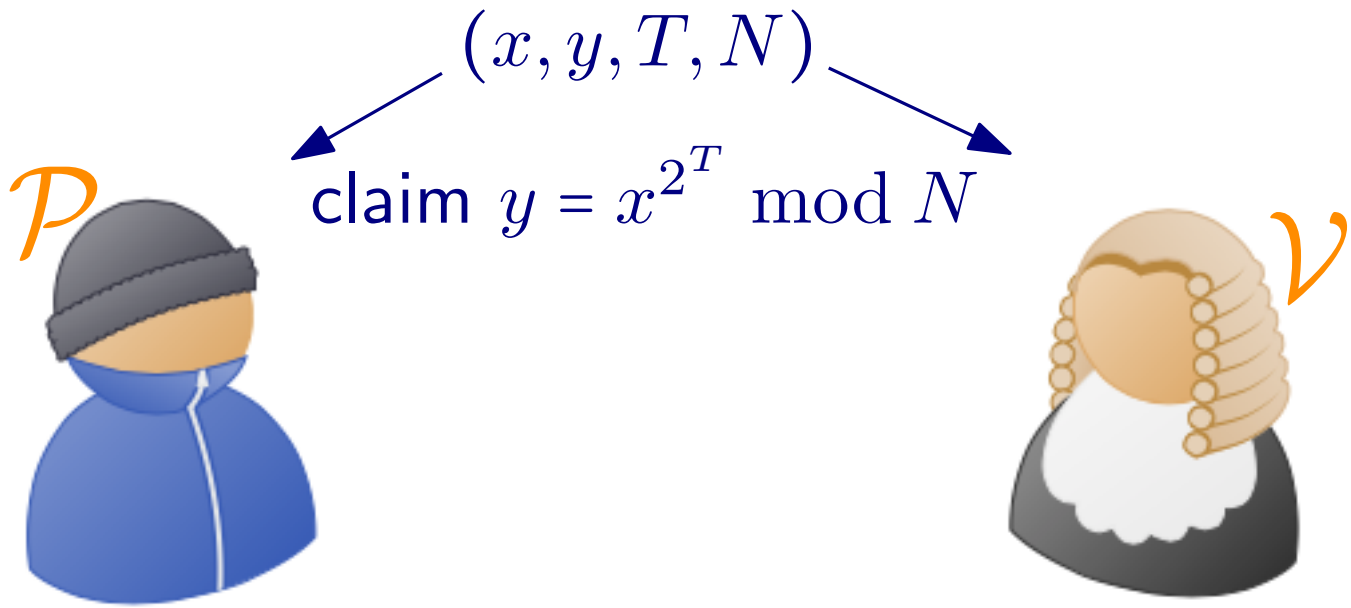
☺ simple/efficient VDFs based on the RSW time-lock puzzle

A Survey of Two Verifiable Delay Functions

Dan Boneh Benedikt Bünz Ben Fisch

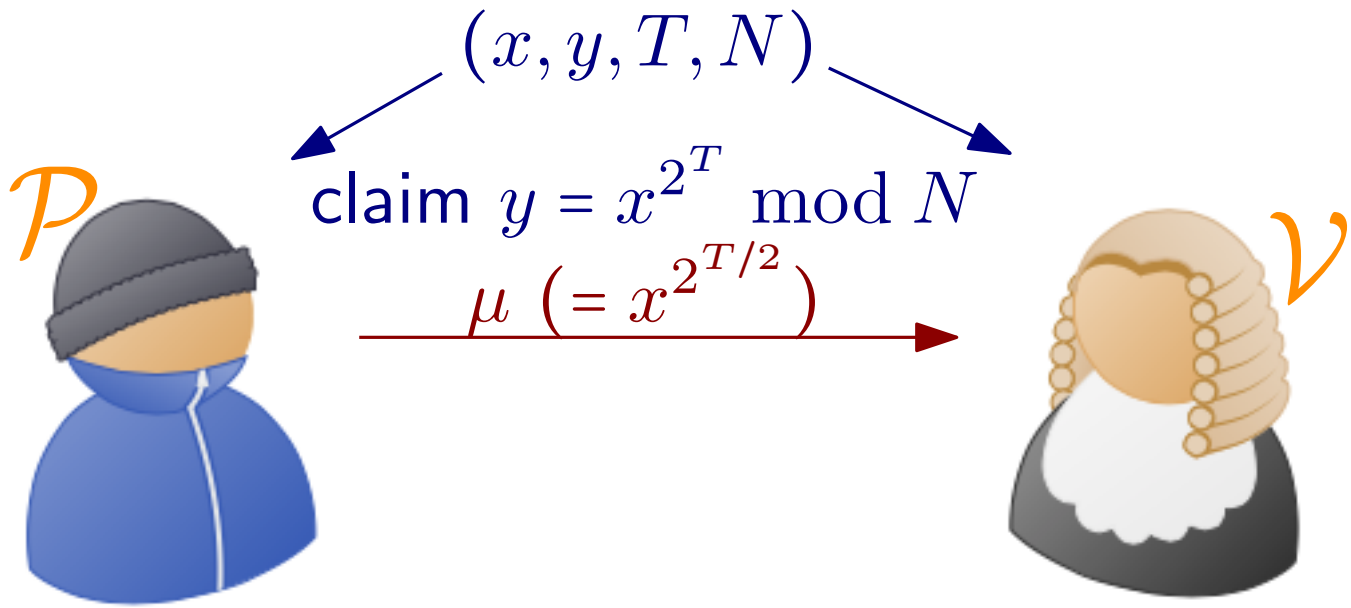
August 27, 2018

Proving $\sigma = x^{2^T}$ in Groups of Unknown Order



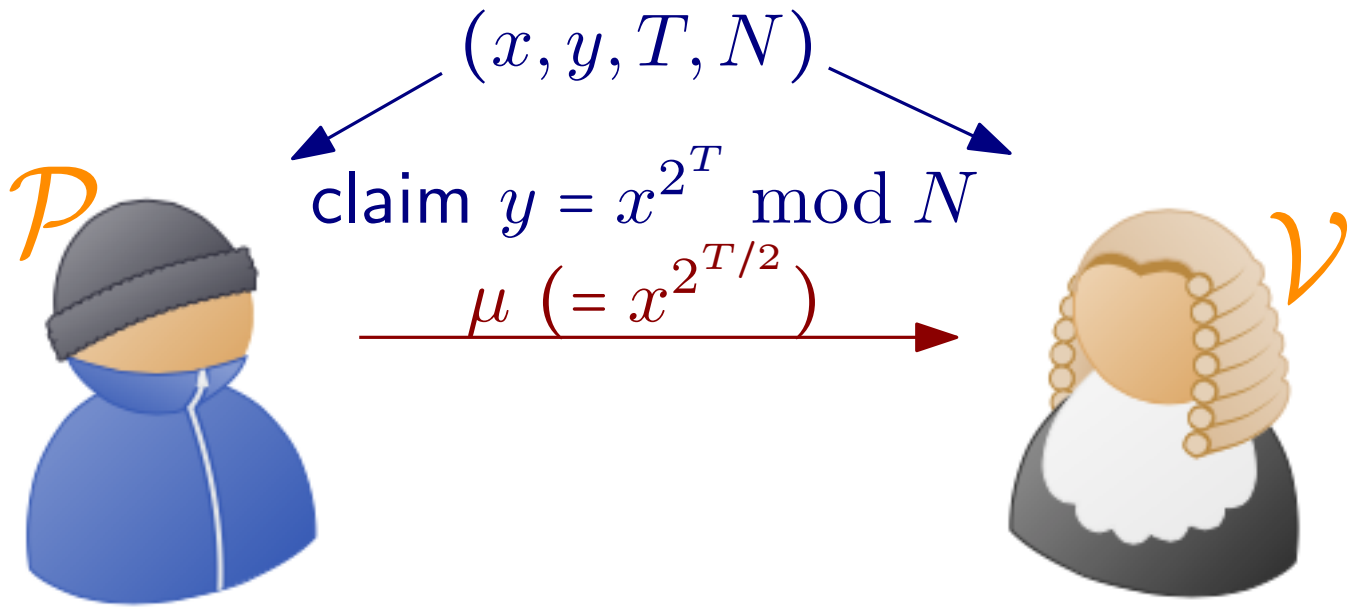
$x \; x^2 \; x^{2^2} \; x^{2^3} \; \ldots$ $\ldots \; x^{2^{T-1}} \; x^{2^T}$

Proving $\sigma = x^{2^T}$ in Groups of Unknown Order



$x\ x^2\ x^{2^2}\ x^{2^3}\ \ldots \qquad \mu\ (= x^{2^{T/2}}) \qquad \ldots\ x^{2^{T-1}}\ x^{2^T}$

Proving $\sigma = x^{2^T}$ in Groups of Unknown Order

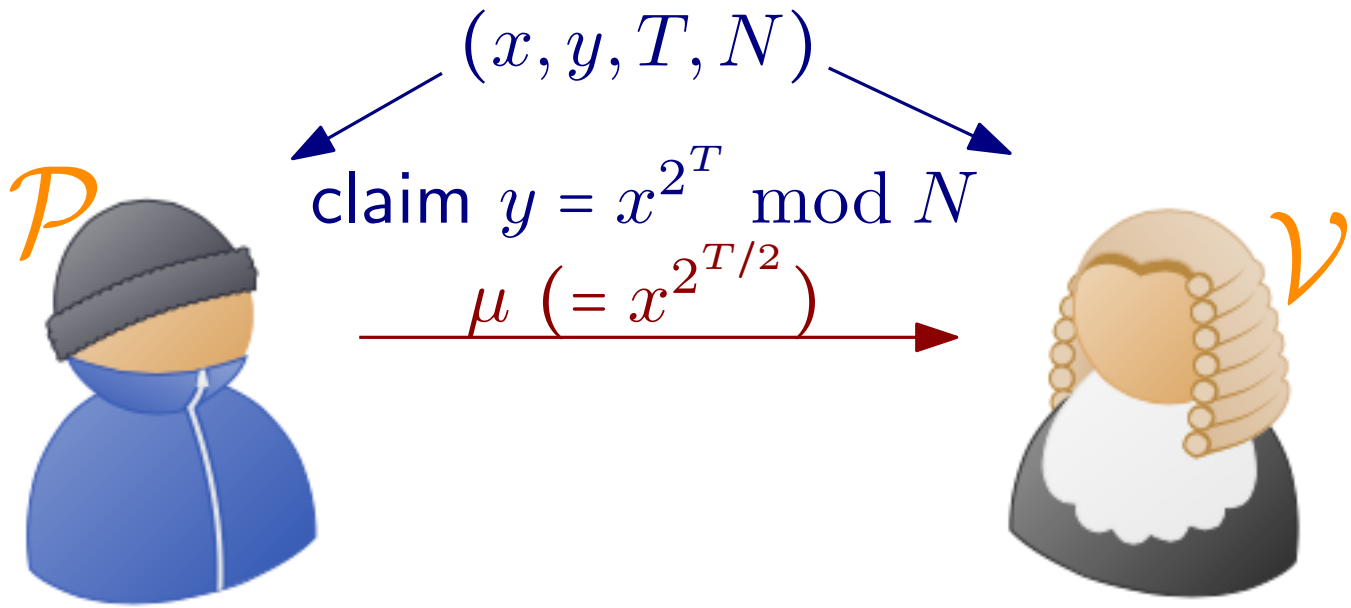


$y = x^{2^T}$ ← $\mu = x^{2^{T/2}} \wedge y = \mu^{2^{T/2}}$

2 claims for $T/2$ for 1 claim for T

$x\ x^2\ x^{2^2}\ x^{2^3}\ \ldots$ $\mu\ (= x^{2^{T/2}})$ $\ldots\ x^{2^{T-1}}\ x^{2^T}$

Proving $\sigma = x^{2^T}$ in Groups of Unknown Order



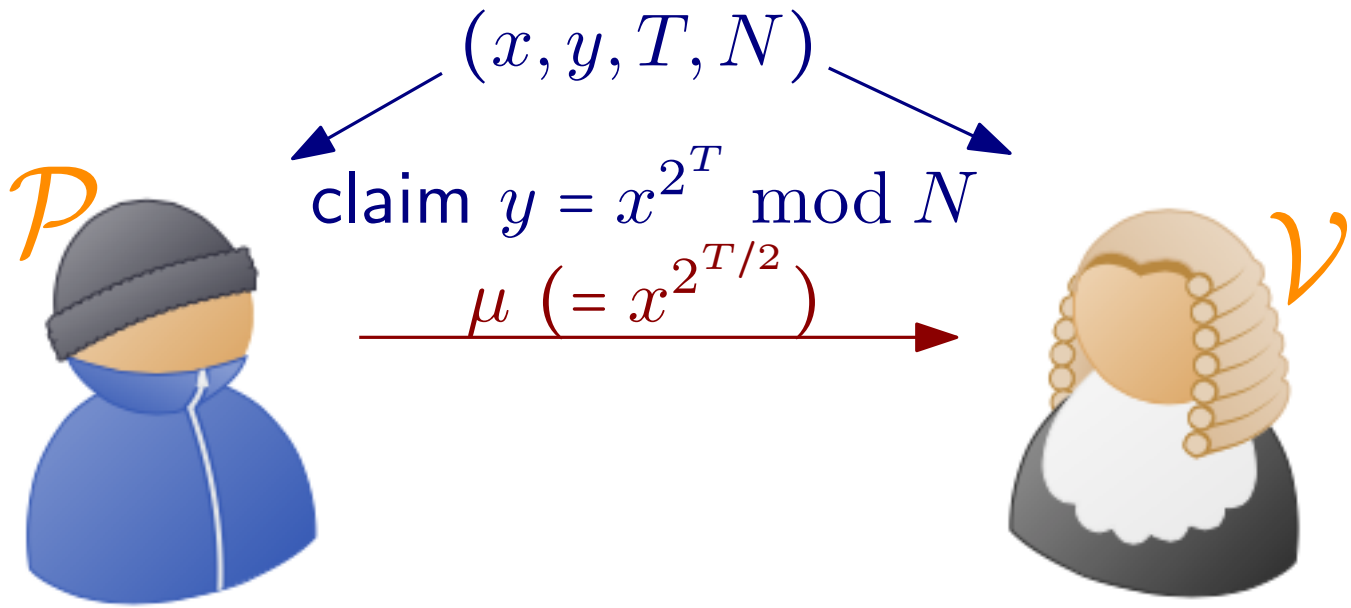
$$y = x^{2^T} \quad \longleftarrow \quad \boxed{\mu = x^{2^{T/2}} \ \wedge \ y = \mu^{2^{T/2}}}$$

$$\mu^r \cdot y = (x^r \cdot \mu)^{2^{T/2}} \quad \longleftarrow \quad \boxed{\mu = x^{2^{T/2}} \ \wedge \ y = \mu^{2^{T/2}}}$$

2 claims for $T/2$ for
1 claim for T

$$x \ x^2 \ x^{2^2} \ x^{2^3} \ \ldots \qquad \mu\ (= x^{2^{T/2}}) \qquad \ldots \ x^{2^{T-1}} \ x^{2^T}$$

Proving $\sigma = x^{2^T}$ in Groups of Unknown Order



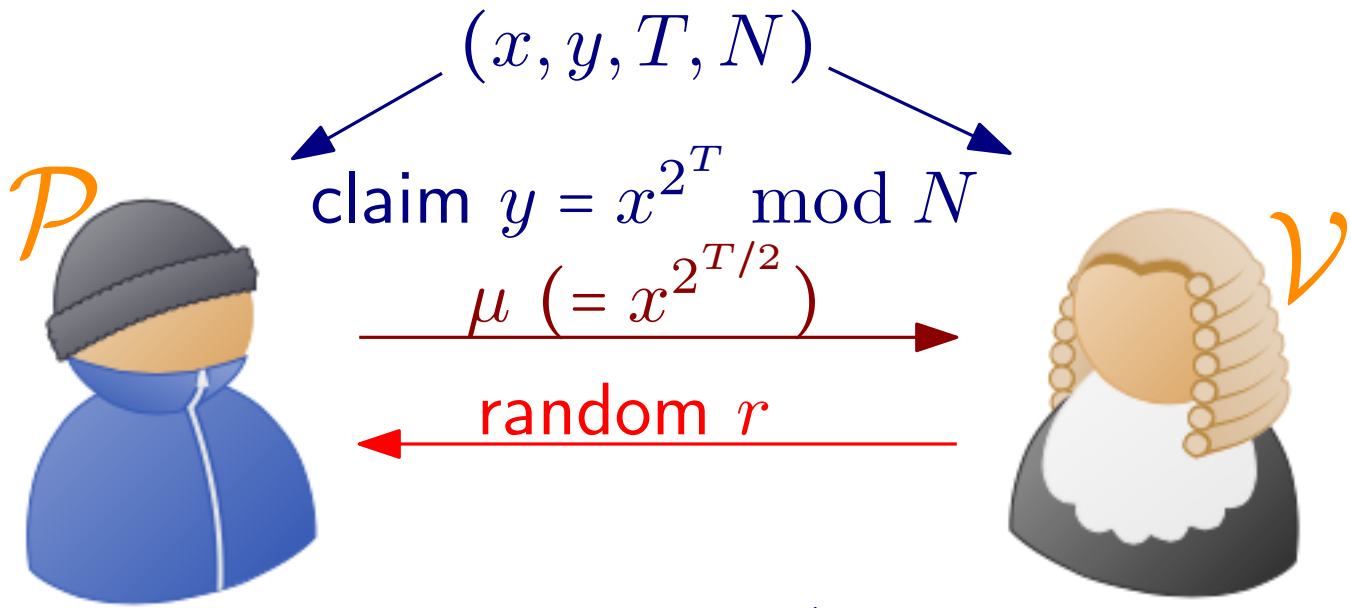
$$y = x^{2^T} \quad \longleftarrow \quad \boxed{\mu = x^{2^{T/2}} \ \wedge \ y = \mu^{2^{T/2}}}$$

$$\mu^r \cdot y = (x^r \cdot \mu)^{2^{T/2}} \quad \not\longleftarrow$$

for almost all r $\mu^r \cdot y \neq (x^r \cdot \mu)^{2^{T/2}} \quad \longleftarrow \quad \boxed{\mu \neq x^{2^{T/2}} \ \vee \ y \neq \mu^{2^{T/2}}}$

$$x \ x^2 \ x^{2^2} \ x^{2^3} \ \ldots \qquad \mu\ (= x^{2^{T/2}}) \qquad \ldots \ x^{2^{T-1}} \ x^{2^T}$$

Proving $\sigma = x^{2^T}$ in Groups of Unknown Order



new claim $y' = x'^{2^{T/2}} \bmod N$

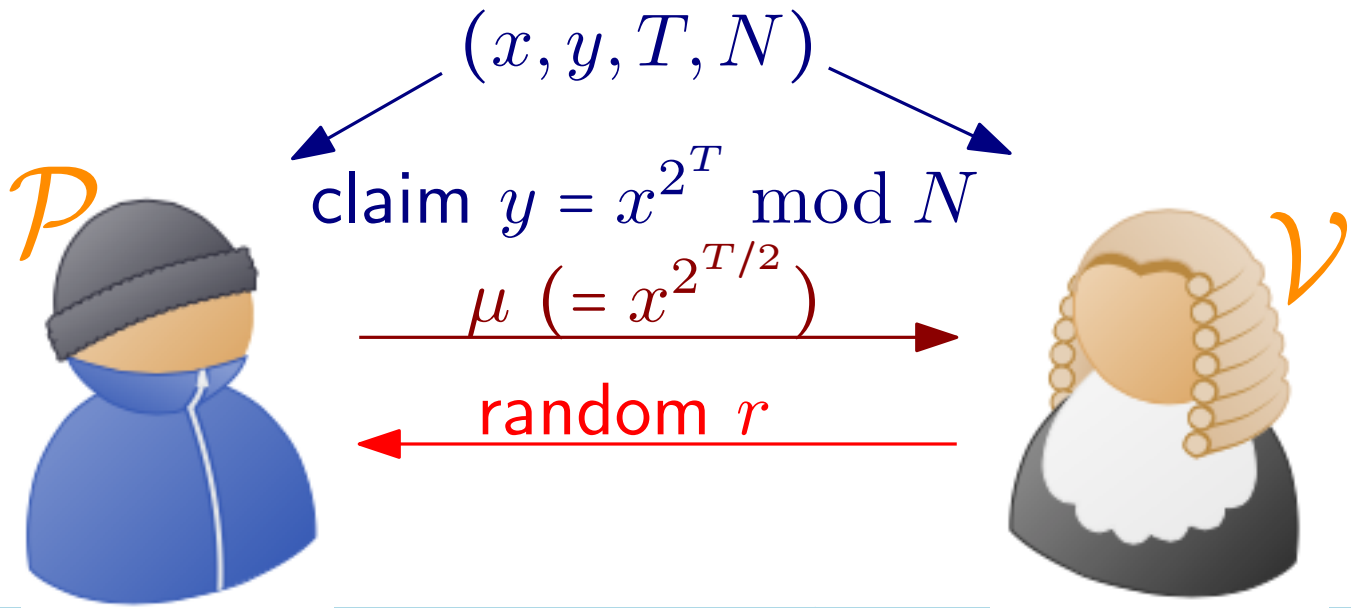
where

$$x' := \mu^r \cdot y$$

$$y' := (x^r \cdot \mu)^{2^{T/2}}$$

$x\ x^2\ x^{2^2}\ x^{2^3} \ldots \qquad \mu\ (= x^{2^{T/2}}) \qquad \ldots\ x^{2^{T-1}}\ x^{2^T}$

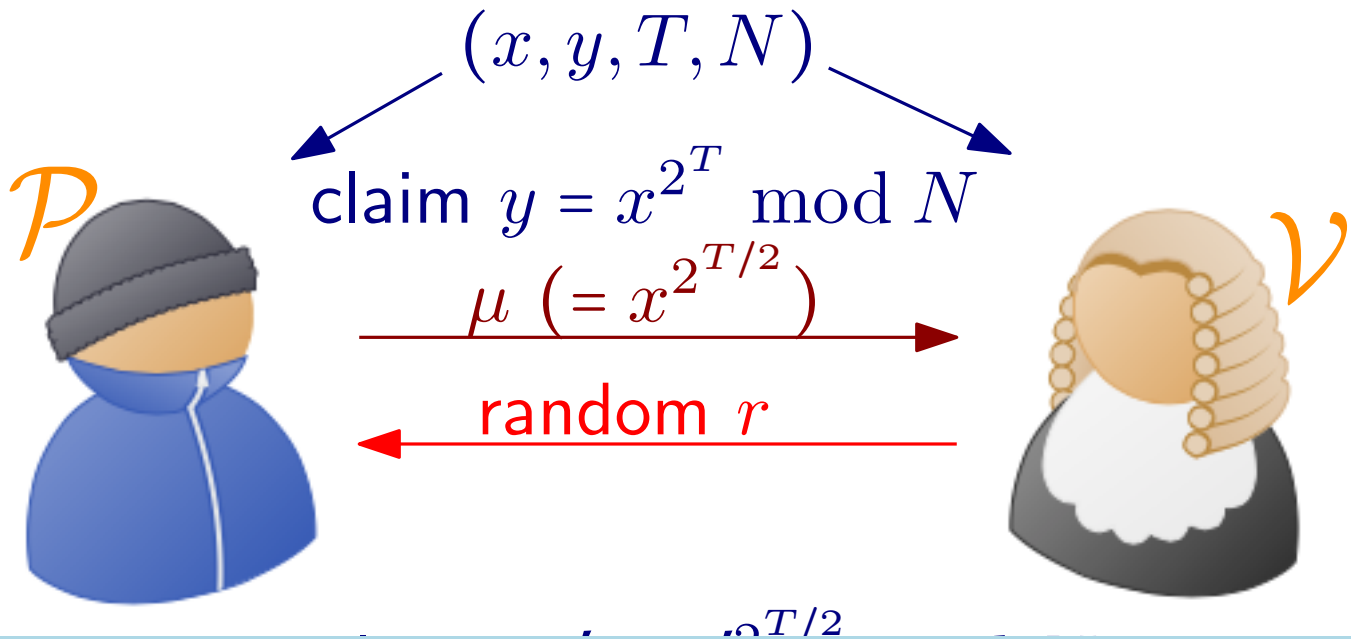
Proving $\sigma = x^{2^T}$ in Groups of Unknown Order



- Repeat $\log(T)$ times until $\mathcal{V}$ can verify claim for $T = 1$.
- Can be made non-interactive in random-oracle model.
- **Complexity**: $\mathcal{P}$ needs T time to compute y, and then $\sqrt{T}$ time and space to compute proof.
- **Proof Size**: $\log T$ elements (the μ values).

$x\ x^2\ x^{2^2}\ x^{2^3}\ \ldots \qquad \mu\ (= x^{2^{T/2}}) \qquad \ldots\ x^{2^{T-1}}\ x^{2^T}$

Proving $\sigma = x^{2^T}$ in Groups of Unknown Order



Concurrent work by Wesolowski[a] has proof of size 1 element (not $\log T$), but requires computational assumption: for any $\alpha \neq 1$, given random B it's hard to compute Bth root $\alpha^{1/B}$.

[a] https://eprint.iacr.org/2018/623.pdf

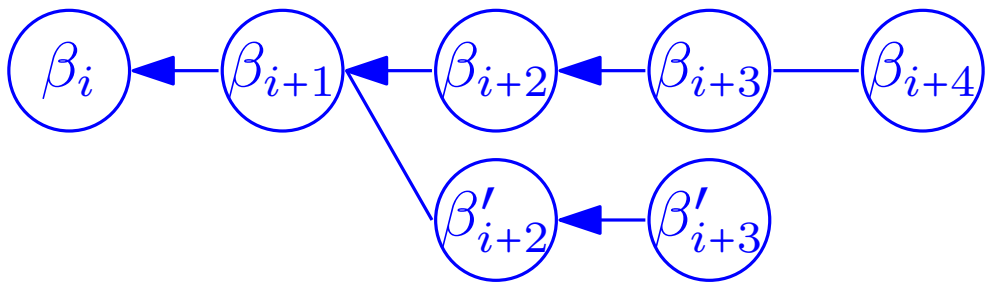
$x\ x^2\ x^{2^2}\ x^{2^3}\ \ldots \qquad \mu\ (= x^{2^{T/2}}) \qquad \ldots\ x^{2^{T-1}}\ x^{2^T}$

https://chia.net/

Bitcoin Mining Recap

Bticoin block $\beta_i = (\tilde{\beta}_i, \phi)$, $\tilde{\beta}_i = (i, H(\beta_{i-1}), pk, \tau_i)$ contains

- Transactions τ_i (consistent with chain so far).
- pk (block-reward and transactions fees go to pk)
- hash of previous block $H(\beta_{i-1})$
- PoW ϕ where $H(\phi, \tilde{\beta}_i) \leq treshhold$



Bitcoin mining:

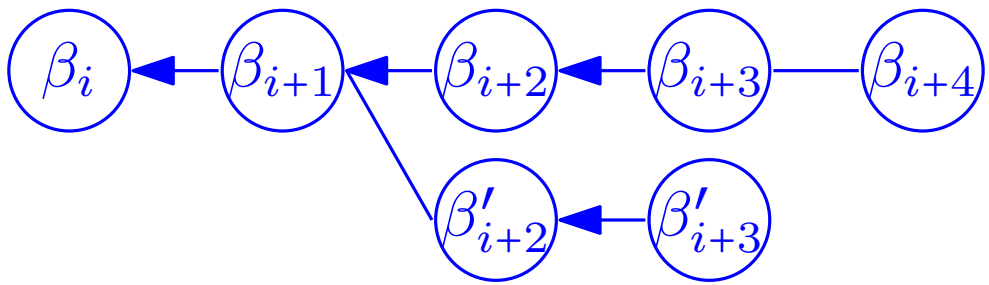
Init: sample signature key-pair (pk, sk)

1. Find head of longest chain β_{i-1} & compile block of transactions τ_i.
2. (PoW) hash distinct ϕ until $H(\phi, \ldots) \leq treshhold$
   - announce new block $\beta_i = (\phi, \ldots)$ and goto step 1.
   - if new longer chain observed immediately go to step 1.

Bitcoin Mining Recap

Bticoin block $\beta_i = (\tilde{\beta}_i, \phi)$, $\tilde{\beta}_i = (i, H(\beta_{i-1}), pk, \tau_i)$ contains

- Transactions τ_i (consistent with chain so far).
- pk (block-reward and transactions fees go to pk)
- hash of previous block $H(\beta_{i-1})$
- PoW ϕ where $H(\phi, \tilde{\beta}_i) \leq treshhold$



Bitcoin mining:

Init: sample signature key-pair (pk, sk)

1. Find head of longest chain β_{i-1} & compile block of transactions τ_i.
2. (PoW) hash distinct ϕ until $H(\phi, \ldots) \leq treshhold$
   - announce new block $\beta_i = (\phi, \ldots)$ and goto step 1.
   - if new longer chain observed immediately go to step 1.

Chia Building Blocks

Unique Digital Signatures

$$\forall m \ : \ \Pr[\text{Sig.verify}(pk, m, \phi) = \text{accept}] = 1$$

$$\text{where } (pk, sk) \leftarrow \text{Sig.keygen} \ ; \ \phi \leftarrow \text{Sig.sign}(sk, m)$$

$$\text{Unique} : (\text{Sig.verify}(pk, m, \phi) = \text{Sig.verify}(pk, m, \phi') = \text{accept}) \Rightarrow (\phi = \phi')$$

Proofs of Space

$$S \leftarrow \text{PoSpace.init}(pk, N)$$

$$\forall pk, N \ : \ \text{PoSpace.verify}(c, \text{PoSpace.prove}(S, pk, c)) = \text{accept}$$

$$\text{Weakly Unique} : \mathbb{E}_c\left[\{\sigma \ : \ \text{PoSpace.verify}(pk, c, \sigma) = \text{accept}\}\right] = 1$$

$$\text{Signed} : \sigma = (\sigma', \text{Sig.sign}(sk, \sigma'))$$

Verifiable Delay Functions

$$\forall t, c \ : \ \text{VDF.verify}(c, t, \text{VDF.prove}(c, t)) = \text{accept}$$

$\text{VDF.prove}(c, t)$ should take almost sequential time t to compute

A Blockchain from Proofs of Space and VDFs

Space Farmers

Initialization:$(pk, sk) \leftarrow$ Sig.KeyGen, $\Sigma \leftarrow$ PoSpace.Init(pk, N)

Mining: When new longest chain with head β_i observed:

compute $\phi \leftarrow$ PoSpace(Σ, c) for challenge $c := H(i, \beta_i, \tau_i, pk)$

gossip ϕ and define "quality" of ϕ as $q(\phi) := H(\phi)$.

A Blockchain from Proofs of Space and VDFs

Space Farmers

Initialization:$(pk, sk) \leftarrow$ Sig.KeyGen, $\Sigma \leftarrow$ PoSpace.Init(pk, N)

Mining: When new longest chain with head β_i observed: compute $\phi \leftarrow$ PoSpace(Σ, c) for challenge $c := H(i, \beta_i, \tau_i, pk)$

gossip ϕ and define "quality" of ϕ as $q(\phi) := H(\phi)$.

Time Lords

If PoSpace ϕ received, start computing

$$\tau \leftarrow \text{VDF}(\text{challenge} = \phi, \text{time} = q(\phi) \cdot \text{hardness parameter})$$

ONLY IF (given local view) this will be the first VDF to finalize a block at this level.

Gossip τ once finished.

A Blockchain from Proofs of Space and VDFs

Space Farmers

Initialization:$(pk, sk) \leftarrow$ Sig.KeyGen, $\Sigma \leftarrow$ PoSpace.Init(pk, N)

Mining: When new longest chain with head β_i observed: compute $\phi \leftarrow$ PoSpace(Σ, c) for challenge $c := H(i, \beta_i, \tau_i, pk)$

gossip ϕ and define "quality" of ϕ as $q(\phi) := H(\phi)$.

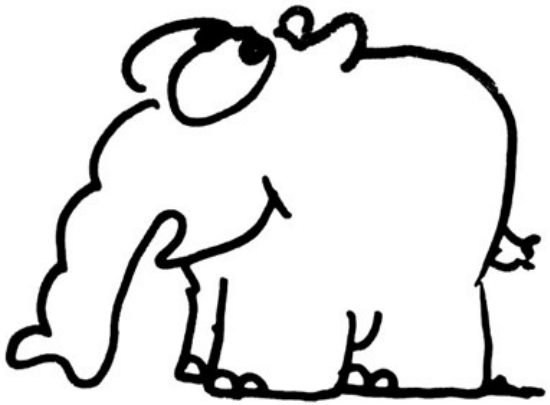
Time Lords

If PoSpace ϕ received, start computing

$$\tau \leftarrow \text{VDF}(\text{challenge} = \phi, \text{time} = q(\phi) \cdot \text{hardness parameter})$$

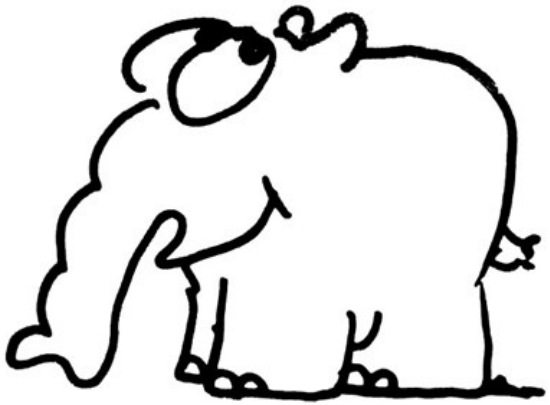
ONLY IF (given local view) this will be the first VDF to finalize a block at this level.

Gossip τ once finished.



Attacks

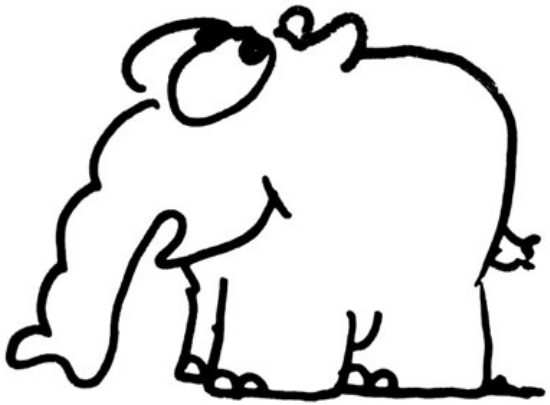
Farming: When new longest chain with head β_i observed: compute $\phi \leftarrow \mathsf{PoSpace}(\Sigma, c)$ for challenge $c := H(i, \beta_i, \tau_i, pk)$



Attacks

Farming: When new longest chain with head β_i observed: compute $\phi \leftarrow \mathsf{PoSpace}(\Sigma, c)$ for challenge $c := H(i, \beta_i, \tau_i, pk)$

Extending Multiple Chains: As (unlike PoW) computing a PoSpace is cheap, the miner can try to extend all blocks it learns about.



Attacks

Farming: When new longest chain with head β_i observed: compute $\phi \leftarrow \mathsf{PoSpace}(\Sigma, c)$ for challenge $c := H(i, \beta_i, \tau_i, pk)$

Extending Multiple Chains: As (unlike PoW) computing a PoSpace is cheap, the miner can try to extend all blocks it learns about.
provably not such a big problem (wait two slides)

Attacks

Farming: When new longest chain with head β_i observed: compute $\phi \leftarrow \text{PoSpace}(\Sigma, c)$ for challenge $c := H(i, \beta_i, \tau_i, pk)$

Extending Multiple Chains: As (unlike PoW) computing a PoSpace is cheap, the miner can try to extend all blocks it learns about.
provably not such a big problem (wait two slides)

Grinding: try out many τ_i's to get different c's until one found which gives me a super high quality PoSpace for next round $\Rightarrow$ can hijack chain forever!

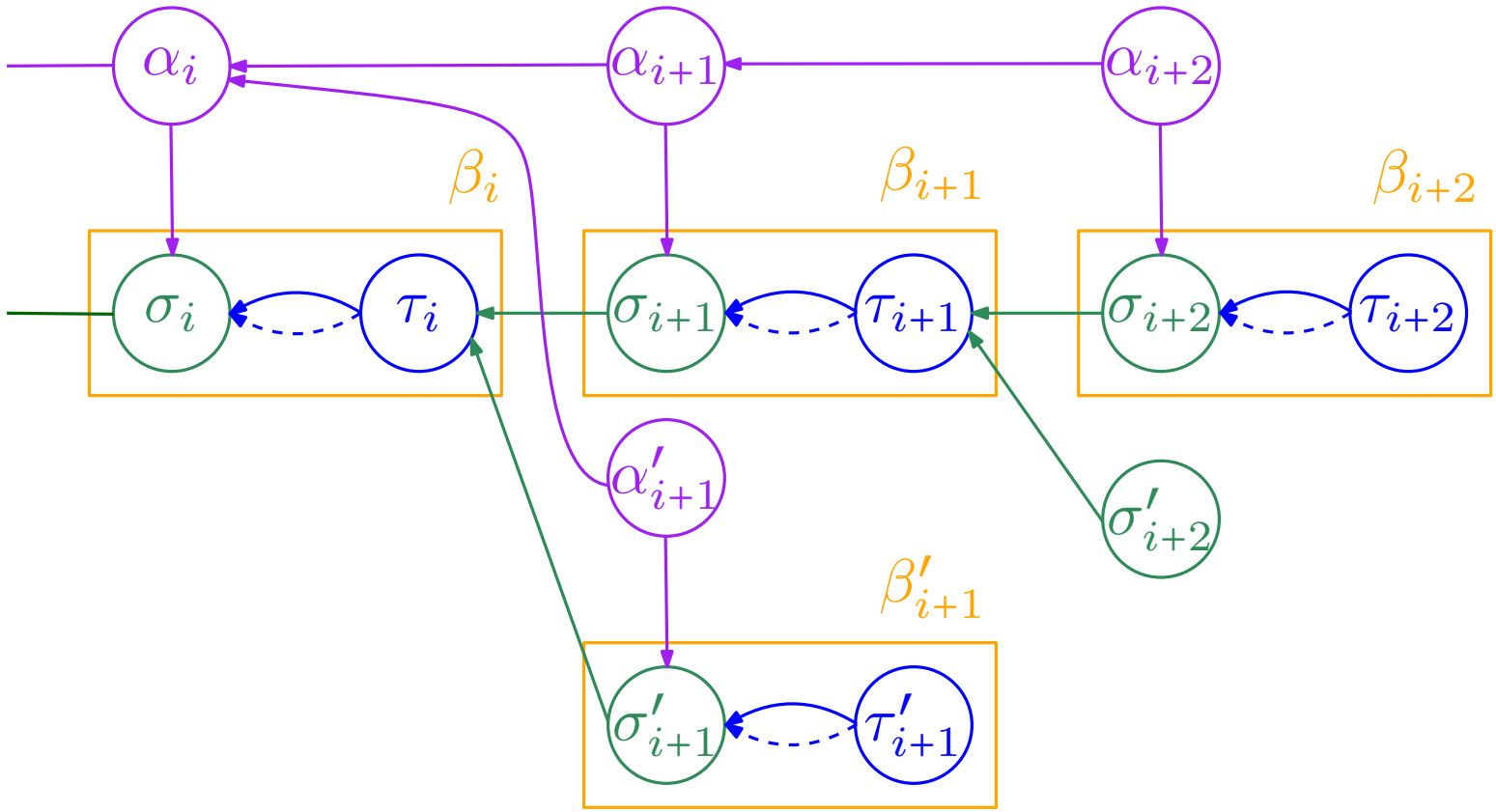
Attacks

Farming: When new longest chain with head β_i observed: compute $\phi \leftarrow \text{PoSpace}(\Sigma, c)$ for challenge $c := H(i, \beta_i, \tau_i, pk)$

Extending Multiple Chains: As (unlike PoW) computing a PoSpace is cheap, the miner can try to extend all blocks it learns about.
provably not such a big problem (wait two slides)

Grinding: try out many τ_i's to get different c's until one found which gives me a super high quality PoSpace for next round $\Rightarrow$ can hijack chain forever!
separate proofs from everything "graindable", Chia block format (next slide) kills the problem!

The Chia Block Format & (Non-)Grinding

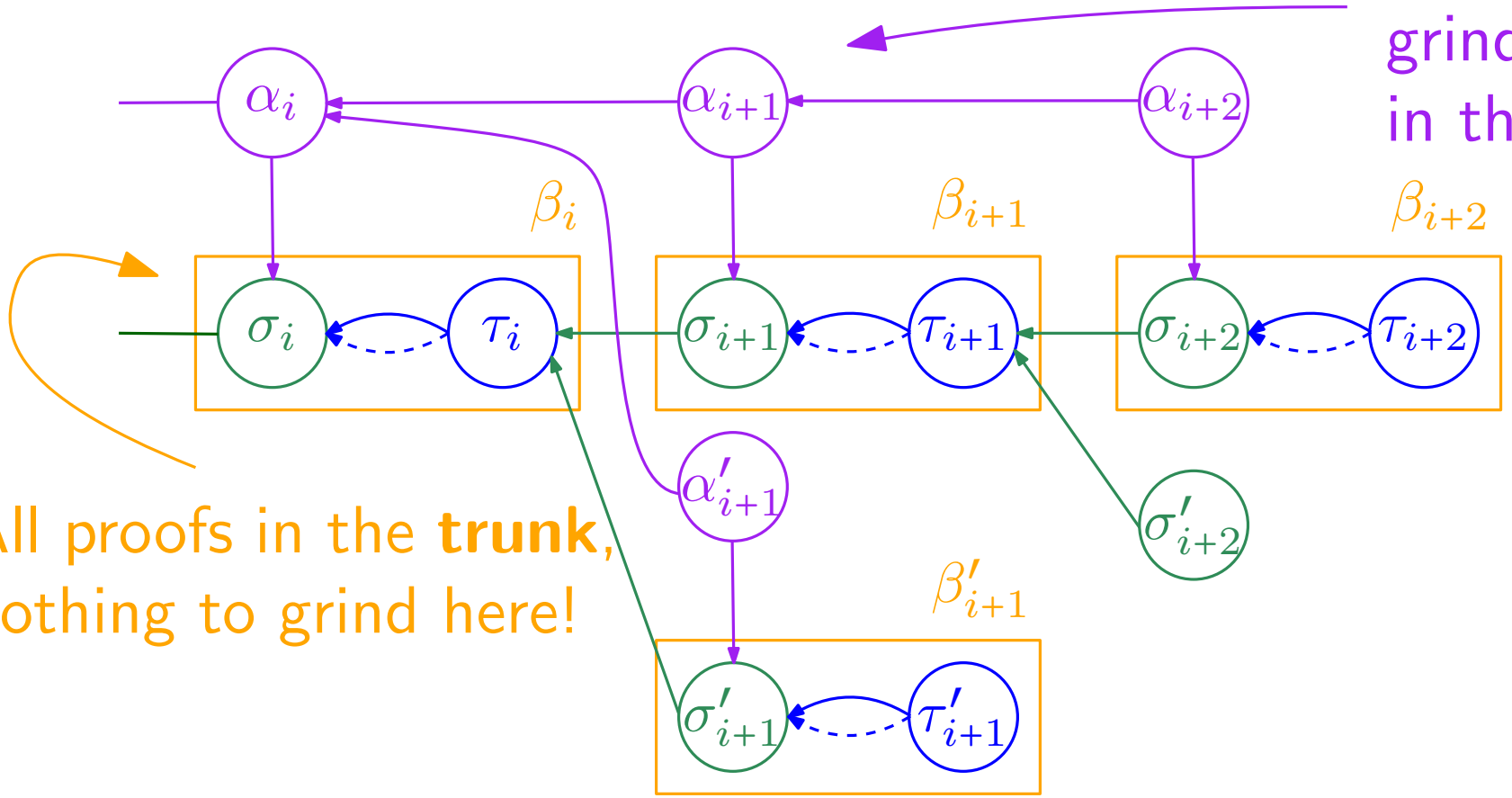


A full block $\gamma_i = (\beta_i, \alpha_i)$ contains
$\beta_i = (i, (pk_i, \sigma_i), \tau_i)$ and $\alpha_i = (\phi_i, data_i)$

1. PoSpace.verify$(pk_i, H(\tau_{i-1}), \sigma_i, N) = 1$
2. VDF.verify$(c, t, \tau_i) = 1$ where $c = H(\sigma_i)$, $t = 0.H(\sigma_i) \cdot T$
3. Sig.verify$(pk_i, H(\alpha_{i-1}, \sigma_i, data_i), \phi_i) = 1$

The Chia Block Format & (Non-)Grinding

Transactions and other grindable stuff in the **foliage**



All proofs in the **trunk**, nothing to grind here!

A full block $\gamma_i = (\beta_i, \alpha_i)$ contains
$\beta_i = (i, (pk_i, \sigma_i), \tau_i)$ and $\alpha_i = (\phi_i, data_i)$

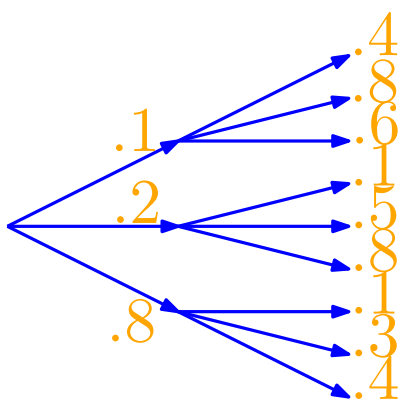
1. PoSpace.verify$(pk_i, H(\tau_{i-1}), \sigma_i, N) = 1$
2. VDF.verify$(c, t, \tau_i) = 1$ where $c = H(\sigma_i)$, $t = 0.H(\sigma_i) \cdot T$
3. Sig.verify$(pk_i, H(\alpha_{i-1}, \sigma_i, data_i), \phi_i) = 1$

Analysing Chain Growth

- h honest miners, each has one unit of space.
- adversarial miner with m units of space.
- every unit of space for every challenge gives a proof of quality uniform in $[0, 1]$.
- to finalize a proof of quality α takes time α (all VDF equally fast).
- adversary can run infinite number of VDF.
- no network delays.

Analysing Chain Growth

- h honest miners, each has one unit of space.
- adversarial miner with m units of space.
- every unit of space for every challenge gives a proof of quality uniform in $[0, 1]$.
- to finalize a proof of quality α takes time α (all VDF equally fast).
- adversary can run infinite number of VDF.
- no network delays.
- Consider h-ary tree of depth ℓ.
- Label every edge with random value from $[0, 1]$.
- Random Variable $C^{\ell}_{\kappa,h}$ is length of shortest path we find when always following the κ best edges from root to a leave.

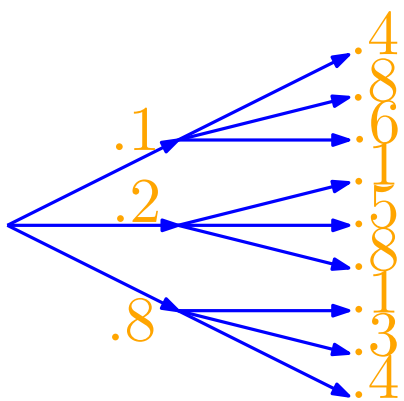


$C^{2}_{1,3} = .5$

$C^{2}_{\infty,3} = .3$

Analysing Chain Growth

- h honest miners, each has one unit of space.
- adversarial miner with m units of space.
- every unit of space for every challenge gives a proof of quality uniform in $[0, 1]$.
- to finalize a proof of quality α takes time α (all VDF equally fast).
- adversary can run infinite number of VDF.
- no network delays.
- Consider h-ary tree of depth ℓ.
- Label every edge with random value from $[0, 1]$.
- Random Variable $C^{\ell}_{\kappa,h}$ is length of shortest path we find when always following the κ best edges from root to a leave.



$C^{2}_{1,3} = .5$

$C^{2}_{\infty,3} = .3$

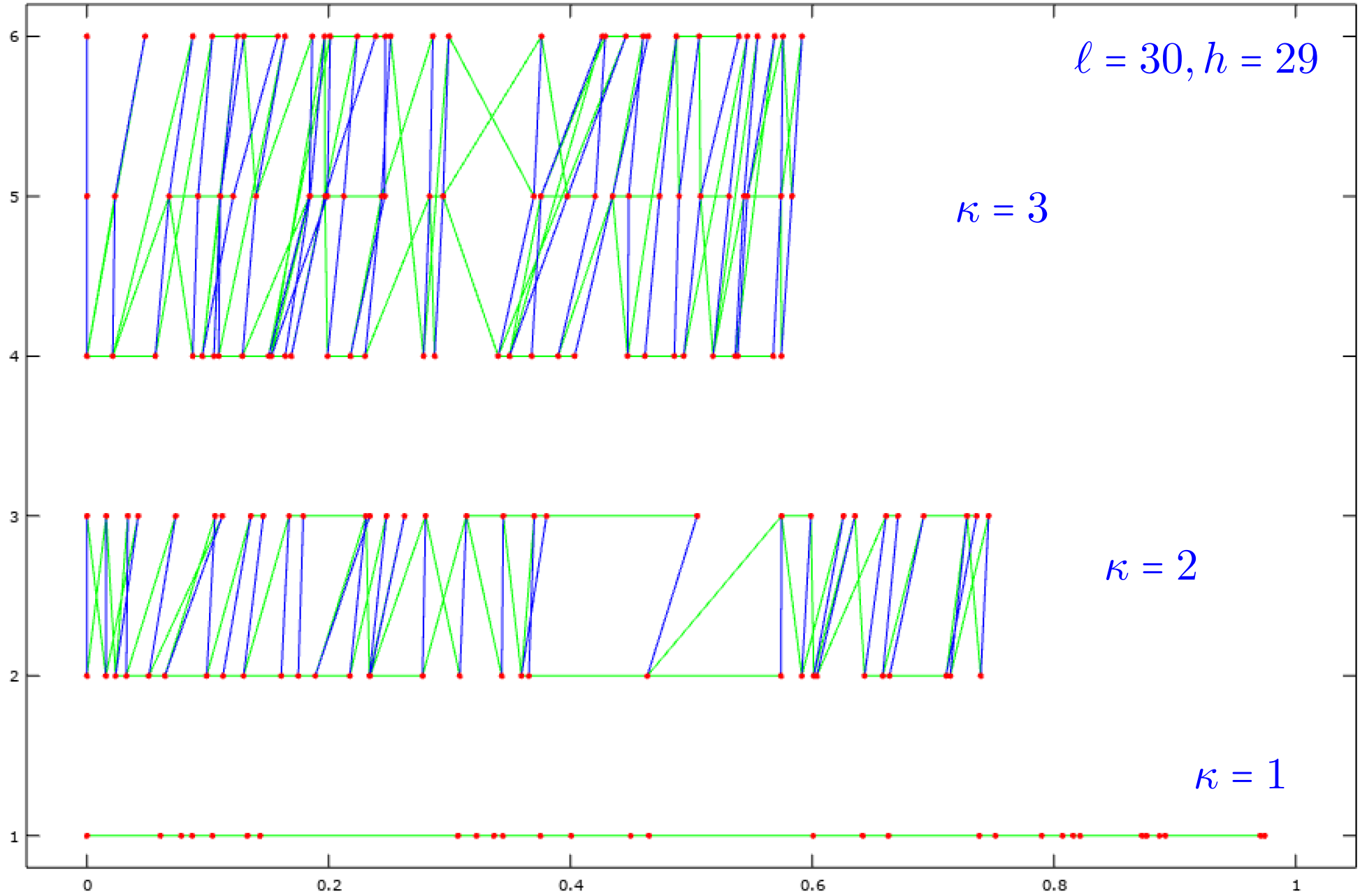
- $C^{\ell}_{\kappa,h}$ is expected time h honest miners need to grow chain of length ℓ.
- $C^{\ell}_{\infty,m}$ is expected time adversary controlling m space needs to grow chain of length ℓ.

Pseudocode For Sampling $C^{\ell}_{\kappa,h}$

Algorithm 1 sample $C^{\ell}_{\kappa,h}$

Input: κ, ℓ, h
$s[1,\ldots,\kappa] = 0$ ▷ initially we have κ paths of length 0
for $i = 1$ to ℓ **do** ▷ sample ℓ steps
  for $j = 1$ to κ **do** ▷ extend each of the κ states...
    for $k = 1$ to h **do** ▷ by h values...
      $p[j,k] = s[j] + \text{rand}([0,1])$ ▷ chosen uniform from $[0,1]$
    end for
  end for
  $z = \text{sort}(p[1,1],\ldots,p[\kappa,h])$ ▷ sort the $\kappa \cdot h$ values
  $s = z[1,\ldots,\kappa]$ ▷ new state are the κ shortest paths
end for
Return min(s)

Simulation of $C^{\ell}_{\kappa,h}$



What we know about $C^{\ell}_{\kappa,h}$

1. $C^{\ell}_{\kappa,h}$ is expected time h honest miners need to grow chain of length ℓ without adversarial interference
2. **No Slowdown Lemma**: an adversary with unbounded space and parallelism (but which cannot break the underlying signature scheme) cannot slow down the rate at which this chain grows.
3. We know exact expectation for $\kappa = 1$

$$E[C^{\ell}_{1,h}] = \frac{\ell}{h+1}$$

4. We can lower bound for $\kappa = \infty$

$$E[C^{\ell}_{\infty,h}] \geq \frac{\ell}{h+1} \cdot \frac{1}{e}$$

What we know about $C^{\ell}_{\kappa,h}$

1. $C^{\ell}_{\kappa,h}$ is expected time h honest miners need to grow chain of length ℓ without adversarial interference
2. **No Slowdown Lemma**: an adversary with unbounded space and parallelism (but which cannot break the underlying signature scheme) cannot slow down the rate at which this chain grows.
3. We know exact expectation for $\kappa = 1$

$$E[C^{\ell}_{1,h}] = \frac{\ell}{h+1}$$

4. We can lower bound for $\kappa = \infty$

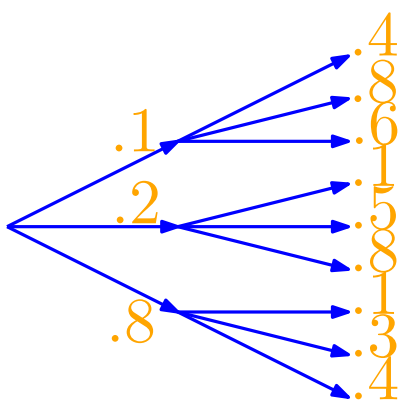
$$E[C^{\ell}_{\infty,h}] \geq \frac{\ell}{h+1} \cdot \frac{1}{e}$$

(Weak) Chain Quality Lemma: If $m < h/e$ (m space controlled by adversary, h honest space) then the fraction of honestly mined blocks is > 0.

Proof Sketch

- We can lower bound for $\kappa = \infty$

$$E[C^{\ell}_{\infty,h}] \geq \frac{\ell}{h+1} \cdot \frac{1}{e}$$



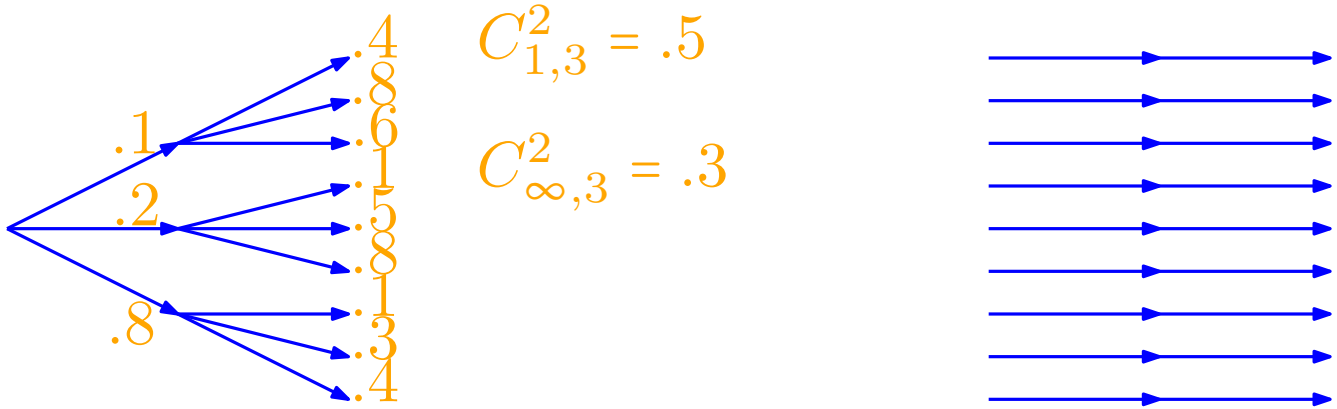
$C^2_{1,3} = .5$

$C^2_{\infty,3} = .3$

Proof Sketch

- We can lower bound for $\kappa = \infty$

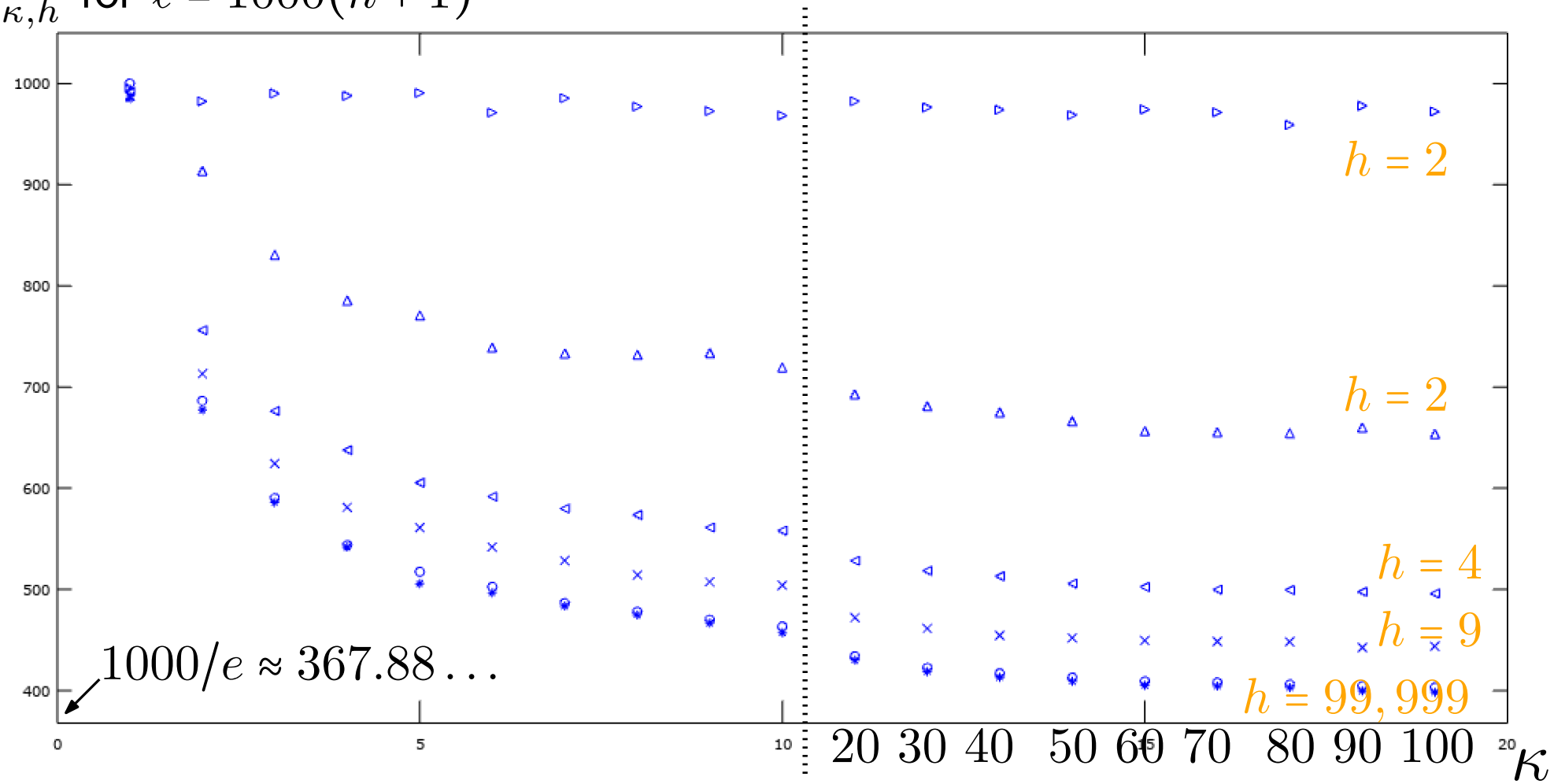
$$E[C^{\ell}_{\infty,h}] \geq \frac{\ell}{h+1} \cdot \frac{1}{e}$$



- Instead of analyzing shortest path in h-ary tree of depth ℓ, consider h^ℓ independent paths, prove that this tilts the bound in right direction.
- (Chernoff) Show that probability that any of those is shorter than x is $\ll \frac{1}{h^\ell}$.
- (Union Bound) Whp. all h^ℓ of them are shorter than x.

Choosing κ

$C^{\ell}_{\kappa,h}$ for $\ell = 1000(h+1)$



κ :	1	2	3	4	5	6	7	8	9
$C^{\ell}_{1,h}/C^{\ell}_{\kappa,h}$:	1	1.45	1.69	1.83	1.93	1.99	2.05	2.09	2.
$e_\kappa \approx \frac{e}{C^{\ell}_{1,h}/C^{\ell}_{\kappa,h}}$:	2.71	1.86	1.60	1.47	1.40	1.36	1.32	1.29	1.

Pseudocode

Space Miner Pseudocode (1/2)

Algorithm 2 SpaceMiner.init

Global Parameters: N
$\mathcal{C} \leftarrow$ Chain.init ▷ extract view from network
$(pk, sk) \leftarrow$ Sig.keygen ▷ generate a signature key pair
$S \leftarrow$ PoSpace.init(N, pk). ▷ run PoSpace initialisation with space N and identity pk to get a file S of size $|S| = N$.
Initalize a vector pos_count to all 0 ▷ see Remark **??**
Output:
$(pk, sk), S,$ pos_count ▷ State for SpaceMiner.mine
$\mathcal{C}$ ▷ State for Chain.update

Algorithm 3 SpaceMiner.loop

loop
    Wait for block(s) Γ to be received from the network
    $(\Gamma_f, \Gamma_n) \leftarrow$ Chain.update(Γ)
    $\forall \gamma \in \Gamma_f$: SpaceMiner.mine(γ) ▷ Algorithm 4
end loop

Space Miner Pseudocode (2/2)

Algorithm 4 SpaceMiner.mine

Global Parameters: κ
Input: $\gamma_i = (\beta_i = (i, \sigma_i, \tau_i), \alpha_i)$. ▷ finalized, fresh & valid block for slot i
State: (pk, sk), S, pos_count
if pos_count$(i) = \kappa$ **then** ▷ already generated κ PoS for slot i
    return without output
end if
pos_count$(i) \leftarrow$ pos_count$(i) + 1$
$\sigma_{i+1} \leftarrow$ PoSpace.prove$(S, pk, \mathsf{H}(\tau_i))$ ▷ produce PoSpace
Generate data$_{i+1}$ ▷ application specific
$\phi_{i+1} \leftarrow$ Sig.sign$(sk, (\alpha_i, \sigma_{i+1}, \text{data}_{i+1})$ ▷ signature for signature chain
Chain.update$((i+1, \sigma_{i+1}), \alpha_{i+1} = (\phi_{i+1}, \text{data}_{i+1}))$ ▷ Cf. §**??**

Time Miner Pseudocode (1/3)

Algorithm 5 TimeMiner.init

1: $\mathcal{C} \leftarrow$ Chain.init ▷ extract view from network
2: Initalize a vectors finalized and running to all 0
3: **Output:**
4: finalized, running ▷ State for TimeMiner.mine/finalized/runPoSW
5: $\mathcal{C}$ ▷ State for Chain.update

Algorithm 6 TimeMiner.loop

1: **loop**
2: Wait for block(s) Γ to be received from the network
3: $(\Gamma_f, \Gamma_n) \leftarrow$ Chain.update(Γ)
4: $\forall((i, \sigma), \alpha) \in \Gamma_n$: TimeMiner.mine(i, σ) ▷ Algorithm 7
5: $\forall((i, \sigma, \tau), \alpha) \in \Gamma_f$: TimeMiner.finalized(i) ▷ Algorithm 9
6: **end loop**

Algorithm 7 TimeMiner.mine

1: **Global Parameters:** T, κ
2: **Input:** $\beta_i = (i, \sigma_i)$ ▷ non-finalized, fresh & valid block for slot i received
3: **State:** finalized, running
4: **if** finalize$[i] = \kappa$ **then** ▷ already finalized κ blocks for this slot
5: return with no output
6: **end if**
7: $t := 0.\mathsf{H}(\sigma_i) \cdot T$ ▷ time required to finalize this block
8: **if** finalize$[i]$ + running$[i] < \kappa$ **then** ▷ $< \kappa$ proofs finalized or running
9: start thread TimeMiner.runPoSW$(i, \mathsf{H}(\sigma_i), t)$ ▷ to finish at time now + t
10: running$[i]$ = running$[i] + 1$
11: **end if**
12: **if** finalize$[i]$ + running$[i] = \kappa$ **then** ▷ exactly κ proofs finalized or running
13: **if** the slowest PoSW for slot i will finish at time $> t +$ now **then**
14: abort the thread of this PoSW
15: start thread TimeMiner.runPoSW$(i, \mathsf{H}(\sigma_i), t)$
16: **end if**
17: **end if**

Time Miner Pseudocode (3/3)

Algorithm 8 TimeMiner.runPoSW

State: finalized, running
Input: $i, (c, t)$
$\tau_i \leftarrow \text{PoSW}(c, t)$ ▷ start PoSW, if not aborted will output proof τ_i in time t
finalized$[i]$ = finalized$[i] + 1$
running$[i]$ = running$[i] - 1$
Chain.update(τ_i)

Algorithm 9 TimeMiner.finalized

State: finalized, running
Input: i ▷ fresh, valid & finalized block for slot i was received
if running$[i] > 0$ and running$[i]$+finalized$[i] = \kappa$ **then**
    abort the thread TimeMiner.runPoSW for slot i scheduled to finish last
    running$[i]$ = running$[i] - 1$
end if
finalized$[i] = \min\{\text{finalized}[i] + 1, \kappa\}$

Some Open Problems

- (PoSW) Construct unique proofs of sequential work without heavy crypto machinery (SNARKs).
- (PoS) Is there a proof of space with non-interactive initialization and (at least asymptotically) optimal bounds?
- (Analysis) Better chain quality, persistience etc. analysis? Can we say something about rational (not just honest) miners?