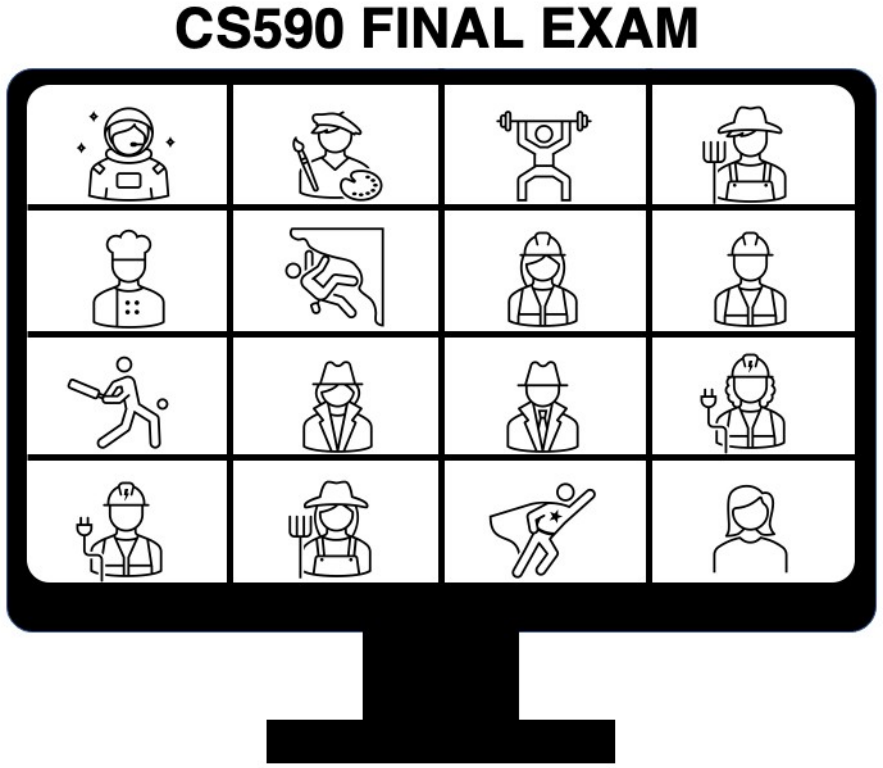
Advanced Cryptography
CS 655

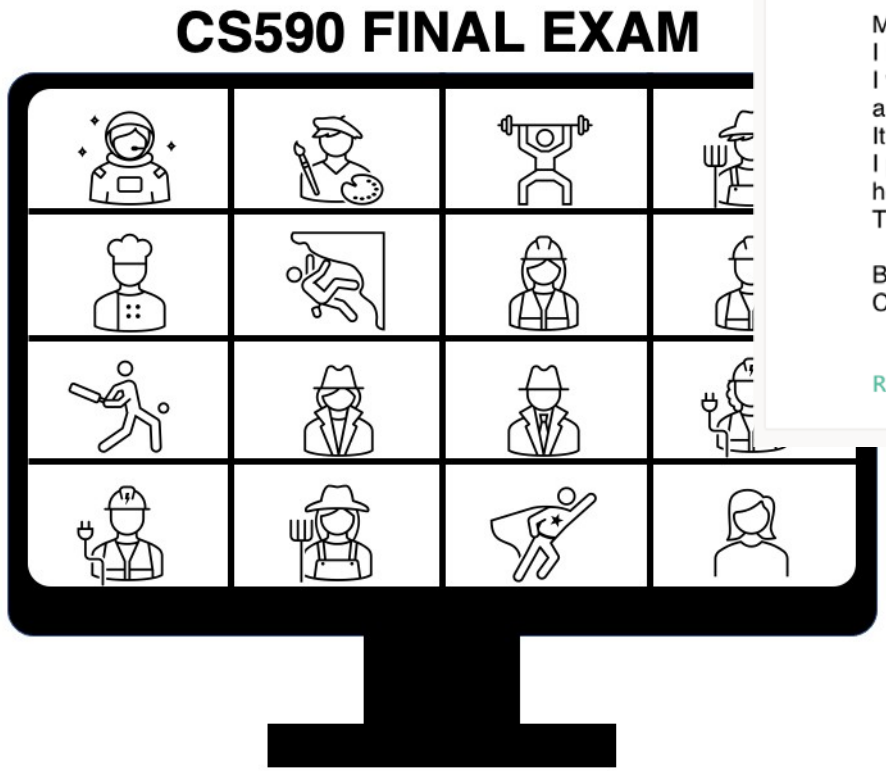
Week 9:

- SCRYPT (wrapup)
- Proof of Sequential Work/Proof of Space

Motivation: Online Exams during the Pandemic



Motivation: Online Exam nic



[CS590] 5 mins late - having internet issue

CG

Cinseer Goodman
Tue 5/2/2021 9:05 PM
To: Seunghoon Lee

answer-goodman.pdf
157 KB

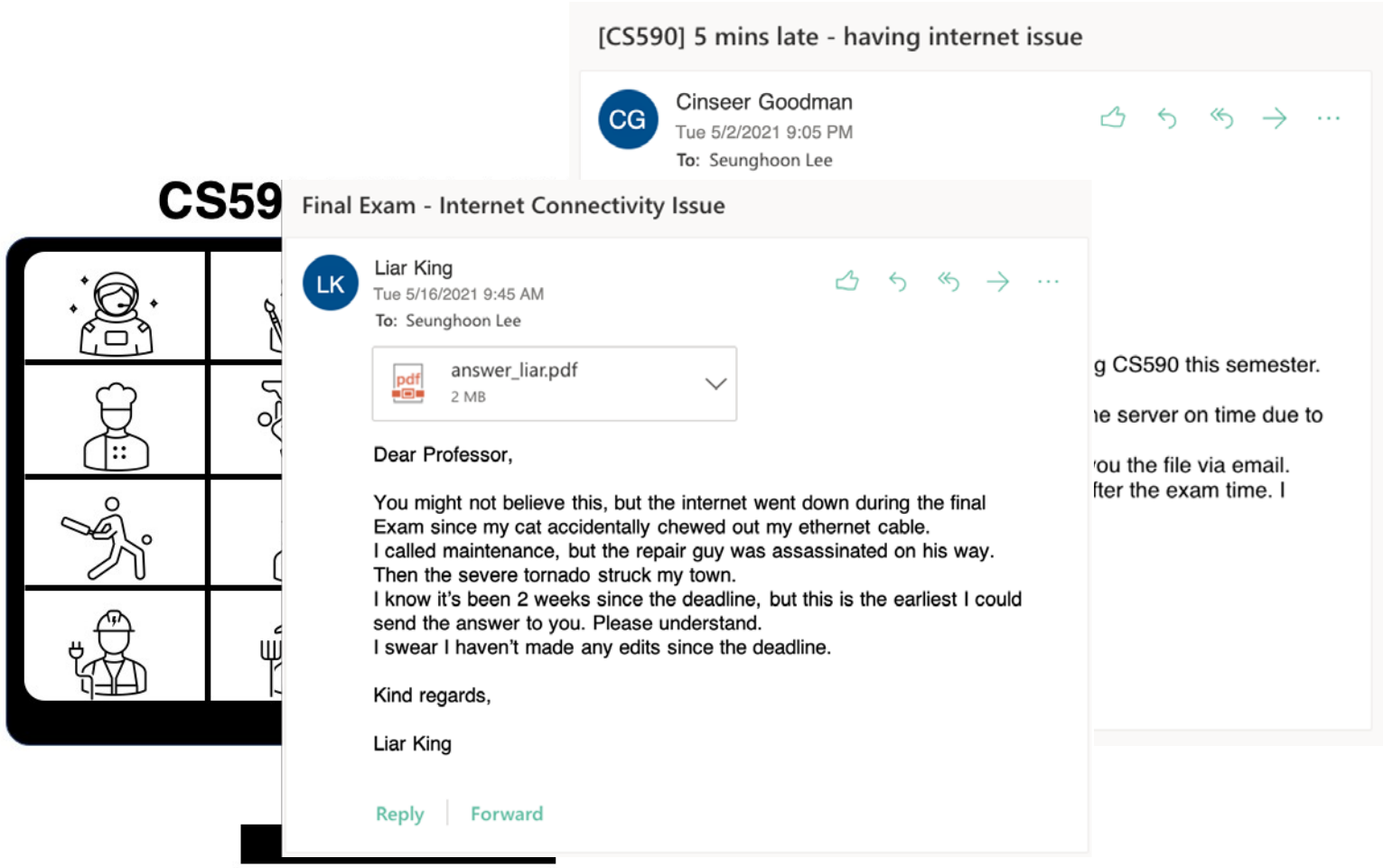
Dear Professor,

My name is Cinseer Goodman who is taking CS590 this semester.
I hope this email finds you well.
I was not able to submit the final exam to the server on time due to an unexpected internet connectivity loss.
It just went back 5 minutes later so I send you the file via email.
I promise I have not done any extra work after the exam time. I hope it works.
Thank you.

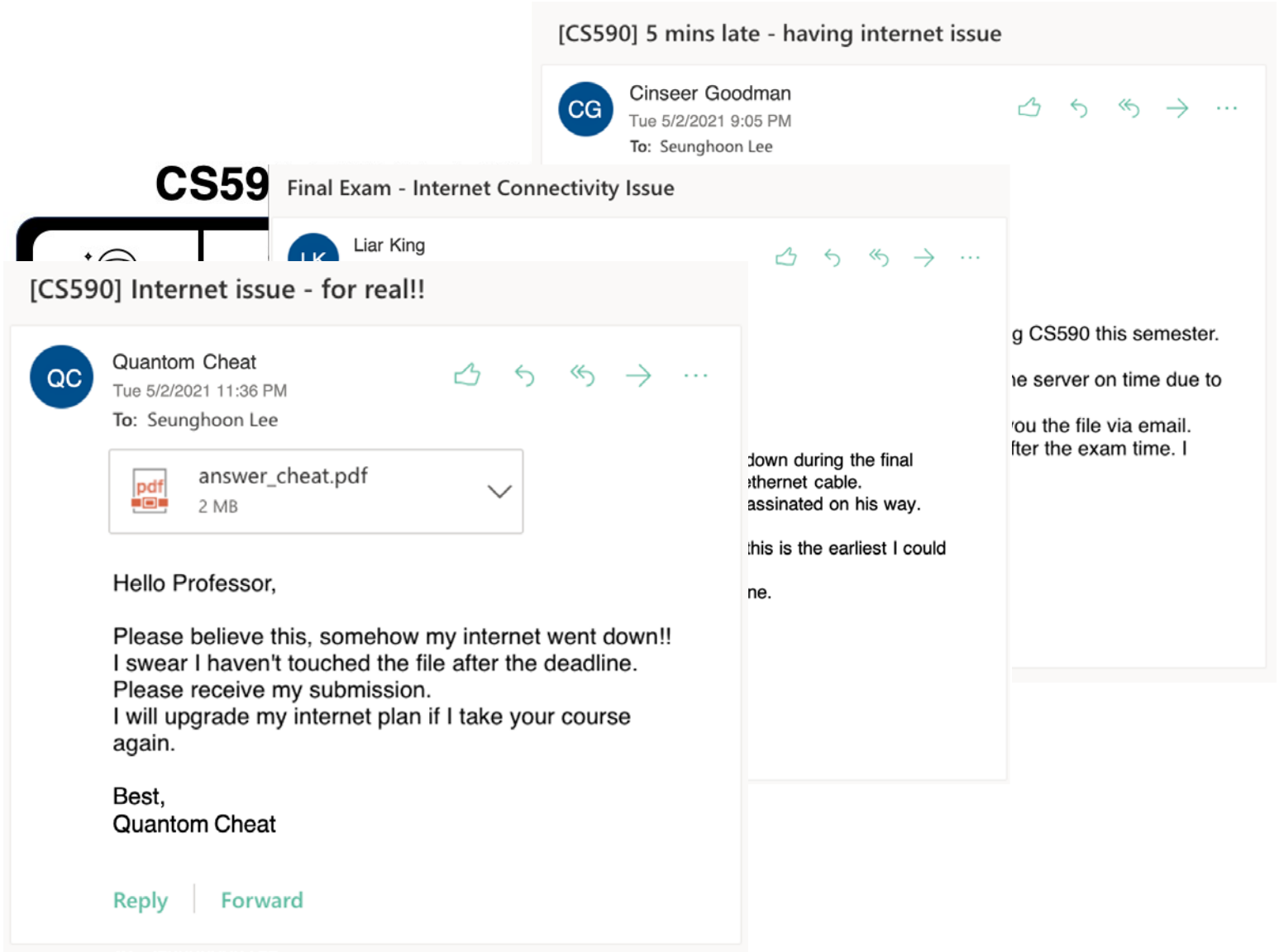
Best,
Cinseer Goodman

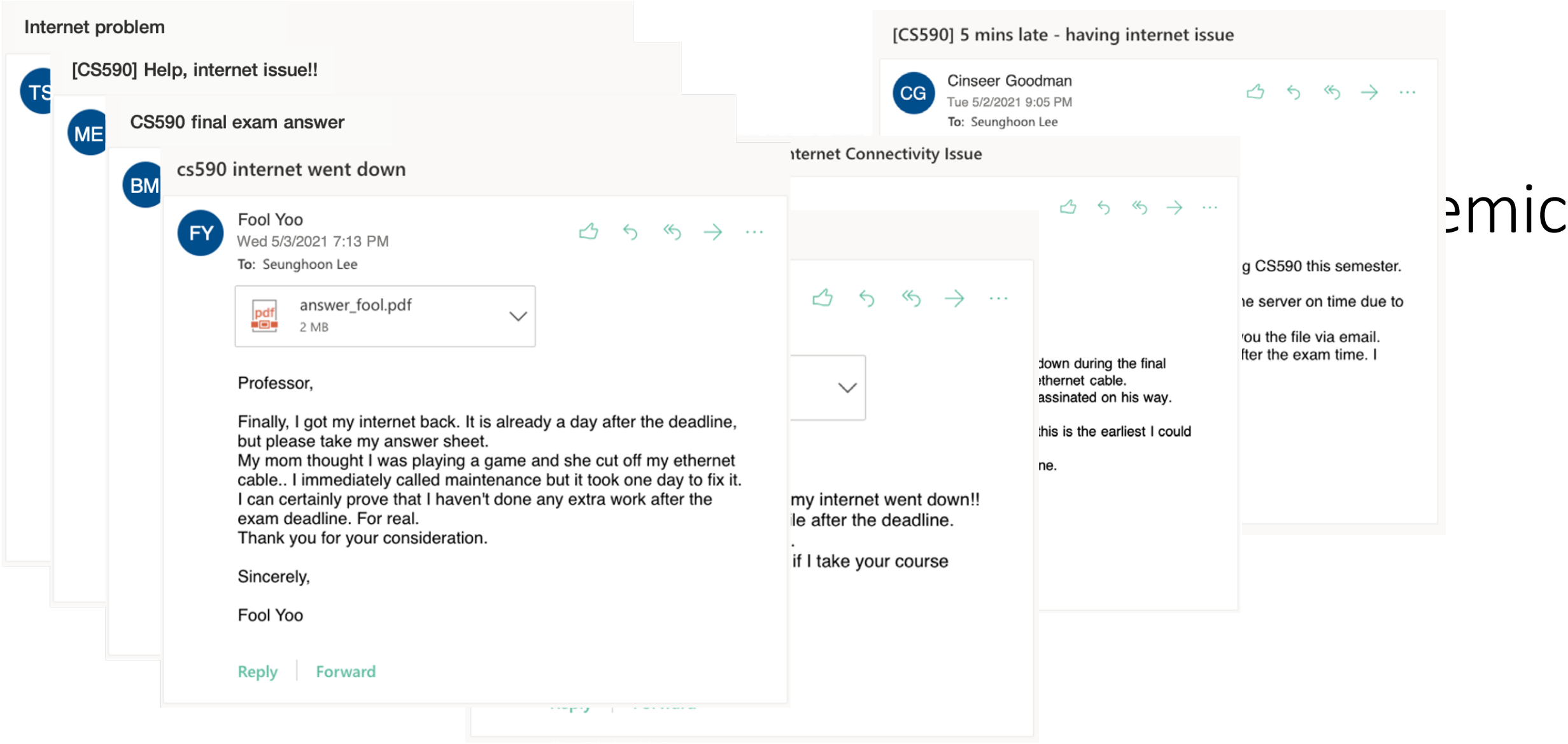
Reply | Forward

Motivatio emic



Motivation: Online Exams during the Pandemic




Internet problem
[CS590] Help, internet issue!!
CS590 final exam answer
cs590 internet went down
Fool Yoo
Wed 5/3/2021 7:13 PM
To: Seunghoon Lee
answer_fool.pdf
2 MB
Professor,
Finally, I got my internet back. It is already a day after the deadline, but please take my answer sheet.
My mom thought I was playing a game and she cut off my ethernet cable.. I immediately called maintenance but it took one day to fix it.
I can certainly prove that I haven't done any extra work after the exam deadline. For real.
Thank you for your consideration.
Sincerely,
Fool Yoo
Reply
Forward
[CS590] 5 mins late - having internet issue
Cinseer Goodman
Tue 5/2/2021 9:05 PM
To: Seunghoon Lee
g CS590 this semester.
e server on time due to
ou the file via email.
fter the exam time. I
down during the final
thernet cable.
assinated on his way.
this is the earliest I could
ne.
my internet went down!!
le after the deadline.
if I take your course

Internet problem

[CS590] Help, internet issue!!

CS590 final exam answer

cs590 internet went down

Fool Yoo
Wed 5/3/2021 7:13 PM
To: Seunghoon Lee

answer_fool.pdf
2 MB

Professor,

Finally, I got my internet back. It is already a day after the deadline, but please take my answer sheet.
My mom thought I was playing a game and she cut off my ethernet cable.. I immediately called maintenance but it
I can certainly prove that I haven't do
exam deadline. For real.
Thank you for your consideration

Sincerely,

Fool Yoo

Reply | Forward

[CS590] 5 mins late - having internet issue

Cinseer Goodman
Tue 5/2/2021 9:05 PM
To: Seunghoon Lee

nternet Connectivity Issue

g CS590 this semester.
e server on time due to
ou the file via email.
fter the exam time. I
down during the final
ethernet cable.
assinated on his way.
this is the earliest I could
ne.

Which students are telling the truth?

emic

Proofs of Sequential Work

aka. Verifiable Delay Algorithm

Prover P

statement x

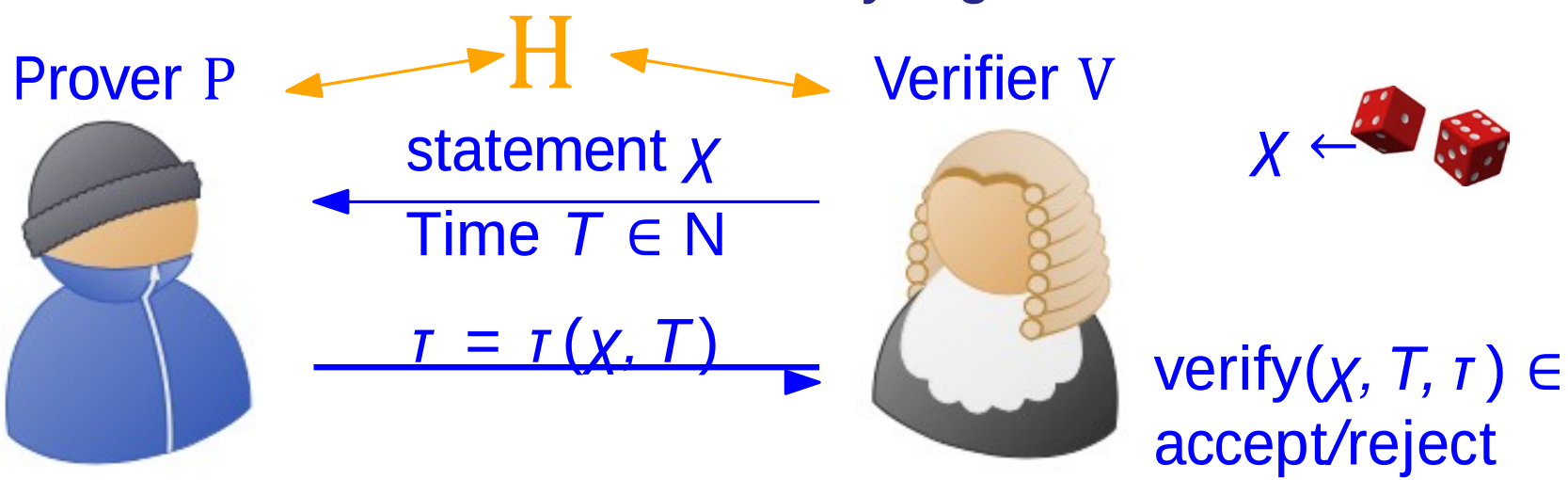
Time $T \in \mathrm{N}$

Verifier V

$X \leftarrow$ 

Proofs of Sequential Work

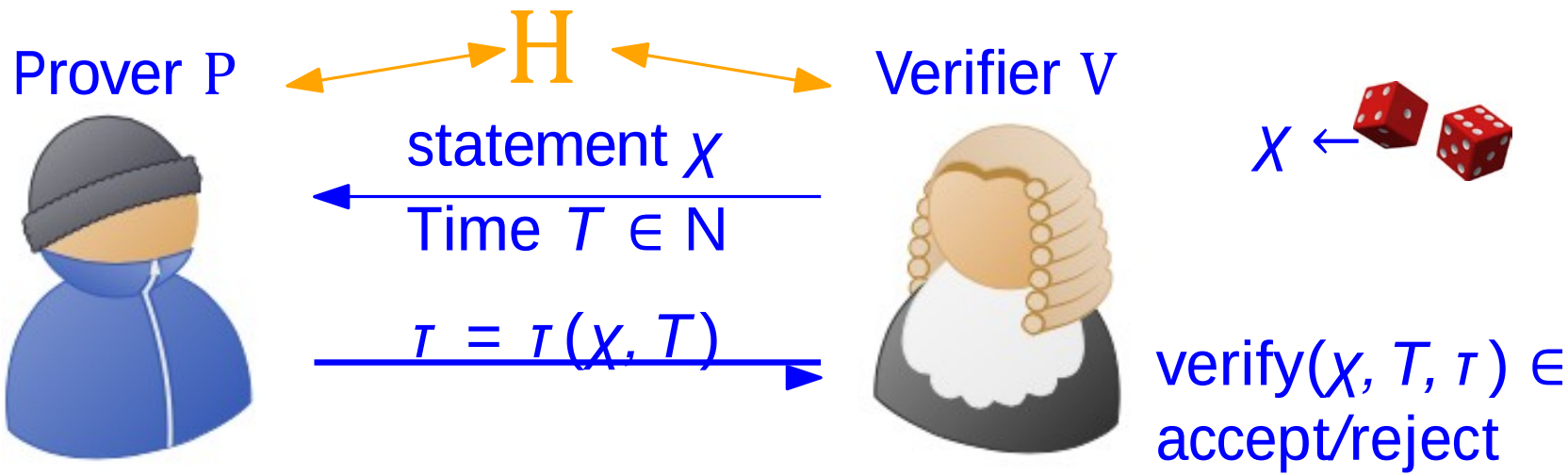
aka. Verifiable Delay Algorithm



Completeness and Soundness *in the random oracle model:*

Proofs of Sequential Work

aka. Verifiable Delay Algorithm



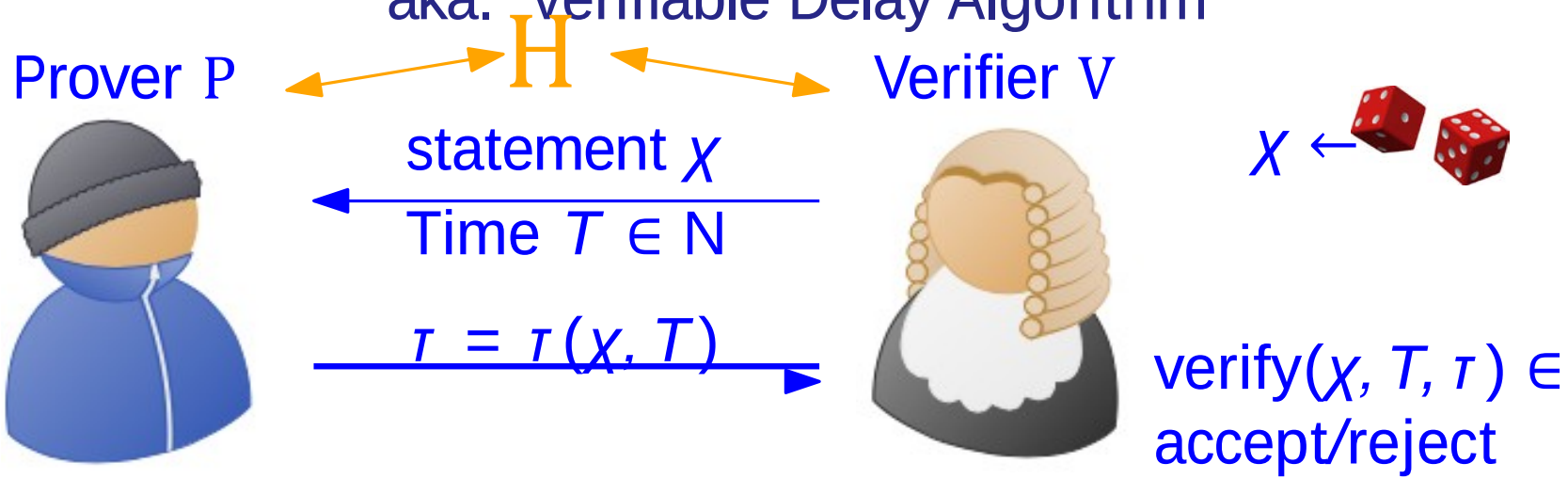
Completeness and Soundness *in the random oracle model:*

Completeness: $\tau(c, T)$ can be computed making T queries to H

Soundness: Computing any τ' s.t. verify(χ, T, τ') =accept for random χ requires almost T *sequential* queries to H

Proofs of Sequential Work

aka. Verifiable Delay Algorithm



Completeness and Soundness *in the random oracle model:*

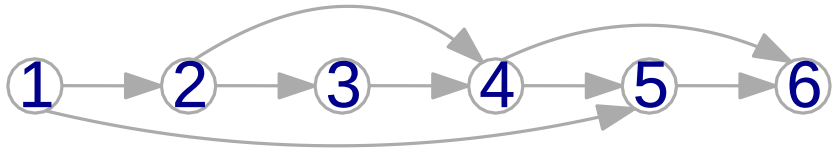
Completeness: $\tau(c, T)$ can be computed making T queries to H

Soundness: Computing any τ' s.t. verify(χ, T, τ') =accept for random χ requires almost T *sequential* queries to H

massive parallelism useless to generate valid proof faster ⇒
prover must make almost T sequential queries ~ T time

Three Basic Concepts

Depth-Robust Graphs (only [MMV'13])

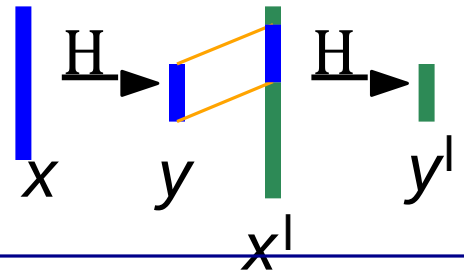


DAG $G = (V, E)$ is (e, d) **depth-robust** if after removing any e nodes a path of length d exists.

Graph **Labelling**

label $\pounds_i = \mathrm{H}(\pounds_{parents(i)})$, e.g. $\pounds_4 = \mathrm{H}(\pounds_3, \pounds_2)$

Random Oracles are **Sequential**

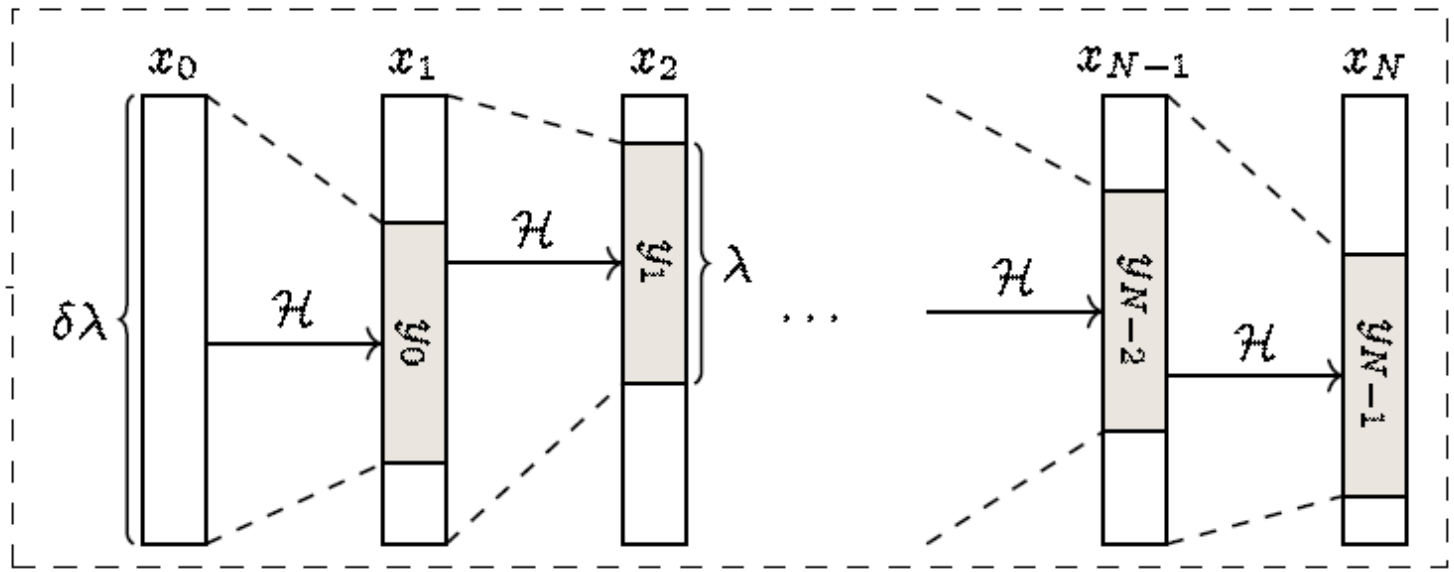


queries $y = \mathrm{H}(x)$, $y' = \mathrm{H}(x')$ where
$y \subseteq x' \Rightarrow$ query x' was made after x

H-Sequence

Definition 3 (H-sequence). *An* H *sequence of length* s *is a sequence* $x_0, \ldots, x_s \in \{0,1\}^*$ *where for each* $i, 1 \leq i < s$, $\mathsf{H}(x_i)$ *is contained in* x_{i+1} *as continuous substring, i.e.,* $x_{i+1} = a \| \mathsf{H}(x_i) \| b$ *for some* $a, b \in \{0,1\}^*$.

$x_0, x_1, \ldots, x_N \in \{0,1\}^*$ s.t. for each $1 \leq i \leq N$, there exist $a, b \in \{0,1\}^*$ such that $x_i = a \| \mathcal{H}(x_{i-1}) \| b$.



Random Oracles are Sequential

- Let $H: \{0,1\}^{\leq \delta\lambda} \rightarrow \{0,1\}^{\lambda}$ be a random oracle
- Suppose that the attacker may make $s-1$ rounds of sequential queries
- **Attacker Goal:** output an H-sequence $x_0, \ldots, x_s$ of length s with each $|x_i| \leq \delta\lambda$
- Suppose that attacker makes at most q RO queries

Lemma: The attacker succeeds with probability at most $\frac{q^2\delta\lambda + qs\delta\lambda}{2^{\lambda}}$

Random Oracles are Sequential

Lemma: The attacker succeeds with probability at most $\frac{q^2\delta\lambda + qs\delta\lambda}{2^\lambda}$

Proof Sketch:

Let LuckyGuess denote the event that for some i the string $H(x_i)$ is a substring of x_{i+1} but the attacker never actually made the query $H(x_i)$.

Claim 1: $\Pr[\text{LuckyGuess}] \leq \frac{s(\delta-1)\lambda}{2^\lambda}$

Proof of Claim 1: Fix any index i and any $j \leq (\delta - 1)\lambda$ we have

$$\Pr\left[H(x_i) = x_{i+1}[j, j+\lambda-1]\right] \leq \frac{1}{2^\lambda}$$

We now union bound over all indices $i \leq s$ and all $j \leq (\delta - 1)\lambda$

Random Oracles are Sequential

Lemma: The attacker succeeds with probability at most $\frac{q^2\delta\lambda + qs\delta\lambda}{2^\lambda}$

Proof Sketch:

Let Collision denote the event that for some $1 \leq i < j \leq s-1$ there is a query a_i made in round i and a query a_j made in round j where $H(a_j)$ is a substring of a_i

Claim 2: $\Pr[\text{Collision}] \leq \frac{q^2\delta\lambda}{2^\lambda}$

Proof of Claim 2: Fix any pair of queries a_i and a_j and any index $k \leq (\delta-1)\lambda$

Observe that $H(a_j)$ can be viewed as a random string picked after a_i is fixed.

$$\Pr\left[H(a_j) = a_i[k, k+\lambda-1]\right] \leq \frac{1}{2^\lambda}$$

We now union bound over all $\binom{q}{2}$ pairs of queries and all $j \leq (\delta-1)\lambda$

Random Oracles are Sequential

Lemma: The attacker succeeds with probability at most $\frac{q^2\delta\lambda + qs\delta\lambda}{2^\lambda}$

Proof Sketch:

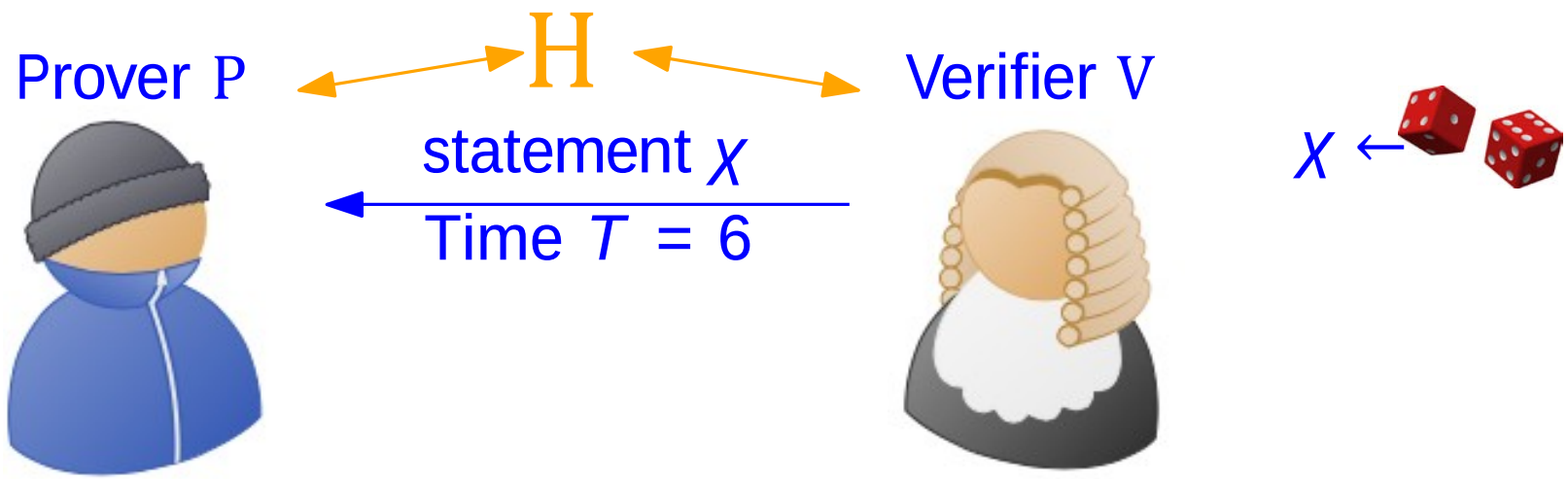
$$\Pr[\text{LuckyGuess}] + \Pr[\text{Collision}] \leq \frac{q^2\delta\lambda + s\delta\lambda}{2^\lambda}$$

If the attacker produces an H-sequence of length s then *at least one* of the events LuckyGuess or Collision must occur.

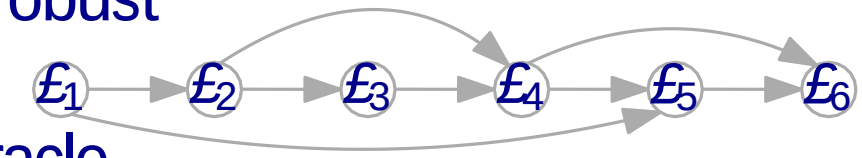
LuckyGuess: for some i the string $H(x_i)$ is a substring of x_{i+1} although attacker never queried $H(x_i)$.

Collision: for some $1 \leq i < j \leq s-1$ $H(a_j)$ is a substring of a_i where the query a_i (resp. a_j) is made in round i (resp. j).

The MMV'13 Construction

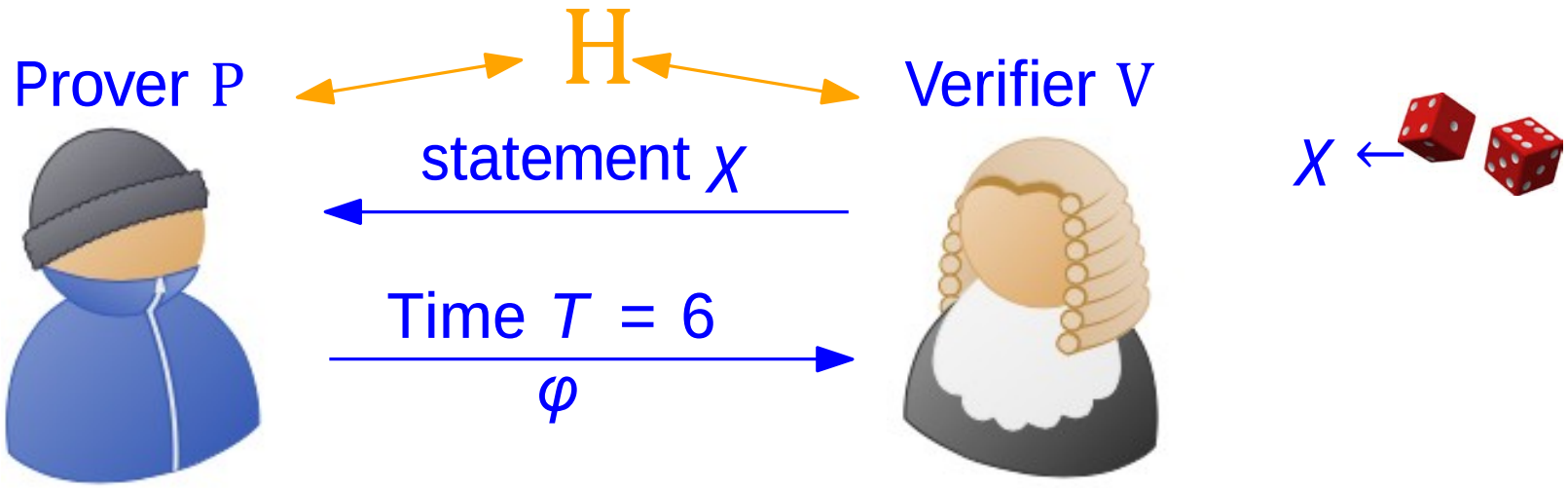


- Protocol specifies depth-robust DAG G on T nodes



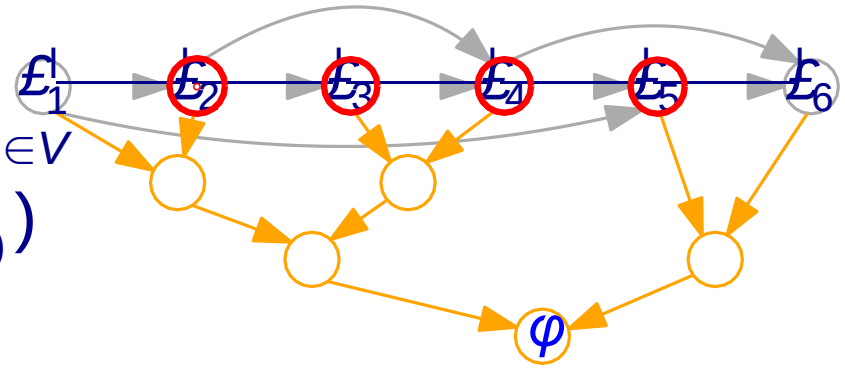
- Define "fresh" random oracle $\mathrm{H}_\chi(\cdot) \equiv \mathrm{H}(\chi\,\cdot)$
- Compute labels of G using H_χ

The MMV'13 Construction



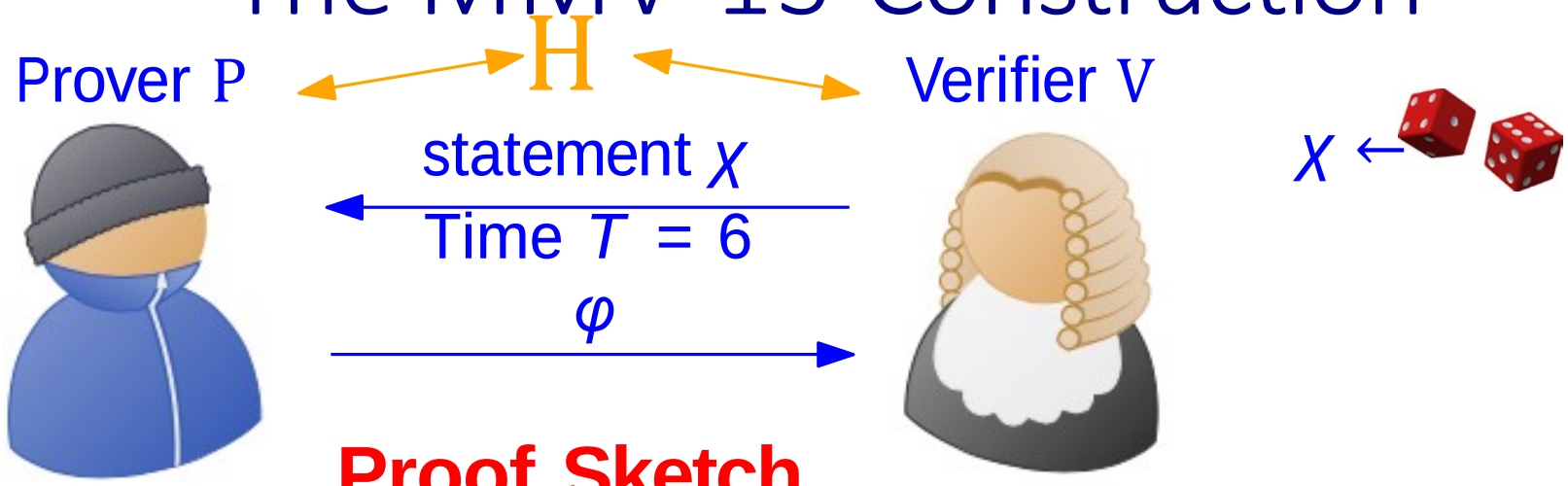
Proof Sketch

- G is (e, d) depth-robust
- φ commits $\tilde{P}$ to labels $\{\ell_i\}_{i \in V}$
- i is **bad** if $\ell_i \neq H(\ell_{parents(i)})$



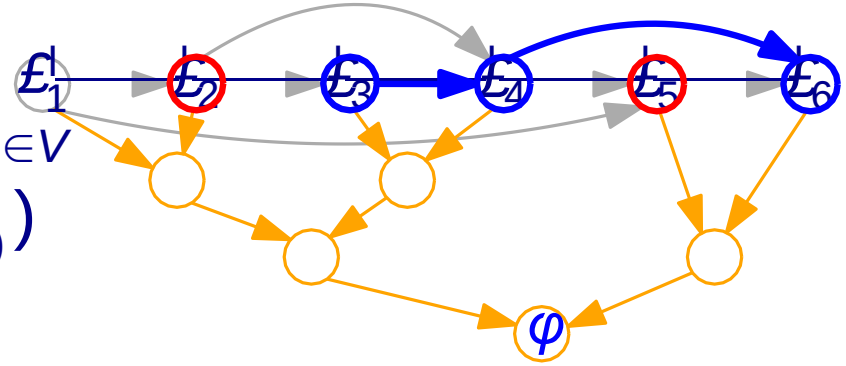
- Case 1: ≥ e bad nodes ⇒ will fail opening phase whp.

The MMV'13 Construction



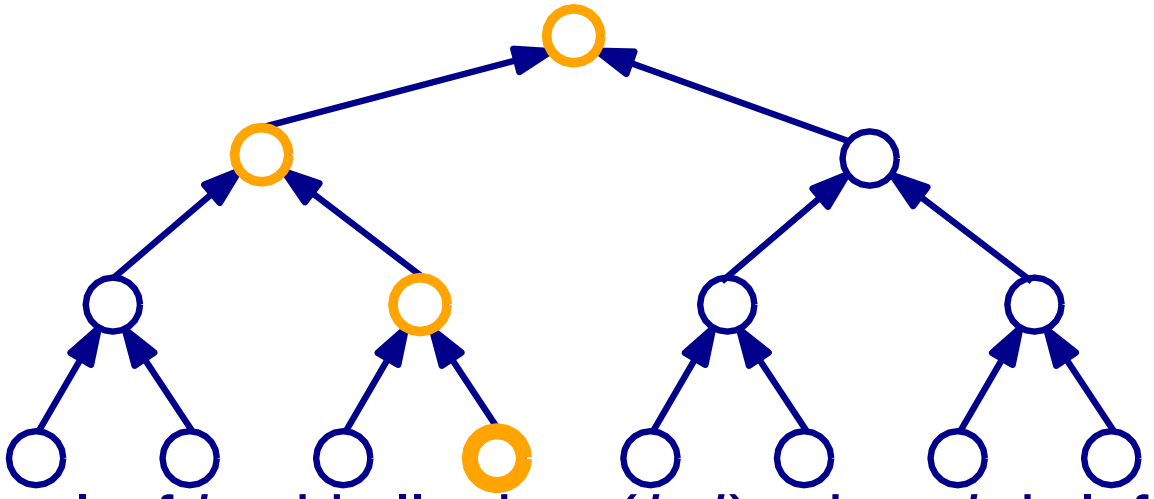
Proof Sketch

- G is (e, d) depth-robust
- φ commits $\tilde{P}$ to labels $\{\ell_i\}_{i \in V}$
- i is **bad** if $\ell_i \neq H(\ell_{parents(i)})$



- Case 1: ≥ e bad nodes ⇒ will fail opening phase whp.
- Case 2: Less than e bad labels ⇒ ∃ path of good nodes (by (e, d) depth-robustness) ⇒ $\tilde{P}$ made d sequential queries (by sequantality of RO)

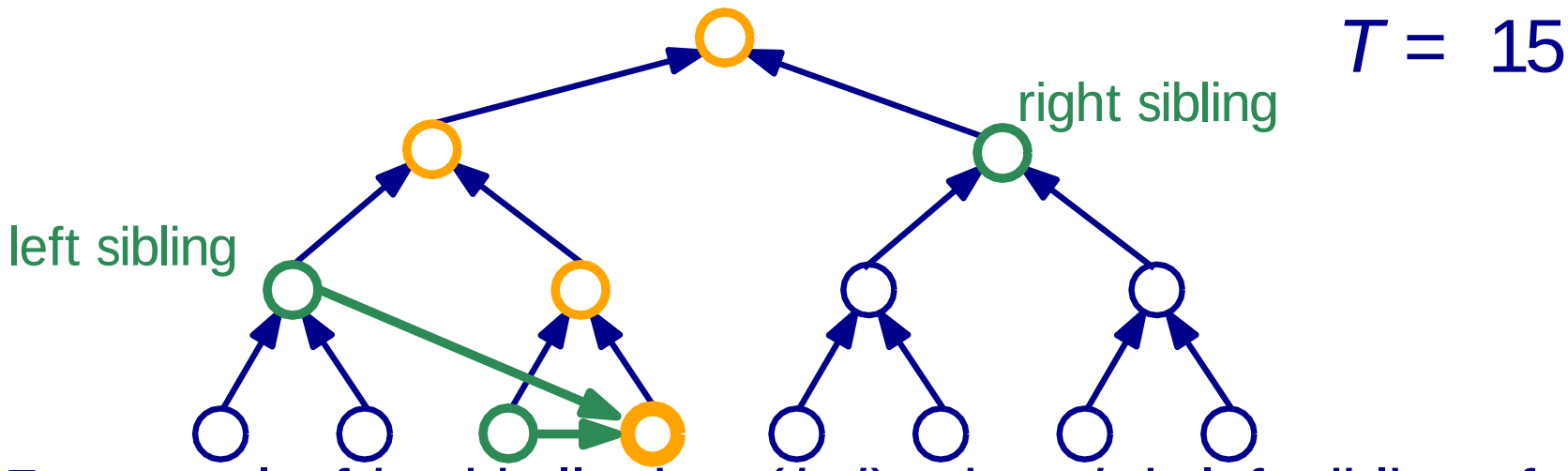
The New Construction



$T = 15$

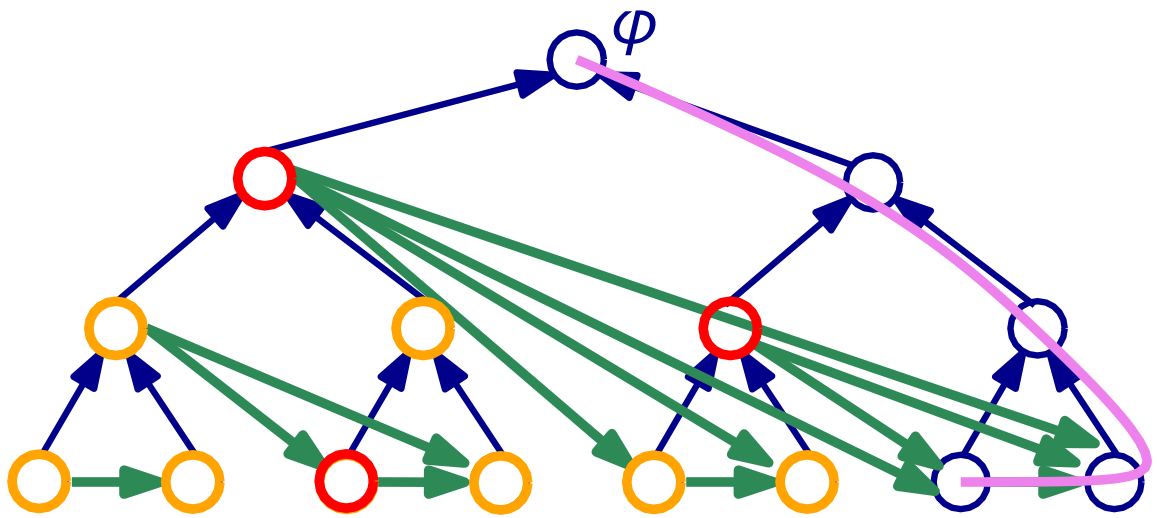
For every leaf i add all edges (j, i) where j is left sibling of node on path $i \rightarrow root$

The New Construction



For every leaf i add all edges (j, i) where j is left sibling of node on path $i \rightarrow root$

Weighted Depth-Robust



weight=8, height 3

weight=4, height 2

weight=2, height 1

weight=1, height 0

Nodes at height h have weight 2^h

#Nodes at height h: 2^{n-h}

Total Weight at Height h: 2^n

Total Weight of all Nodes: $\mathrm{n}2^n$

Weighted Depth-Robust: Let S be any subset of nodes with total weight $\mathrm{wt}(\mathrm{S}) \leq \alpha 2^n$

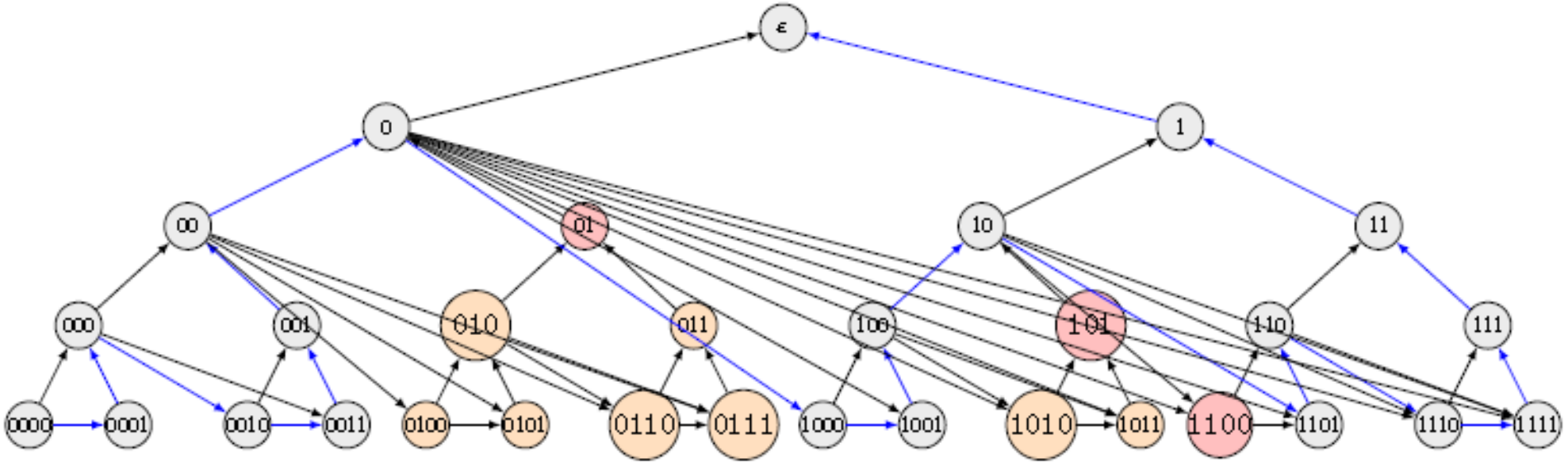
Claim: $G - S$ has a path of length $\mathrm{d} \geq (1-\alpha)2^n$

Intuition: Cannot delete too many nodes close to the root (high weight) Deleting nodes close to the leaf has a small impact on the depth.

Intuition 2: A cheating prover will be caught proportional to the total weight of deleted (inconsistent) nodes

Weighted-Depth-Robustness

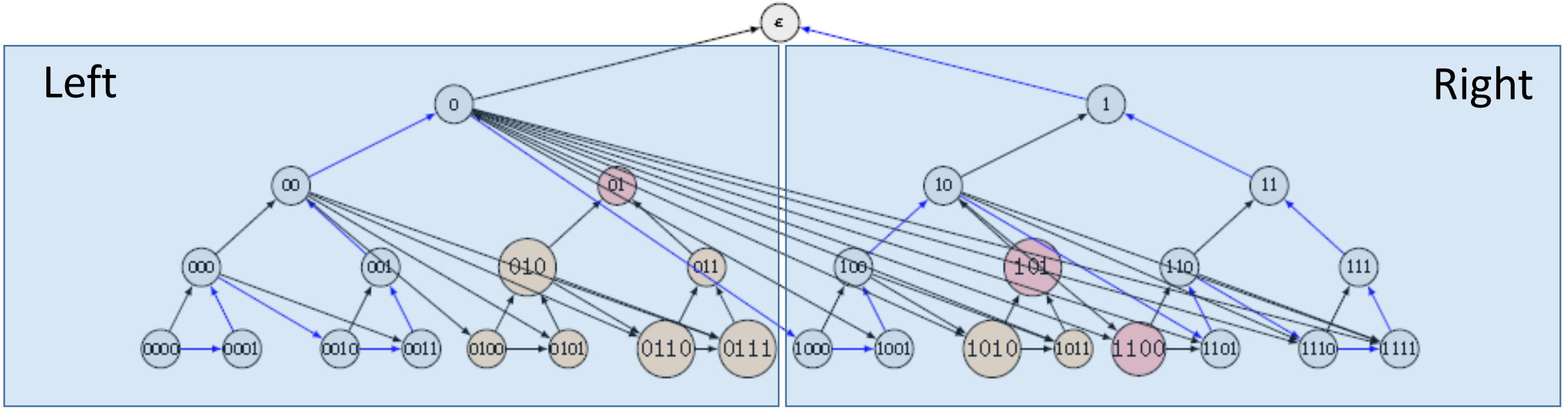
- Suppose we delete **S**. Let D_S be the set of nodes which are in S or below some node in S.
- **Claim:** There is a directed path through all nodes in $V - D_S$



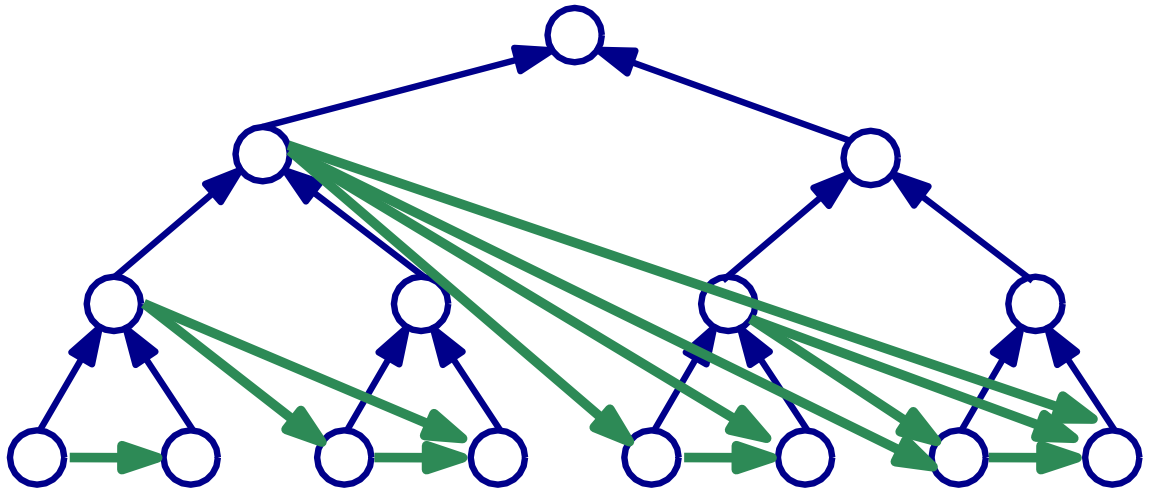
Weighted-Depth-Robustness

- **Claim:** There is a directed path through all nodes in $V - \boldsymbol{D_S}$
- Proof Sketch (Induction on height of tree):
  - By Inductive Assumption there is a path through all nodes on left (same for right)
  - By construction there is a path from left root to every leaf node on right side

  ➔ Can piece paths together (and then connect right root to leaf node)



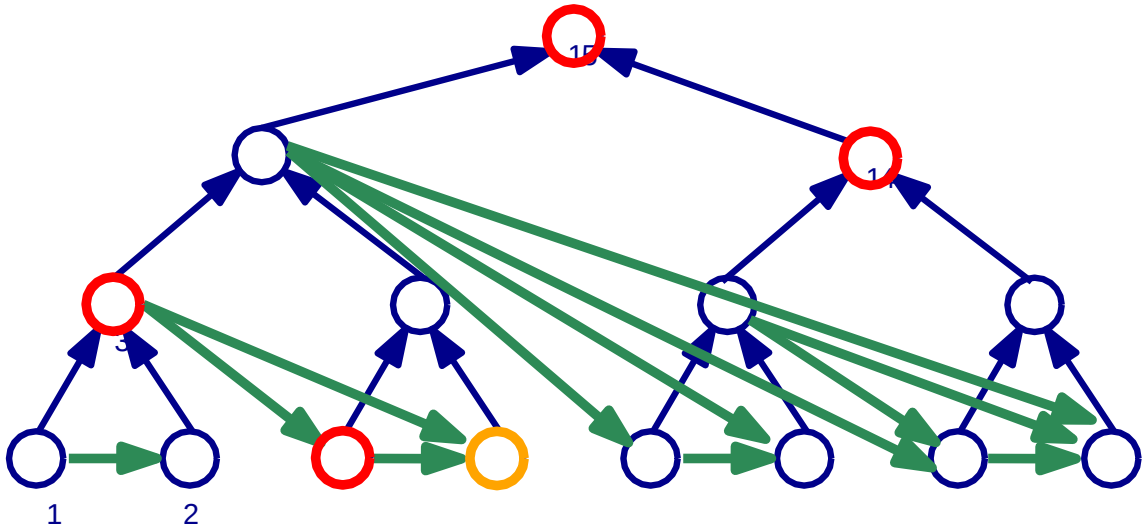
The New Construction



$T = 15$

For every leaf i add all edges (j, i) where j is left sibling of node on path $i \rightarrow root$

The New Construction

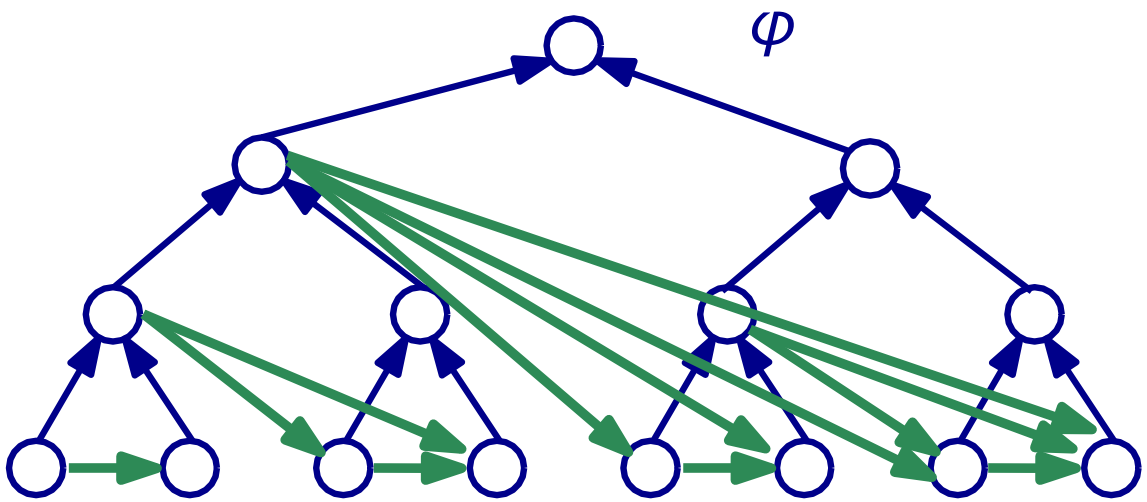


$T = 15$

For every leaf i add all edges (j, i) where j is left sibling of node on path $i \rightarrow root$

- P computes labelling $\pounds_i = \mathrm{H}(\pounds_{parents(i)})$ and sends root label $\varphi = \pounds_T$ to V. Can be done storing only $\log(T)$ labels.
- V challenges P to open a subset of leaves and checks consistency (blue and green edges!)

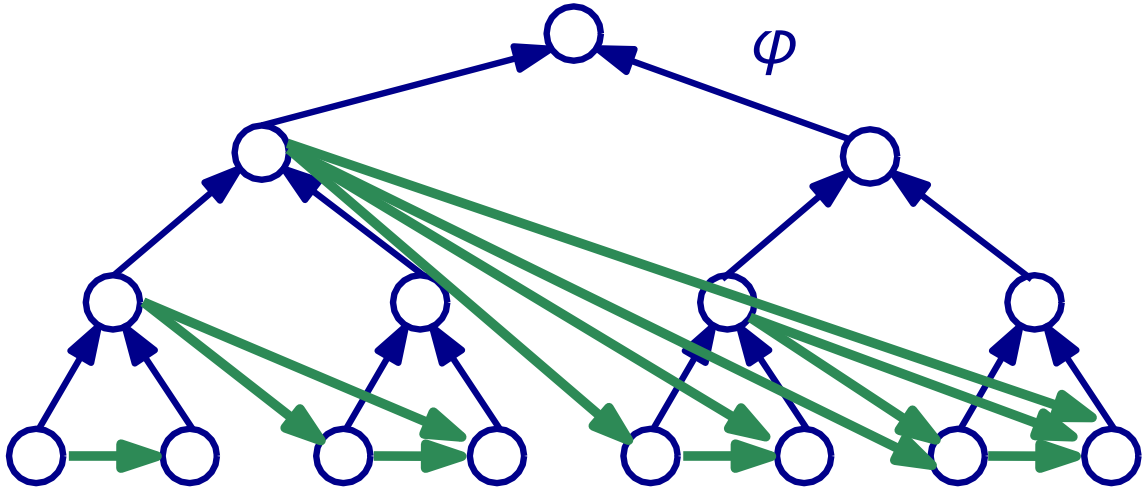
The New Construction



$T = 15$

Proof Sketch

The New Construction

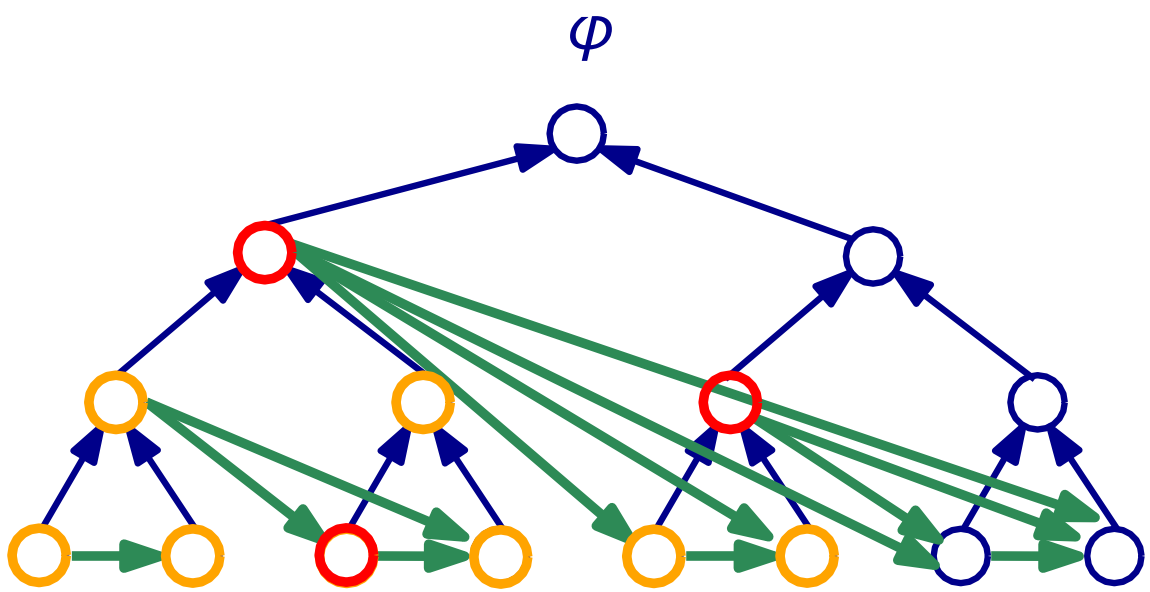


$T = 15$

Proof Sketch

- $\tilde{P}$ committed to all labels L_i after sending $\varphi = L_{15}$.
- i is bad if L_i is not consistent i.e., i's parents are $x_1, \ldots, x_\delta$ *but* $L_i \neq H(L_{x_1}, \ldots, L_{x_\delta})$

The New Construction

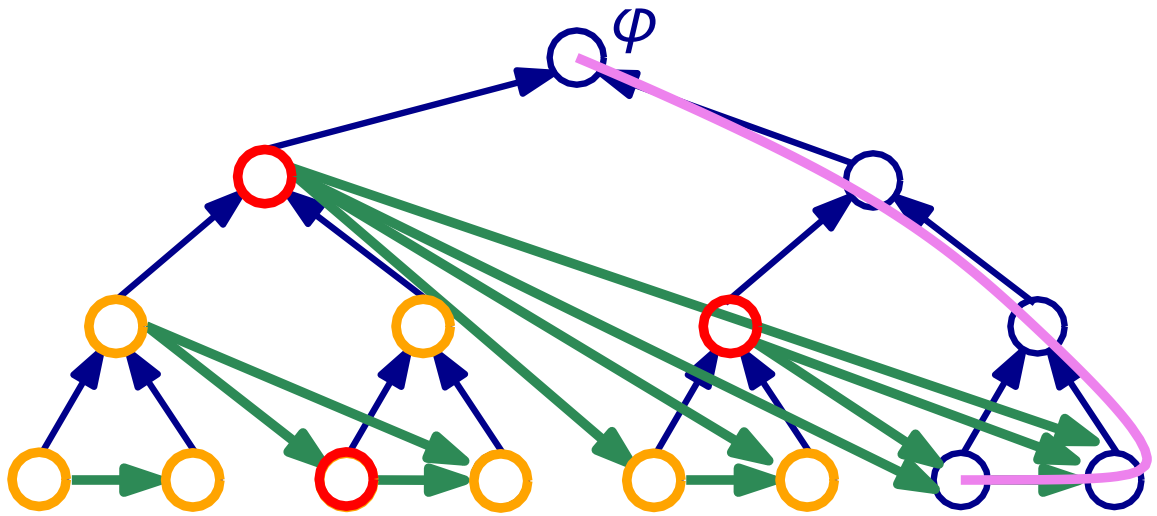


$T = 15$

Proof Sketch

- $\tilde{P}$ committed to labels $\mathcal{L}_i$ after sending $\varphi = \mathcal{L}_{15}$.
- i is bad if L_i is not consistent i.e., i's parents are $x_1, \ldots, x_\delta$ $but\ L_i \neq H(L_{x_1}, \ldots, L_{x_\delta})$
- Let $S \subset V$ denote the bad nodes and all nodes below.

The New Construction



$T = 15$

Proof Sketch

- $\tilde{P}$ committed to labels ℓ_i after sending $\varphi = \ell_{15}$.
- i is bad if $\ell_i \neq \mathrm{H}(\ell_{parents(i)})$.
- Let $S \subset V$ denote the bad nodes and all nodes below.
- Claim 1: $\exists$ path going through $V - S$ (of length $T - |S|$).
- Claim 2: $\tilde{P}$ can't open $|S|/T$ fraction of leafs.

Theorem: $\tilde{P}$ made only $T(1 - c)$ sequential queries
$\Rightarrow$ will pass opening phase with prob. $\leq (1 - c)^{\#\text{of challenges}}$

Wei

Three Problems of the [MMV'13] PoSW

1) **Space Complexity :** Prover needs massive (linear in T) space to compute proof.
2) **Poor/Unclear Parameters** due to usage of sophisticated combinatorial objects.
3) **Uniqueness :** Once an accepting proof is computed, many other valid proofs can be generated (not a problem for time-stamping, but for blockchains).

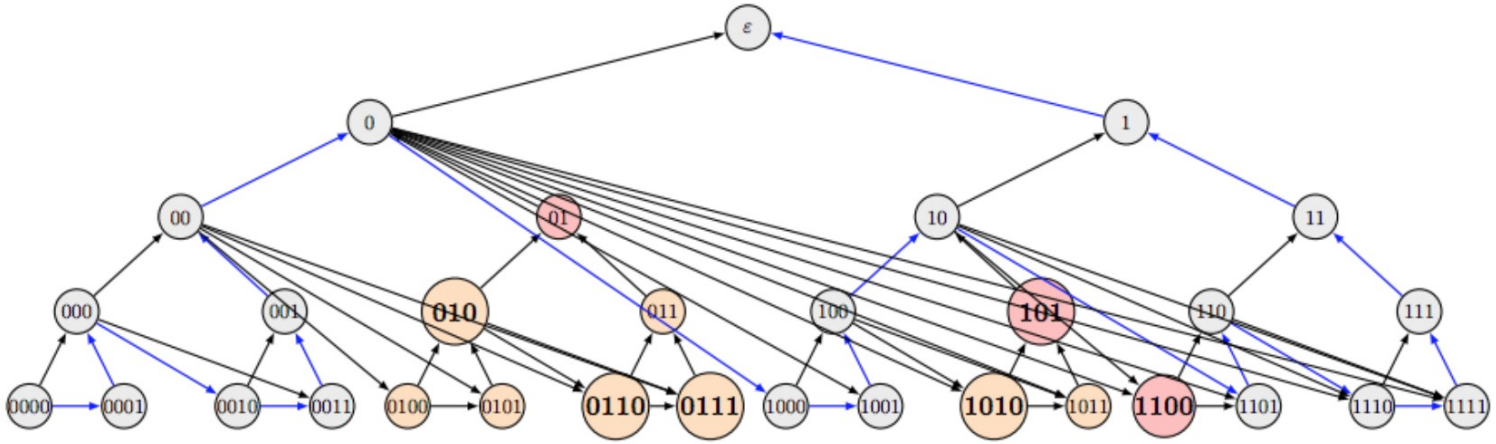
Three Problems of the [MMV'13] PoSW

1) **Space Complexity :** Prover needs massive (linear in T) space to compute proof.
2) **Poor/Unclear Parameters** due to usage of sophisticated combinatorial objects.
3) **Uniqueness :** Once an accepting proof is computed, many other valid proofs can be generated (not a problem for time-stamping, but for blockchains).

New Construction

1) Prover needs only $O(log(T))$ (not $O(T)$) space, e.g. for $T = 2^{42}$ ($\approx$ a day) that's $\approx 10KB$ vs. $\approx 1PB$.
2) Simple construction and proof with good concrete parameters.
3) Awesome open problem!

Construction and Proof Sketch

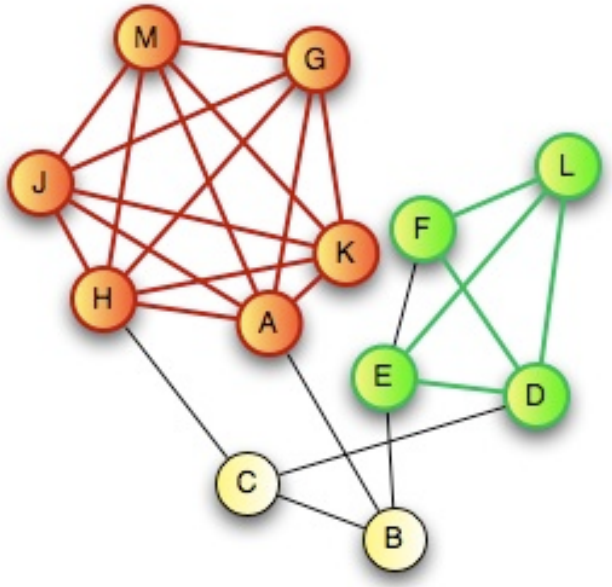


Mining Bitcoin (Proofs of Work)

Mining Bitcoin (Proofs of Work)

Can we have a more "sustainable" Blockchain?

Zero-Knowledge Proof for all NP

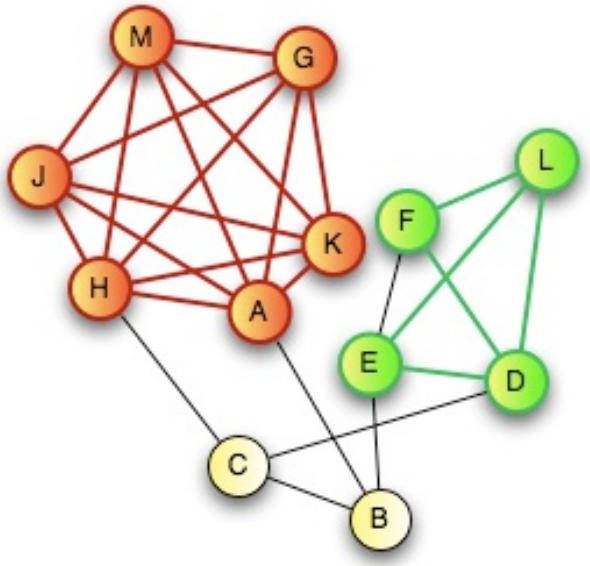


- CLIQUE
  - Input: Graph G=(V,E) and integer k>0
  - Question: Does G have a clique of size k?

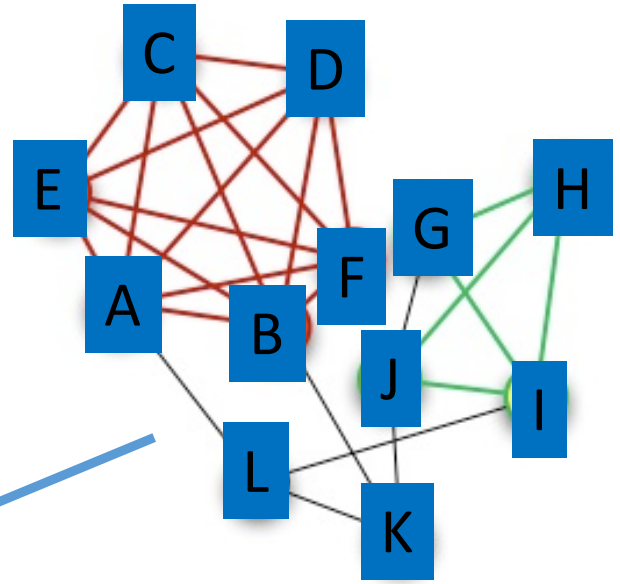
- CLIQUE is NP-Complete
  - Any problem in NP reduces to CLIQUE
  - A zero-knowledge proof for CLIQUE yields proof for all of NP via reduction

- Prover:
  - Knows k vertices $v_1,\ldots,v_k$ in G=(V,E) that form a clique

Zero-Knowledge Proof for all NP



$\sigma(G)$



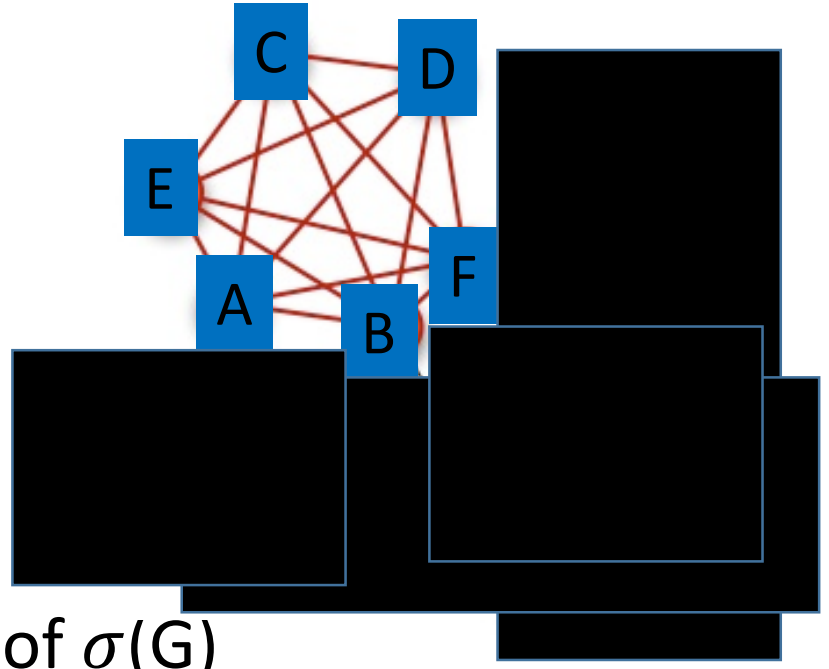
Adjacency matrix $A_{\sigma(G)}$

$$\begin{array}{c} \\ A \\ \\ L \end{array} \begin{array}{c} \begin{array}{ccc} A & \cdots & L \end{array} \\ \begin{pmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{pmatrix} \end{array}$$

Commitment to $A_{\sigma(G)}$

$$\begin{array}{c} \\ A \\ \\ L \end{array} \begin{array}{c} \begin{array}{ccc} A & \cdots & L \end{array} \\ \begin{pmatrix} Com(0, r_{A,A}) & \cdots & Com(1, r_{A,L}) \\ \vdots & \ddots & \vdots \\ Com(1, r_{L,A}) & \cdots & Com(0, r_{L,L}) \end{pmatrix} \end{array}$$

Zero-Knowledge Proof for all NP



- Prover:
  - Knows k vertices $v_1,\ldots,v_k$ in G=(V,E) that for a clique

1. Prover commits to a permutation σ over V
2. Prover commits to the adjacency matrix $A_{\sigma(G)}$ of σ(G)
3. Verifier sends challenge c (either 1 or 0)
4. If c=0 then prover reveals σ and adjacency matrix $A_{\sigma(G)}$
   1. Verifier confirms that adjacency matrix is correct for σ(G)
5. If c=1 then prover reveals the submatrix formed by first rows/columns of $A_{\sigma(G)}$ corresponding to $\sigma(v_1), \ldots, \sigma(v_k)$
   1. Verifier confirms that the submatrix forms a clique.

Zero-Knowledge Proof for all NP

- **Completeness**: Honest prover can always make honest verifier accept
- **Soundness**: If prover commits to adjacency matrix $A_{\sigma(G)}$ of σ(G) and can reveal a clique in submatrix of $A_{\sigma(G)}$ then G itself contains a k-clique. Proof invokes binding property of commitment scheme.
- **Zero-Knowledge:** Simulator cheats and either commits to wrong adjacency matrix or cannot reveal clique. Repeat until we produce a successful transcript. Indistinguishability of transcripts follows from hiding property of commitment scheme.

NIZK Security (Random Oracle Model)

- Simulator is given statement to prove (e.g., G is 3-COLORABLE)
- Simulator must output a proof π'_z and a random oracle H'

- Distinguisher D
  - World 1 (Simulated): Given z, π'_z and oracle access to H'
  - World 2 (Honest): Given z, π_z (honest proof) and oracle access to H
  - Advantage: $\mathrm{ADV}_{\mathrm{D}} = |Pr[D^{H}(\mathrm{z}, \pi_z) = 1] - Pr[D^{H'}(\mathrm{z}, \pi'_z) = 1]|$
- **Zero-Knowledge:** Any PPT distinguisher D should have negligible advantage.
- NIZK proof π_z is transferrable (contrast with interactive ZK proof)

Σ-Protocols

- Prover Input: instance/claim x and witness w
- Verifier Input: Instance x
- Σ-Protocols: three-message structure
  - Prover sends first message $m=P_1(x,w; r_1)$
  - Verifier responds with random challenge c
  - Prover sends response $R=P_2(x,w,r_1,c; r_2)$
  - Verifier outputs decision V(x,m,c,R)
  - **Completeness:** If w is a valid witness for instance x then Pr[V(x,c,R)=1]=1
  - **Soundness:** If the claim x is false then V(x,c,R)=0 with probability at least ½
  - **Zero-Knowledge:** Simulator can produce computationally indistinguishable transcript

Σ-Protocols and Fiat-Shamir Transform

- Convert Σ-Protocols into Non-Interactive ZK Proof
- Prover Input: instance/claim x and witness w
- Verifier Input: Instance x
- **Step 1:** Prover generates first messages for n instances of the protocol
  - $m_i = P_1(x,w; r_i)$ for each i=1 to n
- **Step 2:** Prover uses random oracle to extract random coins $z_j=H(x,j, m_1,\ldots,m_n)$ for j=1 to n
  - Prover samples challenges $c_1,\ldots,c_n$ using random strings $z_1,\ldots,z_n$ i.e., c_i=SampleChallenge(z_i)
- **Step 3:** Prover computes responses $R_1,\ldots,R_n$
  - $R_i \leftarrow P_2(x,w,r_i,c_i)$
- **Step 4:** Prover outputs the proof $\{(m_i, c_i, z_i)\}_{i\leq n}$

Σ-Protocols and Fiat-Shamir Transform

- **Step 1:** Prover generates first messages for n instances of the protocol
  - $m_i = P_1(x,w; r_i)$ for each i=1 to n
- **Step 2:** Prover uses random oracle to extract random coins $z_i=H(x,i, m_1,....,m_n)$ for i=1 to n
  - Prover samples challenges $c_1,...,c_n$ using random strings $z_1,...,z_n$ i.e., c_i=SampleChallenge(z_i)
- **Step 3:** Prover computes responses $R_1,...,R_n$
  - $R_i \leftarrow P_2(x,w,r_i,c_i)$
- **Step 4:** Prover outputs the proof $\pi = \{(m_i, c_i, R_i)\}_{i \leq n}$

Verifier: $V_{NI}(x,\pi)$ check that for all $i \leq n$

1. **$V(x, (m_i, c_i, R_i))=1$ and**
2. c_i=SampleChallenge(z_i) **where** $z_i=H(x,i, m_1,....,m_n)$

Σ-Protocols and Fiat-Shamir Transform

- **Step 1:** Prover generates first messages for n instances of the protocol
  - $m_i = P_1(x,w; r_i)$ for each i=1 to n
- **Step 2:** Prover uses random oracle to extract random coins $z_i=H(x,i, m_1,\ldots,m_n)$ for i=1 to n
  - Prover samples challenges $c_1,\ldots,c_n$ using random strings $z_1,\ldots,z_n$ i.e., $c_i=\text{SampleChallenge}(z_i)$
- **Step 3:** Prover computes responses $R_1,\ldots,R_n$
  - $R_i \leftarrow P_2(x,w,r_i,c_i)$
- **Step 4:** Prover outputs the proof $\pi = \{(m_i, c_i, R_i)\}_{i\leq n}$

Zero-Knowledge (Idea):

Step 1: Run simulator for Σ n-times to obtain n transcripts (m_i, c_i, R_i) for each $i \leq n$.

Step 2: Program the random oracle so that $H(x,i, m_1,\ldots,m_n)=z_i$ where $c_i=\text{SampleChallenge}(z_i)$

Non-Interactive Proof of Sequential Work

- **Key Idea:** Apply Fiat-Shamir Transform!
- **Interactive Verifier:** Picks uniformly random challenge nodes $c_1, \ldots, c_k$
- **Non Interactive Version:** Let h_x denote the root of the Merkle-Tree output by the prover. Define $c_i = H(i, h_x)$. Non-Interactive Proof includes root h_x and responses $r_1, \ldots, r_k$
- **Non Interactive Verifier:** generates the challenges $c_i = H(i, h_x)$ and verifies the responses $r_1, \ldots, r_k$

- Security Analysis (sketch): If the attacker makes q RO queries over at most T' $< N(1-\varepsilon)$ sequential rounds then s/he finds a valid PoSW probability at most

$$\mathrm{q}(1-\varepsilon)^k + \frac{2\,\lambda\,q^2 \log N}{2^\lambda}$$

Probability of finding ``lucky" challenges

Probability of finding an H-sequence longer than T'

Verifiable Delay Functions

Dan Boneh, Joe Bonneau,

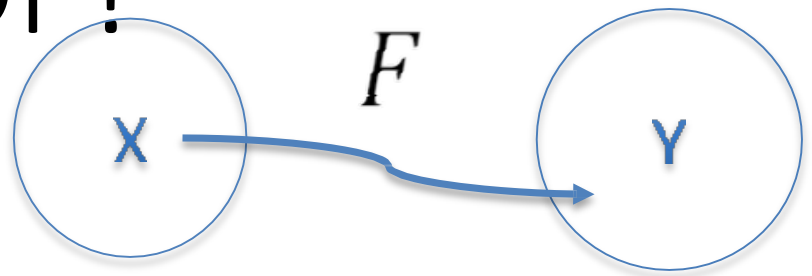
Benedikt Bünz, Ben Fisch

Crypto 2018

What is a VDF?

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

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



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 ***π***
  - PRAM runtime *T* with polylog(T) processors
- **Verify(*pp*, *x*, *y*, *π*)** $\longrightarrow$ { *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

RSW Timelock Puzzle (Repeated Squaring)

Puzzle Generation: $N = pq$ *and sends puzzle* $Z = (N, H(f(x)) \oplus secret)$

Trapdoor: p, q *must not be known to others*

$$f(x) = x^{2^T} mod\ N$$

- **Goals:**
  - Puzzle can be generated quickly in time $O(\text{polylog}\, T)$.
  - Other parties can recover secret in sequential time $\Omega(T)$.
  - Secret is hidden from (massively parallel) attackers running in sequential time $o(T)$.
- **Assumptions:** Factoring N is hard and (without prime factors) it takes sequential time $\Omega(T)$ to compute $f(x) = x^{2^T} mod\ N$

RSW Timelock Puzzle (Repeated Squaring)

Puzzle Generation: $N = pq$ *and sends puzzle* $Z = (N, H(f(x)) \oplus secret)$
Trapdoor: p, q *must not be known to prover*

$$f(x) = x^{2^T} mod\ N$$

Computing $f(x)$ (Puzzle Solver):

$x_0 = x$ // $x_0 = x^{2^0} mod\ N$

for i=1 to T

$x_i = x_{i-1} * x_{i-1} mod\ N$ // $x_i = x^{2^{i-1}} x^{2^{i-1}} mod\ N = x^{2^i}\ mod\ N$

output x_T

RSW Timelock Puzzle(Repeated Squaring)

Puzzle Generation: $N = pq$ *and sends puzzle* $Z = (N, H(f(x)) \oplus secret)$
Trapdoor: p, q *must not be known to prover*

$$f(x) = x^{2^T} mod\ N$$

Computing $f(x)$ (Puzzle Solver):

$z = x$ // $z = x^{2^0} mod\ N$
for i=1 to T
$z = z * z\ mod\ N$ // $z = x^{2^i}\ mod\ N$
output z

RSW Timelock Puzzle (Repeated Squaring)

Puzzle Generation: $N = pq$ *and sends puzzle* $Z = (N, H(f(x)) \oplus secret)$

Trapdoor: p, q *must not be known to prover*

$$f(x) = x^{2^T} mod\ N$$

Computing $f(x)$ with Trapdoor (Puzzle Generation):

Compute $\varphi(N) = (p-1)(q-1)$ and $y = 2^T\ mod\ \varphi(N)$

output $x^y\ mod\ N$

$O(\log N) \ll T$ multiplication queries mod N

RSW Timelock Puzzle (Repeated Squaring)

- **Assumptions:** Factoring N is hard and (without prime factors) it takes time $\Omega(T)$ to compute $f(x) = x^{2^T} mod\ N$

- Is this a Verifiable Delay Function?

- **Answer:** Not publicly verifiable!
  - Verifier who does not have prime factors (p,q) has to re-compute

$$f(x) = x^{2^T}\ mod\ N$$

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) < $\sigma(T)$,

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

VDF security more formally…

Sequentiality Game

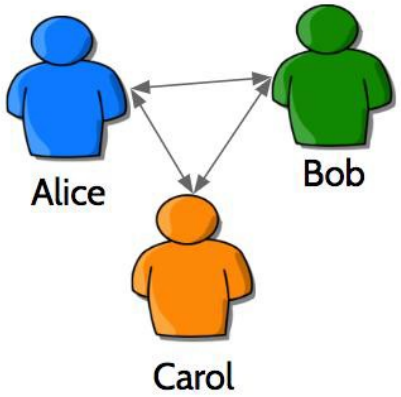
$pp \leftarrow Setup(\lambda, T)$ //sample setup params
$L \leftarrow A_0(pp, T)$ //adversary preprocesses params
$x \leftarrow X$ //choose a random challenge input x
$y_A \leftarrow A_1(L, pp, x)$ //adversary computes output y

$A = (A_0, A_1)$ "**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

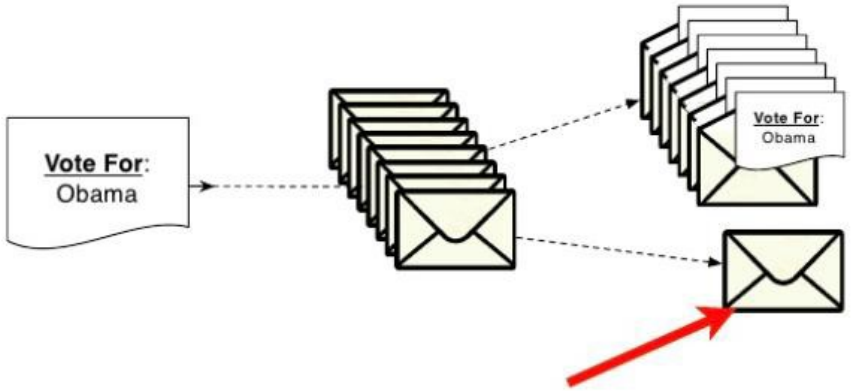
- Rabin ‘83

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

Many uses for random beacons

Games

Lotteries



Cryptographic proofs

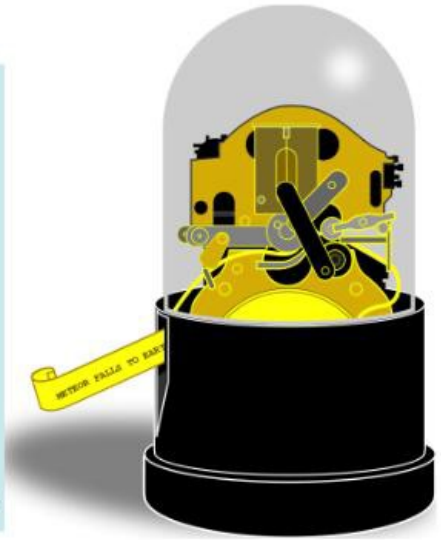
Leader election

Randomness beacon

``Public displays”
are easily corrupted

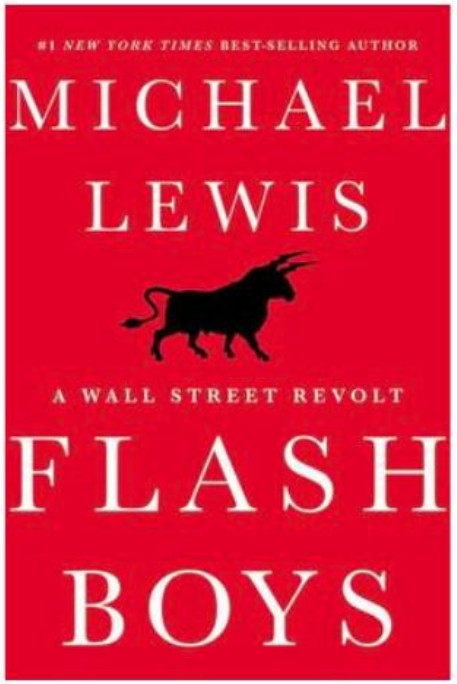
Public entropy source

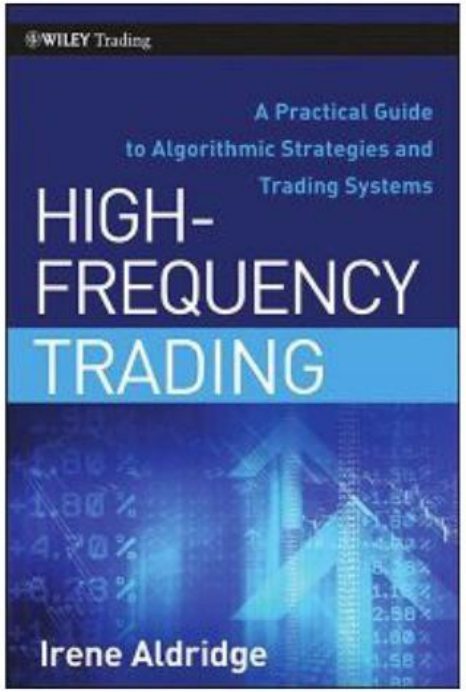
Stock prices [Clark, Hengartner 2010]

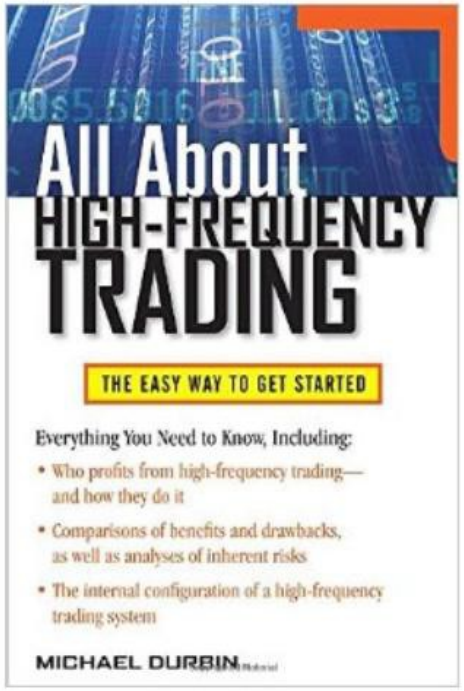


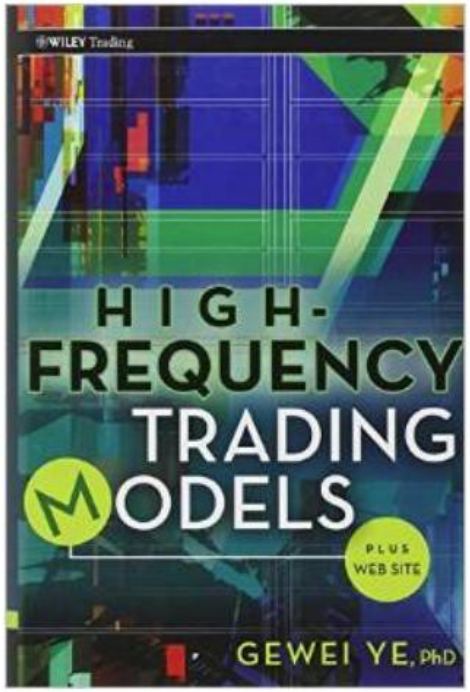
<u>Assumption</u>: (1) unpredictable, (2) adversary cannot fix stock prices

Stock price manipulation









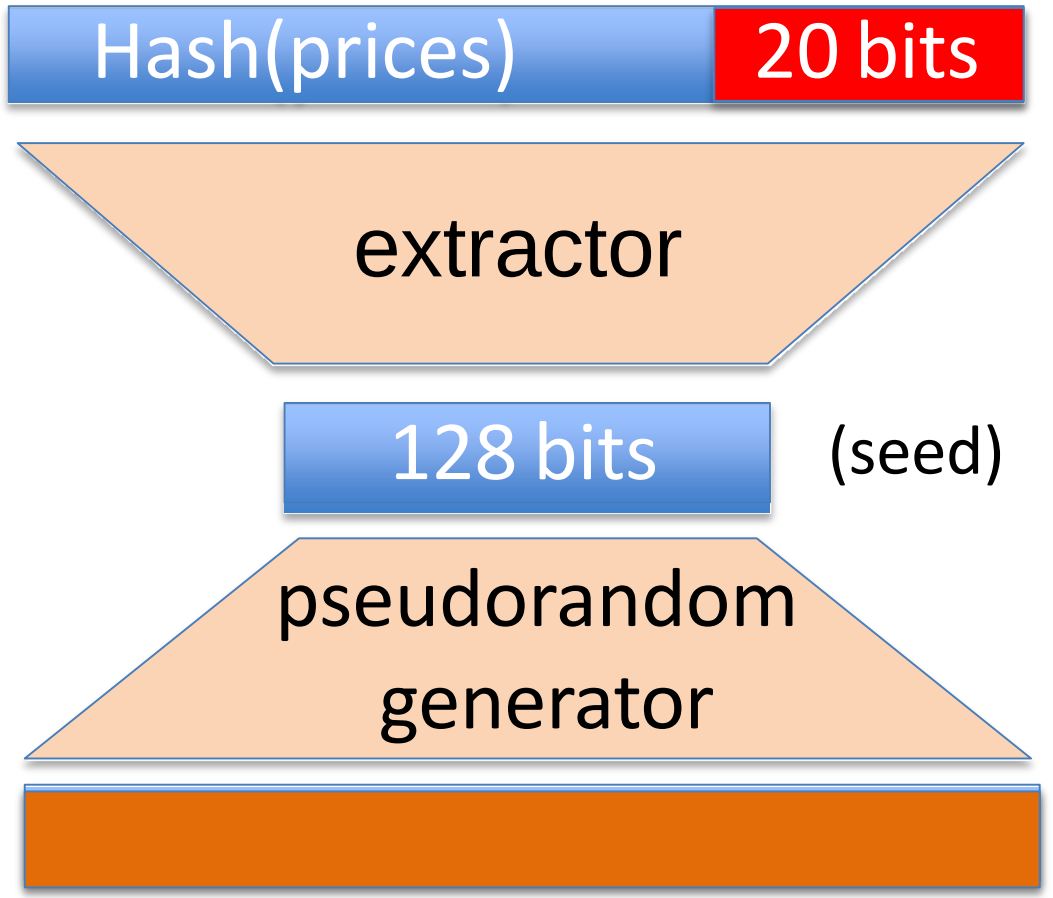
Stock price randomness beacon

Closing prices of 100 stocks:

<u>The problem</u>:

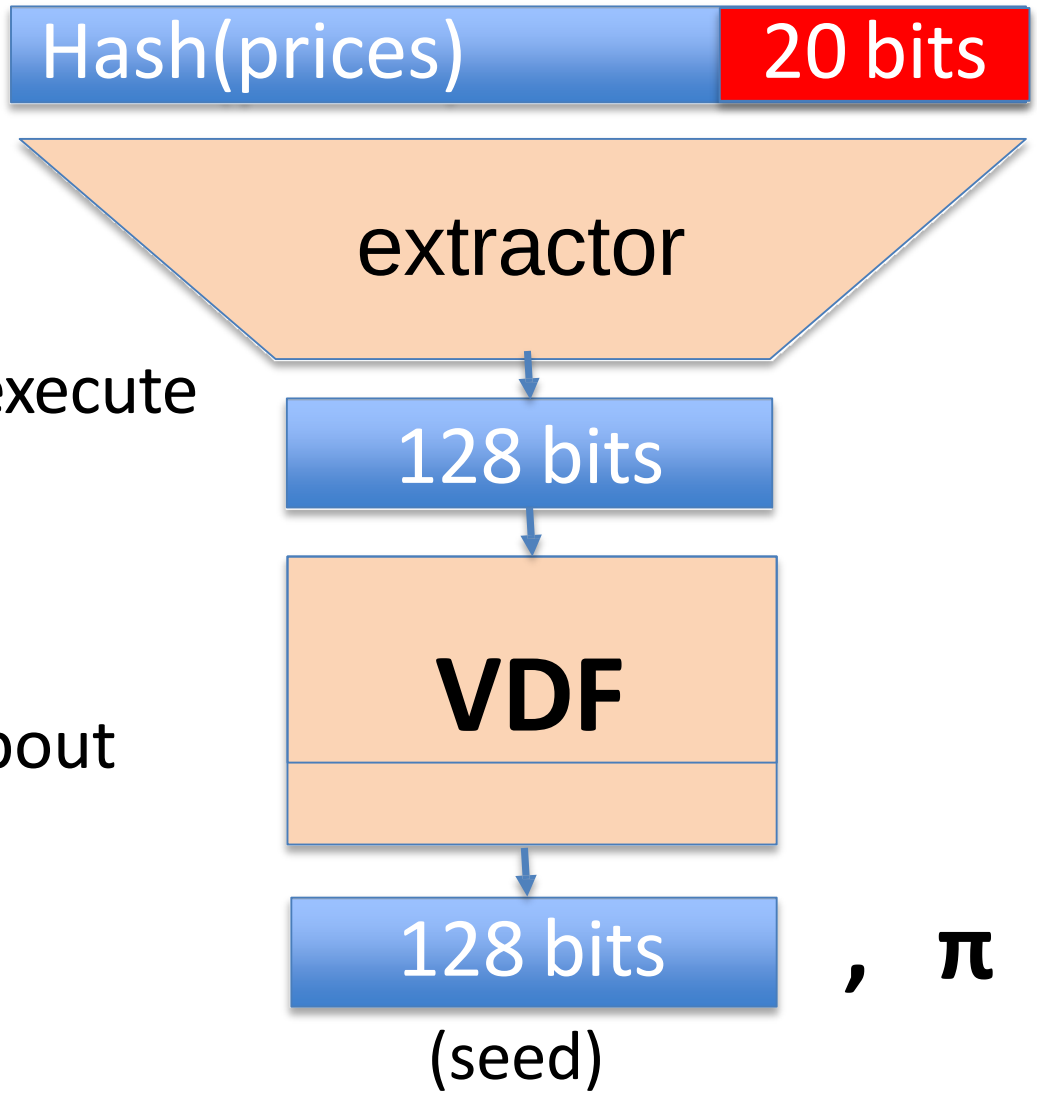
- Once prices settle a minute before closing, attacker executes 20 last-minute trades to influence seed.

- Attacker can **predict** outcome of trades and choose favorable trades to **bias** result

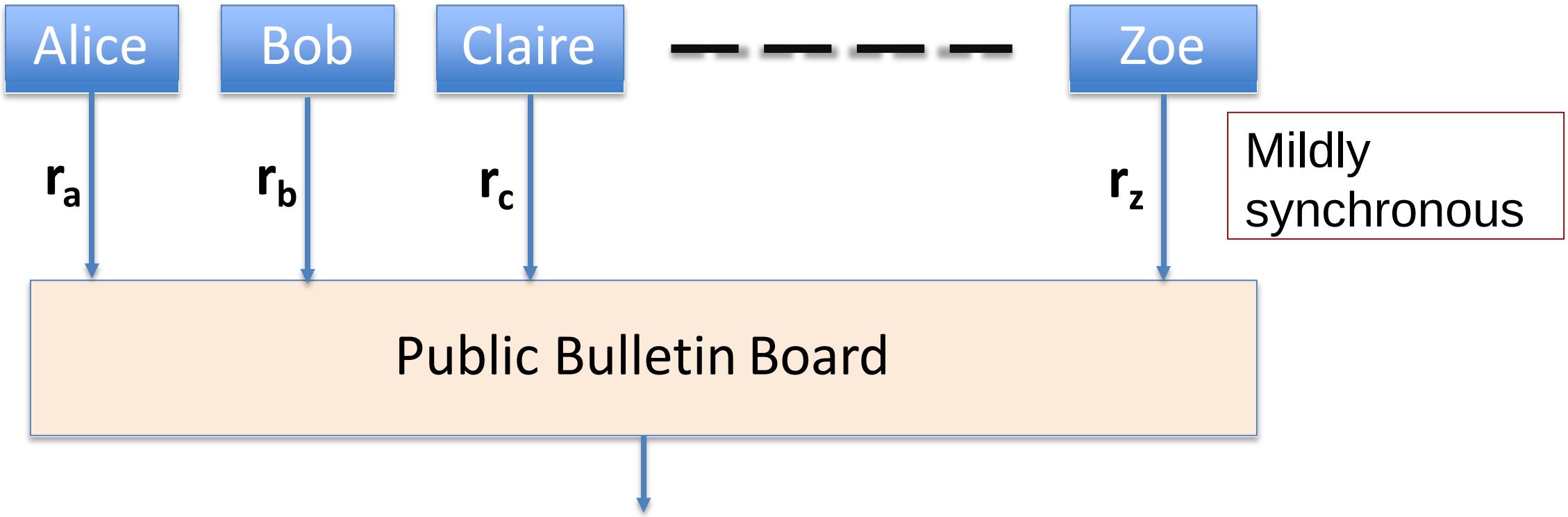


Solution: slow things down with a VDF

- A solution: **one hour VDF**
- Attacker cannot tell what trades to execute before market closes
- Uniqueness: ensures no ambiguity about output



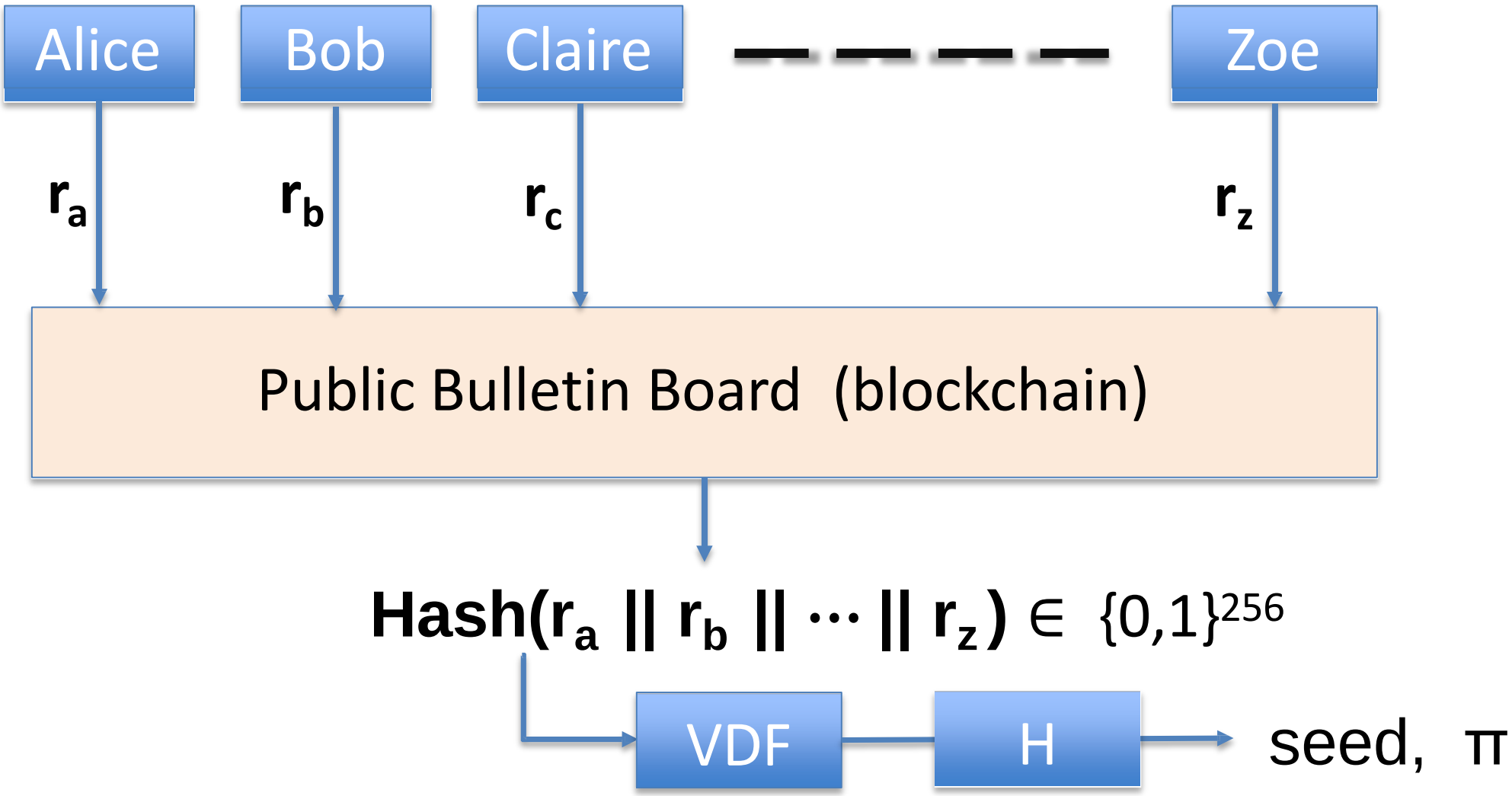
Simple Bulletin Board



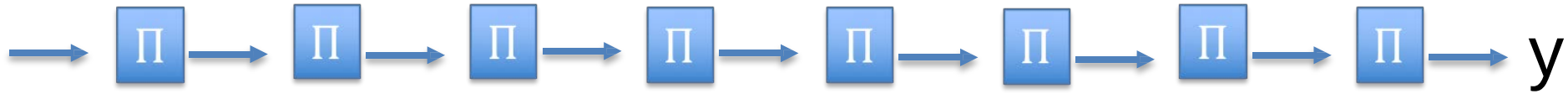
output $\mathbf{seed = Hash(r_a \,||\, r_b \,||\, \cdots \,||\, r_z)} \in \{0,1\}^{256}$

Problem: Zoe controls the final seed !!

Solution: slow things down with a VDF



Part II: Constructions

I. 

(reverse permutation)

SNARK/STARK proof π

This work

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

π = {proof of correct exponentiation}

Assumption: the group G has unknown size

Followup:
Pietrzak'18, Wesolowski'18

Hash Chain w/ Verifiable Computation

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

- 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

$$\mathbf{x} \rightarrow H(x) \rightarrow H\big(H(x)\big) \rightarrow \cdots \rightarrow H^{(t)}(x) = y$$

$$\pi$$

Problem

- Proof generation slower than hash chain, without massive parallelism

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, H: X ⟶ 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

RSW Timelock Puzzle (Repeated Squaring)

Puzzle Generation: $N = pq$ *and sends puzzle* $Z = (N, H(f(x)) \oplus secret)$

Trapdoor: p, q *must not be known to others*

$$f(x) = x^{2^T} mod\ N$$

- **Goals:**
  - Puzzle can be generated quickly in time $O(\text{polylog}\, T)$.
  - Other parties can recover secret in sequential time $\Omega(T)$.
  - Secret is hidden from (massively parallel) attackers running in sequential time $o(T)$.
- **Assumptions:** Factoring N is hard and (without prime factors) it takes sequential time $\Omega(T)$ to compute $f(x) = x^{2^T} mod\ N$

RSW Timelock Puzzle (Repeated Squaring)

Puzzle Generation: $N = pq$ *and sends puzzle* $Z = (N, H(f(x)) \oplus secret)$
Trapdoor: p, q *must not be known to prover*

$$f(x) = x^{2^T} mod\ N$$

Computing $f(x)$ (Puzzle Solver):

$x_0 = x$ // $x_0 = x^{2^0} mod\ N$

for i=1 to T

$x_i = x_{i-1} * x_{i-1} mod\ N$ // $x_i = x^{2^{i-1}} x^{2^{i-1}} mod\ N = x^{2^i}\ mod\ N$

output x_T

RSW Timelock Puzzle(Repeated Squaring)

Puzzle Generation: $N = pq$ *and sends puzzle* $Z = (N, H(f(x)) \oplus secret)$
Trapdoor: p, q *must not be known to prover*

$$f(x) = x^{2^T} mod\ N$$

Computing $f(x)$ (Puzzle Solver):

$z = x$ // $z = x^{2^0} mod\ N$
for i=1 to T
$z = z * z\ mod\ N$ // $z = x^{2^i}\ mod\ N$
output z

RSW Timelock Puzzle (Repeated Squaring)

Puzzle Generation: $N = pq$ *and sends puzzle* $Z = (N, H(f(x)) \oplus secret)$

Trapdoor: p, q *must not be known to prover*

$$f(x) = x^{2^T} mod\ N$$

Computing $f(x)$ with Trapdoor (Puzzle Generation):

Compute $\varphi(N) = (p-1)(q-1)$ and $y = 2^T\ mod\ \varphi(N)$

output $x^y\ mod\ N$

$O(\log N) \ll T$ multiplication queries mod N

RSW Timelock Puzzle (Repeated Squaring)

- **Assumptions:** Factoring N is hard and (without prime factors) it takes time $\Omega(T)$ to compute $f(x) = x^{2^T} mod\ N$

- Is this a Verifiable Delay Function?

- **Answer:** Not publicly verifiable!
  - Verifier who does not have prime factors (p,q) has to re-compute

$$f(x) = x^{2^T}\ mod\ N$$

Wesolowski's VDF Construction

- **Public Parameter:** $N = pq$ (Generated by trusted party/MPC)
- **Trapdoor Discarded:** No one knows p, q

$$f(x) = x^{2^T} mod\ N$$

- **Prover:** Computes y $= f(x) = x^{2^T} mod\ N$ and sends y to verifier
- **Verifier:** picks random prime B
- **Prover:** Computes $\pi = x^{\left\lfloor \frac{2^T}{B} \right\rfloor} mod\ N$ and sends π to the verifier.
- **Verifier:** Checks that $y = \pi^B x^{2^T mod\ B} mod\ N$
- **Soundness:** For any number B we have $2^T = (2^T mod\ B) + B \left\lfloor \frac{2^T}{B} \right\rfloor$

Wesolowski's VDF Construction

- **Prover:** Computes y $= f(x) = x^{2^T} mod\ N$ and sends y to verifier
- **Verifier:** picks random prime B
- **Prover:** Computes $\pi = x^{\left\lfloor \frac{2^T}{B} \right\rfloor} mod\ N$ and sends π to the verifier.
- **Verifier:** Checks that $y = \pi^B x^{2^T mod\ B} mod\ N$
- **Completeness:** For any number B we have $2^T = (2^T mod\ B) + B \left\lfloor \frac{2^T}{B} \right\rfloor$

$$\pi^B x^{2^T mod\ B} = x^{B\left\lfloor \frac{2^T}{B} \right\rfloor + (2^T mod\ B)} mod\ N = x^{2^T} mod\ N = f(x)$$

Wesolowski's VDF Construction

- **Prover:** Computes y $= f(x) = x^{2^T} mod\ N$ and sends y to verifier
- **Verifier:** picks random prime B
- **Prover:** Computes $\pi = x^{\left\lfloor \frac{2^T}{B} \right\rfloor} mod\ N$ and sends π to the verifier.
- **Verifier:** Checks that $y = \pi^B x^{2^T mod\ B} mod\ N$
- **Soundness:**
  - **Assumption** for any $z \neq 1$ it is hard to find y such that $y^B = z\ mod\ N$ when B is random.

Wesolowski's VDF Construction (Non-Interactive VDF)

- **Prover:**
  - Computes y $= f(x) = x^{2^T} mod\ N$ and sends y to verifier
  - **Sets random coins** $R = H\big(f(x)\big)$ $\quad B = \mathrm{GenPrime}(R)$
  - Computes $\pi = x^{\left\lfloor \frac{2^T}{B} \right\rfloor} mod\ N$
  - Output $(x, y = f(x), \pi)$
- **Verifier:**
  - Compute $B = \mathrm{GenPrime}(H(y))$
  - Checks that $y = \pi^B x^{2^T mod\ B} mod\ N$

Wesolowski's VDF Construction (Non-Interactive VDF)

- **Verifier:**
  - Compute $B = \mathrm{GenPrime}(H(y))$
  - Checks that $\mathrm{y} = \pi^{B} x^{2^{T} mod\ B} mod\ N$
- **Efficiency:**
  - **Proof Size:** π is very short (just O(log N) bits) ☺
  - **Prover Efficiency:** extra $\mathbf{O}(T)$ multiplications ☹
    - $\mathbf{O}(T/\log T)$ multiplications 😐
  - **Verifier Efficiency:** $\mathbf{O}(\log T)$ multiplications ☺

Pietrzak's Construction [ITCS19]

- **Safe Prime:** prime $p = 2p' + 1$ such that p' is also prime
- Assume $N = pq$ where $p = 2p' + 1$ and $q = 2q' + 1$
- **Quadratic Residues:** $QR_N = \{z^2 \ mod \ N | z \in \mathbb{Z}_N^*\}$
- **Signed Quadratic Residues**
  - *Represent elements of* $\mathbb{Z}_N^*$ *as* $\left\{-\frac{N-1}{2}, \ldots, \frac{N-1}{2}\right\}$
  - $QR_N^+ = \{|x| \mid x \in QR_N\}$
  - **Fact:** *The map* $|.|$ *is an isomorphism from* QR_N *to* QR_N^+
  - **Fact:** QR_N^+ *is a cyclic group with operation* $\circ$ *defined as*
    $$a \circ b := |ab \ mod \ N|$$
  - **Redefine Notation:** $x \in QR_N^+$ ➔ $x^{i+1} := x \circ x^i$ *e.g.,* $x^2 := x \circ x$

Pietrzak's Construction [ITCS19]

- **Safe Prime:** prime $p = 2p' + 1$ such that p' is also prime
- Assume $N = pq$ where $p = 2p' + 1$ and $q = 2q' + 1$
- **Quadratic Residues:** $QR_N = \{z^2 | z \in \mathbb{Z}_N^*\}$
- **Signed Quadratic Residues**
  - *Represent elements of* $\mathbb{Z}_N^*$ *as* $\left\{-\frac{N-1}{2}, \dots, \frac{N-1}{2}\right\}$
  - $QR_N^+ = \{|x| \mid x \in QR_N\}$
  - **Fact:** *The map* $|.|$ *is an isomorphism from* QR_N *to* QR_N^+
  - **Fact:** QR_N^+ *is a cyclic group with operation* $\circ$ *defined as*
    $$a \circ b := |ab \bmod N|$$
  - **Fact:** Membership in QR_N^+ can be efficiently tested (unlike QR_N)
  - **Fact:** If primes p and q are safe then QR_N^+ (and QR_N) has not sub-group of small order.

Pietrzak's Construction [ITCS19]

- **Prover:** Given $x \in QR_N^+, N, T$
  - Computes $\mathrm{f(x)} = x^{2^T}$ (repeated squaring with operation $y^2 := y \circ y$)
- **Question:** Does repeated squaring assumption change now that we use QR_N^+?
- **Answer:** Not significantly…
- **Observation 1:** $|QR_N| \geq \frac{|\mathbb{Z}_N^*|}{4}$ so a random element in $\mathbb{Z}_N^*$ is in QR_N with probability at least $\frac{1}{4}$

  ➔ An algorithm that can compute $\mathrm{f(x)} = x^{2^T} mod\ N$ correctly with probability at least ε (over the selection of x in QR_N) the same algorithm computes $\mathrm{f(x)} = x^{2^T} mod\ N$ correctly with probability at least $\frac{\varepsilon}{4}$

Pietrzak's Construction [ITCS19]

- **Prover:** Given $x \in QR_N^+, N, T$
  - Computes $\mathrm{f(x)} = x^{2^T}$ (repeated squaring with operation $y^2 \coloneqq y \circ y$)
- Does repeated squaring assumption change now that we use QR_N^+?

- **Observation 2: Computing over ($QR_N^+, \circ$) is not significantly easier than ($QR_N, \times$)**
  - Suppose x in QR_N and $\mathrm{y} = x^{2^T} mod\ N$
  - Let $\mathrm{y}' = |\mathrm{y}|$ and $x' = |x|$ be the corresponding group elements in QR_N^+
  - We have $y' = x'^{2^T}$
  - We have $y \in \{y', N - y'\}$
  - Flip a coin and output y' or $N - y'$ (correct with probability $\frac{1}{2}$)

Pietrzak's Construction [ITCS19]

- **Prover:** Given $x \in QR_N^+, N, T$
  - Computes $\mathrm{f(x)} = x^{2^T}$ (repeated squaring with operation $y^2 := y \circ y$)
- Does repeated squaring assumption change now that we use QR_N^+?

- **Observation 2: Computing over (QR_N^+, $\circ$) is not significantly easier than (QR_N , $\times$)**
  - Suppose x in QR_N and $\mathrm{y} = x^{2^T} mod\ N$
  - Let $\mathrm{y}' = |\mathrm{y}|$ and $x' = |x|$ be the corresponding group elements in QR_N^+
  - We have $y' = x'^{2^T}$
  - We have $y \in \{y', N - y'\}$
  - Flip a coin and output y' or $N - y'$ (correct with probability $\frac{1}{2}$)
  - An algorithm that computes $\mathrm{f(x)} = x^{2^T}$ over QR_N^+ with probability δ (over the random choice of x in QR_N^+) yields equally efficient (essentially) algorithm which computes $x^{2^T} mod\ N$ with probability $\frac{\delta}{2}$ (over the random choice of x in QR_N)

Pietrzak's Construction [ITCS19]

- **Prover:** Given $x \in QR_N^+, N, T$
  - Computes $\mathrm{f(x)} = x^{2^T}$ (repeated squaring with operation $y^2 := y \circ y$)
- Does repeated squaring assumption change now that we use QR_N^+?

- **Observation 2: Computing over (QR_N^+, $\circ$) is not significantly easier than (QR_N , $\times$)**
  - An algorithm that computes $\mathrm{f(x)} = x^{2^T}$ over QR_N^+ with probability δ (over the random choice of x in QR_N^+) yields equally efficient (essentially) algorithm which computes $x^{2^T} mod\ N$ with probability $\frac{\delta}{2}$ (over the random choice of x in QR_N)
- **Combining Observations:** An algorithm that computes computes $\mathrm{f(x)} = x^{2^T}$ over QR_N^+ with probability δ (over the random choice of x in QR_N^+) yields equally efficient (essentially) algorithm which computes $x^{2^T} mod\ N$ with probability $\frac{\delta}{8}$ (over the random choice of x in $\mathbb{Z}_N^*$)

Pietrzak's Construction [ITCS19]

- HalvingProtocol(N,x,T,y) // Honest Prover: $y = x^{2^T}$
  - If T=1 then Verifier outputs **accept** if $x \circ x = y$; otherwise **reject**
  - Prover sends $\mu = x^{2^{T/2}}$ to verifier
  - If $\mu \notin QR_N^+$ then verifier outputs **reject;** otherwise verifier picks a random integer $r \in \mathbb{Z}_{2^\lambda}$ and sends it to the prover
  - Sender/Prover compute $x' := x^r \circ \mu \qquad (= x^{r+2^{T/2}})$
  - The sender/prover compute $y' = \mu^r \circ y \qquad (= x^{r2^{T/2}+2^T})$
    - If prover is honest then $x' = x^{r+2^{T/2}}$ and $y' = x'^{2^{T/2}}$ and $y' \circ y' = x'^{2^{2^{(1+\frac{T}{2})}}}$
  - If T/2 is even the sender/prover run HalvingProtocol$(N, x', \frac{T}{2}, y')$
  - If T/2 is odd the sender/prover run HalvingProtocol$(N, x', \frac{T}{2}+1, y' \circ y')$

Pietrzak's Construction [ITCS19]

- Non-Interactive version via Fiat-Shamir
- Efficiency of Halving Protocol
  - Terminates after *at most* $O(\log T)$ rounds
    - T replaced by T/2 or (T+1)/2 at each level of recursion
  - Naïve implementation:
    - Prover requires $\frac{T}{2^i} + \log \lambda$ queries to $\circ$ at ith level of recursion
    - Total work $\sum_{i=1}^{\log T} \left(\frac{T}{2^i} + \log \lambda\right) = O(T + \log T \log \lambda)$
  - Optimized Prover requires just $O(\sqrt{T} \log T)$ additional queries to group operations $\circ$
    - Assume $T = 2^t$ for simplicity
    - **Key idea:** Store $\mu_i = x^{2^{\left(\frac{T}{2^i}\right)}}$ for each $i \leq t$ to avoid re-computation

Pietrzak's Construction [ITCS19]

Theorem 1. *If the input (N, x, T) to the protocol satisfies*

1. *$N = p \cdot q$ is the product of safe primes, i.e., $p = 2p' + 1, q = 2q' + 1$ for primes p', q'.*
2. *$\langle x \rangle = QR_N^+$.*[3]
3. *$2^\lambda \leq \min\{p', q'\}$*

Then for any malicious prover $\widetilde{\mathcal{P}}$ who sends as first message y anything else than the solution to the RSW time-lock puzzle, i.e.,

$$y \neq x^{2^T}$$

$\mathcal{V}$ will finally output accept *with probability at most*

$$\frac{3\log(T)}{2^\lambda} .$$

Proof

It will be convenient to define the language

$$\mathcal{L} = \{(N, x, T, y) \ : y \neq x^{2^T} \bmod N \text{ and } \langle x \rangle = QR_N\}$$

We'll establish the following lemma.

Lemma 1. *For N, λ as in Thm. 1, and any malicious prover $\widetilde{\mathcal{P}}$ the following holds. If the input to the halving protocol in §3.1 satisfies*

$$(N, x, T, y) \in \mathcal{L}$$

then with probability $\geq 1 - 3/2^\lambda$ (over the choice of r) $\mathcal{V}$'s output is either reject *or satisfies*

$$(N, x', \lceil T/2 \rceil, y') \in \mathcal{L}$$

Proof of Theorem 1 (assuming Lemma 1)

Proof (Proof of Theorem 1). In every iteration of the halving protocol the time parameter decreases from T to $\lceil T/2 \rceil$ and it stops once $T = 1$, this means we iterate for at most $\lceil \log(T) \rceil$ rounds. By assumption, the input (N, x, T, y) to the first iteration is in $\mathcal{L}$, and by construction, the only case where $\mathcal{V}$ outputs accept is on an input $(N, x, 1, y)$ where $y = x^{2^T} = x^2 \bmod N$, in particular, this input is *not* in $\mathcal{L}$.

So, if $\mathcal{V}$ outputs accept, there must be one iteration of the halving protocol where the input is in $\mathcal{L}$ but the output is not. By Lemma 1, for any particular iteration this happens with probability $\leq 3/2^\lambda$. By the union bound, the probability of this happening in any of the $\lceil \log(T) \rceil - 1$ rounds can be upper bounded by $3 \log(T)/2^\lambda$ as claimed.

It will be convenient to define the language

$$\mathcal{L} = \{(N, x, T, y) \ : y \neq x^{2^T} \bmod N \text{ and } \langle x \rangle = QR_N\}$$

We'll establish the following lemma.

Lemma 1. *For N, λ as in Thm. 1, and any malicious prover $\widetilde{\mathcal{P}}$ the following holds. If the input to the halving protocol in §3.1 satisfies*

$$(N, x, T, y) \in \mathcal{L}$$

then with probability $\geq 1 - 3/2^\lambda$ (over the choice of r) $\mathcal{V}$'s output is either reject *or satisfies*

$$(N, x', \lceil T/2 \rceil, y') \in \mathcal{L}$$

Proof (Proof of Lemma 1). We just consider the case where T is even, the odd T case is almost identical.

Assuming the input to the halving protocol satisfies $(N, x, T, y) \in \mathcal{L}$, we must bound the probability that $\mathcal{V}$ outputs reject or the output $(N, x', T/2, y') \notin \mathcal{L}$.

Lemma 1. *For N, λ as in Thm. 1, and any malicious prover $\widetilde{\mathcal{P}}$ the following holds. If the input to the halving protocol in §3.1 satisfies*

$$(N, x, T, y) \in \mathcal{L}$$

then with probability $\geq 1 - 3/2^{\lambda}$ (over the choice of r) $\mathcal{V}$'s output is either reject *or satisfies*

$$(N, x', \lceil T/2 \rceil, y') \in \mathcal{L}$$

Proof (Proof of Lemma 1). We just consider the case where T is even, the odd T case is almost identical.

Assuming the input to the halving protocol satisfies $(N, x, T, y) \in \mathcal{L}$, we must bound the probability that $\mathcal{V}$ outputs reject or the output $(N, x', T/2, y') \notin \mathcal{L}$.

If $T = 1$ then $\mathcal{V}$ outputs reject and we're done. Otherwise, if $\widetilde{\mathcal{P}}$ sends a $\mu \notin QR_N$ in step 2. then $\mathcal{V}$ outputs reject in step 3. and we're done. So from now we assume $\mu \in QR_N$. We must bound

$$\Pr_r[(y' = x'^{2^{T/2}}) \ \vee \ (\langle x' \rangle \neq QR_N)] \leq 3/2^{\lambda}$$

If $T = 1$ then $\mathcal{V}$ outputs reject and we're done. Otherwise, if $\widetilde{\mathcal{P}}$ sends a $\mu \notin QR_N$ in step 2. then $\mathcal{V}$ outputs reject in step 3. and we're done. So from now we assume $\mu \in QR_N$. We must bound

$$\Pr_r[(y' = x'^{2^{T/2}}) \ \vee \ (\langle x' \rangle \neq QR_N)] \leq 3/2^\lambda$$

using $\Pr[a \vee b] = \Pr[a \wedge \overline{b}] + \Pr[b]$ we rewrite this as

$$\Pr_r[y' = x'^{2^{T/2}} \wedge \langle x' \rangle = QR_N] + \Pr_r[\langle x' \rangle \neq QR_N] \leq 3/2^\lambda \qquad (3)$$

Eq.(3) follows by the two claims below.

Claim. $\Pr_r[\langle x' \rangle \neq QR_N] \leq 2/2^\lambda$.

Claim. $\Pr_r[y' = x'^{2^{T/2}} \bmod N \wedge \langle x' \rangle = QR_N] \leq 1/2^\lambda$.

Comparison for VDFs

- Non-Interactive version via Fiat-Shamir
- Pietrzak's prover requires just $O(\sqrt{T}\log T)$ additional $\circ$ queries ☺
  - Better than $O(T/\log T)$ [Wesolowski]
- Pietrzak's proof size is $O(\log^2 T \log N)$
  - Worse than $O(\log N)$ [Wesolowski]
- Verifier Efficiency is $O((\lambda+1)\log T)$ queries to group operation $\circ$
  - Slightly worse than [Wesolowski] 😐

Thanks for Listening