

# Verifiable Random Functions and Verifiable Delay Functions

Caleb Smith

University of Virginia

# Why do these matter?

Alternative consensus protocols

Applications to public randomness generation

# Leader election

Bitcoin Proof of Work style

Everyone generates a random number, and the largest is the leader?

# Generate random numbers

Assume we have a hash function,  $h$ , and we have a public challenge,  $x$



$$key \leftarrow \{0,1\}^n$$
$$y = h(key || x)$$

$$key \leftarrow \{0,1\}^n$$
$$y = h(key || x)$$

$$key \leftarrow \{0,1\}^n$$
$$y = h(key || x)$$

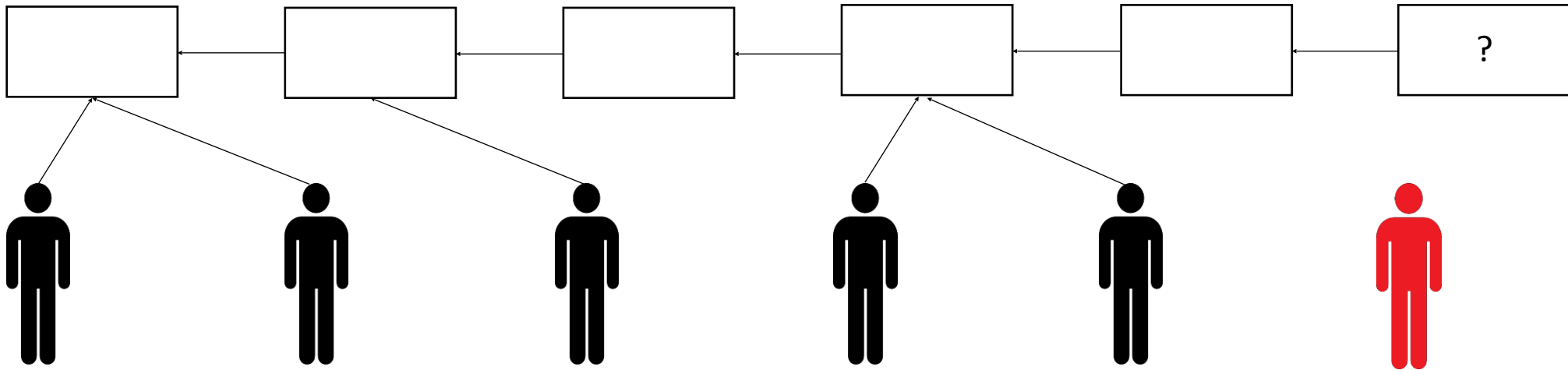
$$key \leftarrow \{0,1\}^n$$
$$y = h(key || x)$$

$$key \leftarrow \{0,1\}^n$$
$$y = h(key || x)$$

$$key \leftarrow \{0,1\}^n$$
$$y = h(key || x)$$

# Generate random numbers

Assume we have a hash function,  $h$ , and we have a public challenge,  $x$



$key \leftarrow \{0,1\}^n$     $key \leftarrow \{0,1\}^n$     $key \leftarrow \{0,1\}^n$     $key \leftarrow \{0,1\}^n$     $key \leftarrow \{0,1\}^n$     $key \leftarrow \{0,1\}^n$   
 $y = h(key || x)$     $y = h(key || x)$     $y = h(key || x)$     $y = h(key || x)$     $y = h(key || x)$     $y = h(key || x)$

# Verifiable Random Function

Introduced by Micali, Rabin, and Vadhan in 1999

$$\text{KeyGen}(1^\lambda) \rightarrow (sk, pk)$$

$$\text{Prove}(sk, x) \rightarrow (y, \pi)$$

Pseudorandom      Proof

$$\text{Verify}(pk, x, y, \pi) \rightarrow \{0,1\}$$

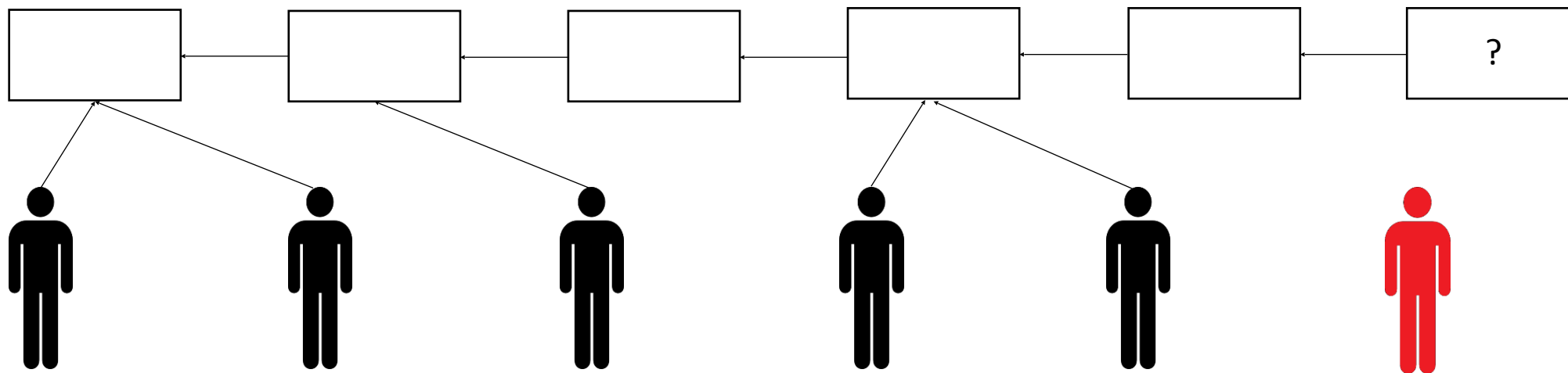
Security Property:

*For all  $x$ , an adversary cannot find  $y_0 \neq y_1$  such that*

$$\text{Verify}(pk, x, y_0, \pi_0) = 1 = \text{Verify}(pk, x, y_1, \pi_1)$$

# Generate random numbers

Assume we have a hash function,  $h$ , and we have a public challenge,  $x$



$sk, pk \leftarrow KeyGen$   $sk, pk \leftarrow KeyGen$   $sk, pk \leftarrow KeyGen$   $sk, pk \leftarrow KeyGen$   $sk, pk \leftarrow KeyGen$   $sk, pk \leftarrow KeyGen$   
 $y, \pi = Prove(sk, x)$   $y, \pi = Prove(sk, x)$   $y, \pi = Prove(sk, x)$   $y, \pi = Prove(sk, x)$   $y, \pi = Prove(sk, x)$   $y, \pi = Prove(sk, x)$

# Verifiable Random Function Assumptions

RSA + Random Oracle [Micali, Rabin, and Vadhan 1999]

Decisional Bilinear Diffie Hellman Inversion [Dodis and Yampolski 2004]

Decisional Diffie Hellman + Random Oracle [Papadopoulos et al 2017]



# Verifiable Delay Functions

Introduced by Boneh, Bonneau, Bünz, and Fisch in 2018

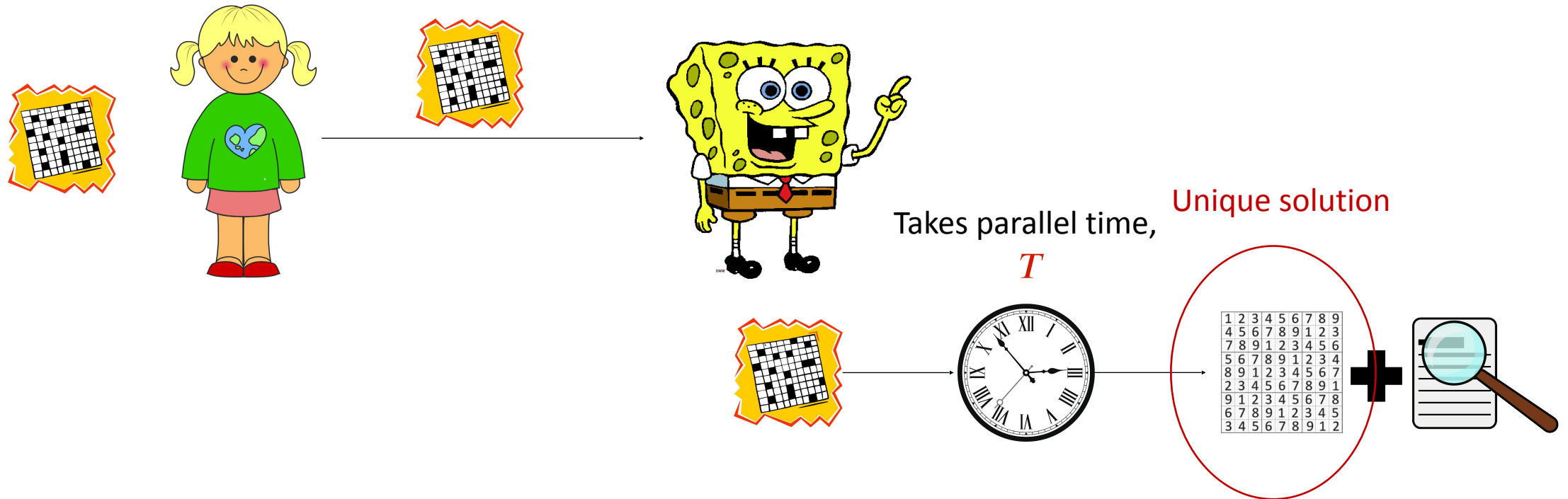
Delay – Takes a minimum amount of *parallel* time to compute

Function – Unique outputs

Verifiable – Third parties can verify that it was evaluated correctly

# Verifiable Delay Function

Alice wants to require Bob to spend  $T$  *parallel time* solving a challenge



# Verifiable Delay Function Syntax

A function that takes a long time to compute, has unique outputs, and can be verified quickly

$Setup(\lambda, T) \rightarrow PP = (ek, vk)$ , specifies input and output space

$Eval(ek, x) \rightarrow (y, \pi)$ , runs in at least *parallel time T*

Proof from the Evaluator to  
help the Verifier

$Verify(vk, x, y, \pi) \rightarrow \{Accept, Reject\}$ , runs in time  $t \ll T$

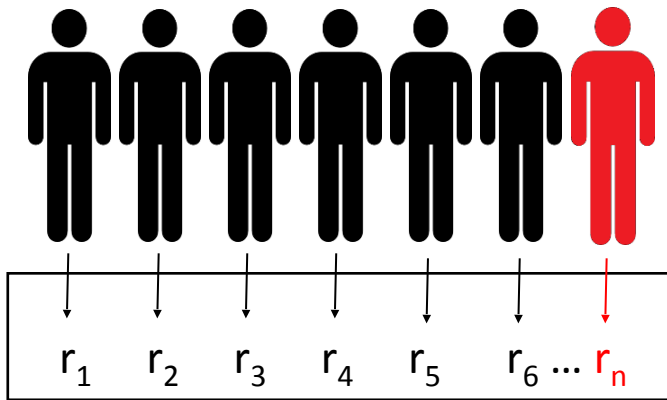
# Verifiable Delay Function Properties

Sequentiality –  $\text{Eval}(x)$  cannot be solved in less than *parallel time*  $T$ , with *poly*( $T$ ) number of processors

Uniqueness – If the adversary runs in time  $O(\text{poly}(T, \lambda))$ , then they are unable to find a  $y \neq \text{Eval}(x)$  that passes verification

# Application - Randomness Beacon

Generate a stream of public random values

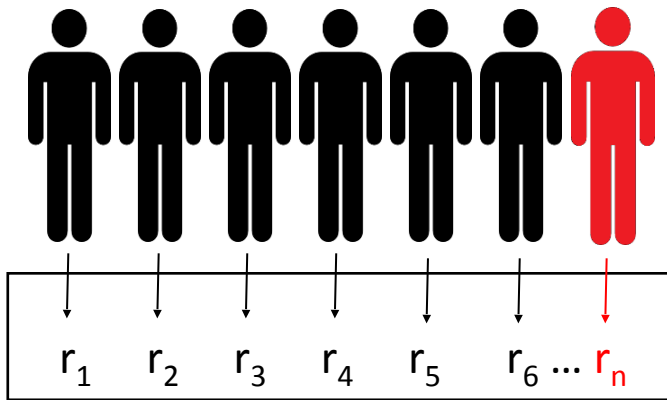


Can submit values from 1:00pm to 1:10pm

$$f(r_1, r_2, \dots, r_n) = r_1 \oplus r_2 \oplus \dots \oplus r_n$$

# Application - Randomness Beacon

Generate a stream of public random values



Can submit values from 1:00pm to 1:10pm

$$h(r_1, r_2, \dots, r_n) = x$$

$$VDF.Eval(x) \rightarrow (y, \pi)$$

$$Extract(y) \rightarrow v$$

# Application – Proof of Space and Time

Cohen and Pietrzak from Chia

Change the assumption from majority of computing power is honest to 2/3 of committed disk space is honest

Proofs of Space will populate some disk space with some function  $f$  and given a challenge, will find their “best” solution almost instantly

# Why not just chain Proofs of Space?

The next Proof of Space challenge is the hash of the previous Proof of Space solution and proof

There are attacks where an adversary can “tweak” elements in their control to bias the next challenge

This does not occur in Bitcoin because of the cost to split resources



# Adding Verifiable Delay Functions

Take the solution and proof of the Proof of Space,  $(y, \pi_{PoS})$ , and compute  $x = h(y \parallel \pi_{PoS})$ ,  $VDF.Eval(x) \rightarrow (y, \pi_{VDF})$

Then  $f(y)$  determines the next Proof of Space challenge

We can now argue that an adversary cannot determine how to “tweak” anything to bias the next challenge


# Verifiable Delay Function Assumptions

Repeated squaring in group of unknown order is inherently sequential

Let  $N$  be an RSA modulus where nobody knows the factorization

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

Conjectured to take  $T$  sequential squarings



Questions?