Constrained Pseudorandom Functions for Unconstrained Inputs

Apoorvaa Deshpande (Brown University)
Venkata Koppula (University of Texas at Austin)
Brent Waters (University of Texas at Austin)

Pseudorandom Functions

(Goldreich-Goldwasser-Micali 84)

Pseudorandom Functions

(Goldreich-Goldwasser-Micali 84)

Keyed Function F

Key space $\mathcal{K}$

Numerous applications in Cryptography

Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Constrained PRFs

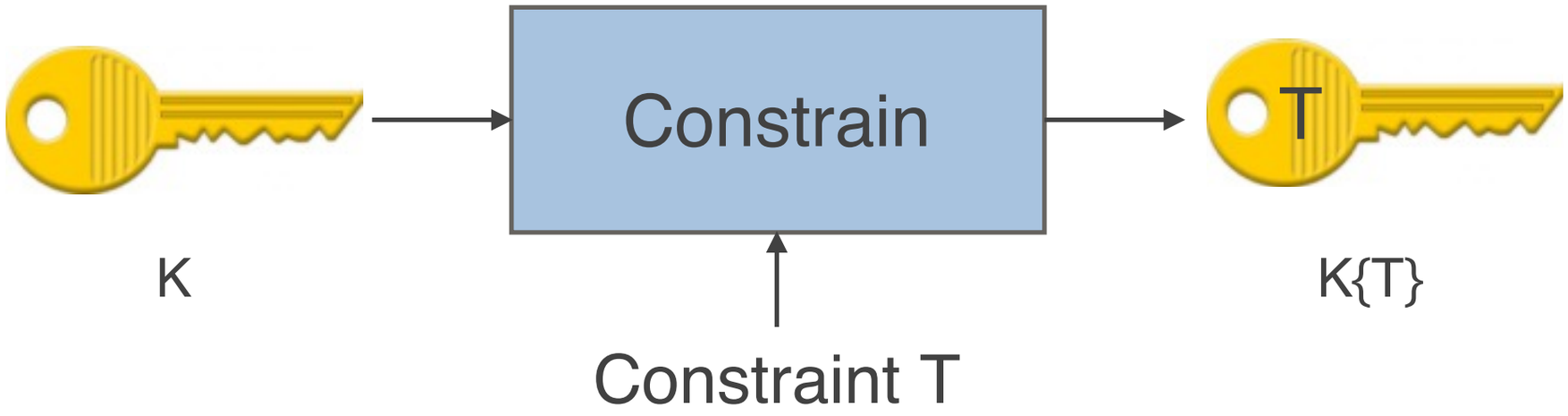
(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Keyed Function F, Key Space $\mathcal{K}$

Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

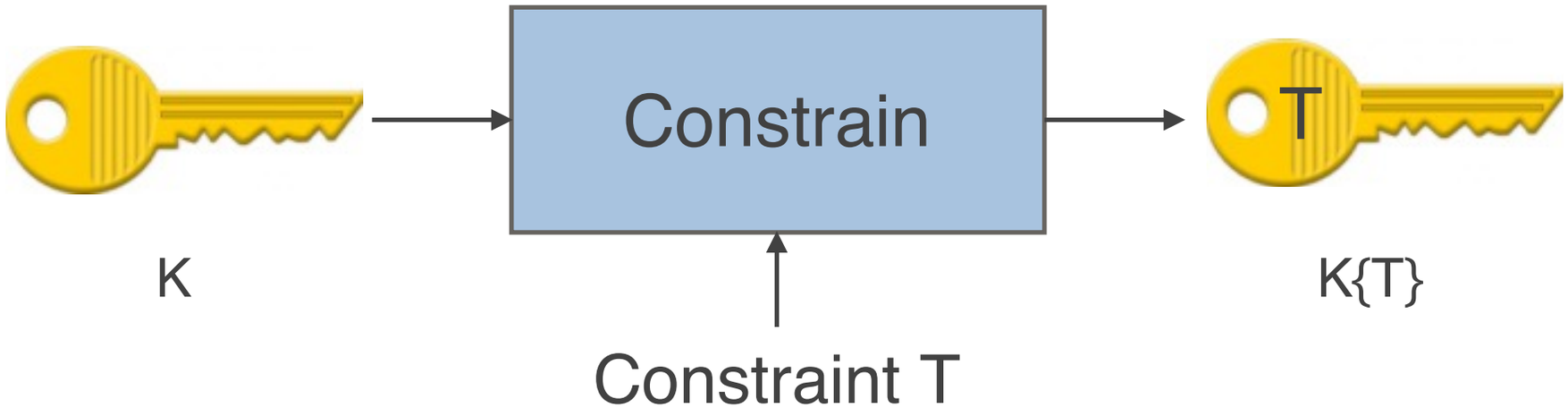
Keyed Function F, Key Space $\mathcal{K}$



Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Keyed Function F, Key Space $\mathcal{K}$



For all x s.t. x satisfies T, F(K , x) = F(K{T} , x)

Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Families of Constraints:

Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Families of Constraints:

- Puncturable PRFs
  - Key can evaluate PRF at all points except ‘punctured point’
  - Goldreich-Goldwasswer-Micali 84 PRFs are puncturable PRFs
  - Punctured programming approach (Sahai-Waters 14)

Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Families of Constraints:

- Puncturable PRFs
  - Key can evaluate PRF at all points except ‘punctured point’
  - Goldreich-Goldwasswer-Micali 84 PRFs are puncturable PRFs
  - Punctured programming approach (Sahai-Waters 14)
- Bit Fixing PRFs
  - Key for a string *s* in $\{0, 1, \perp\}^n$: can evaluate PRF at all points fixed by *s*
  - Multilinear maps based construction (Boneh-Waters 13)
  - Optimal broadcast encryption

Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Families of Constraints:

- Circuit Constrained PRFs
  - Key corresponding to circuit C : can evaluate PRF at input x if C(x) = 1
  - Multilinear maps based construction (Boneh-Waters 13), iO based construction (Boneh-Zhandry 14)
  - Identity based Noninteractive Key Exchange (Boneh-Waters 13)

Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Families of Constraints:

- Circuit Constrained PRFs
  - Key corresponding to circuit C : can evaluate PRF at input x if C(x) = 1
  - Multilinear maps based construction (Boneh-Waters 13), iO based construction (Boneh-Zhandry 14)
  - Identity based Noninteractive Key Exchange (Boneh-Waters 13)

Circuits can handle only bounded length inputs!

Constrained PRFs

(Boneh-Waters, Boyle-Goldwasser-Ivan, Kiayias-Papadopoulos-Triandopoulos-Zacharias)

Families of Constraints:

- Circuit Constrained PRFs
  - Key corresponding to circuit C : can evaluate PRF at input x if C(x) = 1
  - Multilinear maps based construction (Boneh-Waters 13), iO based construction (Boneh-Zhandry 14)
  - Identity based Noninteractive Key Exchange (Boneh-Waters 13)

for bounded number of users

Circuits can handle only bounded length inputs!

Constrained PRFs for Unconstrained Inputs

Constrained PRFs for Unconstrained Inputs

Turing Machine Constrained PRFs

Abusalah, Fuchsbauer, Pietrzak 14

Constrained PRFs for Unconstrained Inputs

Turing Machine Constrained PRFs

Abusalah, Fuchsbauer, Pietrzak 14

- Identity based Noninteractive Key Exchange : unbounded users
- Broadcast encryption : unbounded users

Constrained PRFs for Unconstrained Inputs

Turing Machine Constrained PRFs

Abusalah, Fuchsbauer, Pietrzak 14

- Identity based Noninteractive Key Exchange : unbounded users
- Broadcast encryption : unbounded users

Construction based on knowledge-type assumption

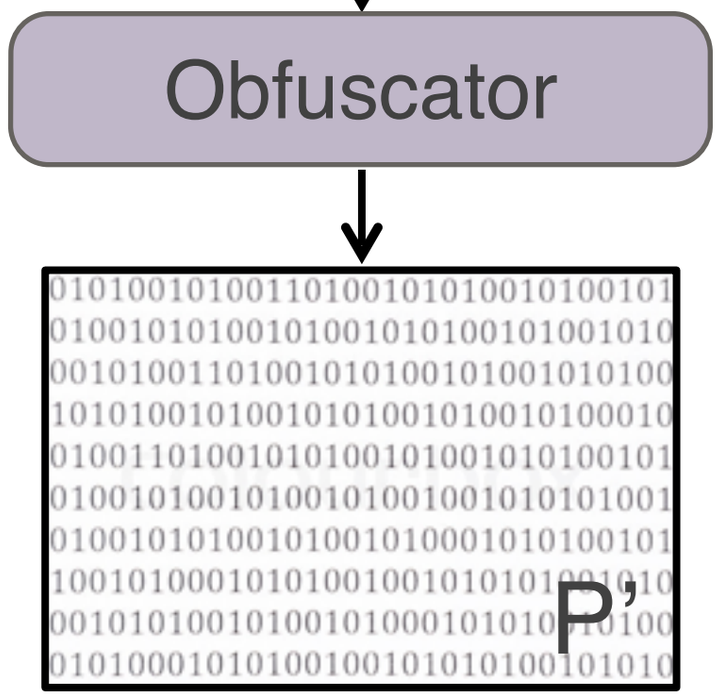
Code Obfuscation

Code Obfuscation

Goal: Make programs maximally unintelligible.

Code Obfuscation

Goal: Make programs maximally unintelligible.



P(x) = P’(x) for all inputs x

Code Obfuscation

Security for obfuscation

Code Obfuscation

Security for obfuscation

Virtual Black Box obfuscation (VBB)

Code Obfuscation

Security for obfuscation

Virtual Black Box obfuscation (VBB)

Obfuscated code

≈

Oracle access to code

Code Obfuscation

Security for obfuscation

Virtual Black Box obfuscation (VBB)

Obfuscated code

≈

Oracle access to code

Impossibility results

(Barak et al. 2001)

Code Obfuscation

Security for obfuscation

	Differing inputs obfuscation (diO)	**Virtual Black Box obfuscation (VBB)**
		Obfuscated code ≈ Oracle access to code
		Impossibility results (Barak et al. 2001)

Code Obfuscation

Security for obfuscation

Differing inputs obfuscation (diO)

If $diO(P_1)$ and $diO(P_2)$ are distinguishable, then one can extract differing input.

Virtual Black Box obfuscation (VBB)

Obfuscated code

≈

Oracle access to code

Impossibility results

(Barak et al. 2001)

Code Obfuscation

Security for obfuscation

	Differing inputs obfuscation (diO)	**Virtual Black Box obfuscation (VBB)**
	If $diO(P_1)$ and $diO(P_2)$ are distinguishable, then one can extract differing input.	Obfuscated code ≈ Oracle access to code
	Implausibility results (Boyle et al, Garg et al, Bellare et al.)	Impossibility results (Barak et al. 2001)

Code Obfuscation

Security for obfuscation

Public coins differing inputs obfuscation (pcdiO)

Differing inputs obfuscation (diO)

Virtual Black Box obfuscation (VBB)

If $diO(P_1)$ and $diO(P_2)$ are distinguishable, then one can extract differing input.

Obfuscated code

≈

Oracle access to code

Implausibility results (Boyle et al, Garg et al, Bellare et al.)

Impossibility results

(Barak et al. 2001)

Code Obfuscation

Security for obfuscation

Public coins differing inputs obfuscation (pcdiO)	**Differing inputs obfuscation (diO)**	**Virtual Black Box obfuscation (VBB)**
No implausibility results, but has ‘extractability’ nature	If $diO(P_1)$ and $diO(P_2)$ are distinguishable, then one can extract differing input.	Obfuscated code ≈ Oracle access to code
	Implausibility results (Boyle et al, Garg et al, Bellare et al.)	Impossibility results (Barak et al. 2001)

Code Obfuscation

Security for obfuscation

Indistinguishability obfuscation (iO)

No implausibility results, but has ‘extractability’ nature

Public coins differing inputs obfuscation (pcdiO)

If $diO(P_1)$ and $diO(P_2)$ are distinguishable, then one can extract differing input.

Implausibility results (Boyle et al, Garg et al, Bellare et al.)

Differing inputs obfuscation (diO)

Obfuscated code

≈

Oracle access to code

Impossibility results

(Barak et al. 2001)

Virtual Black Box obfuscation (VBB)

Code Obfuscation

Security for obfuscation

Indistinguishability obfuscation (iO)

If P_1 and P_2 functionally identical, then $iO(P_1) \approx iO(P_2)$

Public coins differing inputs obfuscation (pcdiO)

No implausibility results, but has ‘extractability’ nature

Differing inputs obfuscation (diO)

If $diO(P_1)$ and $diO(P_2)$ are distinguishable, then one can extract differing input.

Implausibility results (Boyle et al, Garg et al, Bellare et al.)

Virtual Black Box obfuscation (VBB)

Obfuscated code

≈

Oracle access to code

Impossibility results

(Barak et al. 2001)

Code Obfuscation

Security for obfuscation

Indistinguishability obfuscation (iO)	**Public coins differing inputs obfuscation (pcdiO)**	**Differing inputs obfuscation (diO)**	**Virtual Black Box obfuscation (VBB)**
If P_1 and P_2 functionally identical, then $iO(P_1) \approx iO(P_2)$	No implausibility results, but has 'extractability' nature	If $diO(P_1)$ and $diO(P_2)$ are distinguishable, then one can extract differing input.	Obfuscated code ≈ Oracle access to code
		Implausibility results (Boyle et al, Garg et al, Bellare et al.)	Impossibility results (Barak et al. 2001)

Constrained PRFs for Unconstrained Inputs

Turing Machine
Constrained PRFs

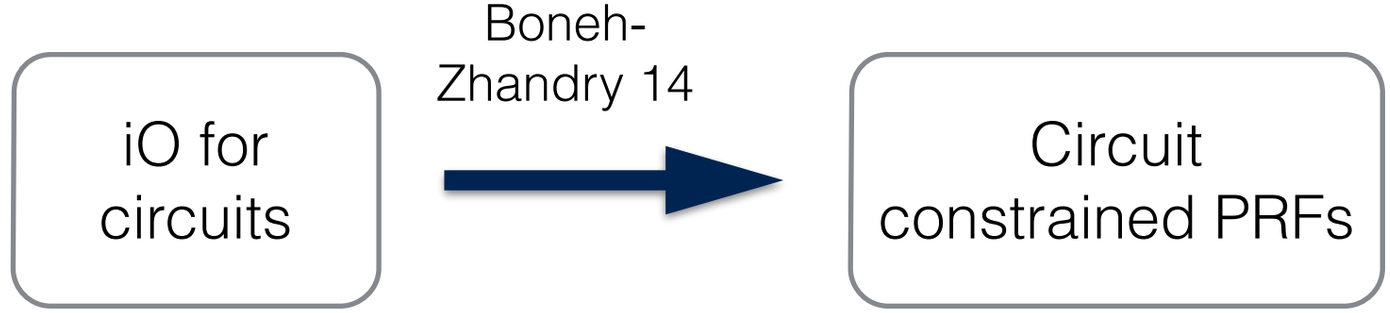
Abusalah, Fuchsbauer, Pietrzak 14

- Identity based Noninteractive Key Exchange : unbounded users
- Broadcast encryption : unbounded users

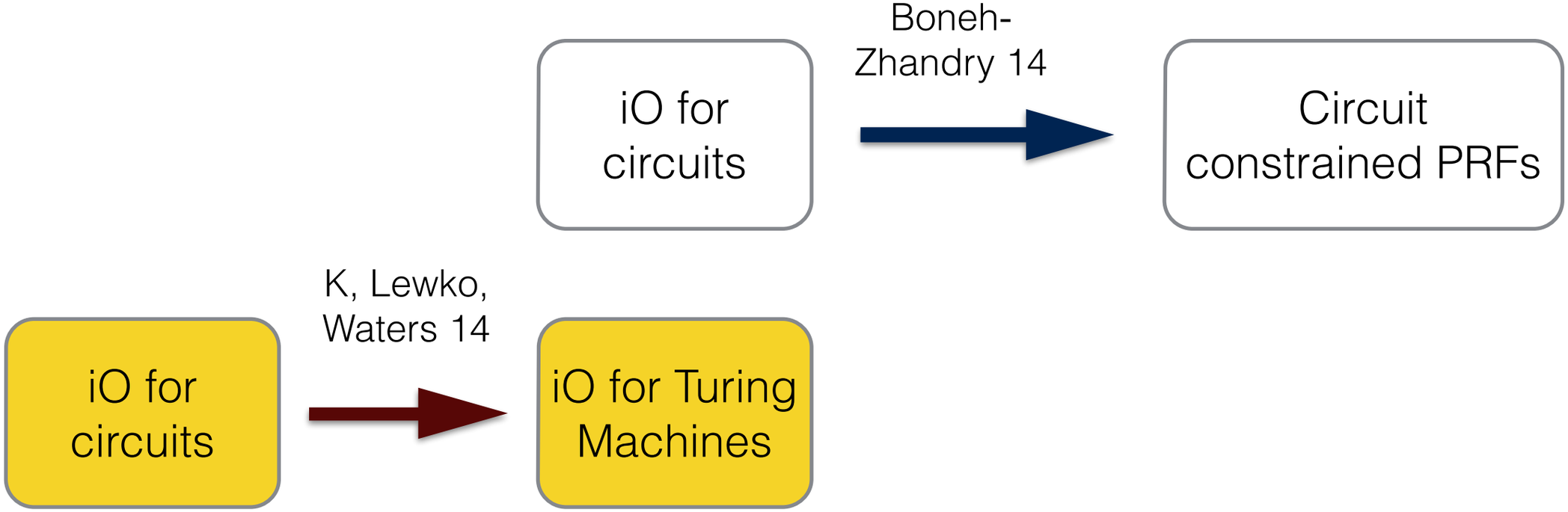
Construction based on
public coins differing inputs obfuscator

Can we build a constrained PRF scheme for Turing machines based on indistinguishability obfuscation (iO)?

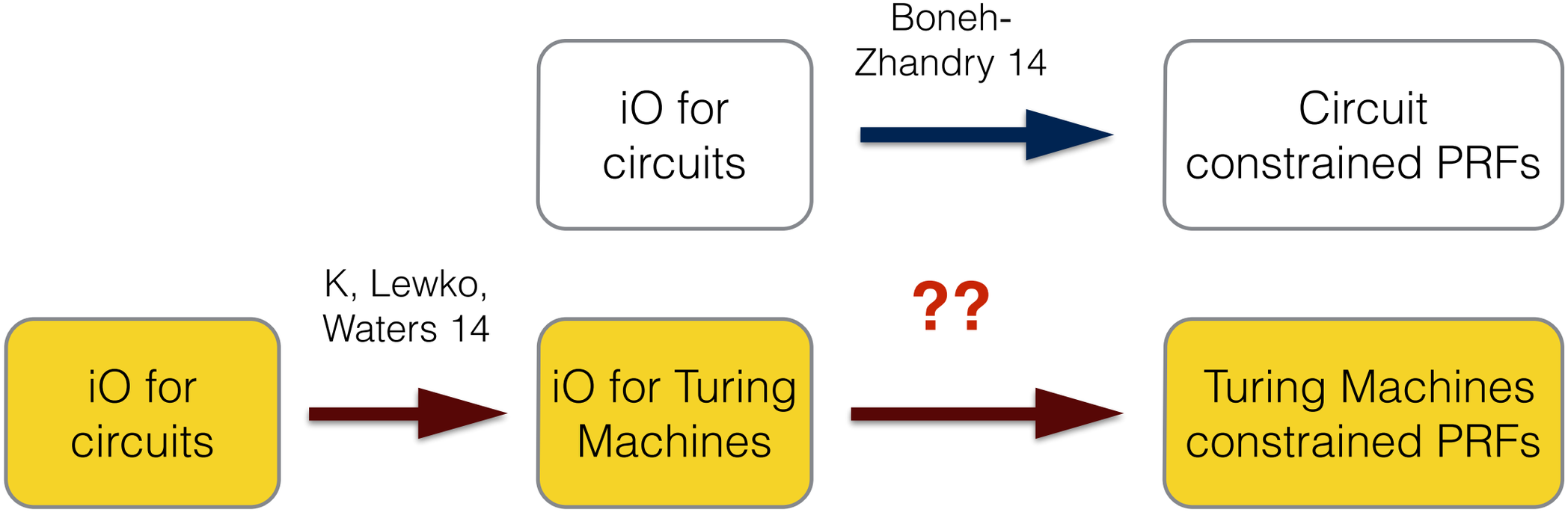
Can we build a constrained PRF scheme for Turing machines based on indistinguishability obfuscation (iO)?



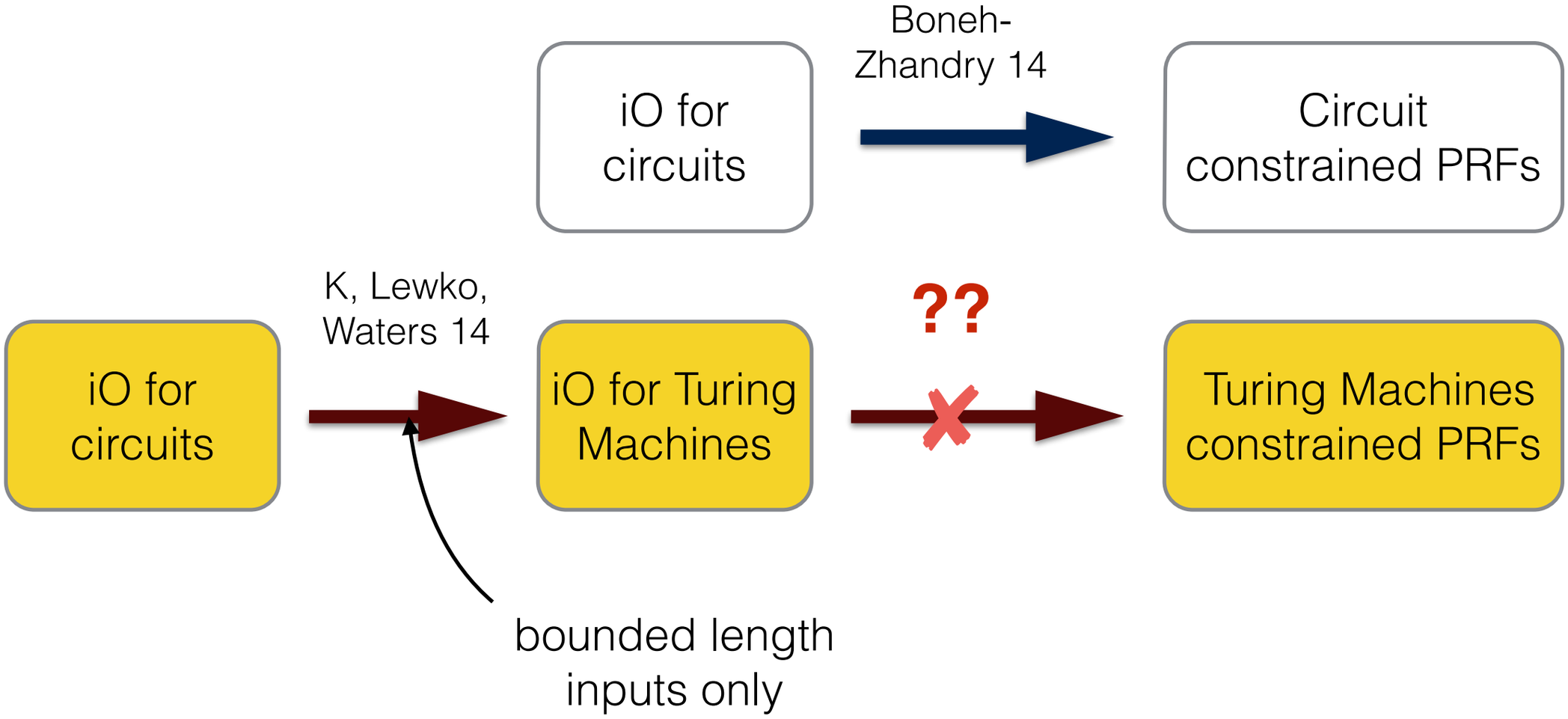
Can we build a constrained PRF scheme for Turing machines based on indistinguishability obfuscation (iO)?



Can we build a constrained PRF scheme for Turing machines based on indistinguishability obfuscation (iO)?



Can we build a constrained PRF scheme for Turing machines based on indistinguishability obfuscation (iO)?



Our Results

Assuming iO (and one way functions), we show a constrained PRF scheme for Turing machines.

Our Results

Assuming iO (and one way functions), we show a constrained PRF scheme for Turing machines.

Assuming iO (and one way functions), we show an Attribute Based Encryption scheme for Turing machines.

FREE !!

Security of Constrained PRFs

Security of Constrained PRFs

Selective Security

Security of Constrained PRFs

Selective Security

Chooses PRF key K.

Security of Constrained PRFs

Selective Security

Chooses PRF key K.

$\leftarrow$ x^*

$y^* = PRF(K, x^*)$ $\rightarrow$
or random

Security of Constrained PRFs

Selective Security

Chooses PRF key K.

x^*

$y^* = PRF(K, x^*)$ or random

M_i

$K\{M_i\}$

$M_i(x^*) = 0$

Security of Constrained PRFs

Selective Security

Chooses PRF key K.

$M_i(x^*) = 0$

Guess PRF/random

Indistinguishability Obfuscation for Circuits

Indistinguishability Obfuscator

C_0, C_1 functionally identical circuits.

$iO(C_0) \approx iO(C_1)$

Indistinguishability Obfuscation for Circuits

Indistinguishability Obfuscator

C_0, C_1 functionally identical circuits.

$iO(C_0) \approx iO(C_1)$

Candidate iO schemes for circuits:

Garg-Gentry-Halevi-Raykova-Sahai-Waters 13

Barak-Garg-Kalai-Paneth-Sahai 14

Zimmerman 14

…

PRFs with Unbounded Inputs

PRFs with Unbounded Inputs

PRF F with bounded length inputs

PRFs with Unbounded Inputs

PRF F with bounded length inputs

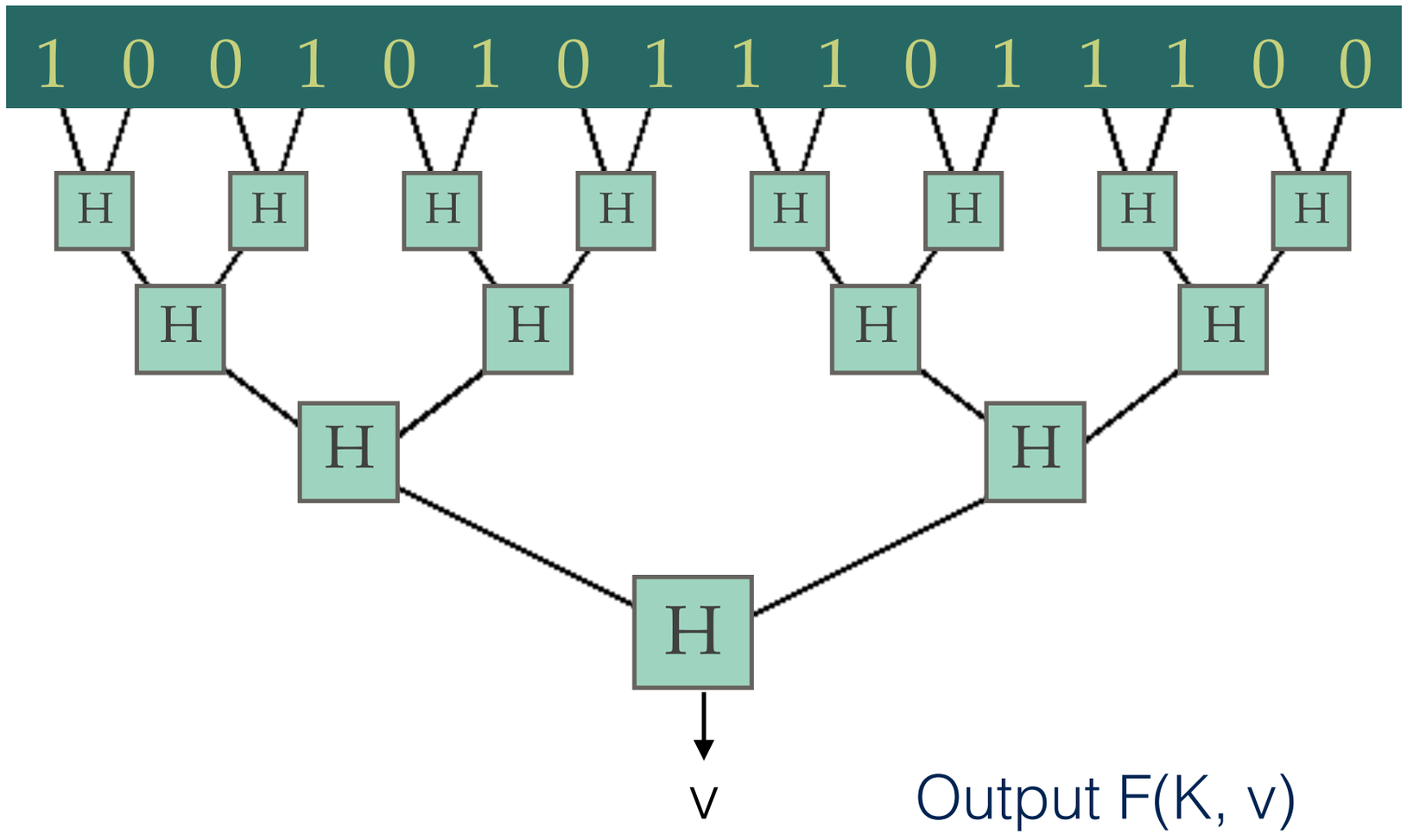
For unbounded inputs : Choose PRF key K, hash function H

PRFs with Unbounded Inputs

PRF F with bounded length inputs

For unbounded inputs : Choose PRF key K, hash function H

Merkle Tree



Output F(K, v)

Our Constrained PRF Construction

Our Constrained PRF Construction

Puncturable PRF F with bounded length inputs

Our Constrained PRF Construction

Puncturable PRF F with bounded length inputs

Choose puncturable PRF key K, special hash function H

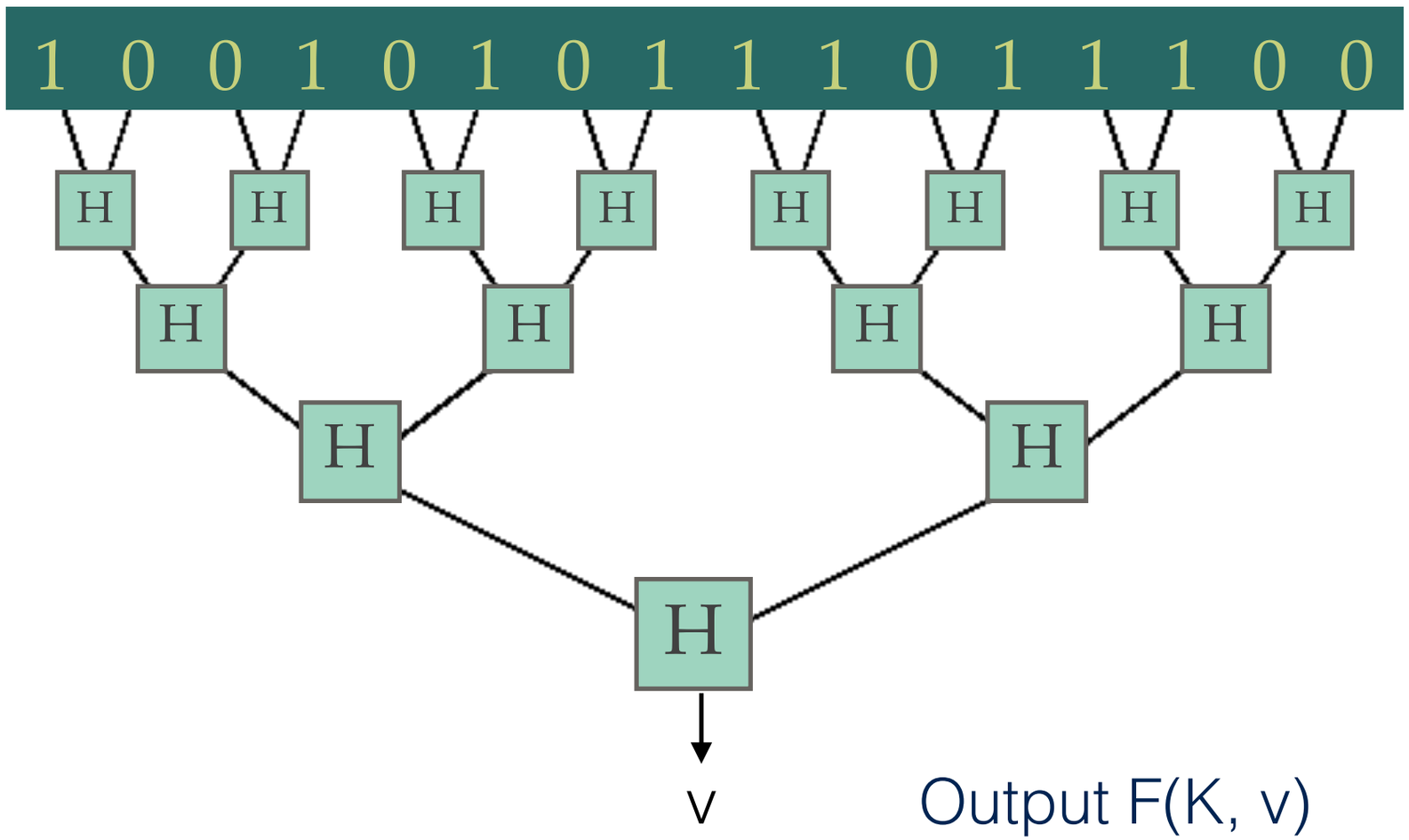
Our scheme's PRF key : (K, H)

Our Constrained PRF Construction

Puncturable PRF F with bounded length inputs

Choose puncturable PRF key K, special hash function H

Our scheme's PRF key : (K, H)



Our Constrained PRF Construction

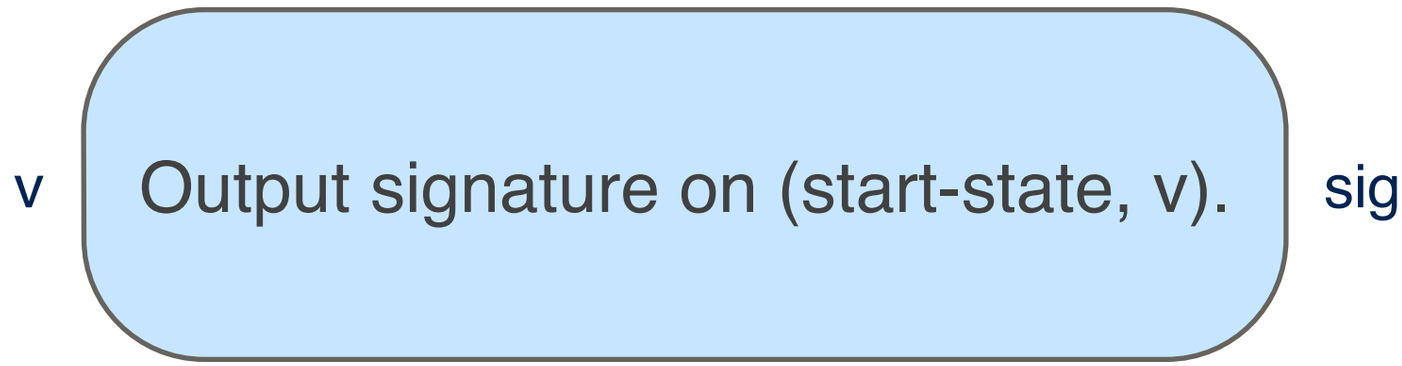
Constrained key for Turing machine M

Our Constrained PRF Construction

Constrained key for Turing machine M

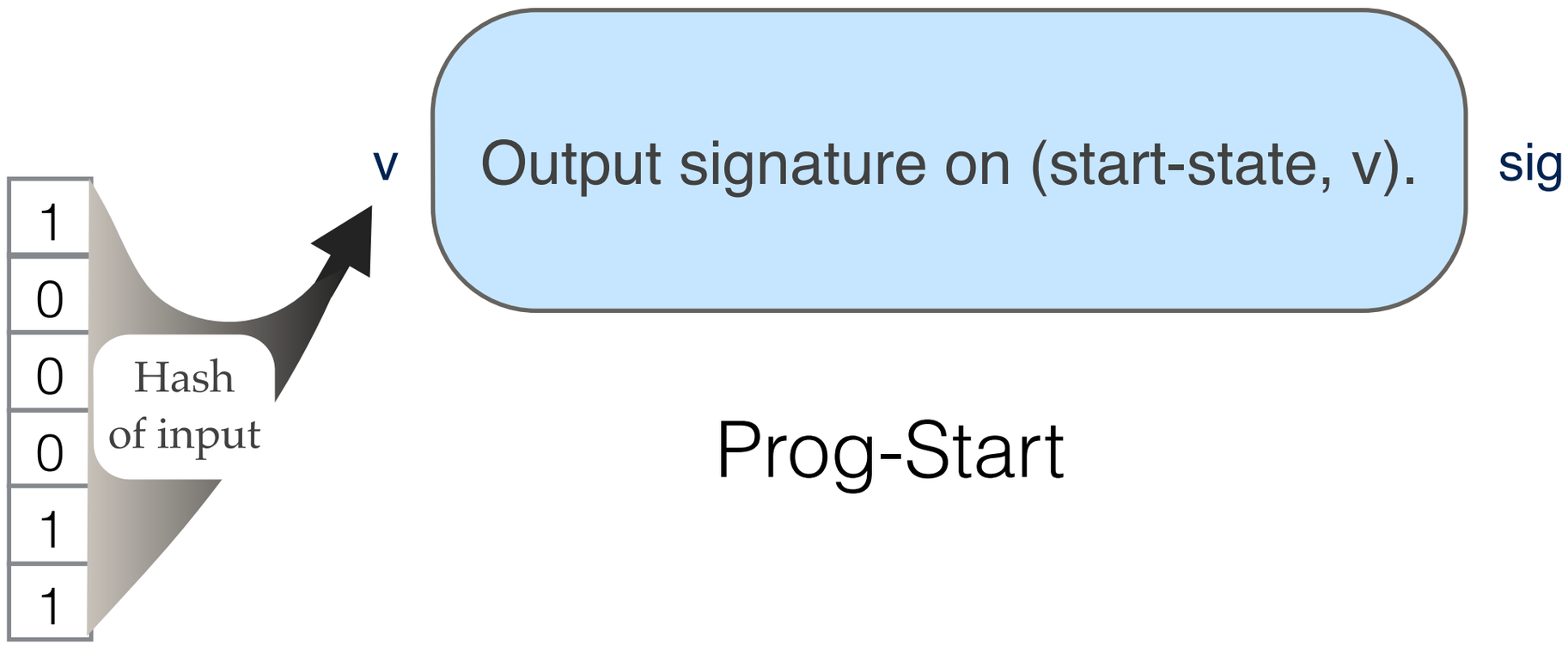
K{M} = H, iO(Prog-Iterate), iO(Prog-Start)

K{M} = H, iO(Prog-Iterate), iO(Prog-Start)



Prog-Start

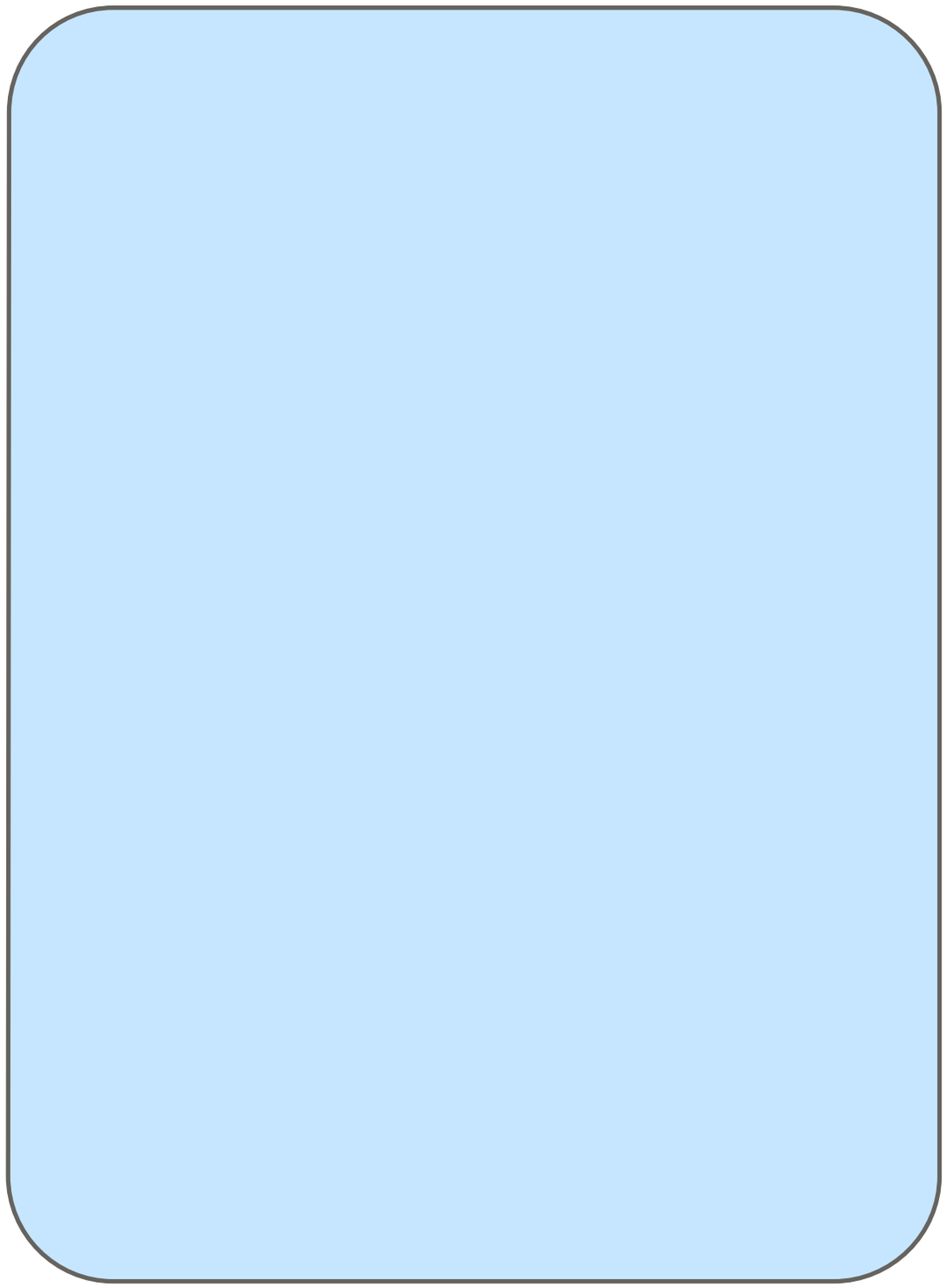
K{M} = H, iO(Prog-Iterate), iO(Prog-Start)



Prog-Start

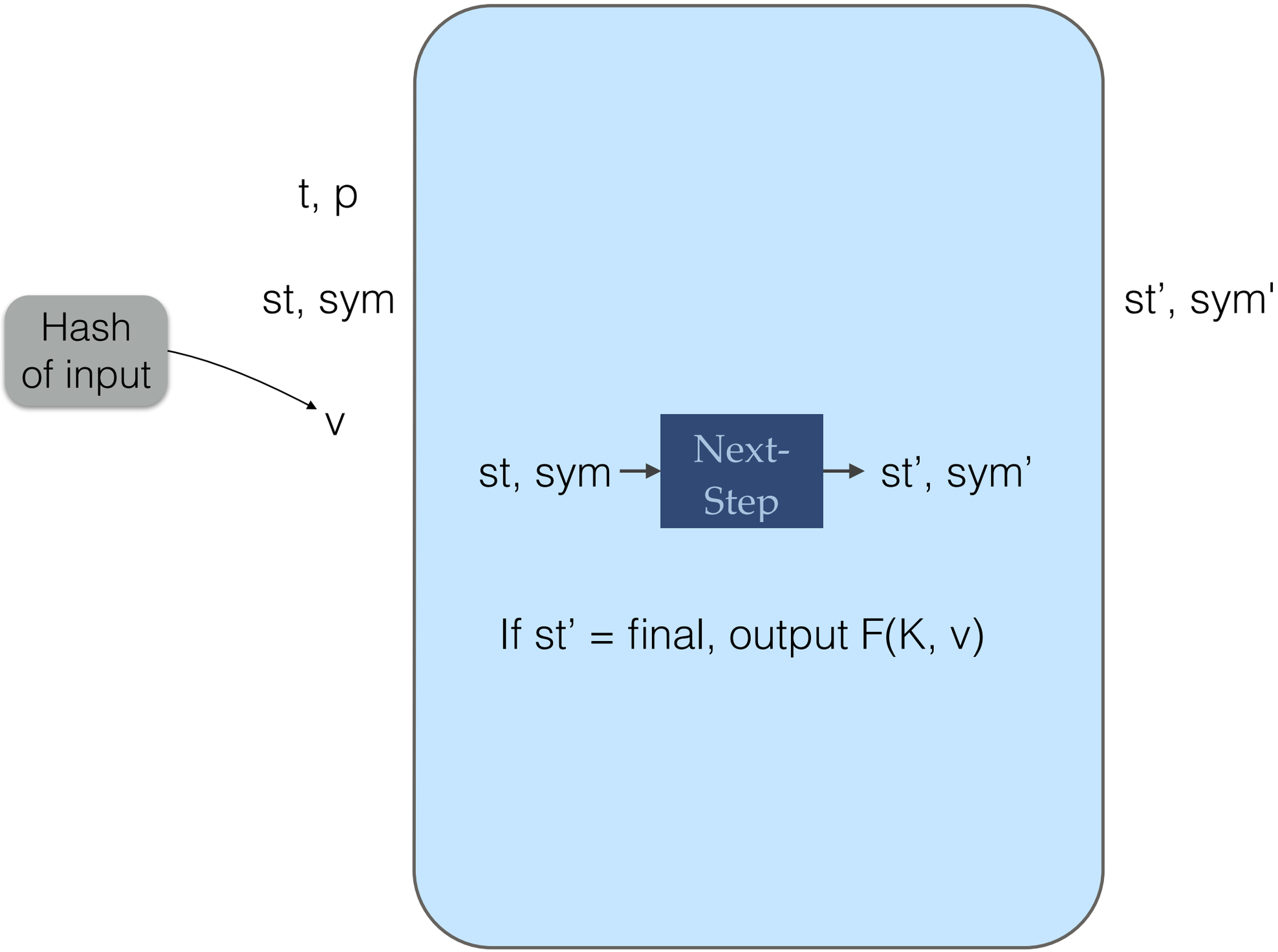
K{M} = H, iO(Prog-Iterate), iO(Prog-Start)

K{M} = H, iO(Prog-Iterate), iO(Prog-Start)



Prog-Iterate

K{M} = H, iO(Prog-Iterate), iO(Prog-Start)



Prog-Iterate

K{M} = H, iO(Prog-Iterate), iO(Prog-Start)

Adversary not bound to correct execution.

t, p

st, sym

Hash of input

v

st', sym'

st, sym → Next-Step → st', sym'

If st' = final, output F(K, v)

Prog-Iterate

K{M} = H, iO(Prog-Iterate), iO(Prog-Start)

Adversary not bound to correct execution.

t, p

st, sym

Hash of input

v

Hash of work tape

h

Using h, verify sym at position p

st, sym → Next-Step → st', sym'

If st' = final, output F(K, v)

Using sym', update h to h'

st', sym'

h'

Prog-Iterate

K{M} = H, iO(Prog-Iterate), iO(Prog-Start)

Adversary not bound to correct execution.

t, p

st, sym

Hash of input

v

Hash of work tape

h

sig

Using h, verify sym at position p

Verify sig on (st, h)

st, sym → Next-Step → st', sym'

If st' = final, output F(K, v)

Using sym', update h to h'

Output signature on (st', h')

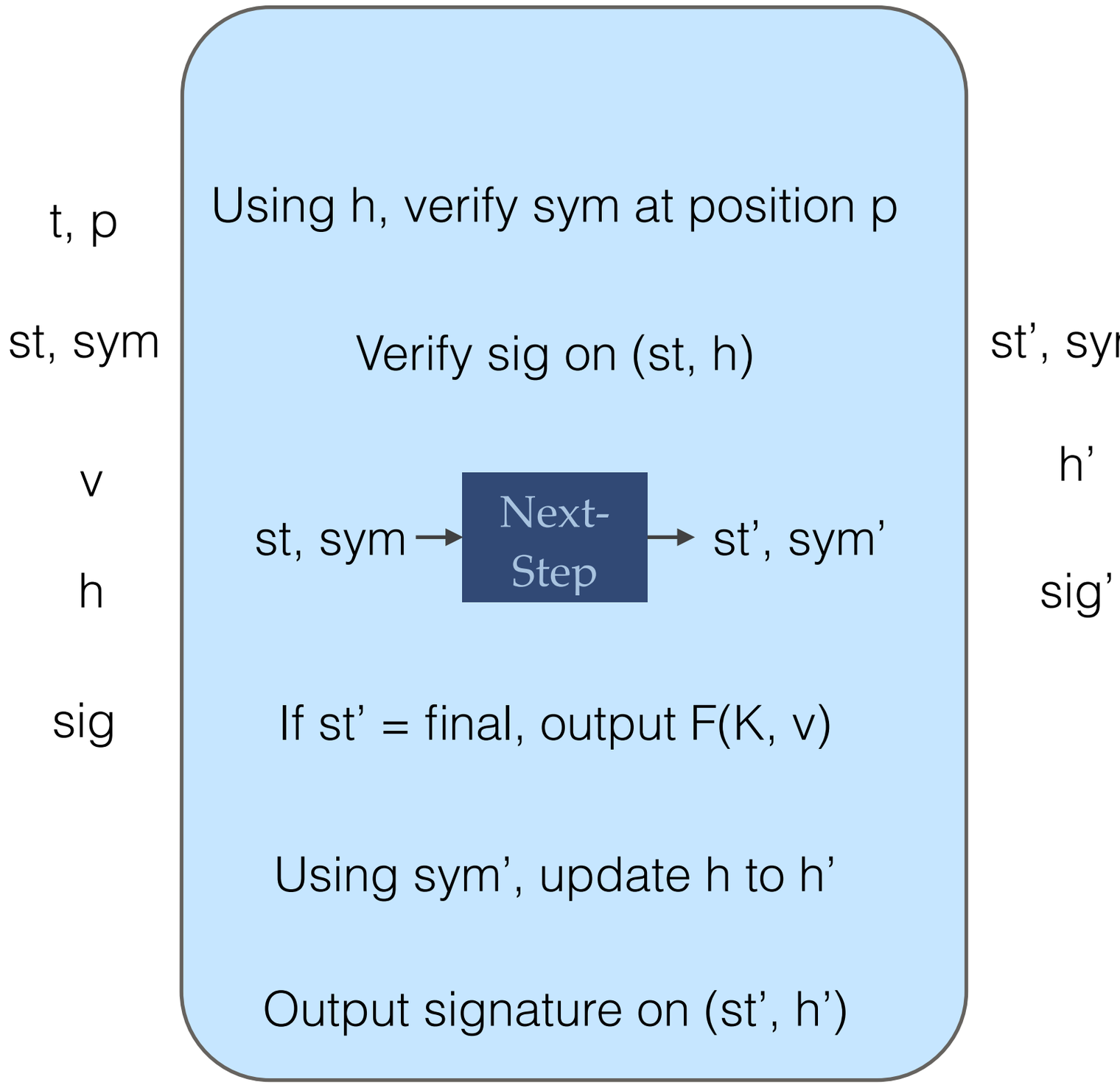
st', sym'

h'

sig'

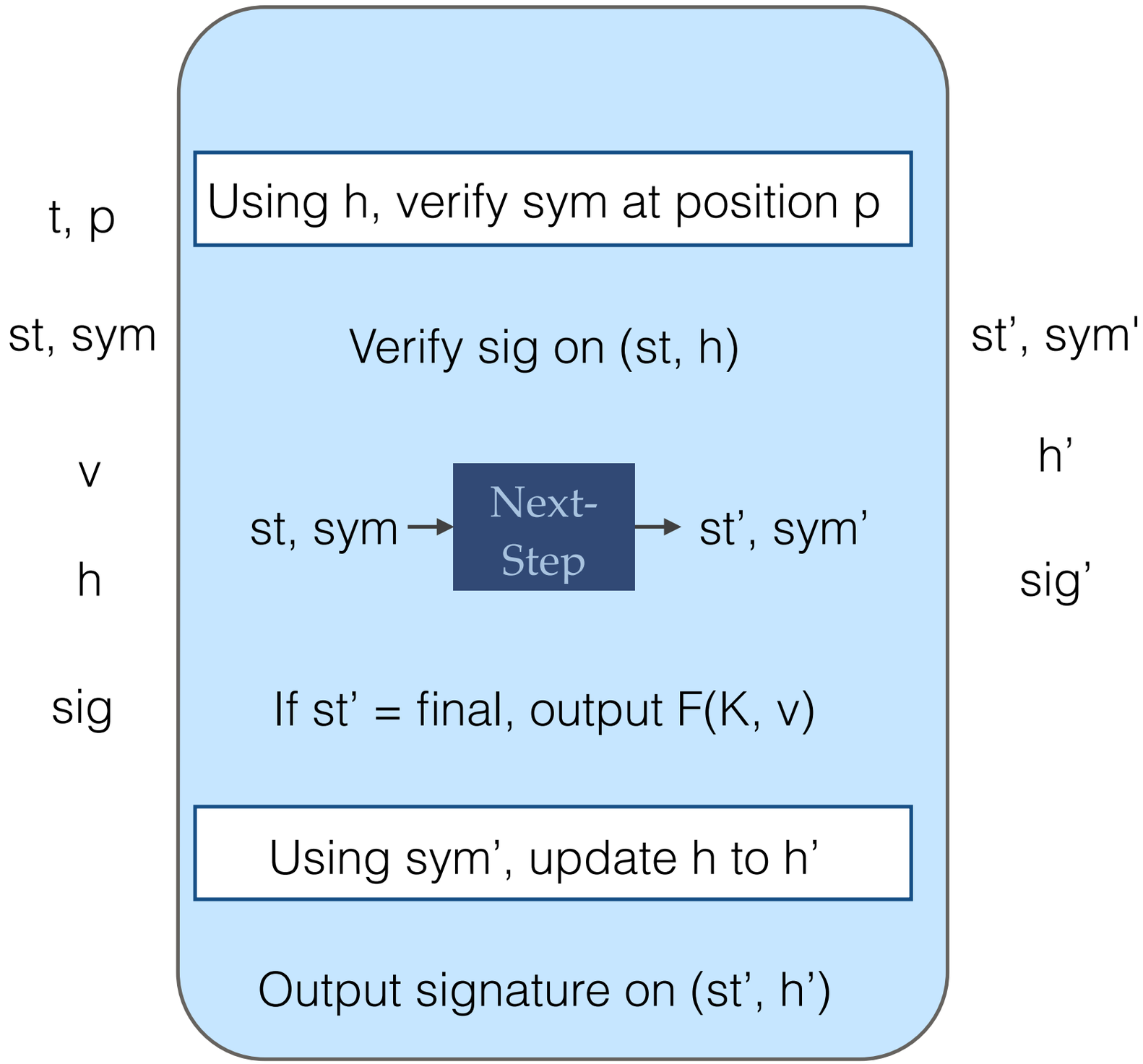
Prog-Iterate

K{M} = H, iO(Prog-Iterate), iO(Prog-Start)



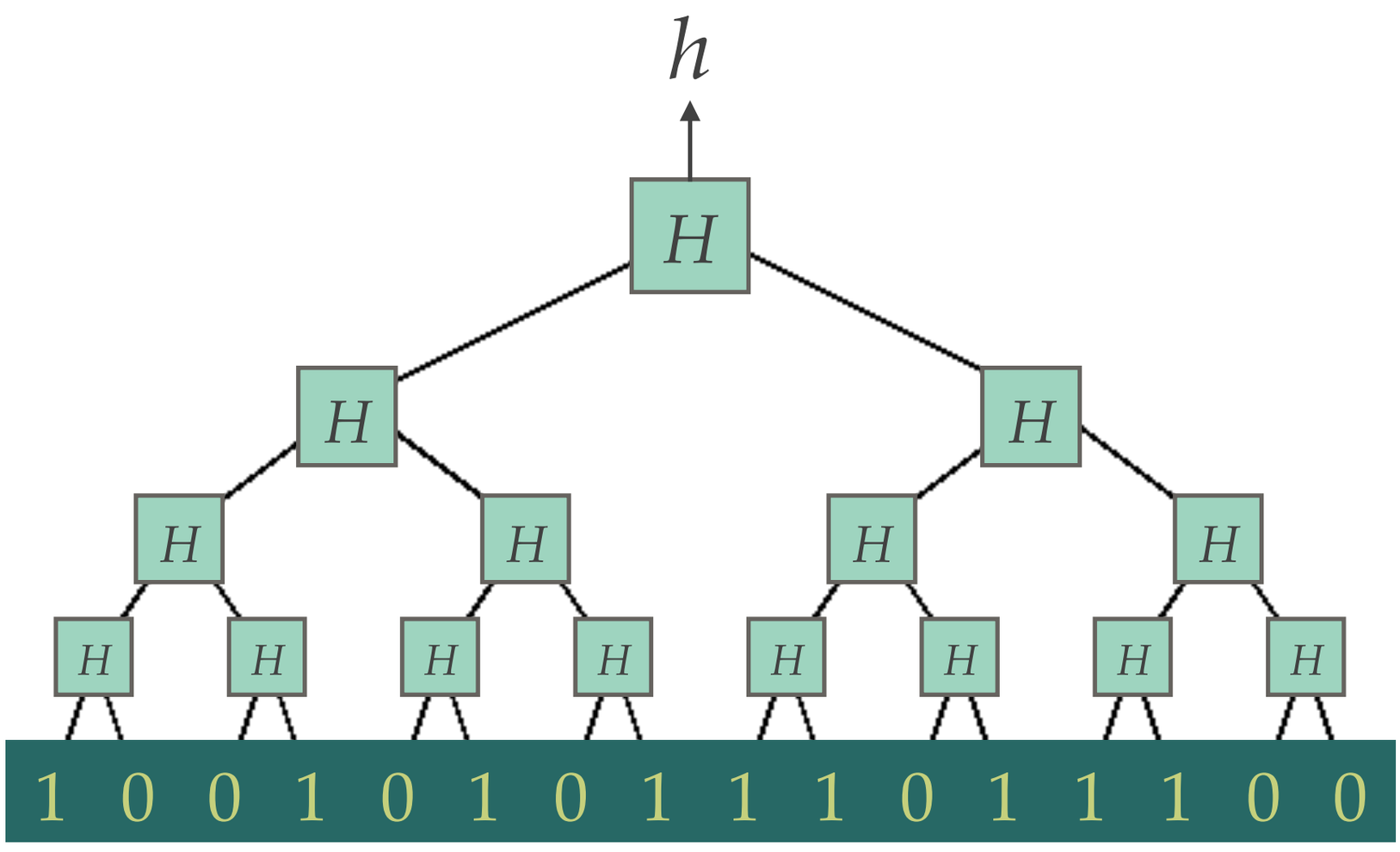
Prog-Iterate

K{M} = H, iO(Prog-Iterate), iO(Prog-Start)



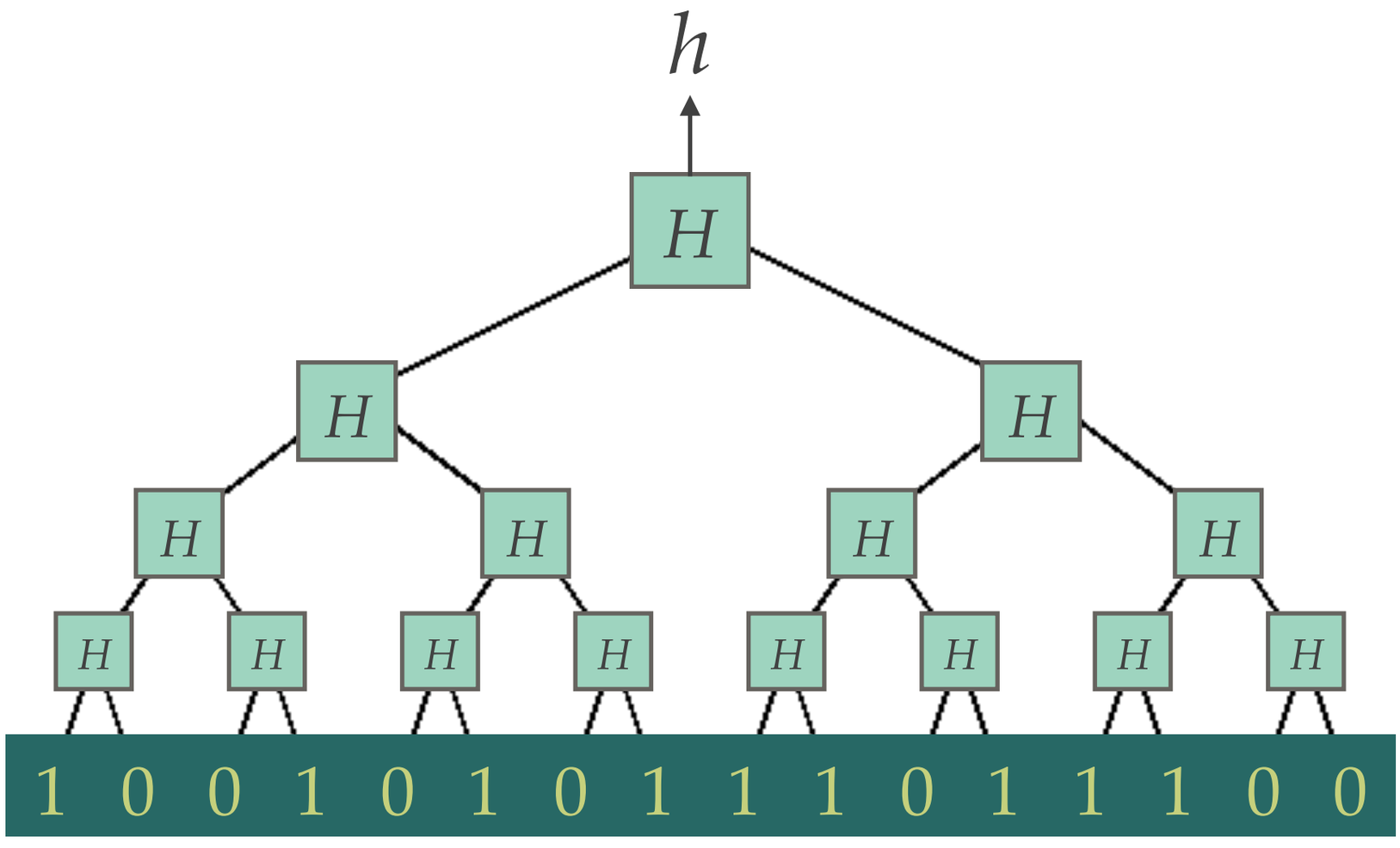
Prog-Iterate

Merkle Trees: Succinct Proof



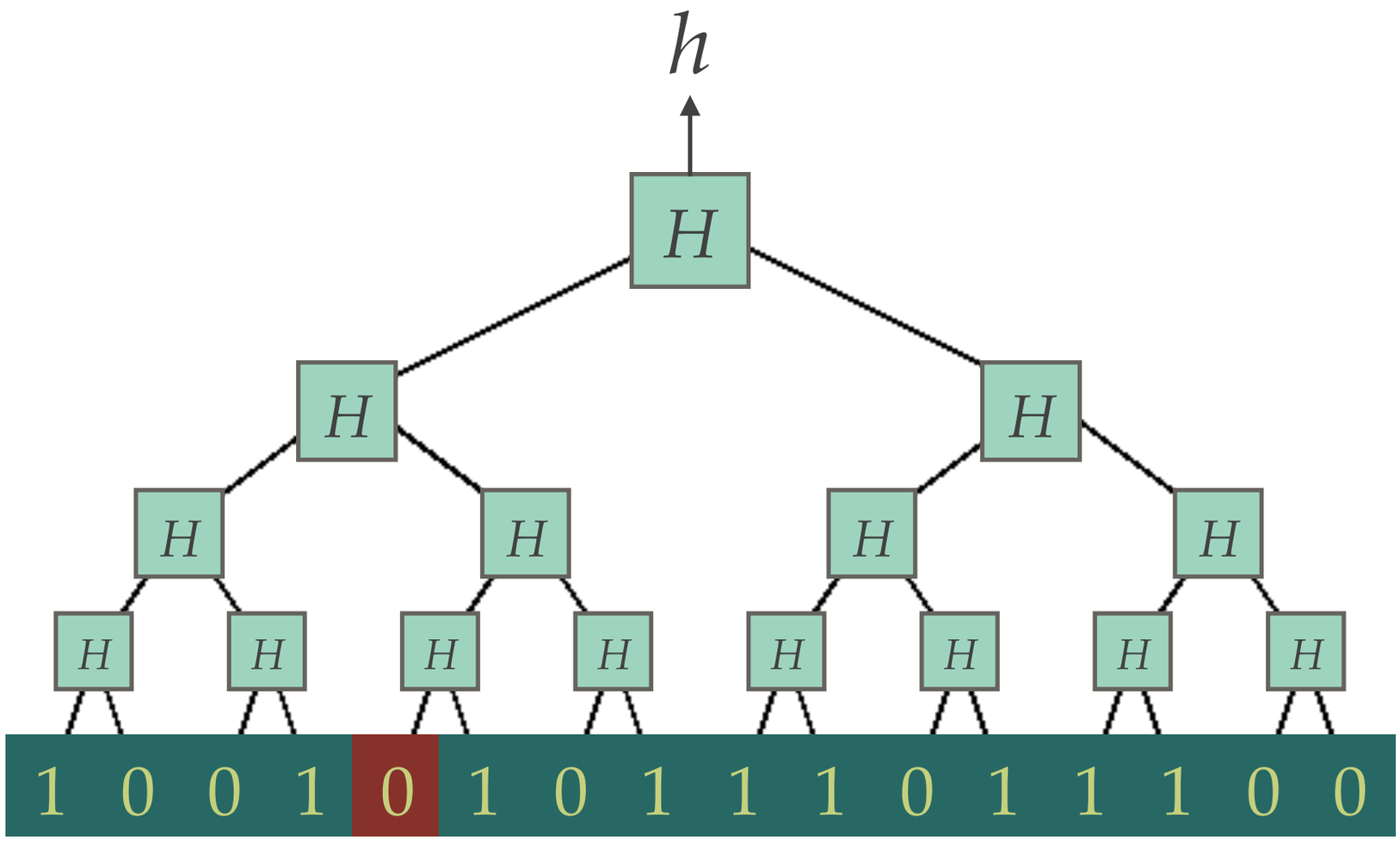
Merkle Trees: Succinct Proof

Prove m is at position p



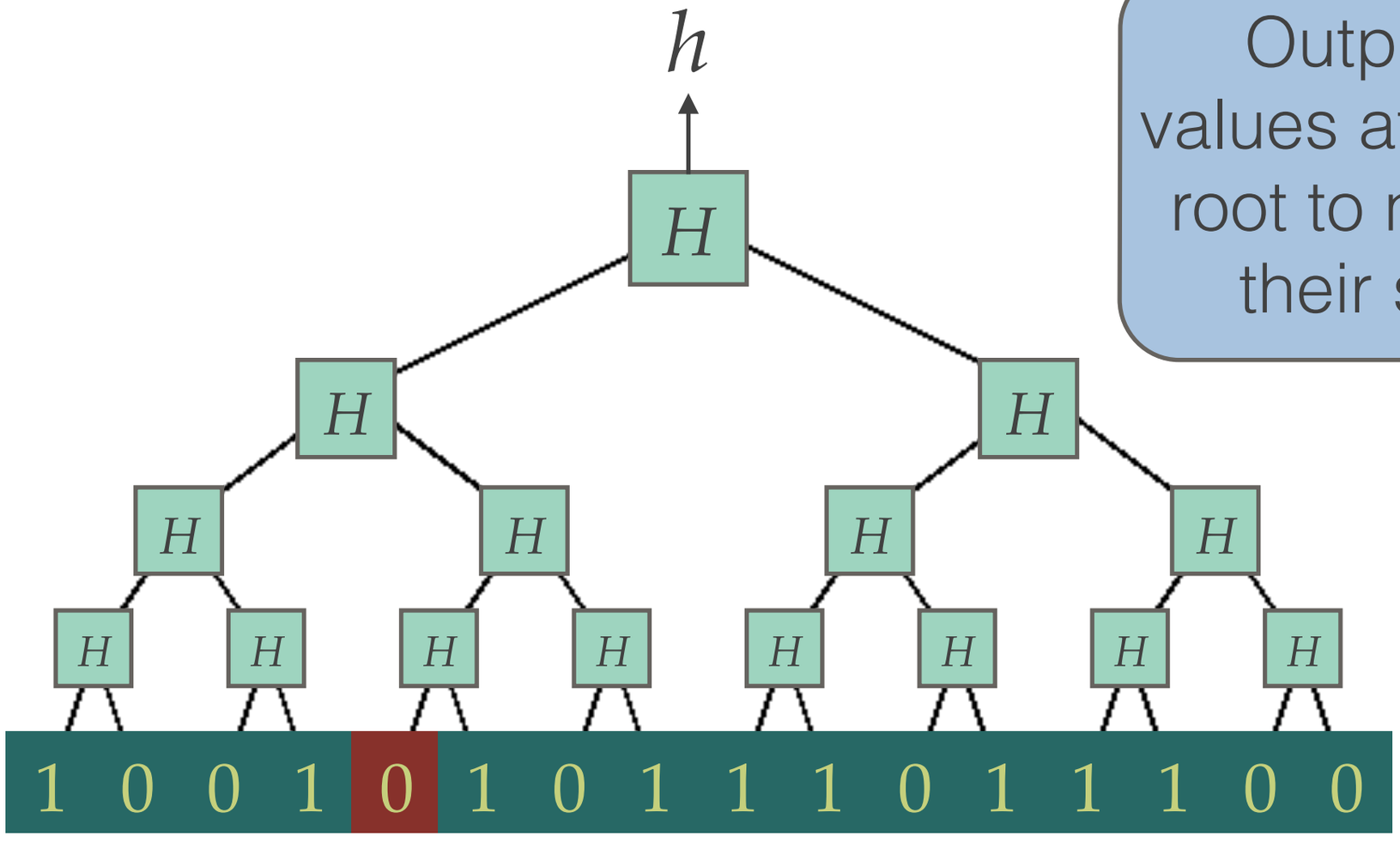
Merkle Trees: Succinct Proof

Prove m is at position p



Merkle Trees: Succinct Proof

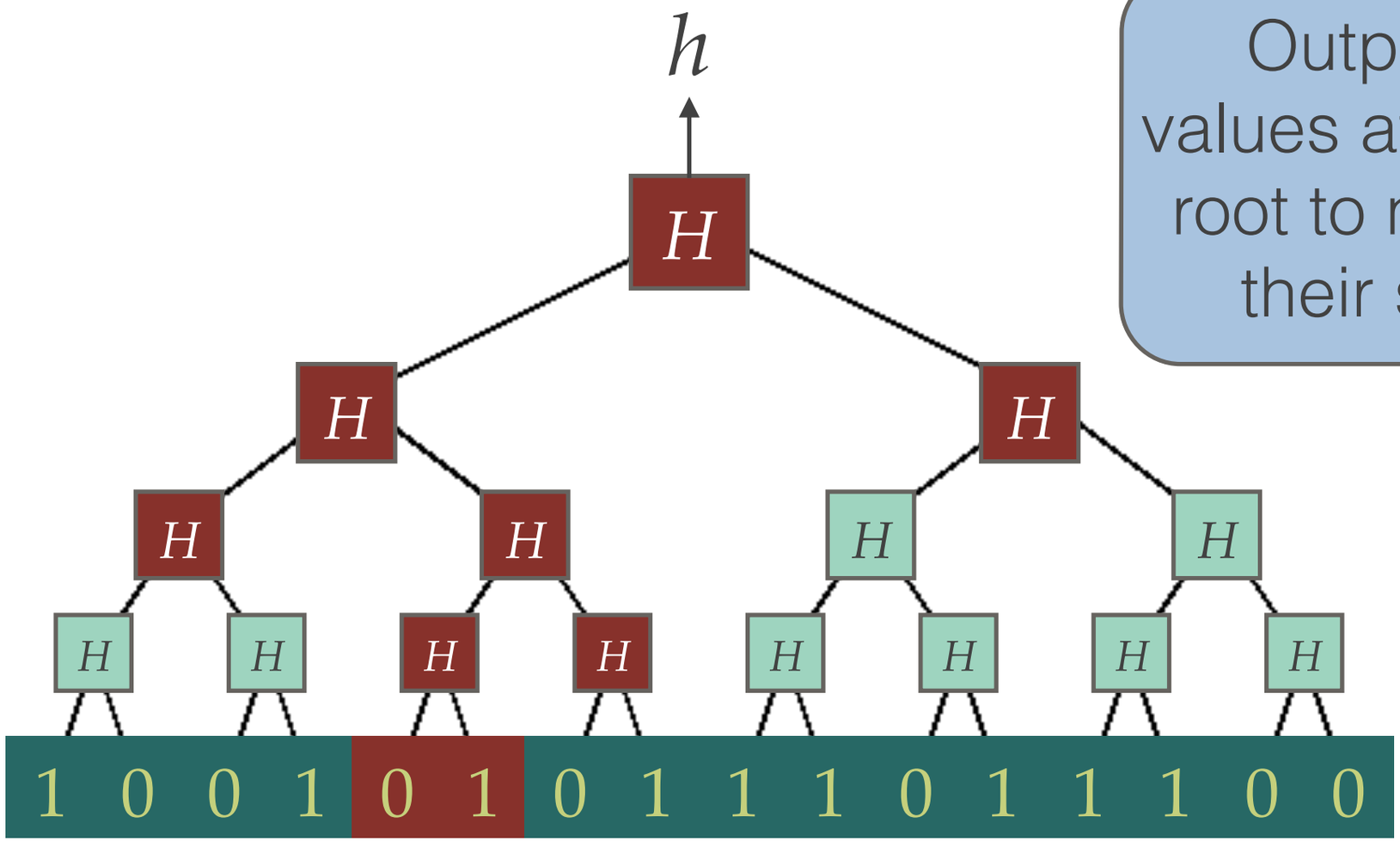
Prove m is at position p



Output hash values at path from root to node, and their siblings

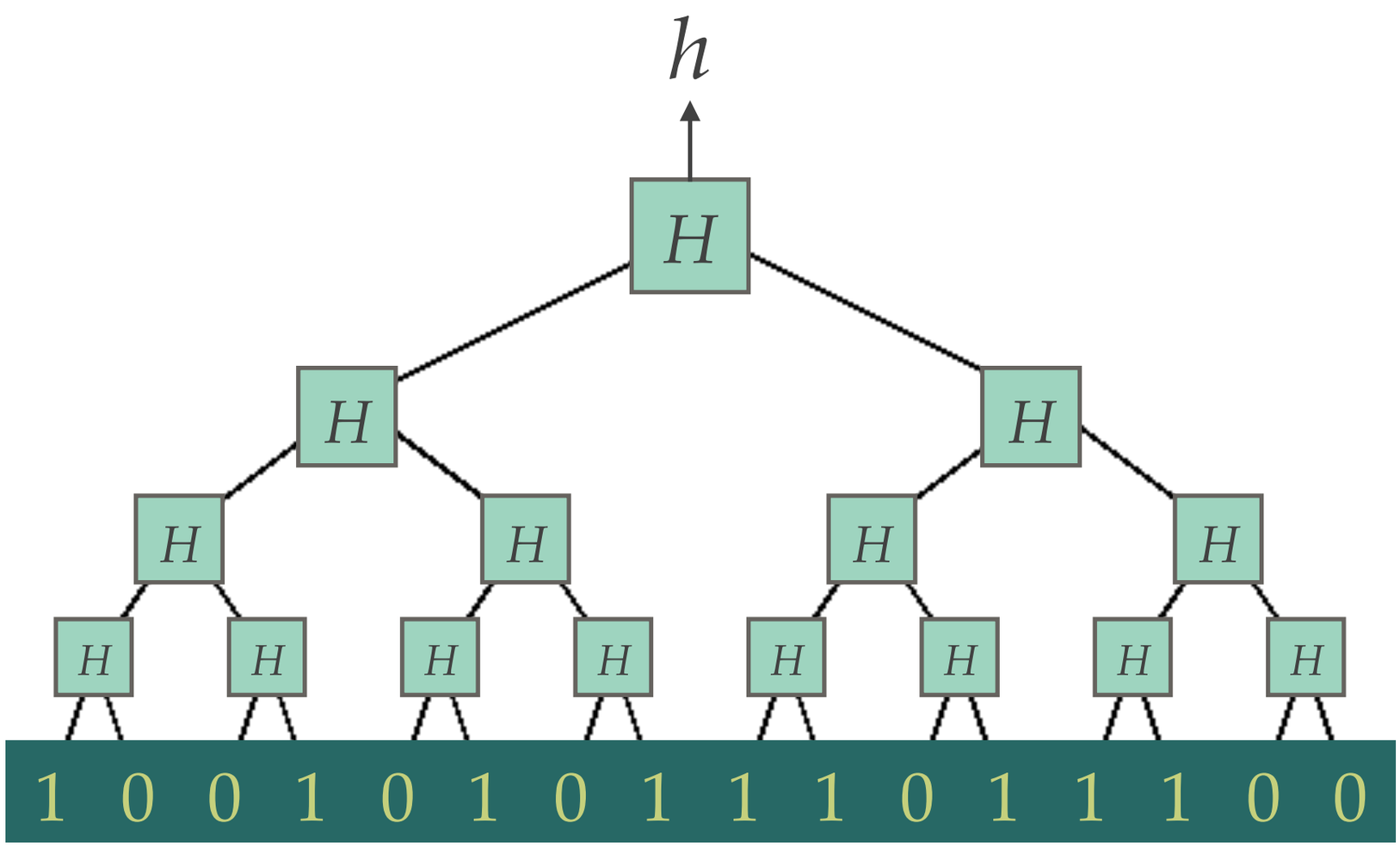
Merkle Trees: Succinct Proof

Prove m is at position p



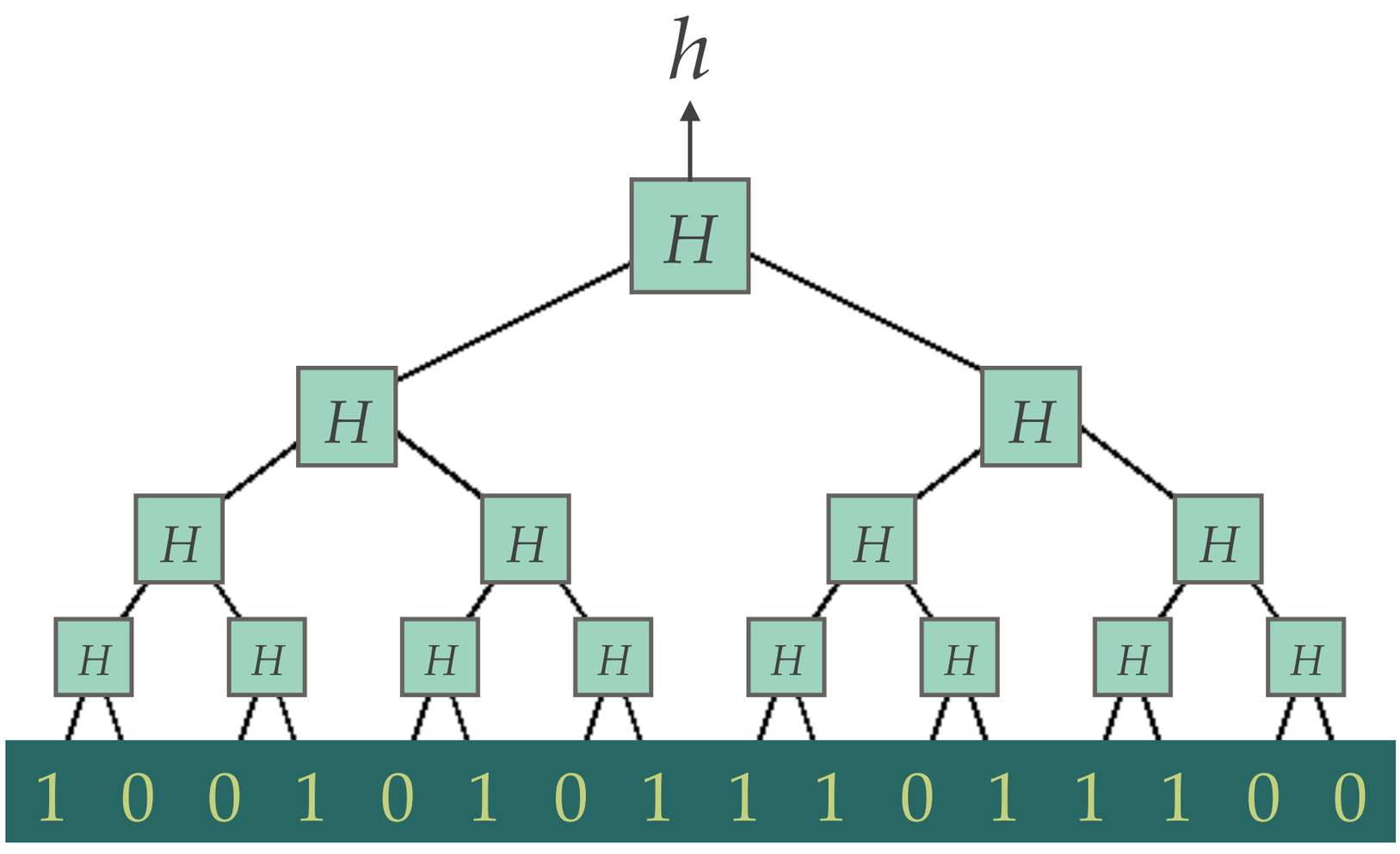
Output hash values at path from root to node, and their siblings

Merkle Trees: Update Hash



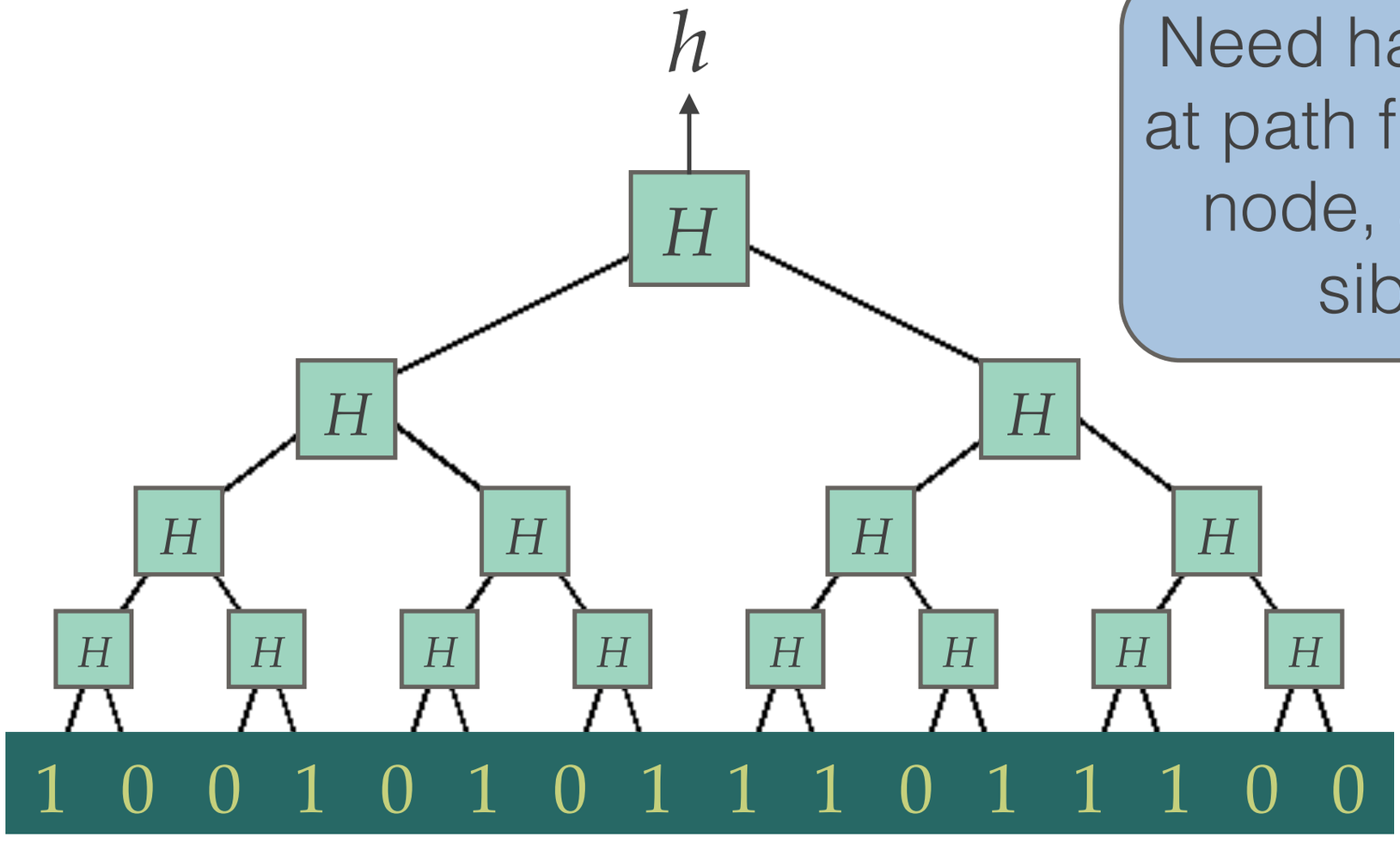
Merkle Trees: Update Hash

Update h to write m at position p



Merkle Trees: Update Hash

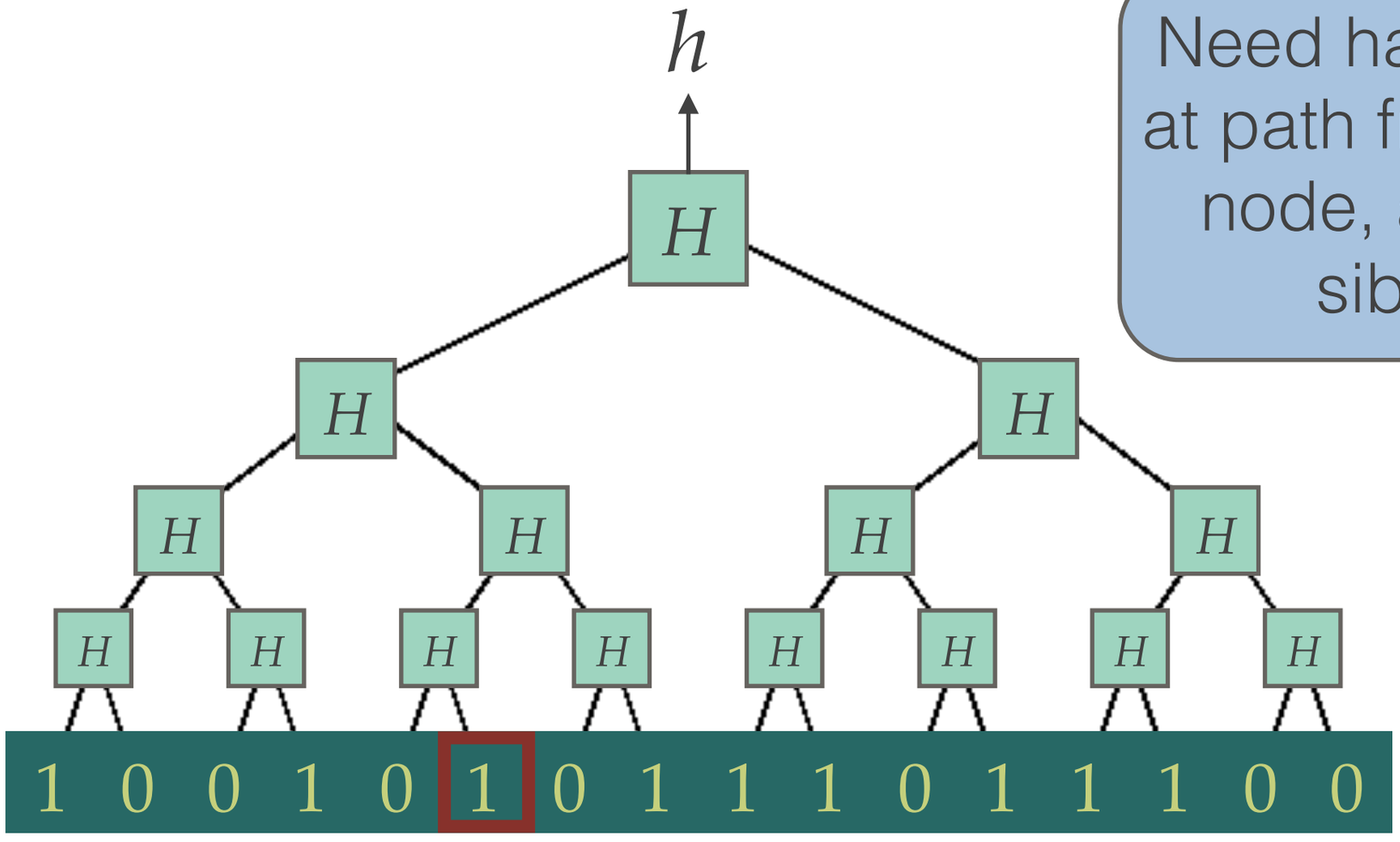
Update h to write m at position p



Need hash values at path from root to node, and their siblings

Merkle Trees: Update Hash

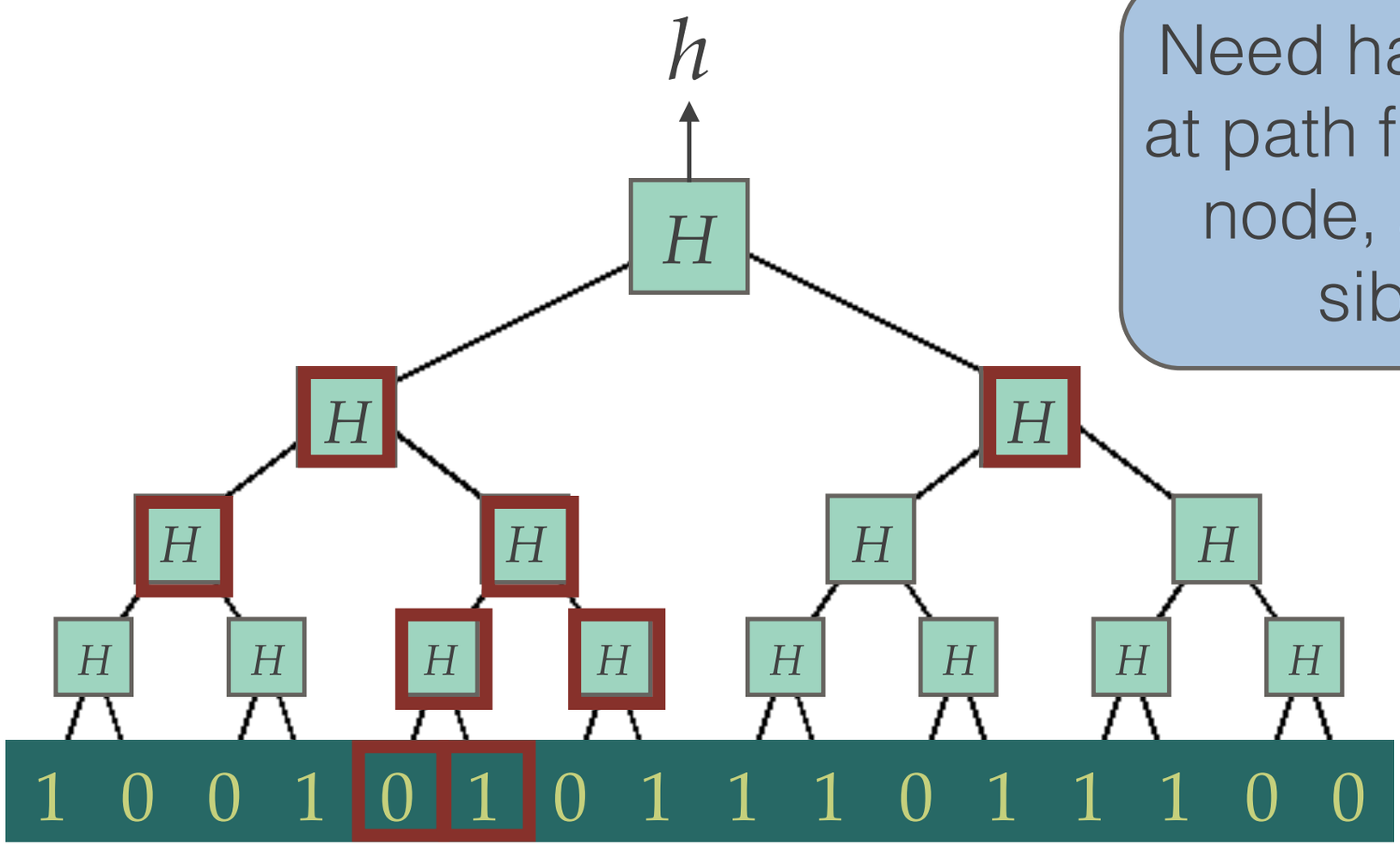
Update h to write m at position p



Need hash values at path from root to node, and their siblings

Merkle Trees: Update Hash

Update h to write m at position p



Need hash values at path from root to node, and their siblings

Proving Selective Security

Proving Selective Security

Chooses PRF key K.

Proving Selective Security

Chooses PRF key K.

x^* ←

$y^* = F(K, \text{hash of } x^*)$ or random →

Proving Selective Security

Chooses PRF key K.

v^*

x^*

$y^* = F(K, \text{hash of } x^*)$
or random

Proving Selective Security

Chooses PRF key K.

V^*

x^*

$y^* = F(K, \text{hash of } x^*)$
or random

M (s.t. $M(x^*) = 0$)

K{M}
= (H, iO(Prog-Start),
iO(Prog-Iterate))

Proving Selective Security

Chooses PRF key K.

V^*

x^*

$y^* = F(K, \text{hash of } x^*)$
or random

M (s.t. $M(x^*) = 0$)

$K\{M\}$
$= (H, iO(\text{Prog-Start}),$
$iO(\text{Prog-Iterate}))$

Guess PRF/random

Proving Selective Security

Chooses PRF key K.

V^*

$y^* = F(K, \text{hash of } x^*)$
or random

x^*

M (s.t. $M(x^*) = 0$)

$K\{M\}$
$= (H, iO(\text{Prog-Start}),$
$iO(\text{Prog-Iterate}))$

Contains PRF key K

Guess PRF/random

Proving Selective Security

Using h, verify sym at position p

Verify sig on st and h

If st' = final, output F(K, v)

Using sym', update h to h'

Output signature on st' and h'

Prog-Iterate

Proving Selective Security

Using h, verify sym at position p

Verify sig on st and h

If st' = final, output F(K, v)

Using sym', update h to h'

Output signature on st' and h'

Prog-Iterate

Using h, verify sym at position p

Verify sig on st and h

If st' = final and v=v*, output ⊥

Else if st' = final, output F(K{v*}, v)

Using sym', update h to h'

Output signature on st' and h'

Prog-Iterate'

Proving Selective Security

Using h, verify sym at position p

Verify sig on st and h

If st' = final, output F(K, v)

Using sym', update h to h'

Output signature on st' and h'

Prog-Iterate

???

≈

Using h, verify sym at position p

Verify sig on st and h

If st' = final and v=v*, output ⊥

Else if st' = final, output F(K{v*}, v)

Using sym', update h to h'

Output signature on st' and h'

Prog-Iterate'

Need iO-Compatible Primitives

Primitives Required for our Construction:

(Collision resistant) Hash functions

Signature Schemes

Indistinguishability Obfuscation

Need iO-Compatible Primitives

Primitives Required for our Construction:

~~(Collision resistant) Hash functions~~

Positional Accumulators

~~Signature Schemes~~

Splittable Signature Schemes

Indistinguishability Obfuscation

Need iO-Compatible Primitives

Primitives Required for our Construction:

~~(Collision resistant) Hash functions~~

Positional Accumulators

~~Signature Schemes~~

Splittable Signature Schemes

Indistinguishability Obfuscation

iO for Turing Machines
(K, Lewko, Waters 14)

Proving Selective Security

Using h, verify sym at position p

Verify sig on st and h

st, sym → Next-Step → st’, sym’

If st’ = final, output F(K, v)

Using sym’, update h to h’

Output signature on st’ and h’

Prog-Iterate

KLW techniques

≈

Using h, verify sym at position p

Verify sig on st and h

st, sym → Next-Step → st’, sym’

If st’ = final and v=v*, output ⊥

Else if st’ = final, output F(K{v*}, v)

Using sym’, update h to h’

Output signature on st’ and h’

Prog-Iterate’

Conclusions

Conclusions

- Constrained PRFs for Turing machines based on iO and one way functions

Conclusions

- Constrained PRFs for Turing machines based on iO and one way functions
- Unbounded broadcast encryption, unbounded ID-NIKE from iO and OWFs
  - unbounded broadcast encryption - Zhandry 14
  - unbounded ID-NIKE - Khurana, Rao, Sahai 15

Conclusions

- Constrained PRFs for Turing machines based on iO and one way functions
- Unbounded broadcast encryption, unbounded ID-NIKE from iO and OWFs
    - unbounded broadcast encryption - Zhandry 14
    - unbounded ID-NIKE - Khurana, Rao, Sahai 15
- Attribute Based Encryption for Turing machines
    - Functional Encryption for Turing machines - Ananth, Sahai 15

Danke!