

# Proving Computational Soundness of the Applied Pi-Calculus without Using Computable Parsing

Staying at LSV, ENS Cachan

Yusuke Kawamoto<sup>1 3</sup>

(Joint work with Hubert Comon-Lundh<sup>3</sup>, Masami Hagiya<sup>2</sup> and Hideki Sakurada<sup>4</sup>)

<sup>1</sup> Research Fellow of the Japan Society for the Promotion of Science

<sup>2</sup> University of Tokyo

<sup>3</sup> ENS Cachan

<sup>4</sup> NTT Communication Science Labs

# Approaches to protocol verification

	formal approach	computational approach
Parties & attackers	symbolic process	PPT Turing machine
Messages	terms	bit strings
Attacker's capabilities	equation theory ( <b>not</b> faithful to the real world)	PPT computation ( <b>more</b> faithful to the real world)

**green:** formal approach

**blue:** computational approach

# Approaches to protocol verification

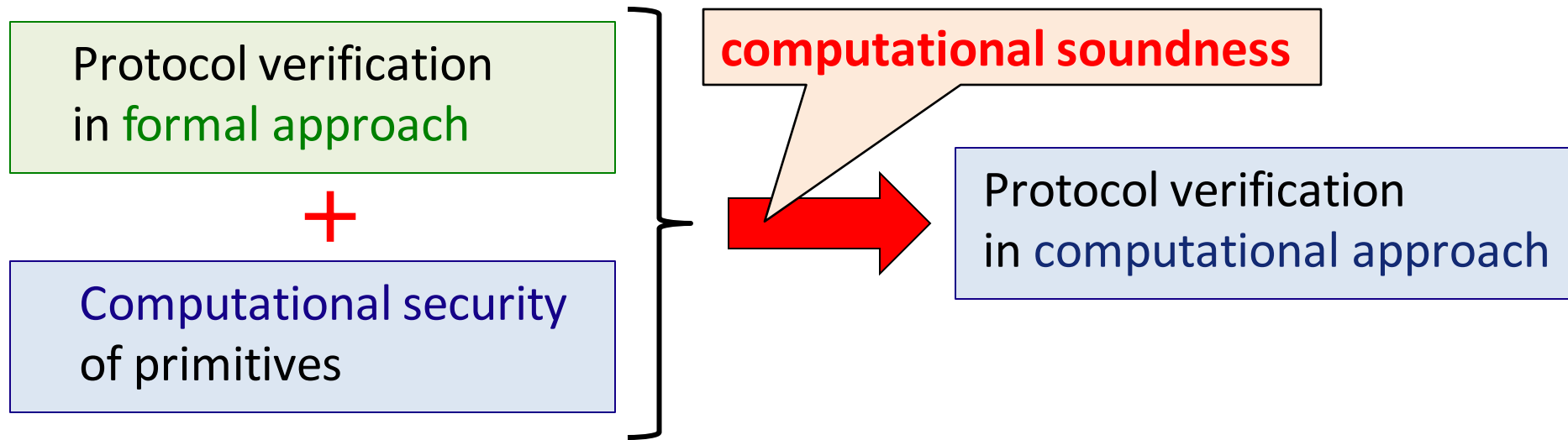
	formal approach	computational approach
Parties & attackers	symbolic process	PPT Turing machine
Messages	terms	bit strings
Attacker's capabilities	equation theory ( <b>not</b> faithful to the real world)	PPT computation ( <b>more</b> faithful to the real world)
Cryptographic assumption	Cryptography is not broken	With a negligible probability, cryptography is broken in poly time
Protocol verification	Might miss some attacks Simple to automate	Consider attacks breaking crypto Difficult and error-prone

**green:** formal approach

**blue:** computational approach

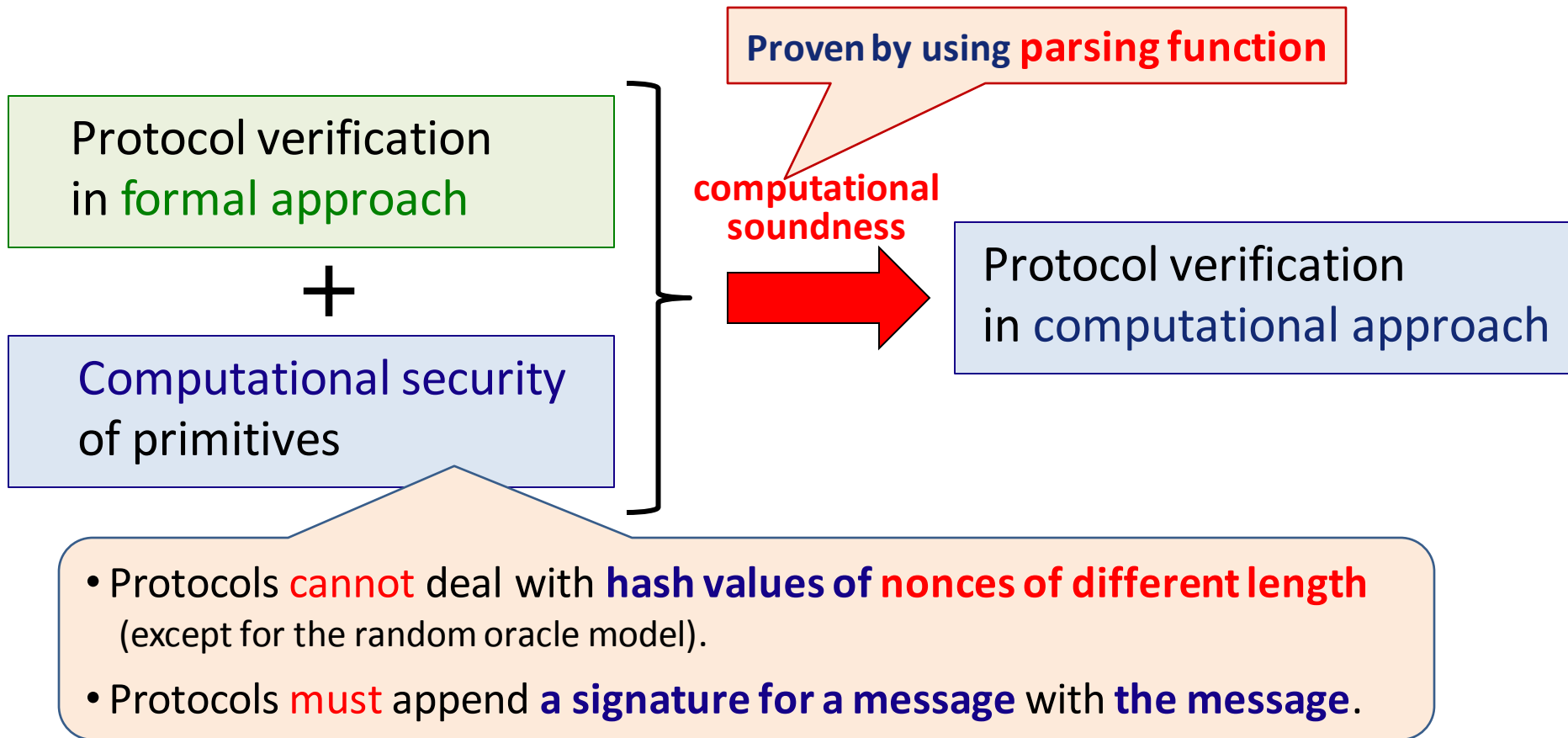
# Computational soundness of formal verification

[Abadi,Rogaway'00], [Baudet,Cortier,Kremer'05], [Comon,Cortier'08] .....

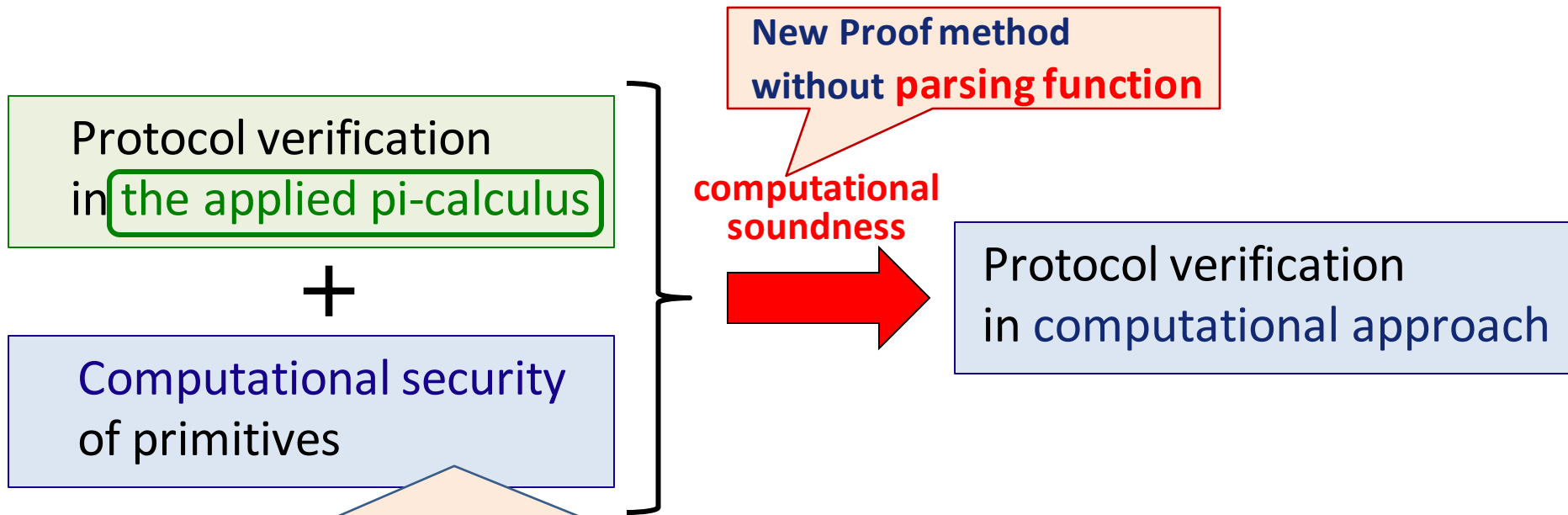


- If a **formal approach** is **computationally sound**, then **the security of protocols in the formal approach implies that in the computational approach**, provided that the primitives satisfy a certain computational security.
- Computational soundness is important to use existing **formal approaches**.

# Previous work



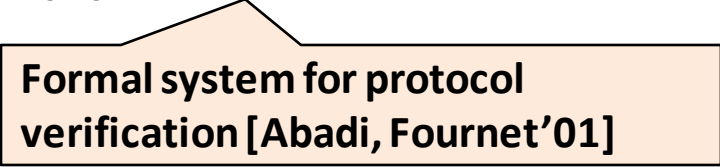
# Our work



- Protocols **may** deal with **hash values of nonces of different length** (assuming only preimage-resistance and collision-resistance).
- Protocols **may not** append **a signature for a message** with **the message**.

# Outline of this talk


- Applied Pi-Calculus

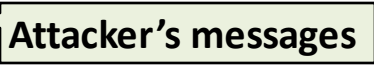
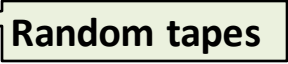
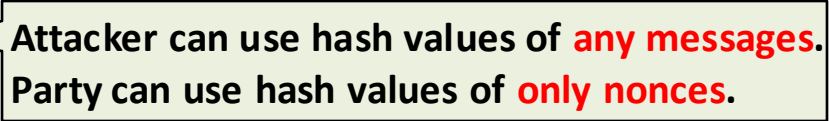
-  Formal system for protocol verification [Abadi, Fournet'01]

ational equivalence

- Overview of soundness proof
- Comparison of proof methods
- Summary and future work

# Syntax of applied pi-calculus

- Terms 

$u ::= y$	Variables 
$r$	Names 
$n^1(r)$	Nonces
$n^2(r)$	Nonces (of different length)
$h(u)$	Hash values 
$\{u\}_{ek(k)}$	Ciphertexts
$dec(u, dk(k))$	Decryption
$\vdots$	


- Equations between terms

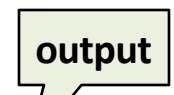
- $dec(\{u\}_{ek(k)}, dk(k)) = u$
  - $\vdots$

# Syntax of applied pi-calculus

- Basic processes

$$B ::= 0$$

  $| i_{in}(y). B$

  $| \text{if } M(u) \text{ then } \overline{i_{out}}(u). B \text{ else } \overline{i_{out}}(*).0$

$$| \text{if } \Phi \text{ then } B_1 \text{ else } B_2$$

(Termination)

(Input  $y$  and execute  $B$ )

(Check and output  $u$ )

(Branching)

- Processes (protocols)

$$P ::= \nu \bar{x}. \nu \bar{n}. c_B(x'). \bar{c}_B(\langle i_{in}, i_{out} \rangle). B$$
$$| \nu \bar{n}. P$$
$$| !P$$
$$| P_1 || P_2$$

(Generate channels and execute  $B$ )

(Generate names  $\bar{n}$  and execute  $P$ )

(Replication of process  $P$ )

(Concurrent execution of processes)

# Outline of this talk

- Applied Pi-Calculus
- **Soundness of observational equivalence**
- Overview of soundness proof
- Comparison of proof methods
- Summary and future work

# Example: formalization of security of protocols

Verification of anonymity  
by applied pi-calculus

Prove observational equivalence

$$P_0 \sim P_1$$

A votes for Candidate<sup>1</sup>  
B votes for Candidate<sup>2</sup>

A votes for Candidate<sup>2</sup>  
B votes for Candidate<sup>1</sup>

Verification of anonymity  
by computational approach

Prove comp. indistinguishability

$$[[P_0]] \approx [[P_1]]$$

A votes for Candidate<sup>1</sup>  
B votes for Candidate<sup>2</sup>

A votes for Candidate<sup>2</sup>  
B votes for Candidate<sup>1</sup>

# Formalization of security of protocols

## Verification of anonymity by applied pi-calculus

Prove observational equivalence

$$P_0 \sim P_1$$

↕ def

For any process  $A$ ,  
 $(P_0 \parallel A) \downarrow_c \Leftrightarrow (P_1 \parallel A) \downarrow_c$

Attacker's process

Attacker's  
observation  
(Receiving messages)

## Verification of anonymity by computational approach

Prove comp. indistinguishability

$$[[P_0]] \approx [[P_1]]$$

↕ def

For any probabilistic polynomial-time Turing Machine  $A$ ,  
 $\Pr[ [[P_b]] \parallel A \rightarrow b' : b' = b ] - \frac{1}{2}$   
is negligible.

# Computational soundness of applied pi-calculus

[Comon, Cortier'08], [Our work]

Verification of anonymity  
by applied pi-calculus

Prove observational equivalence

$$P_0 \sim P_1$$

↕ def

For any process  $A$ ,  
 $(P_0 \parallel A) \downarrow_c \iff (P_1 \parallel A) \downarrow_c$

Attacker's process

Attacker's  
observation  
(Receiving messages)

computational  
soundness

Verification of anonymity  
by computational approach

Prove comp. indistinguishability

$$[[P_0]] \approx [[P_1]]$$

↕ def

For any probabilistic polynomial-time Turing Machine  $A$ ,  
 $\Pr[ [[P_b]] \parallel A \rightarrow b' : b' = b ] - \frac{1}{2}$   
is negligible.

# Outline of this talk

- Applied Pi-Calculus
- Soundness of observational equivalence
- **Overview of soundness proof**
- Comparison of proof methods
- Summary and future work

# Computation tree for soundness proof

Parties' process  $P$

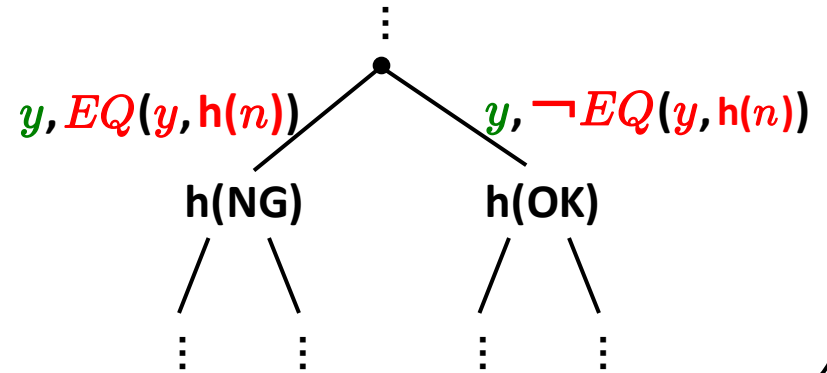
...  $c(y)$ . if  $EQ(y, h(n))$   
then  $\bar{c}(h(NG))$  ...  
else  $\bar{c}(h(OK))$  ...

Party receives  
a message  $y$   
from Attacker

Party check whether  
 $y$  is a hash value of  $n$



Computation tree  $t_P$



# Computation tree for soundness proof

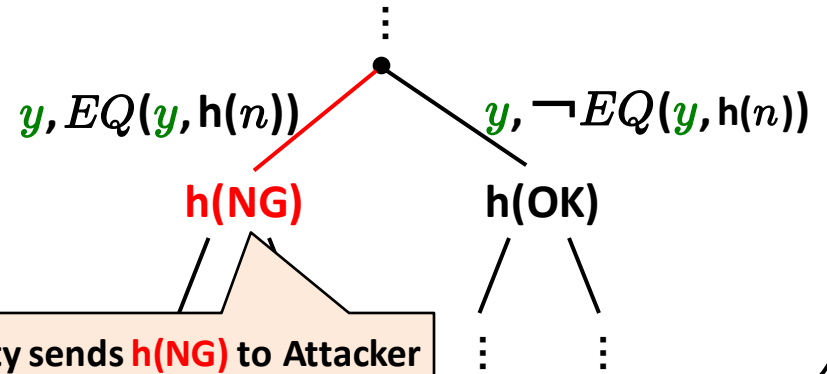
Parties' process  $P$

...  $c(y)$ . if  $EQ(y, h(n))$   
then  $\overline{c}(h(NG))$  ...  
else  $\overline{c}(h(OK))$  ...

Party sends  $h(NG)$



Computation tree  $t_P$



# Computation tree for soundness proof

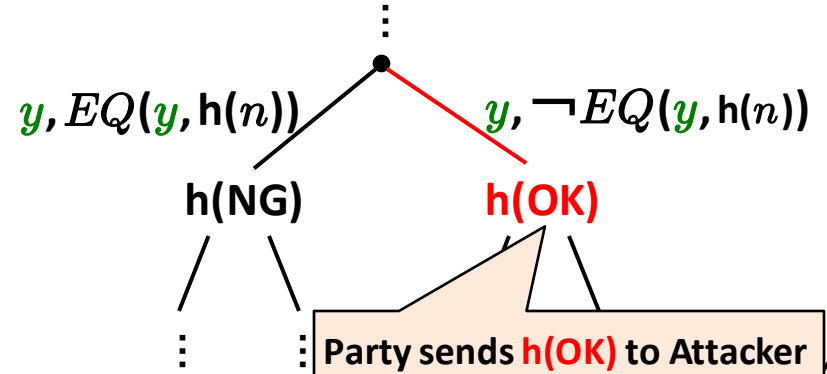
Parties' process  $P$

...  $c(y)$ . if  $EQ(y, h(n))$   
then  $\overline{c}(h(NG))$  ...  
else  $\overline{c}(h(OK))$  ...

Party sends  $h(OK)$

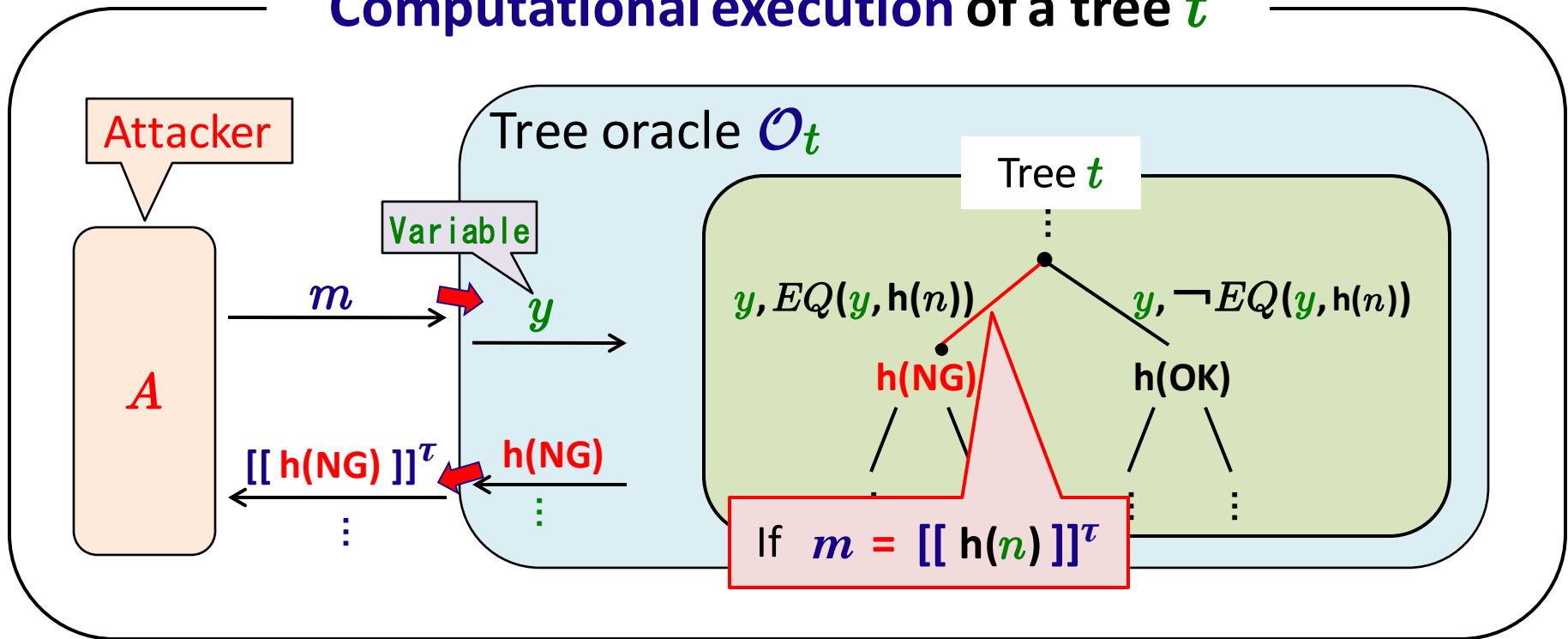


Computation tree  $t_P$



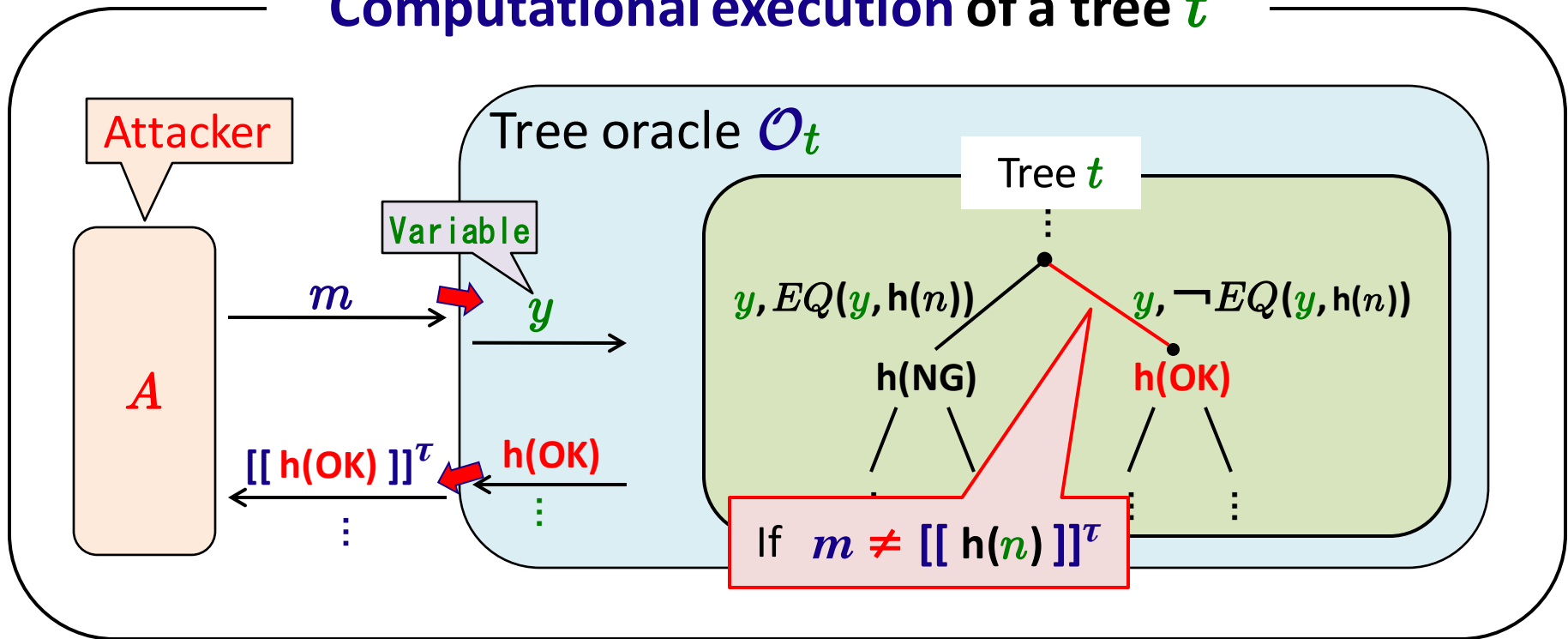
# Tree oracle for soundness proof

## Computational execution of a tree $t$



# Tree oracle for soundness proof

## Computational execution of a tree $t$



# Overview of soundness proof

- Assume  $P \sim Q$ .

Observational equivalence

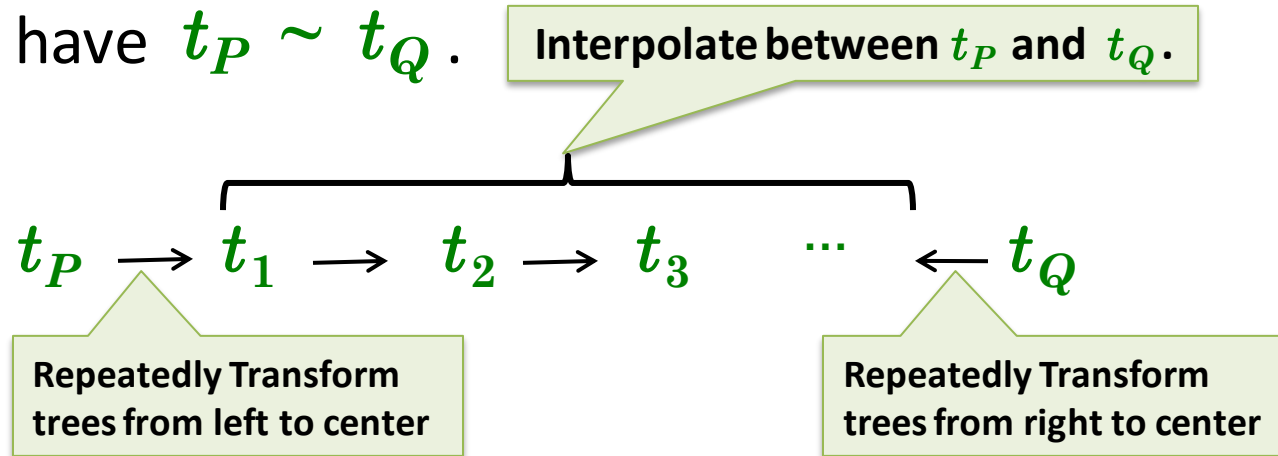
- Then, we have  $t_P \sim t_Q$ .

Corresponds to observational equivalence between processes

# Overview of soundness proof

- Assume  $P \sim Q$ .

- Then, we have  $t_P \sim t_Q$ .



# Overview of soundness proof

- Assume  $P \sim Q$ .
- Then, we have  $t_P \sim t_Q$ .

$$t_P \rightarrow \dots \rightarrow t_n \quad t_{n+1} \leftarrow \dots \leftarrow t_Q$$

- We want to prove

Computational indistinguishability

$[[P]]$

$\approx$

$[[Q]]$

# Overview of soundness proof

- Assume  $P \sim Q$ .
- Then, we have  $t_P \sim t_Q$ .

$$t_P \rightarrow \dots \rightarrow t_n \quad t_{n+1} \leftarrow \dots \leftarrow t_Q$$

- We want to prove

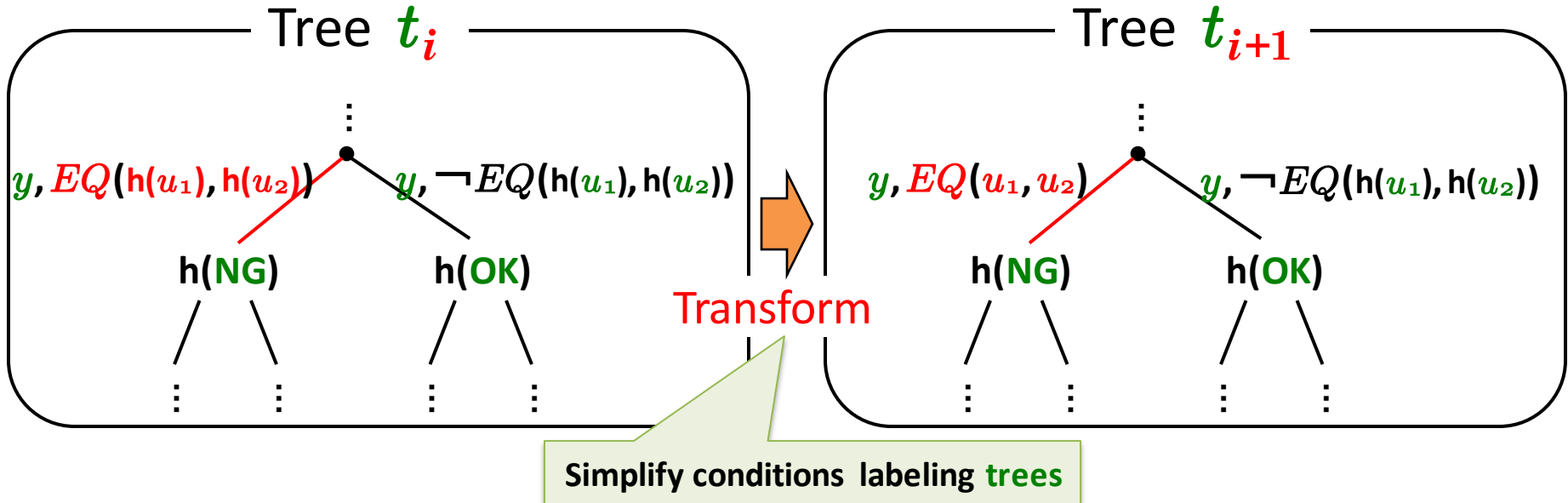
$$[[P]] = \mathcal{O}_{t_P} \approx \dots \approx \mathcal{O}_{t_n} \approx \mathcal{O}_{t_{n+1}} \approx \dots \approx \mathcal{O}_{t_Q} = [[Q]]$$

Computational execution  $[[P]]$  of  $P$

Computational execution  $[[Q]]$  of  $Q$

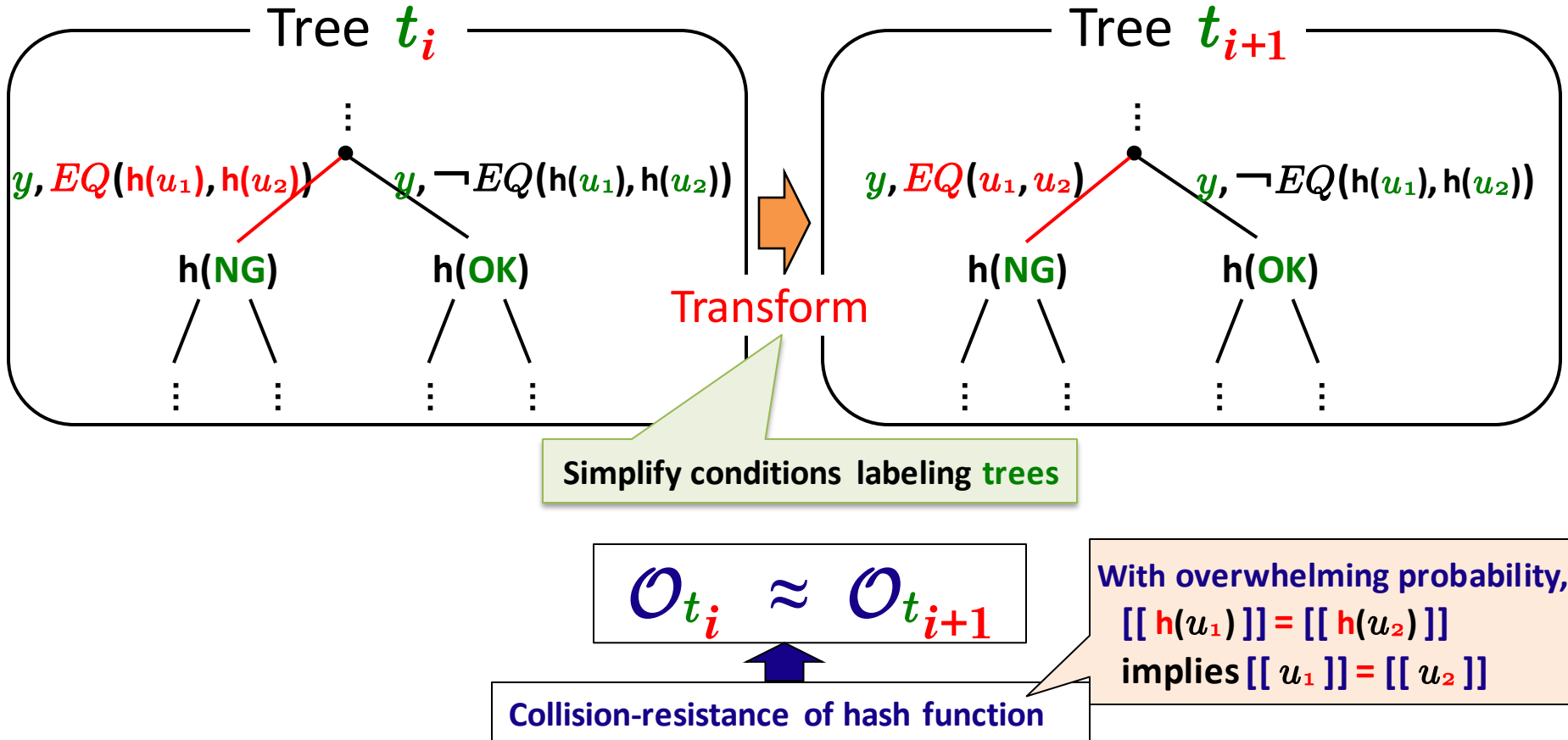
# Overview of soundness proof

## Example of transformation 1



# Overview of soundness proof

## Example of transformation 1

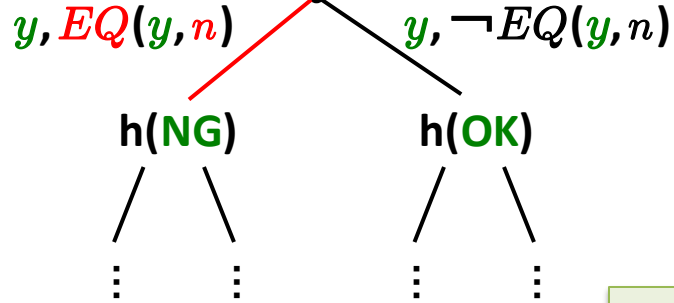


# Overview of soundness proof

## Example of transformation 2

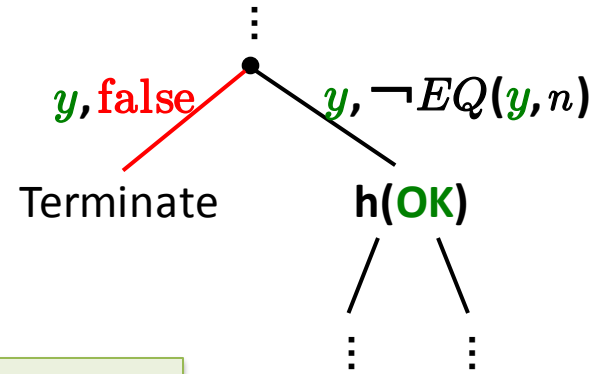
Attacker has received the hash value  $h(n)$  of Party's nonce  $n$ , but not  $n$ .

Tree  $t_j$



Transform

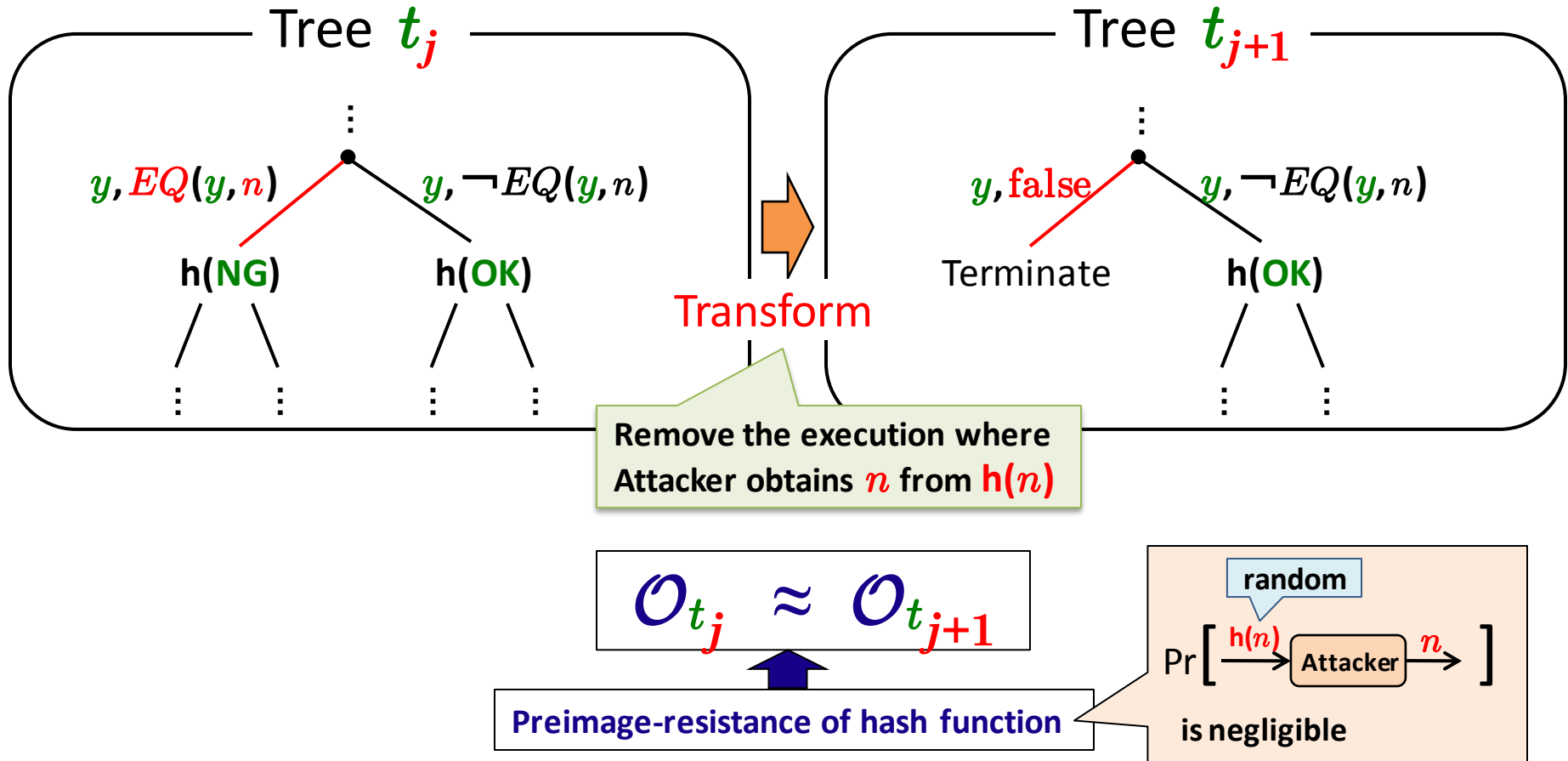
Tree  $t_{j+1}$



Remove the execution where Attacker obtains  $n$  from  $h(n)$

# Overview of soundness proof

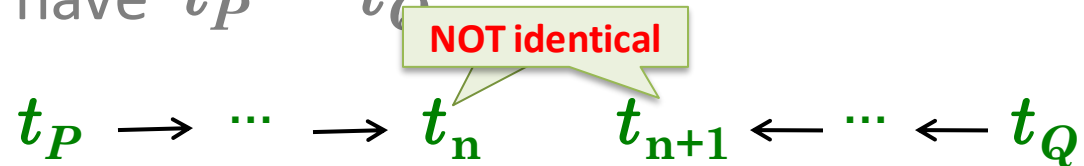
## Example of transformation 2



# Overview of soundness proof

- Assume  $P \sim Q$ .

- Then, we have  $t_P \sim t_Q$



- We want to prove

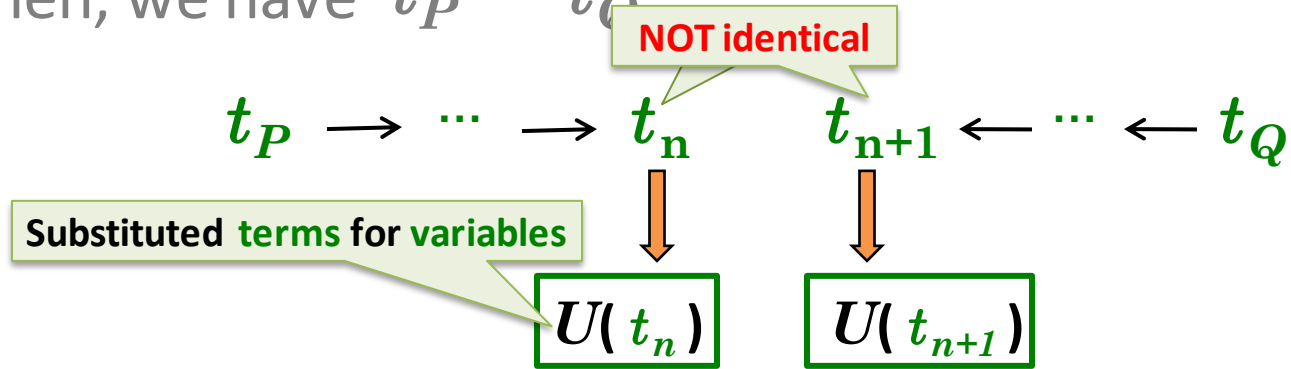
$$[[P]] = \mathcal{O}_{t_P} \approx \dots \approx \boxed{\mathcal{O}_{t_n}} \approx \boxed{\mathcal{O}_{t_{n+1}}} \approx \dots \approx \mathcal{O}_{t_Q} = [[Q]]$$

Need to prove

# Overview of soundness proof

- Assume  $P \sim Q$ .

- Then, we have  $t_P \sim t_Q$



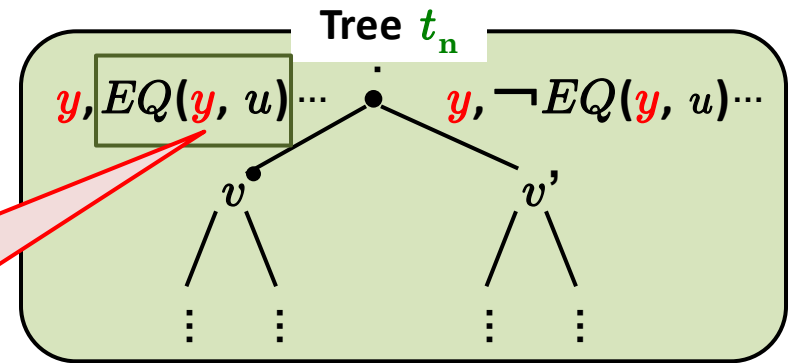
- We want to prove

$$[[P]] = \mathcal{O}_{t_P} \approx \dots \approx \boxed{\mathcal{O}_{t_n}} \approx \boxed{\mathcal{O}_{t_{n+1}}} \approx \dots \approx \mathcal{O}_{t_Q} = [[Q]]$$

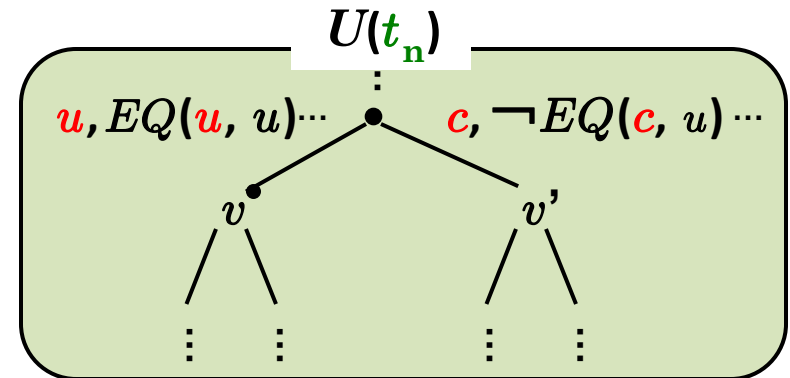
Need to prove

# Overview of soundness proof

Owing to the transformation of conditions, for each variable  $y$ , we finally obtain a subcondition  $EQ(y, u)$  for some term  $u$ .



Substitute terms for variables



# Overview of soundness proof

- Assume  $P \sim Q$ .
- Then, we have  $t_P \sim t_Q$ .

$$t_P \rightarrow \dots \rightarrow t_n \quad t_{n+1} \leftarrow \dots \leftarrow t_Q$$

Substituted **terms** for **variables**

**Identical** up to renaming

$$U(t_n) \approx U(t_{n+1})$$

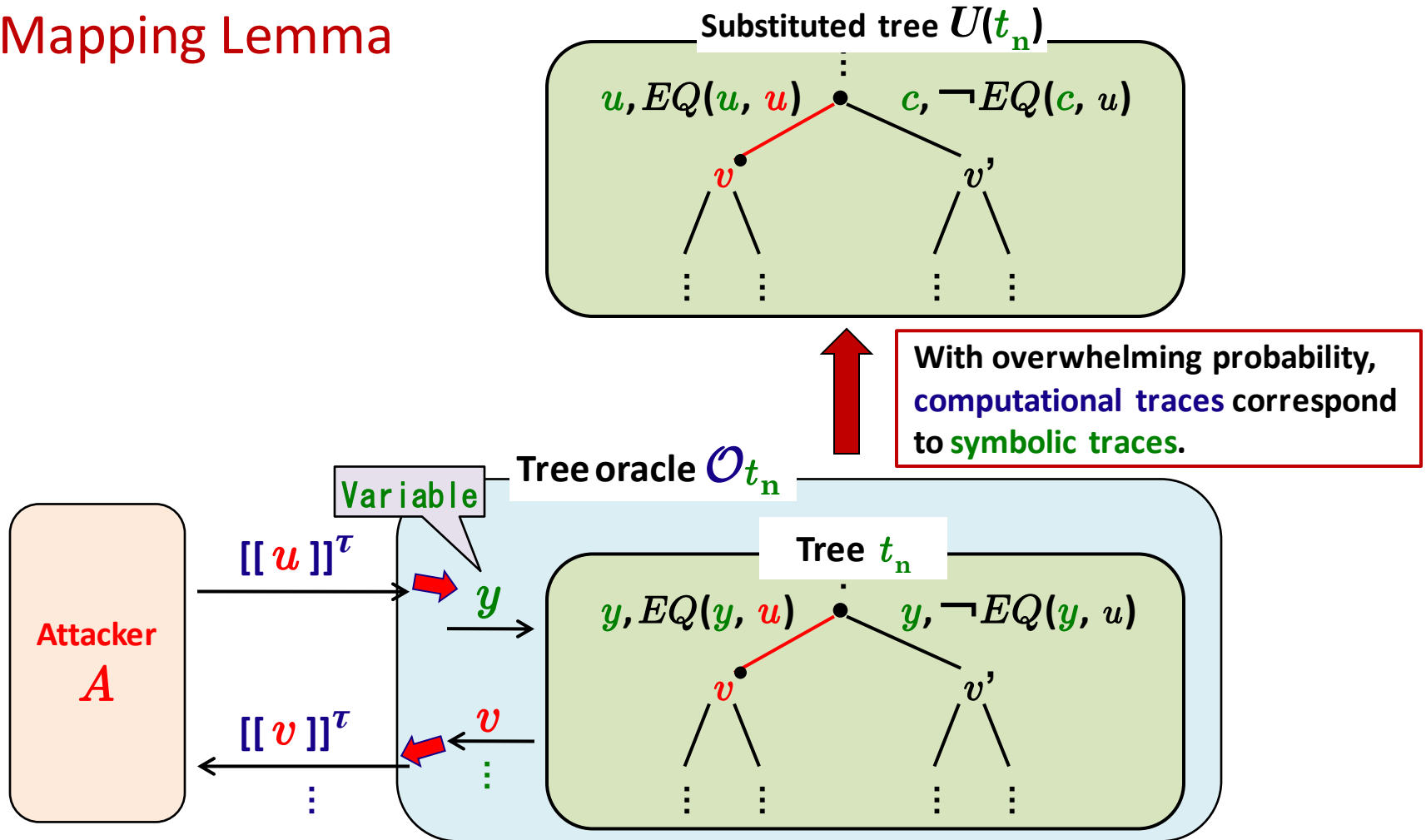
- We want to prove

$$[[P]] = \mathcal{O}_{t_P} \approx \dots \approx \mathcal{O}_{t_n} \quad \mathcal{O}_{t_{n+1}} \approx \dots \approx \mathcal{O}_{t_Q} = [[Q]]$$

Mapping Lemma

# Overview of soundness proof

- Mapping Lemma



# Overview of soundness proof

- Assume  $P \sim Q$ .
- Then, we have  $t_P \sim t_Q$ .

$$t_P \rightarrow \dots \rightarrow t_n \quad t_{n+1} \leftarrow \dots \leftarrow t_Q$$

Substituted **terms** for **variables**

**Identical** up to renaming

$$U(t_n) \approx U(t_{n+1})$$

- We want to prove

$$[[P]] = \mathcal{O}_{t_P} \approx \dots \approx \mathcal{O}_{t_n} \approx \mathcal{O}_{t_{n+1}} \approx \dots \approx \mathcal{O}_{t_Q} = [[Q]]$$

**Mapping Lemma**

Probability distributions are **identical**

# Outline of this talk

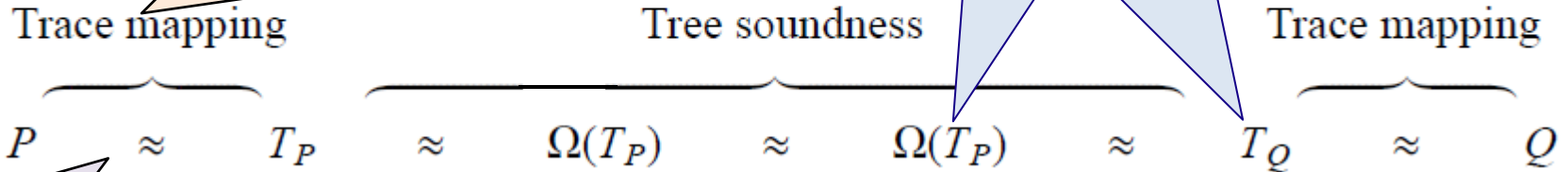
- Applied Pi-Calculus
- Soundness of observational equivalence
- Overview of soundness proof
- **Comparison of proof methods**
- Summary and future work

# Comparison of two soundness proofs

**Proof in [Comon, Coriter'08]**

Remove **all** computational traces not corresponding to **symbolic traces**

Computational traces **correspond** to **symbolic traces**.



Parse bit strings into **terms**

**Proof in our work**

$$P = t_P \approx \Psi(\Omega(t_P)) \approx \Delta(\text{Simp}(\Psi(\Omega(t_P)))) \approx \Delta(\text{Simp}(\Psi(\Omega(t_Q)))) \approx \Psi(\Omega(t_Q)) \approx t_Q = Q$$

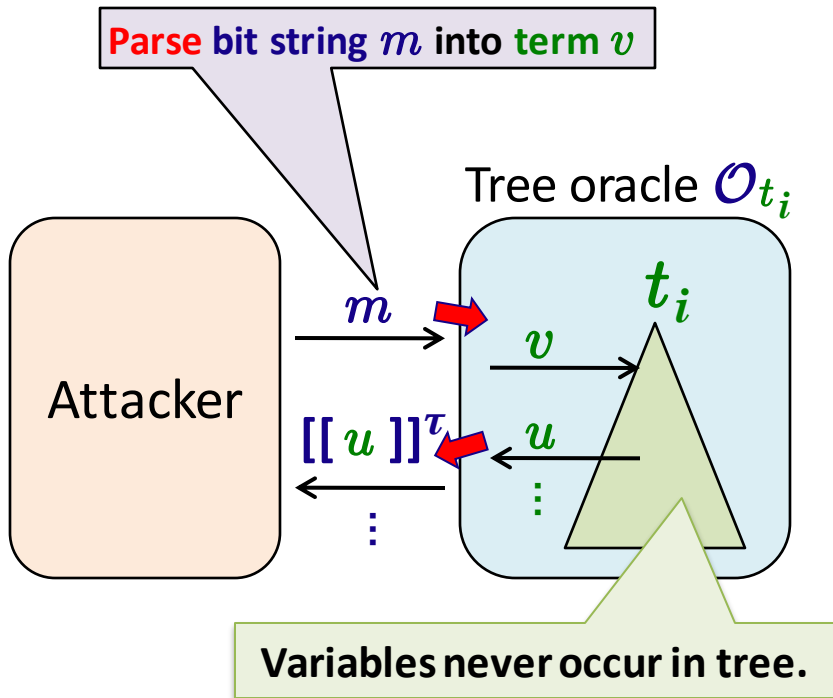
Tree soundness
Trace mapping
Tree soundness

Some computational path corresp

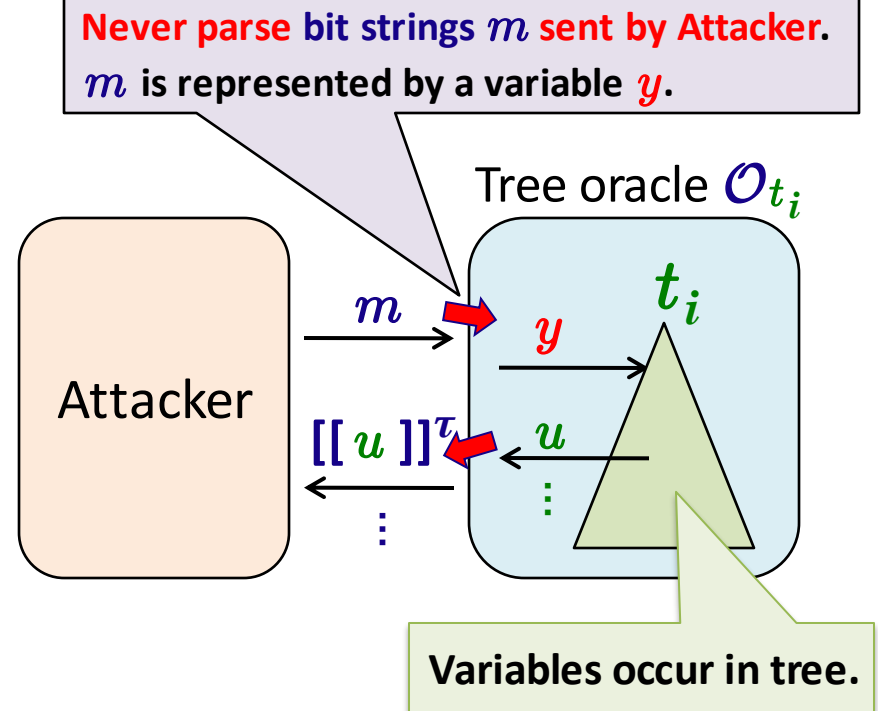


# Comparison of two tree oracles

## Tree oracle in [Comon, Corièr'08]



## Tree oracle in our work

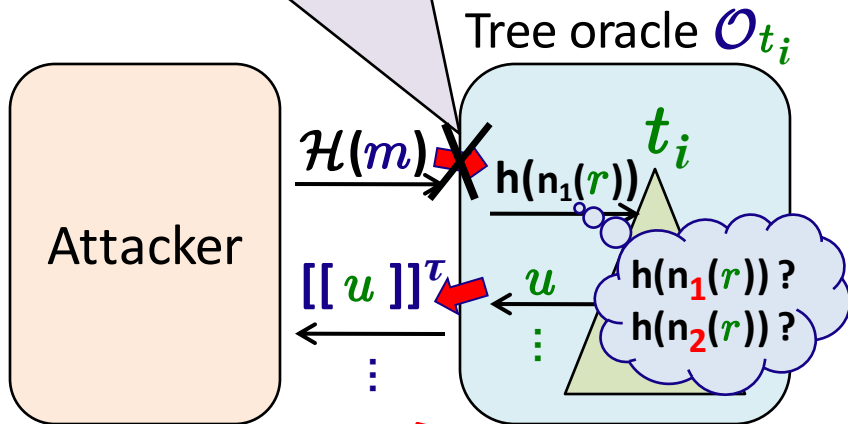


# Comparison of two tree oracles

Case: Attacker sends a **hash value**

Tree oracle in [Comon, Coriter'08]

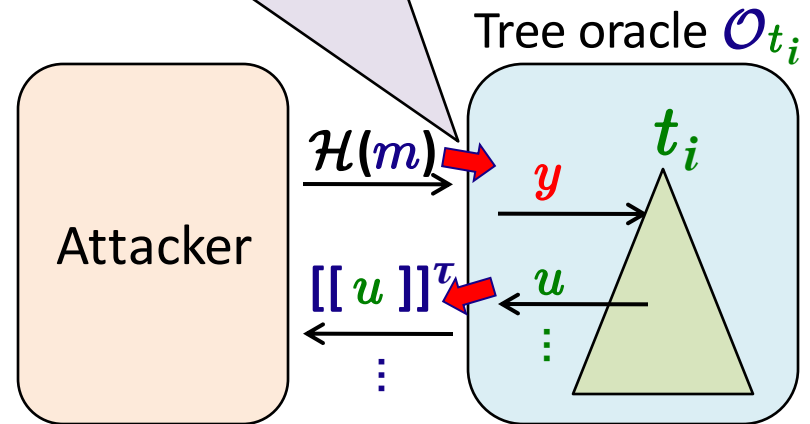
Simulator cannot obtain  $m$  from  $\mathcal{H}(m)$ .  
So he **cannot parse** bit string  $\mathcal{H}(m)$   
into term  $h(n_1(r))$  in polynomial-time.



[Comon, Coriter'08] **cannot** deal with  
hash values of **nonces of different length**.

Tree oracle in **our work**

**Never parse** bit strings  $m$  sent by Attacker.  
 $m$  is represented by a variable  $y$ .






# Outline of this talk

- Applied Pi-Calculus
- Soundness of observational equivalence
- Overview of soundness proof
- Comparison of proof methods
- **Summary and future work**

# Summary

- New soundness proof technique **without parsing function**.
- This makes us to deal with more cryptographic primitives.
  - Hash values of **nonces of different length**
  - Digital signatures for messages **not** appended with **the messages**.

# Future work

- Modify some assumptions to emphasize the benefits of our soundness proofs without parsings.
  - Relax some restrictions on protocols concerning hash values.
  - Allow the case **bit strings** are interpreted as both nonces and ciphertexts.
  - Add the exclusive or (XOR)  In the **next talk** by Hideki Sakurada

---

- New cryptographic primitives
  - Blind signature, Homomorphic encryption, ...
- Extension of **applied pi-calculus**
  - private channels, probabilistic branching, ...

Thank you for your attention.