Threshold Computation in the Head: More Efficient Signatures from MPCitH

Thibauld Feneuil

Journées NAC

February 29, 2024 — Paris (France)

CRYPTOEXPERTS

WE INNOVATE TO SECURE YOUR BUSINESS

Table of Contents

- Introduction
- TC-in-the-Head: general principle
- Applications
- Conclusion

Introduction

How to build signature schemes?

Hash & Sign

$$\begin{tikzcd}
H(m) \arrow[r, bend right, "F_{pk}^{-1}"'] & \sigma \arrow[l, bend right, "F_{pk}"']
\end{tikzcd}$$

Very hard
to compute

- Short signatures
- “Trapdoor” in the public key

How to build signature schemes?

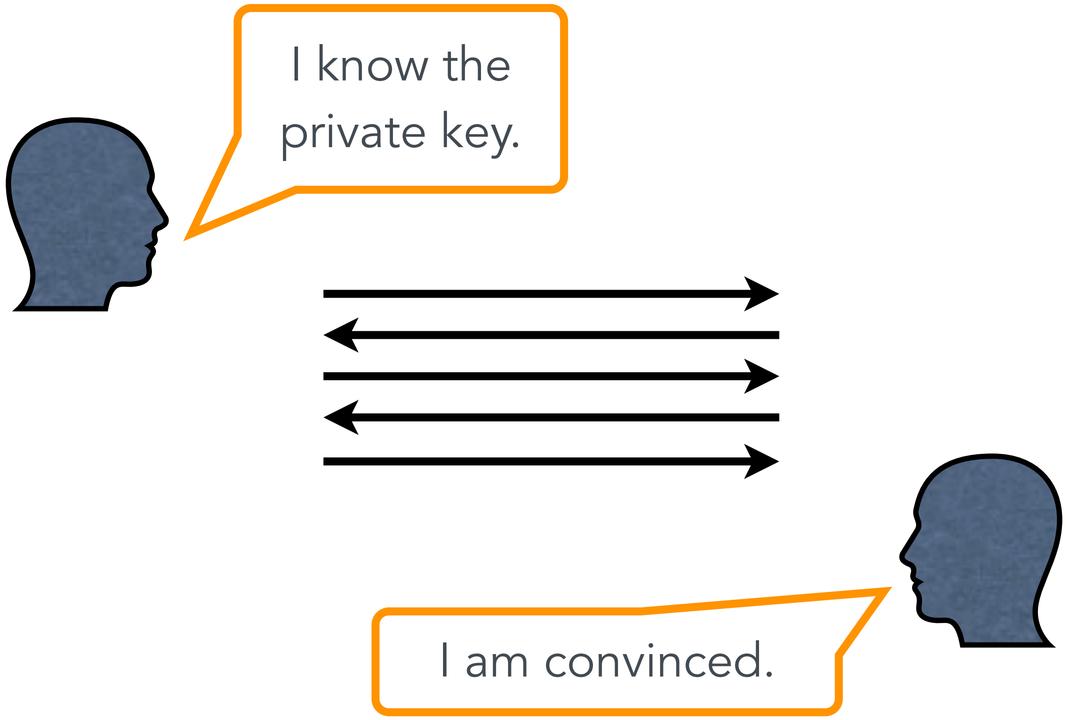
Hash & Sign

$$\begin{tikzcd} H(m) \arrow[r, bend right, "F_{pk}^{-1}"'] & \sigma \arrow[l, bend right, "F_{pk}"'] \end{tikzcd}$$

Very hard to compute

- Short signatures
- "Trapdoor" in the public key

From an identification scheme



- Large(r) signatures
- Short public key

How to build signature schemes?

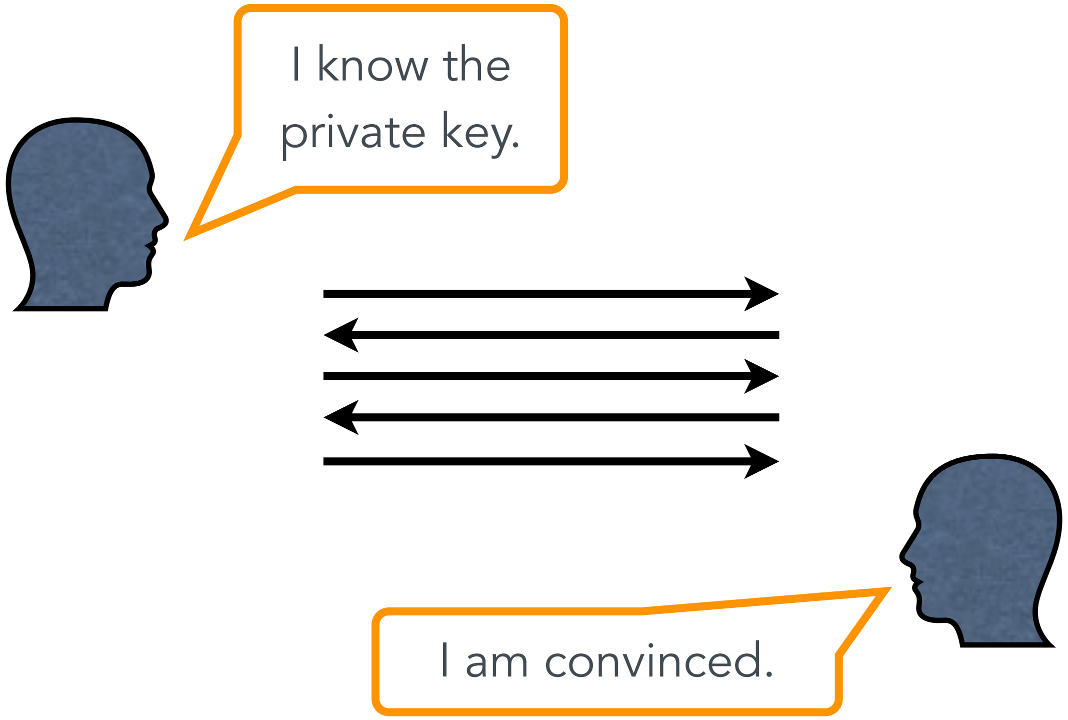
Hash & Sign

$$\begin{tikzcd}
H(m) \arrow[r, bend right, "F_{pk}^{-1}"'] & \sigma \arrow[l, bend right, "F_{pk}"']
\end{tikzcd}$$

Very hard to compute

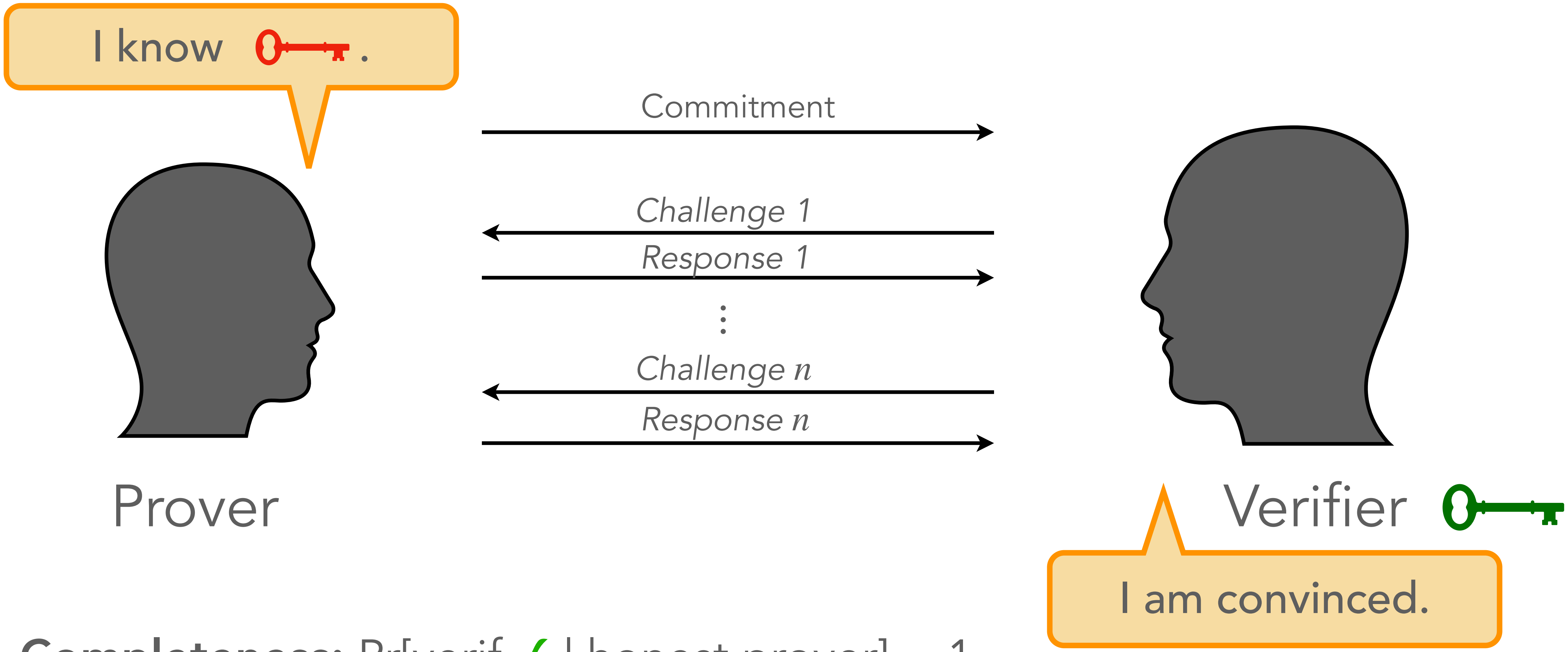
- Short signatures
- "Trapdoor" in the public key

From an identification scheme



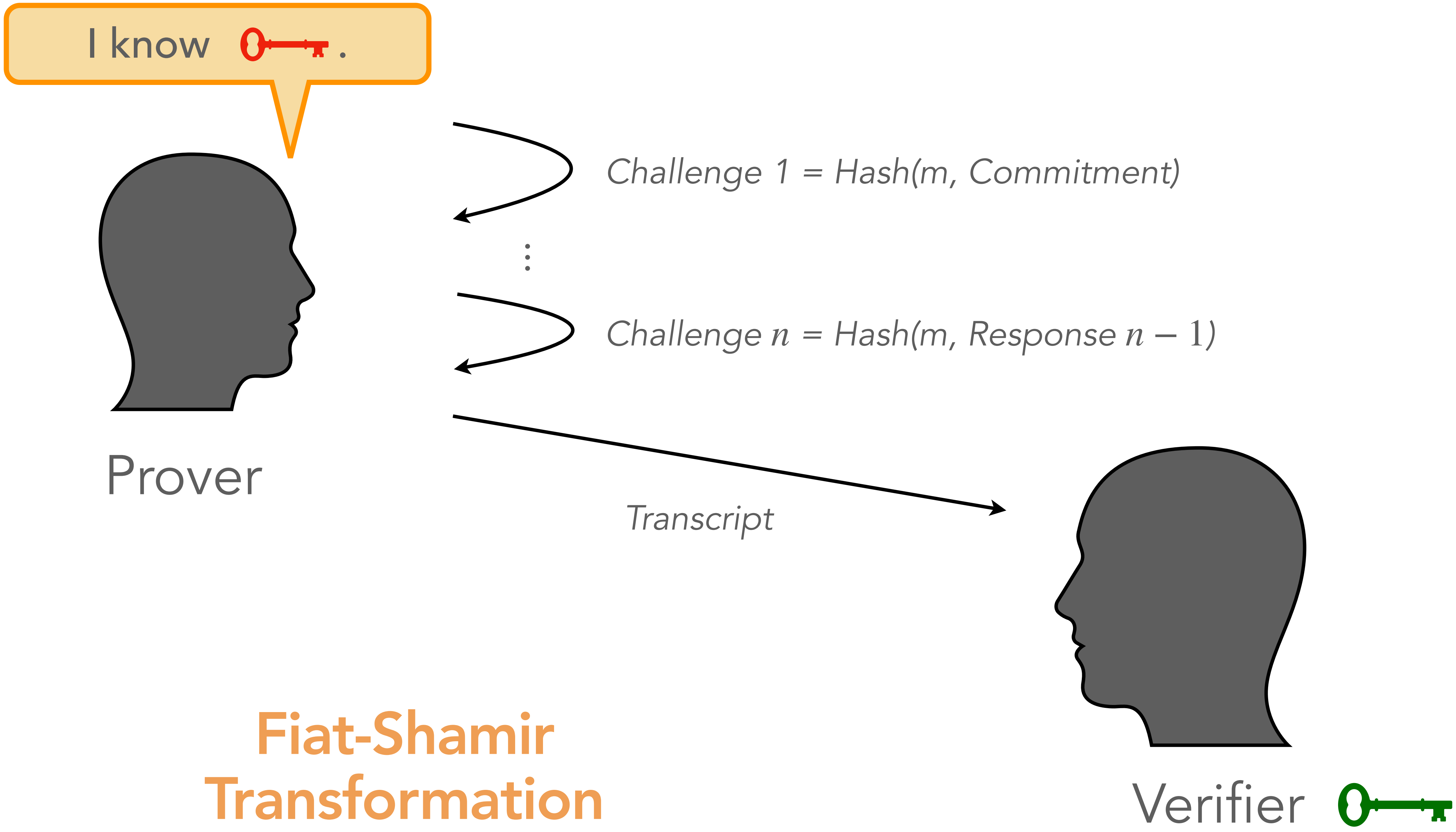
- Large(r) signatures
- Short public key

Identification Scheme



- **Completeness:** Pr[verif ✓ | honest prover] = 1
- **Soundness:** Pr[verif ✓ | malicious prover] $\leq \varepsilon$ (e.g. 2^{-128})
- **Zero-knowledge:** verifier learns nothing on 🔑.

Identification Scheme



I know 🔑.

Challenge 1 = Hash(*m*, Commitment)

⋮

Challenge n = Hash(*m*, Response $n - 1$)

Prover

Transcript

Verifier 🔑

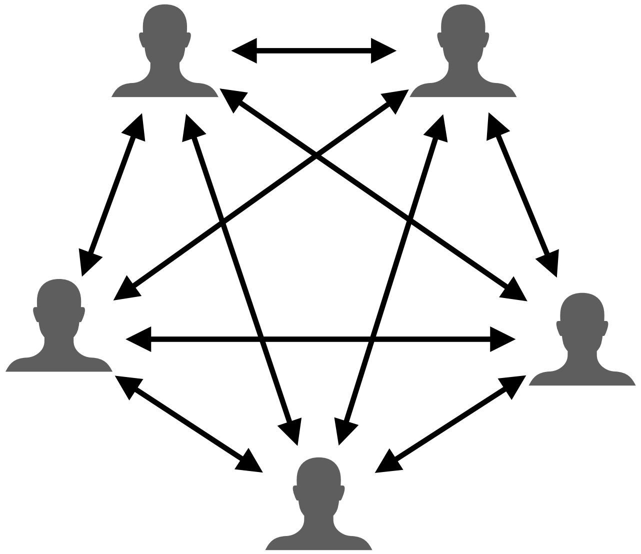
Fiat-Shamir Transformation

m: message to sign

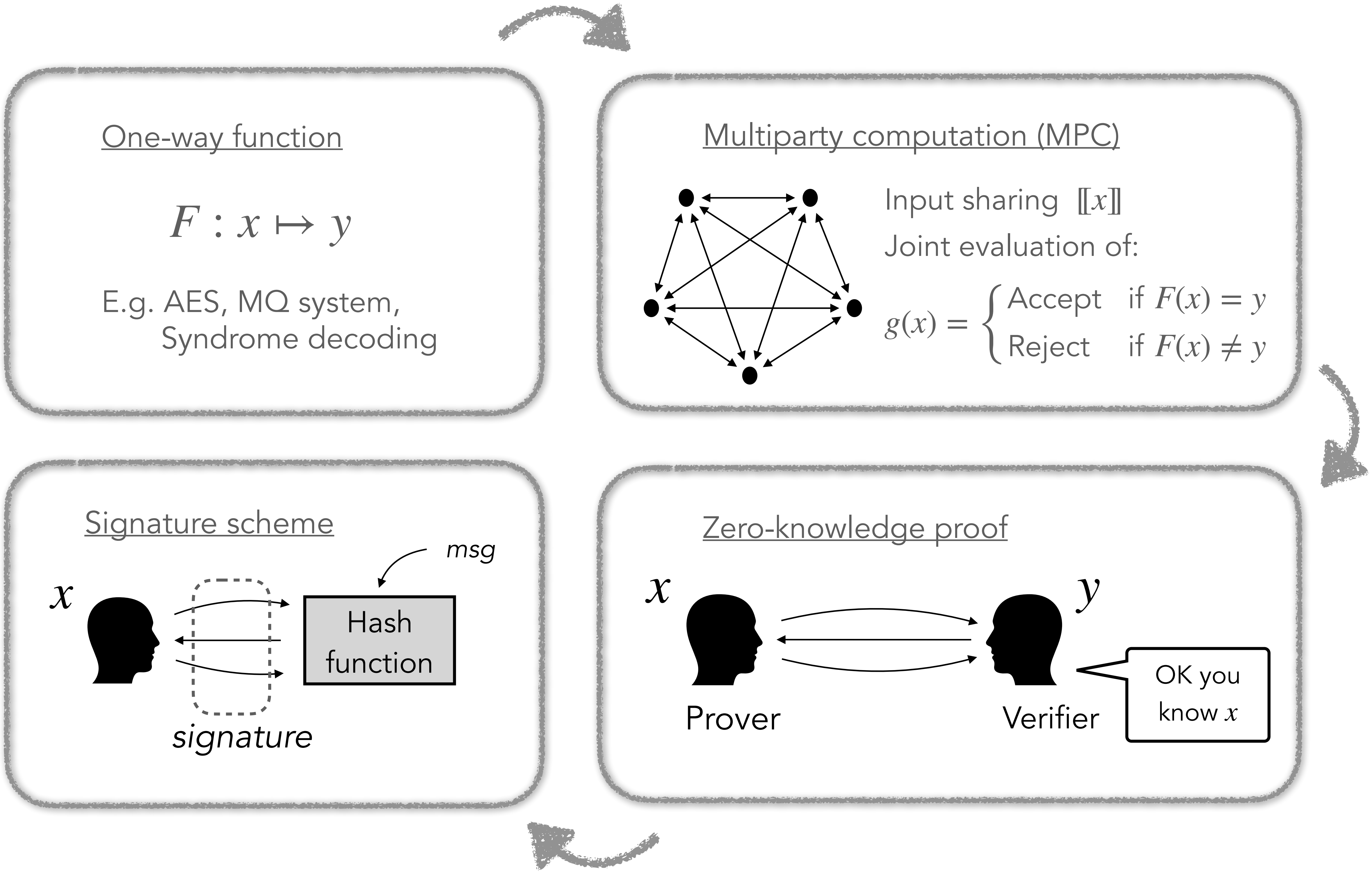
MPC in the Head

- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)

- Turn a *multiparty computation* (MPC) into an identification scheme



- **Generic**: can be apply to any cryptographic problem



One-way function

$F : x \mapsto y$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

Input sharing $[\![x]\!]$

Joint evaluation of:

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Zero-knowledge proof

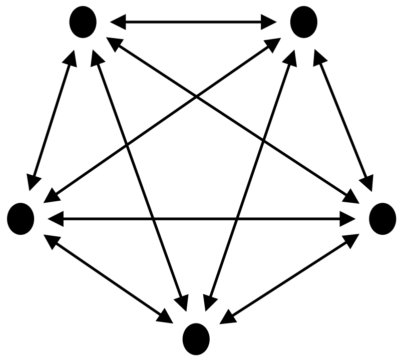
Signature scheme

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

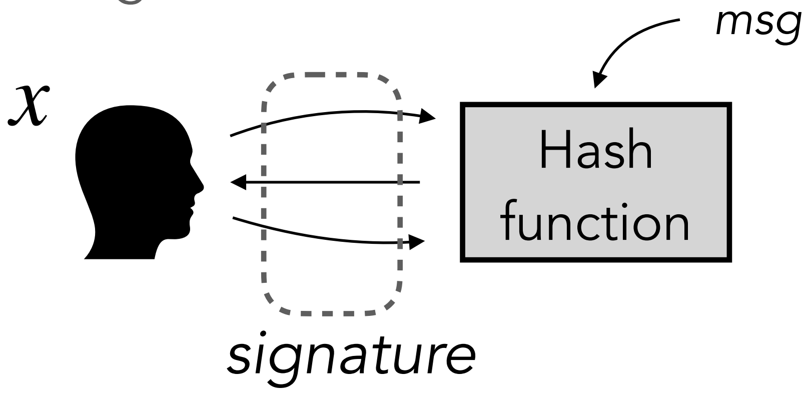


Input sharing $[\![x]\!]$

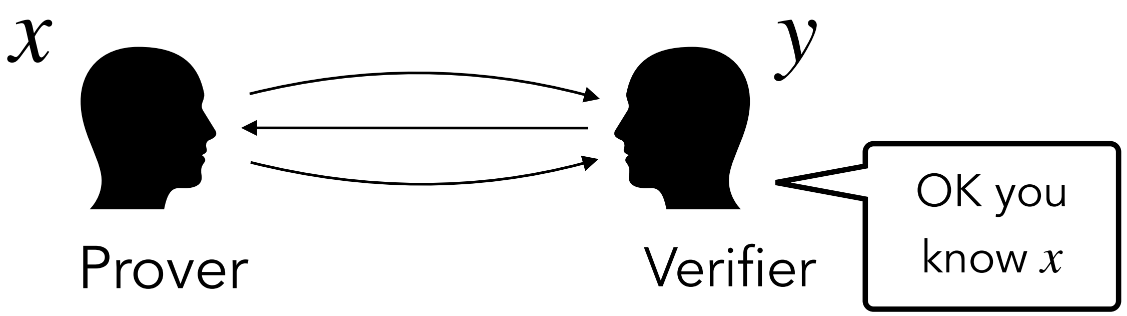
Joint evaluation of:

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Signature scheme



Zero-knowledge proof

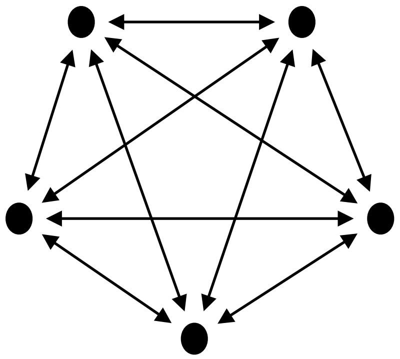


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

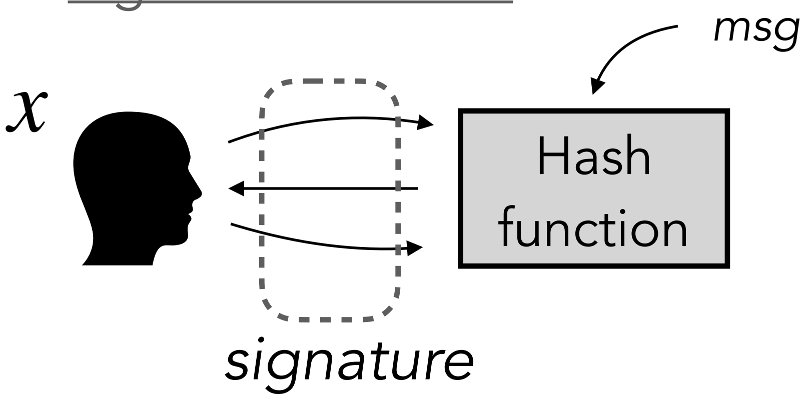


Input sharing $[\![x]\!]$

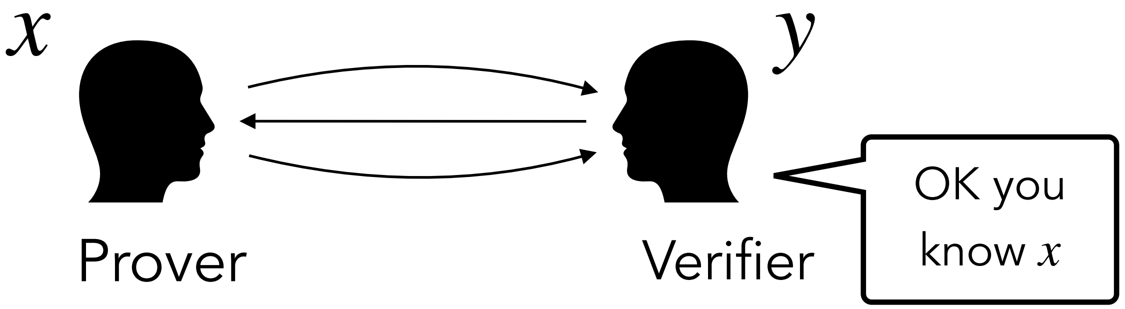
Joint evaluation of:

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Signature scheme



Zero-knowledge proof

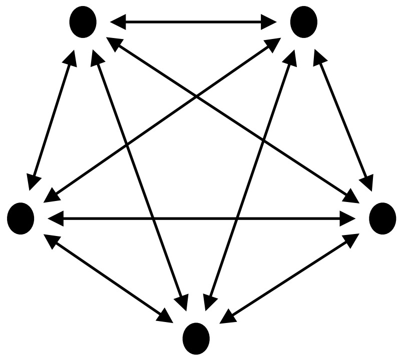


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

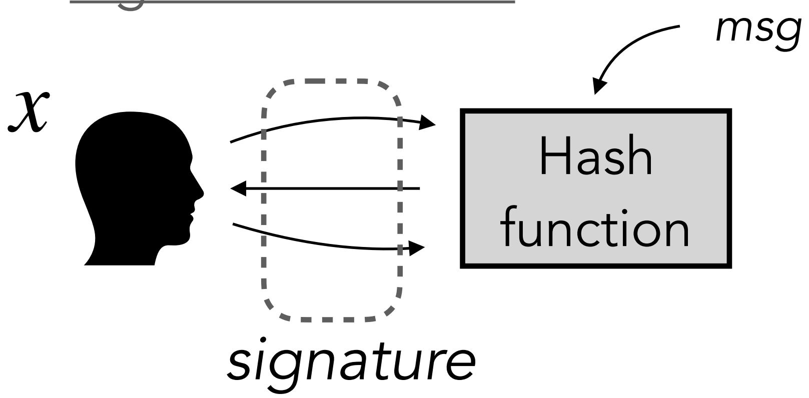


Input sharing $[\![x]\!]$

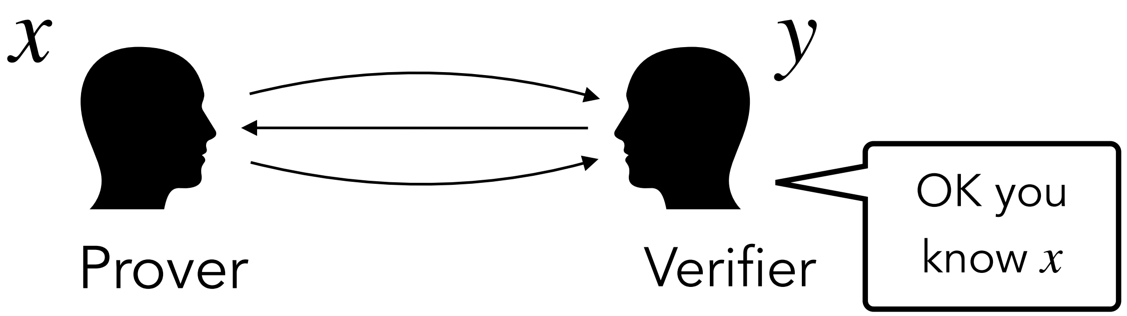
Joint evaluation of:

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Signature scheme



Zero-knowledge proof

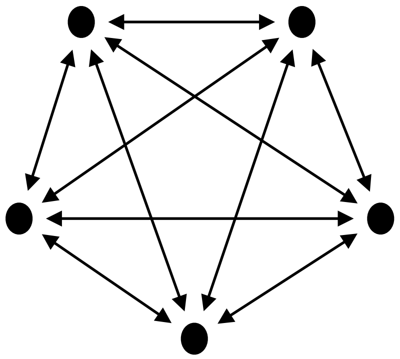


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

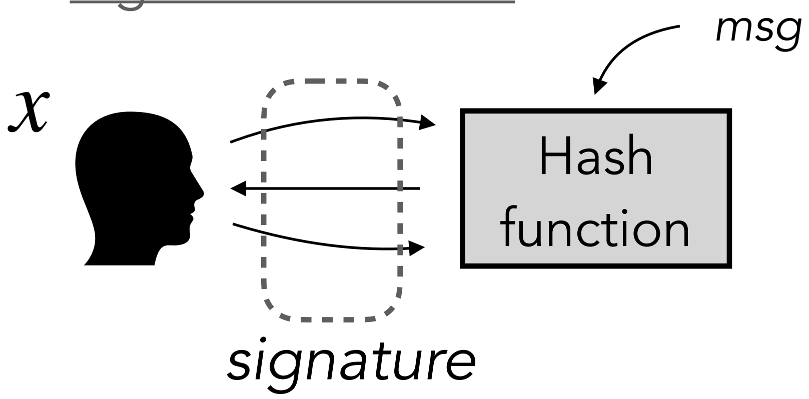


Input sharing $[\![x]\!]$

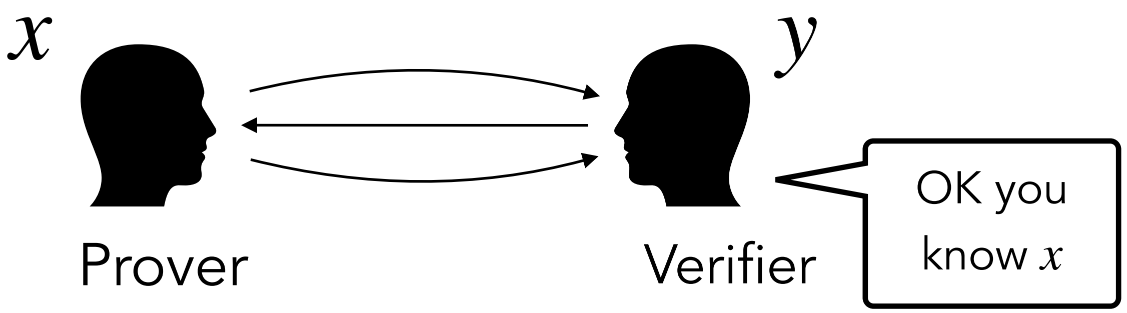
Joint evaluation of:

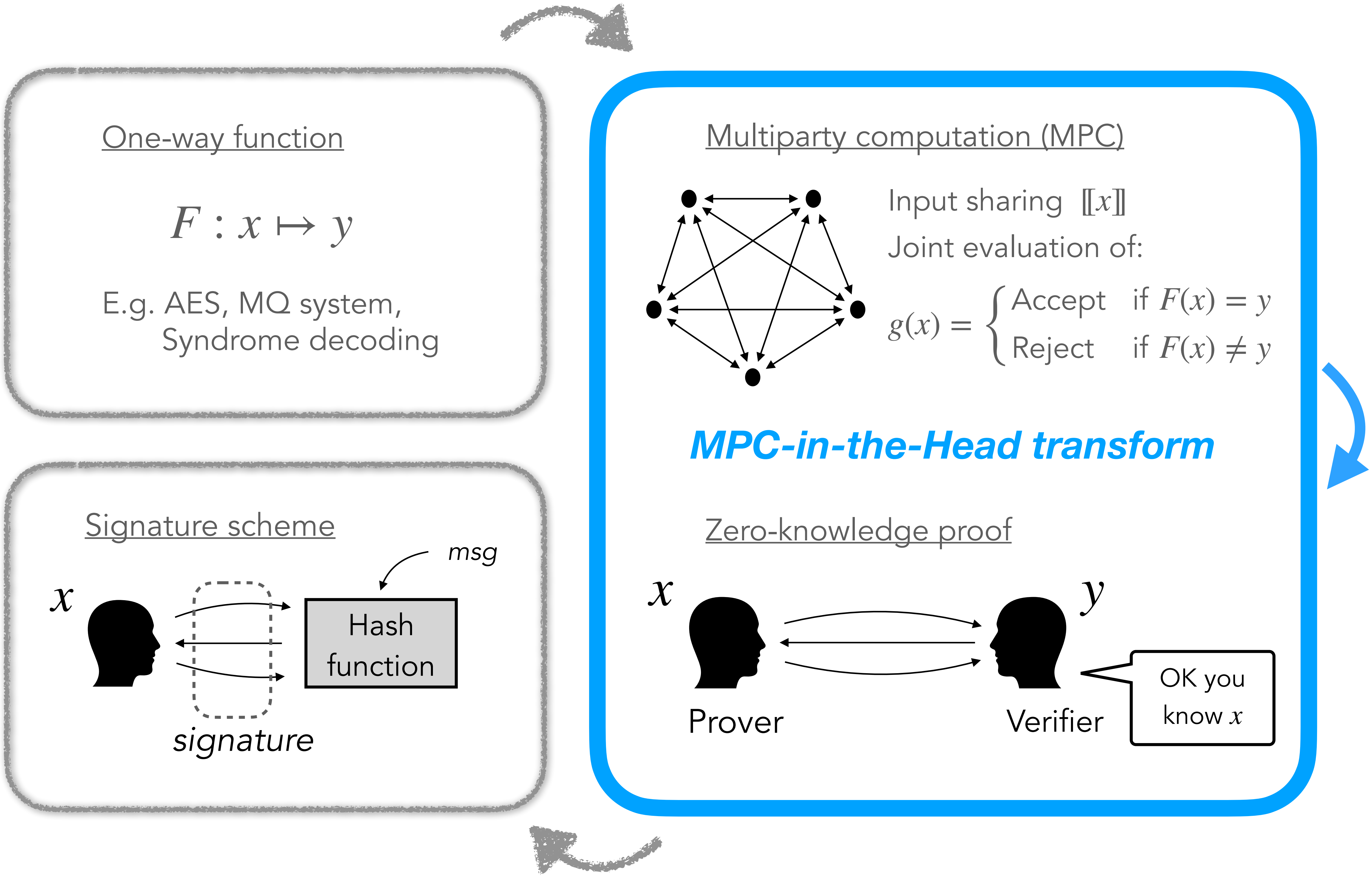
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Signature scheme



Zero-knowledge proof





One-way function

$F : x \mapsto y$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

Input sharing $[\![x]\!]$

Joint evaluation of:

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

MPC-in-the-Head transform

Zero-knowledge proof

Signature scheme

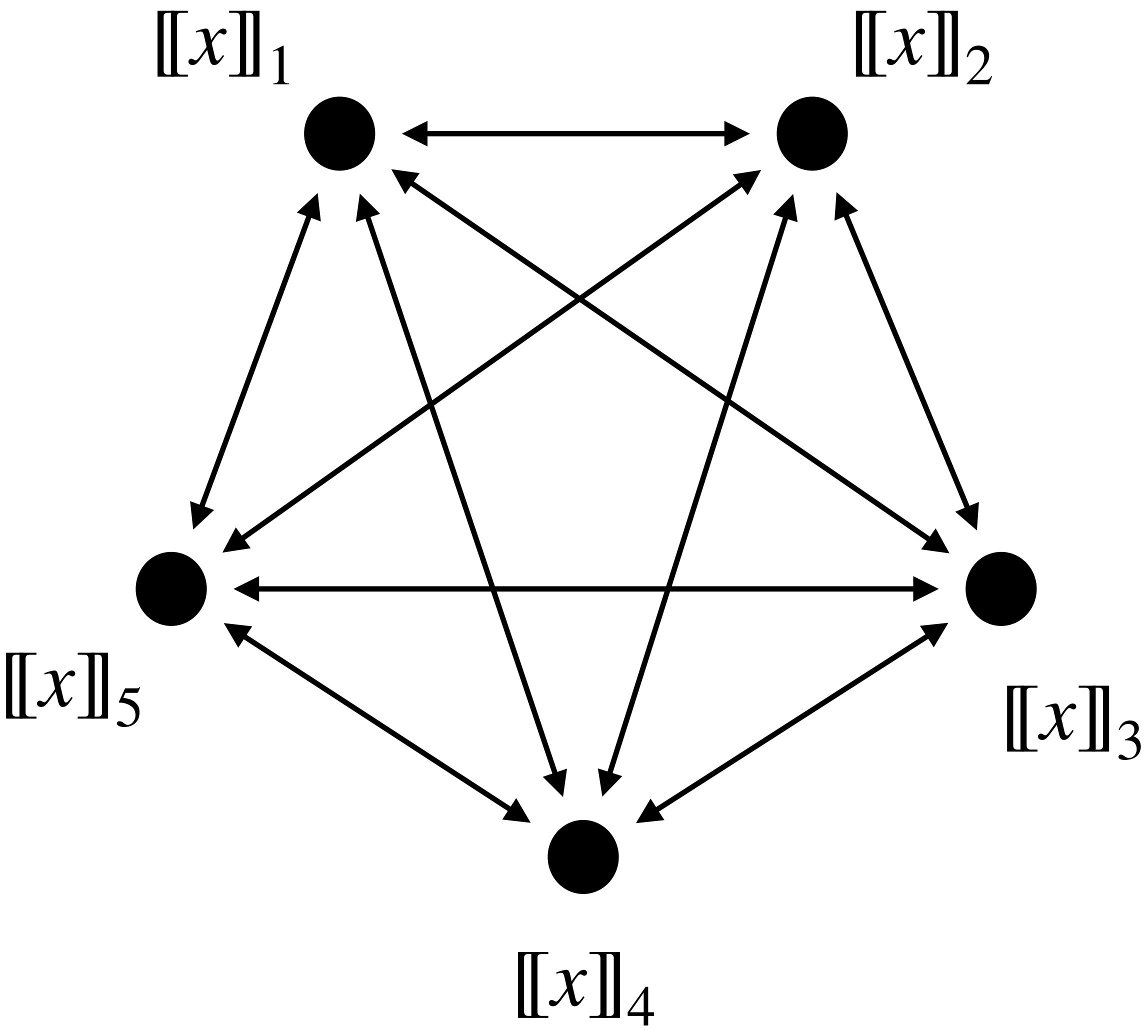
TCitH: general principle

TC: Threshold Computation

[FR23a] Feneuil, Rivain: "Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head" (Asiacrypt 2023)

[FR23b] Feneuil, Rivain: "Threshold Computation in the Head: Improved Framework for Post-Quantum Signatures and Zero-Knowledge Arguments" (Eprint 2023/1573)

MPC model



$[\![x]\!]$ is a degree-ℓ Shamir's secret sharing of x

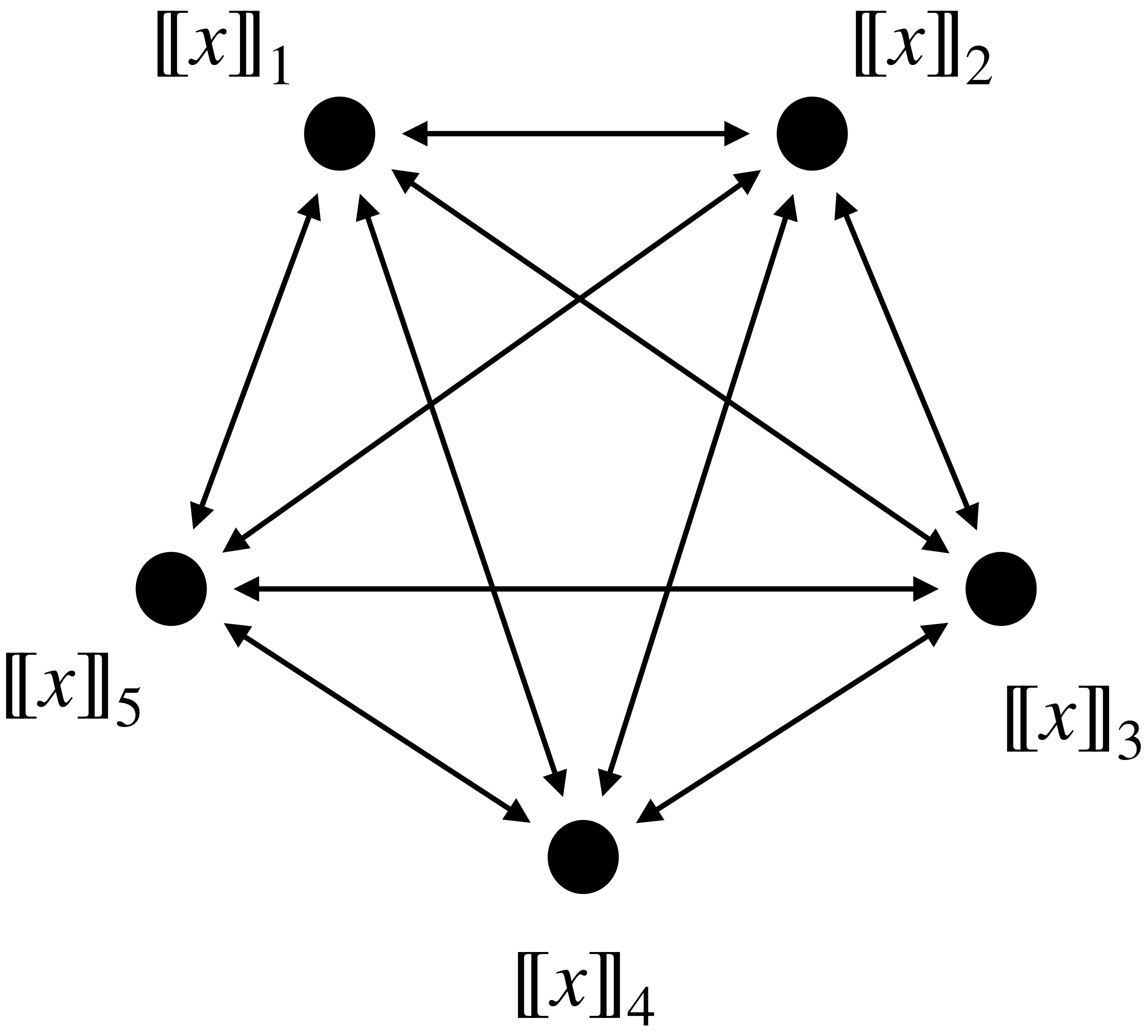
We set the degree-ℓ polynomial P such that

$$P(0) = x$$
$$P(e_1) \leftarrow_\$ \mathbb{F}$$
$$P(e_2) \leftarrow_\$ \mathbb{F}$$
$$\dots$$
$$P(e_\ell) \leftarrow_\$ \mathbb{F}.$$

We define the shares as

$$\forall i \in \{1, \dots, N\}, \quad [\![x]\!]_i = P(e_i).$$

MPC model



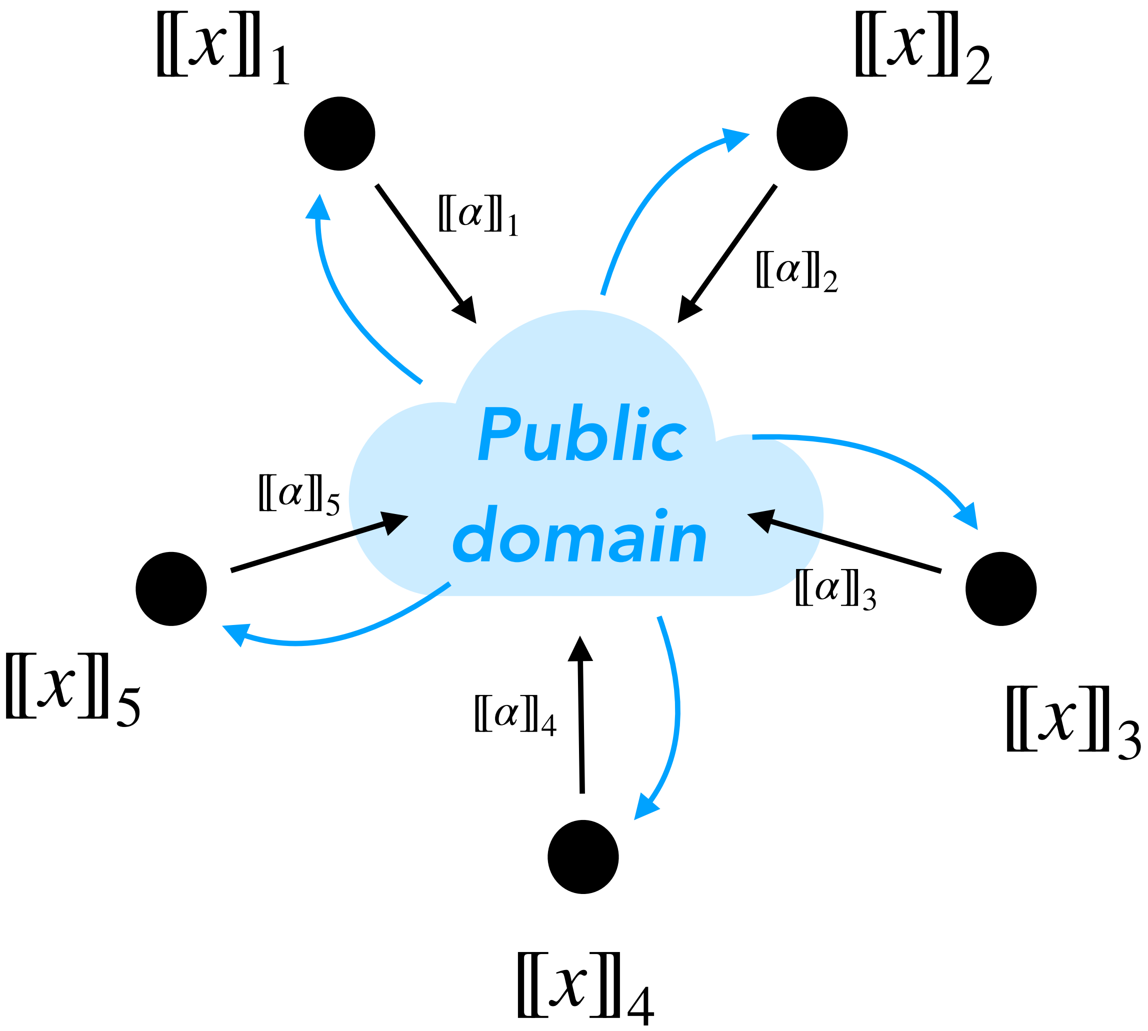
- **Jointly compute**

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

- **ℓ-private:** the views of any ℓ parties provide no information on x
- **Semi-honest model:** assuming that the parties follow the steps of the protocol

$[\![x]\!]$ is a degree-ℓ Shamir's secret sharing of x

MPC model



- **Jointly compute**

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

- **ℓ-private:** the views of any ℓ parties provide no information on x
- **Semi-honest model:** assuming that the parties follow the steps of the protocol
- ***Broadcast model***
  - *Parties locally compute on their shares $[\![x]\!] \mapsto [\![\alpha]\!]$*
  - *Parties broadcast $[\![\alpha]\!]$ and recompute α*
  - *Parties start again (now knowing α)*

$[\![x]\!]$ is a degree-ℓ Shamir's secret sharing of x

TCitH transform

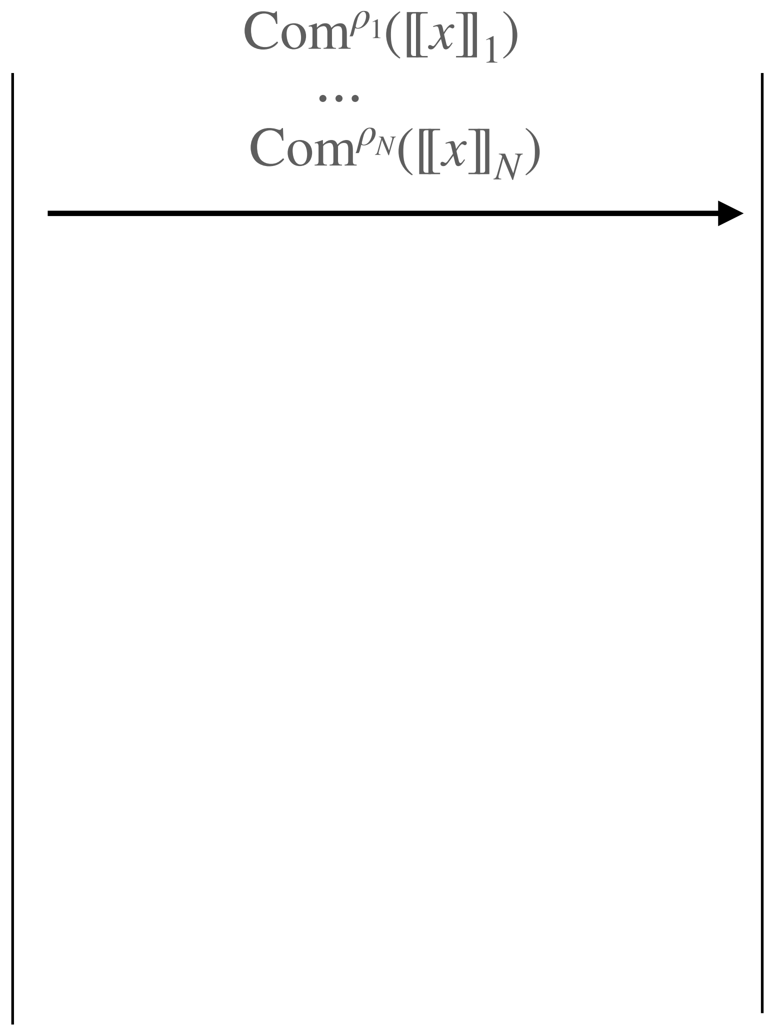
Prover

Verifier

TCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$



$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\ldots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

Prover

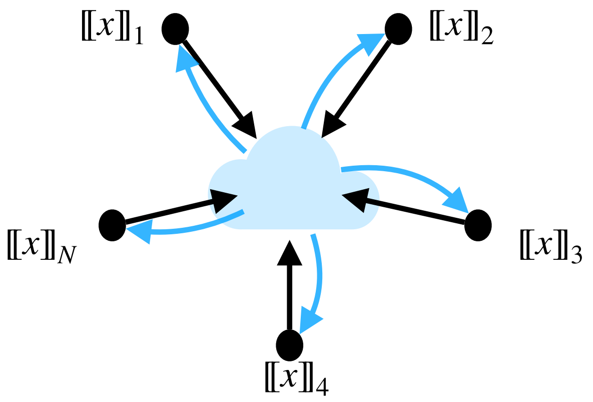
Verifier

TCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$
$\ldots$
$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$ →

send broadcast
$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ →

Prover

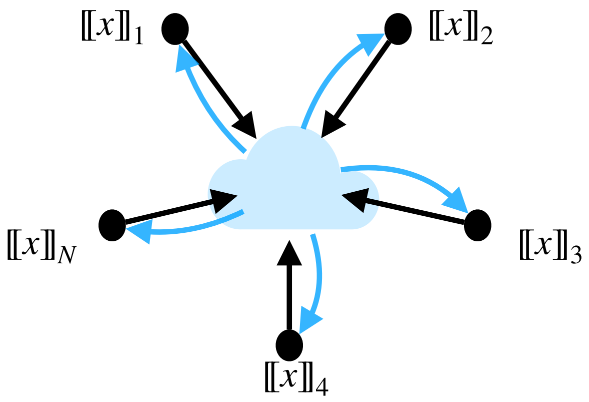
Verifier

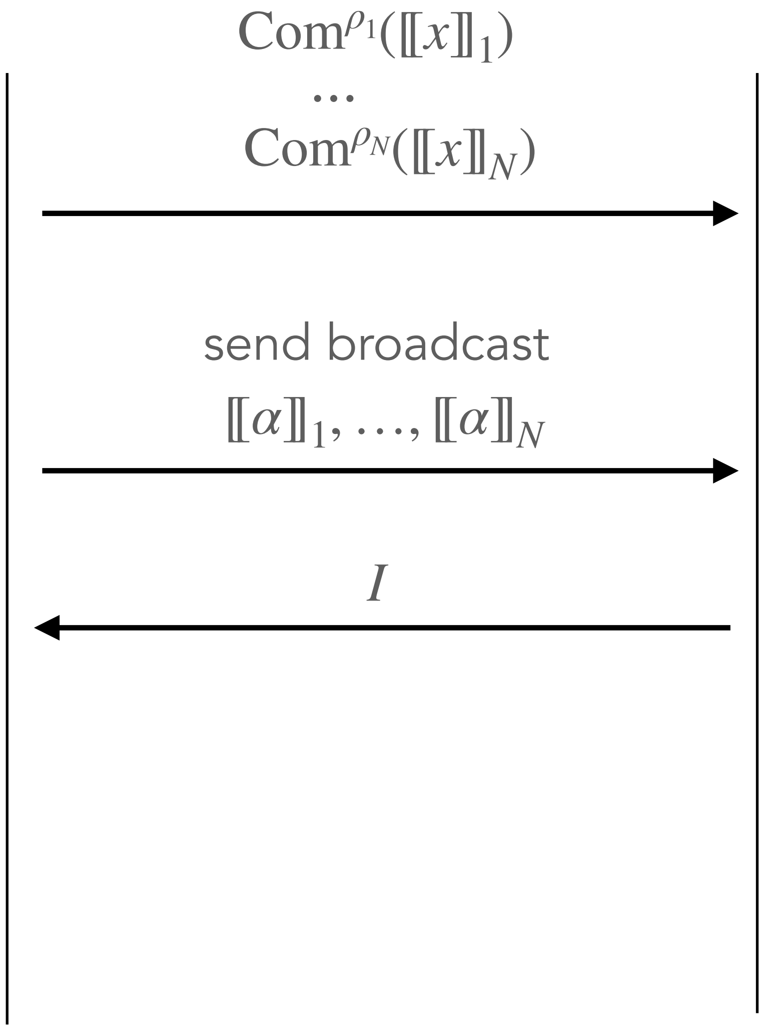
TCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head





$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\ldots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

I

③ Choose a random set of parties $I \subseteq \{1,\ldots,N\}$, s.t. $|I| = \ell$.

Prover

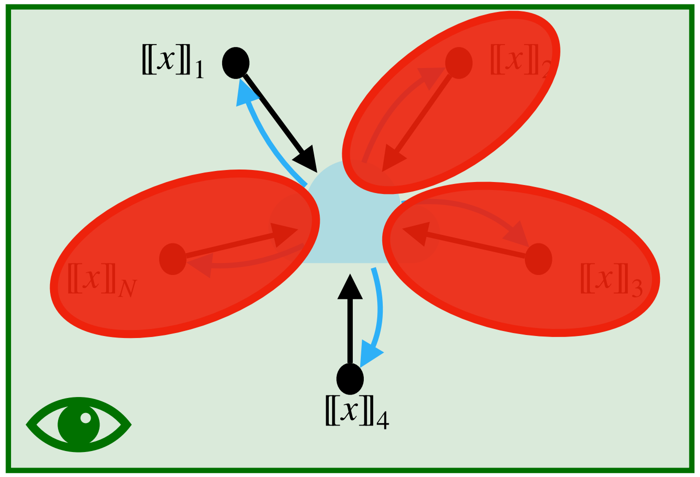
Verifier

TCitH transform

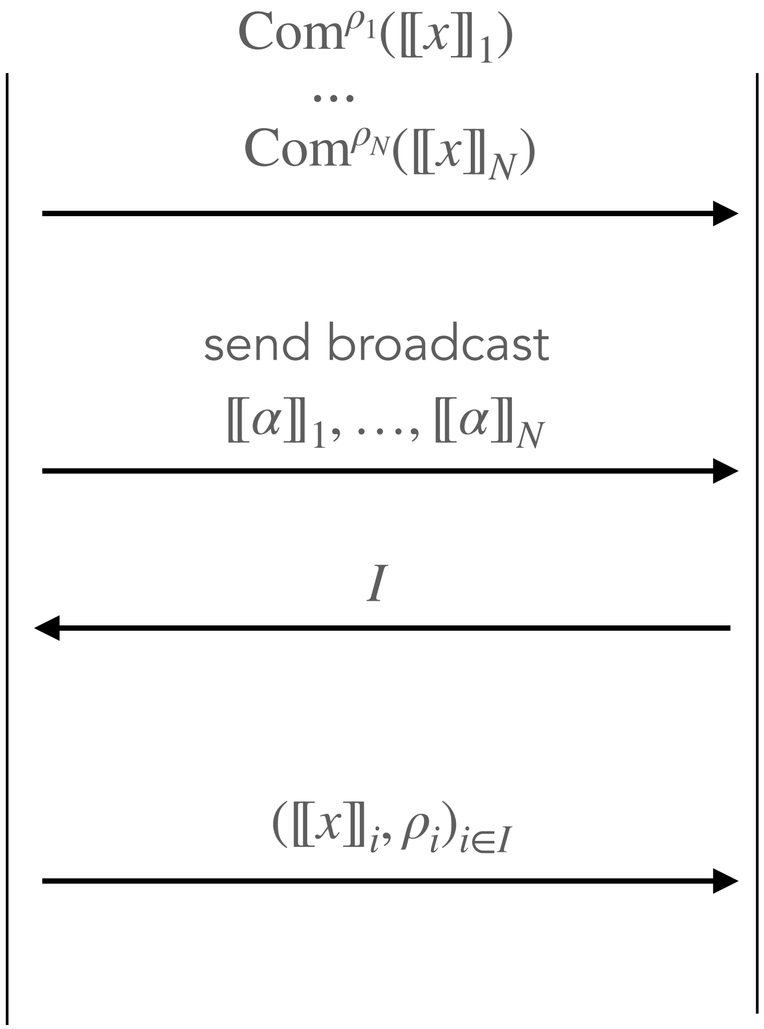
① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



④ Open parties in I



$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

I

$([\![x]\!]_i, \rho_i)_{i \in I}$

③ Choose a random set of parties

$I \subseteq \{1, \ldots, N\}$, s.t. $|I| = \ell$.

Prover

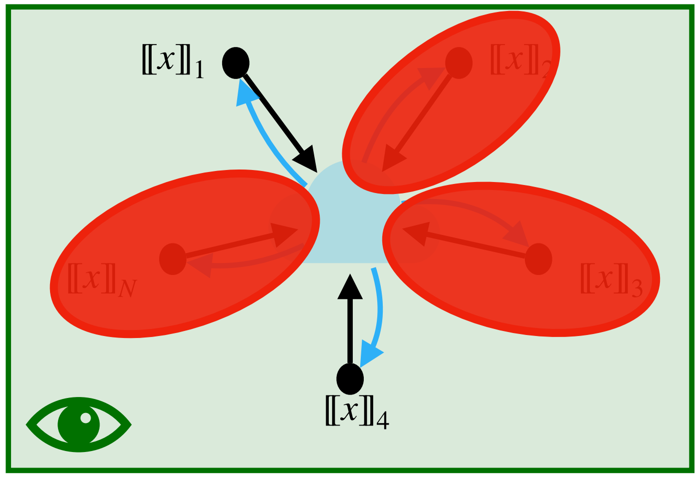
Verifier

TCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



④ Open parties in I

$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$ →

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ →

← I

$([\![x]\!]_i, \rho_i)_{i \in I}$ →

③ Choose a random set of parties

$I \subseteq \{1, \ldots, N\}$, s.t. $|I| = \ell$.

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

Prover

Verifier

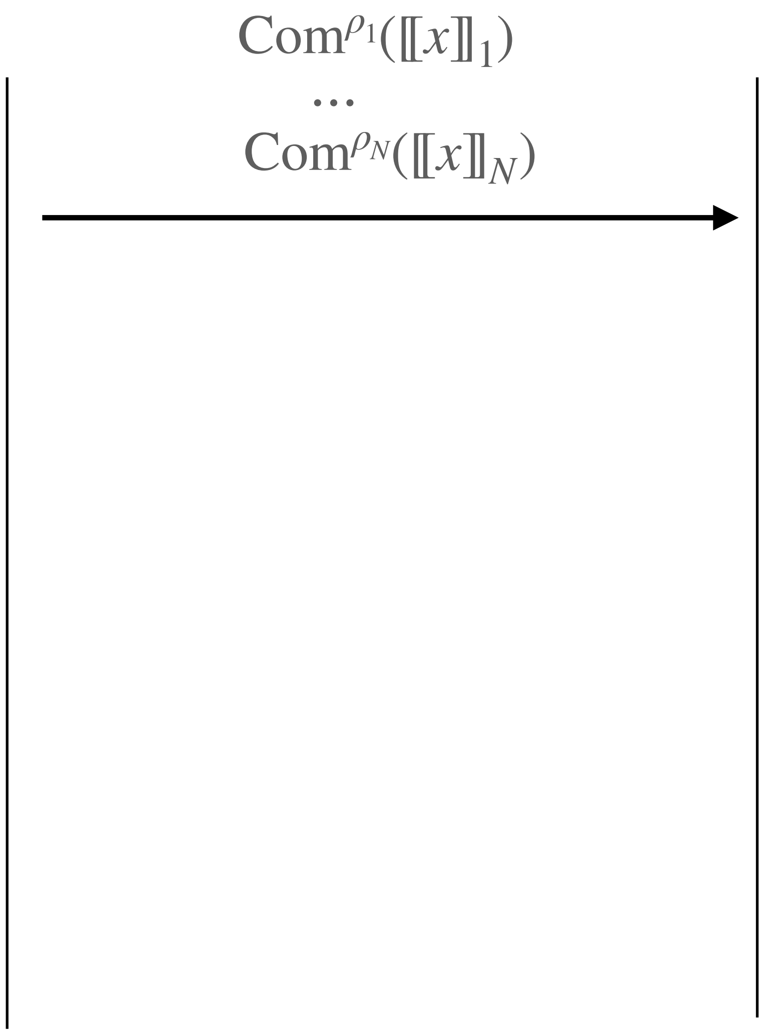
TCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

We have $F(x) \neq y$ *where*

$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$



$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\ldots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

Malicious Prover

Verifier

TCitH transform

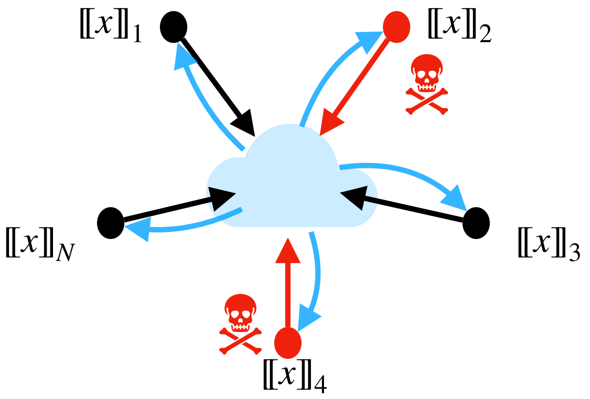
① Generate and commit shares

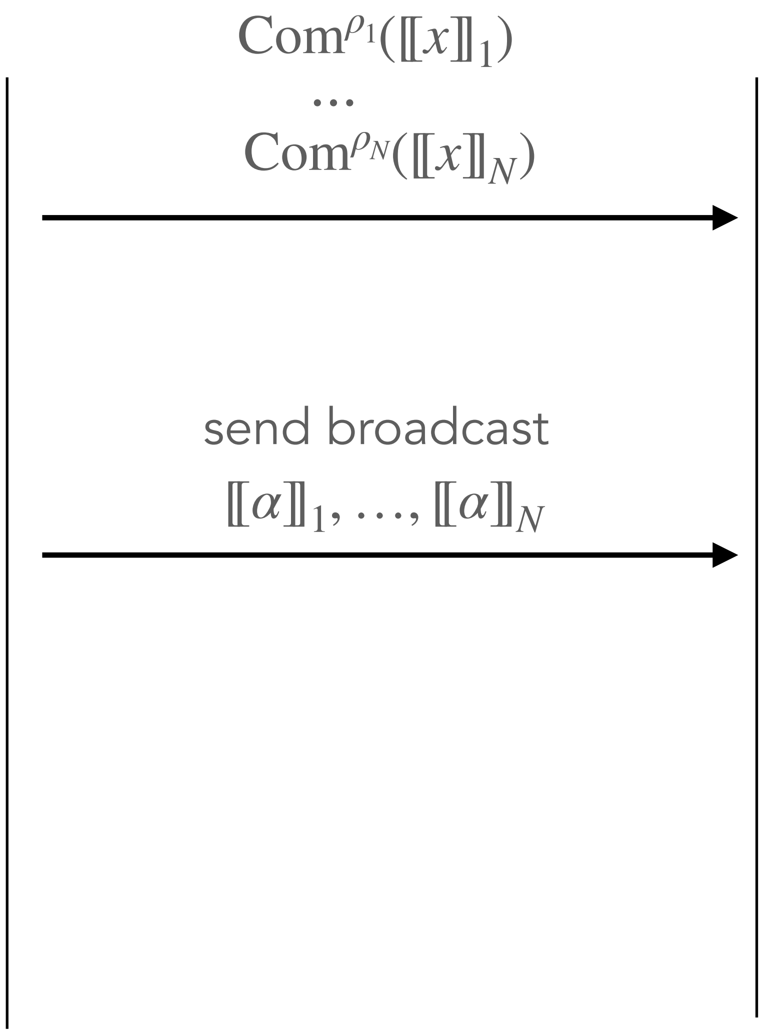
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

We have $F(x) \neq y$ where

$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

② Run MPC in their head





$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

Malicious Prover

Verifier

TCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

We have $F(x) \neq y$ where

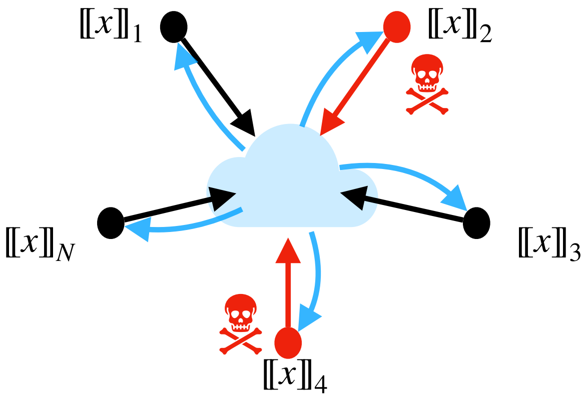
$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

② Run MPC in their head



send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

③ Choose a random set of parties $I \subseteq \{1,\ldots,N\}$, s.t. $|I| = \ell$.

I

Malicious Prover

Verifier

TCitH transform

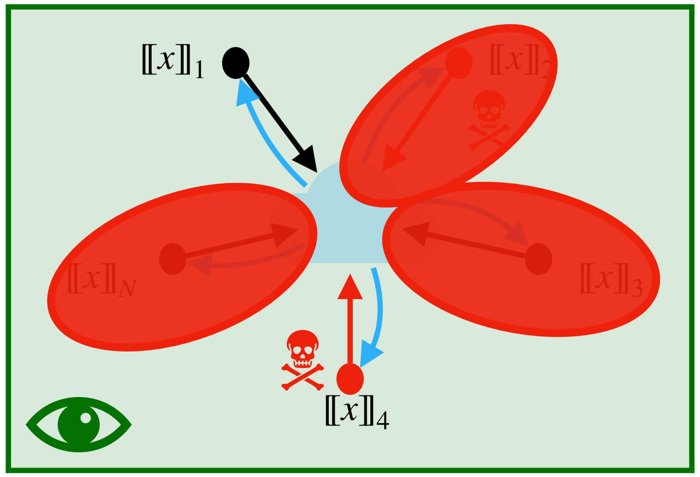
① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

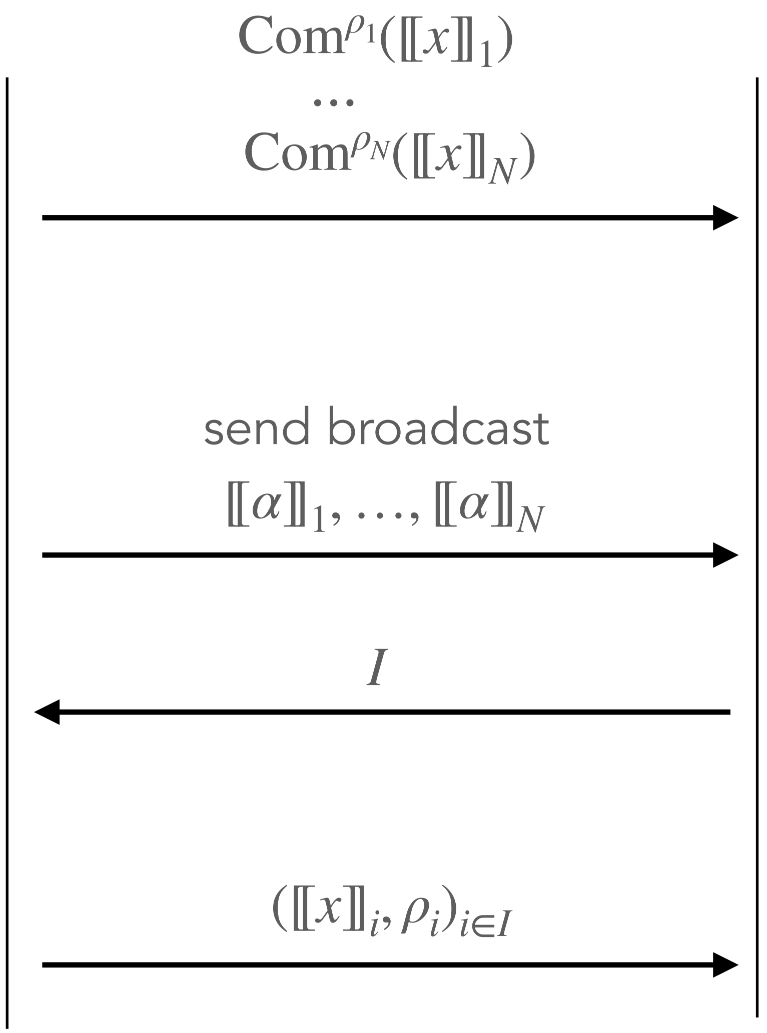
We have $F(x) \neq y$ where

$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

② Run MPC in their head



④ Open parties in I


$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

…

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

I

$([\![x]\!]_i, \rho_i)_{i \in I}$

③ Choose a random set of parties

$I \subseteq \{1,\ldots,N\}$, s.t. $|I| = \ell$.

Malicious Prover

Verifier

TCitH transform

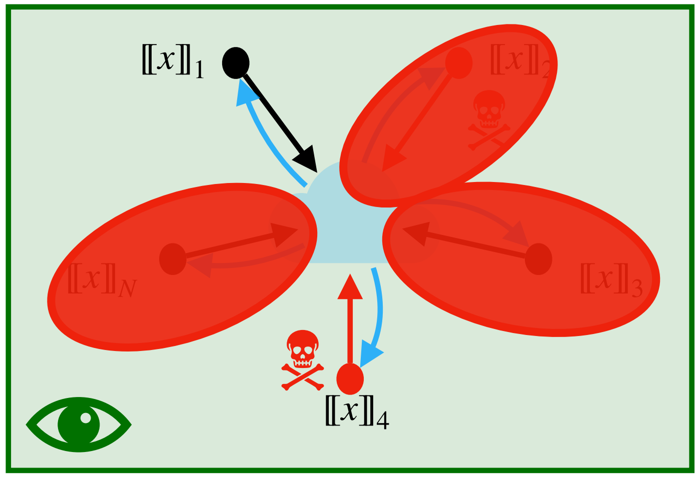
① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

We have $F(x) \neq y$ where

$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

② Run MPC in their head



④ Open parties in I

$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

…

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$

I

$([\![x]\!]_i, \rho_i)_{i \in I}$

③ Choose a random set of parties

$I \subseteq \{1,\ldots,N\}$, s.t. $|I| = \ell$.

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

Malicious Prover

Verifier

✖ **Cheating detected!**

TCitH transform

① Generate and commit shares

$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

We have $F(x) \neq y$ where

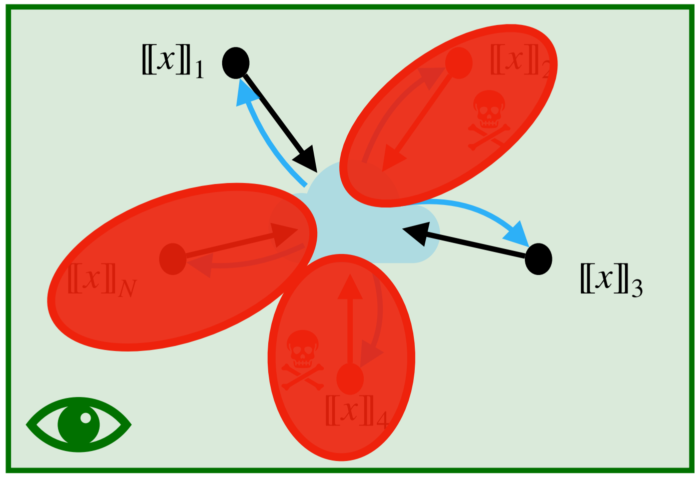
$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$ →

② Run MPC in their head



send broadcast

$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ →

③ Choose a random set of parties

$I \subseteq \{1, \ldots, N\}$, s.t. $|I| = \ell$.

← I

④ Open parties in I

$([\![x]\!]_i, \rho_i)_{i \in I}$ →

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

Check $g(y, \alpha) =$ Accept

Malicious Prover

Verifier

✓ **Seems OK.**

TCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is ℓ-private

TCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is ℓ-private
- **Soundness:** ***if the committed sharing is valid***

$$\begin{aligned}\mathbb{P}(\text{malicious prover convinces the verifier})\\ &= \mathbb{P}(\text{all corrupted parties remain hidden})\\ &= \frac{\binom{d_\alpha}{\ell}}{\binom{N}{\ell}}\end{aligned}$$

d_α is the degree of the sharing $[\![\alpha]\!]$.

TCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is ℓ-private
- **Soundness:** ***if the committed sharing is valid***

$$\begin{aligned}&\mathbb{P}(\text{malicious prover convinces the verifier})\\ &\qquad = \mathbb{P}(\text{all corrupted parties remain hidden})\\ &\qquad = \frac{\binom{d_\alpha}{\ell}}{\binom{N}{\ell}}\end{aligned}$$

d_α is the degree of the sharing $[\![\alpha]\!]$.

- **Parallel repetition**

Protocol repeated τ times in parallel, soundness error $\left(\frac{\binom{d_\alpha}{\ell}}{\binom{N}{\ell}}\right)^{\tau}$

How to commit Shamir's secret sharing?

TCitH-GGM: Using a GGM tree

vs

TCitH-MT: Using a Merkle tree

TCitH-GGM: Using a Seed Tree

Step 1: Generate a *replicated secret sharing* [ISN89]:

$$r = r_1 + r_2 + \ldots + r_N$$

- Party $\mathscr{P}_1$: $r_2, r_3, \ldots, r_N$
- Party $\mathscr{P}_2$: $r_1, r_3, \ldots, r_N$

  $\ldots$

- Party $\mathscr{P}_2$: $r_1, r_2, \ldots, r_{N-1}$

[ISN89] Ito, Saito, Nishizeki: "Secret sharing scheme realizing general access structure" (Electronics and Communications in Japan 1989)

TCitH-GGM: Using a Seed Tree

Step 1: Generate a *replicated secret sharing* [ISN89]:

$$r = r_1 + r_2 + \ldots + r_N$$

- Party $\mathscr{P}_1$: $r_2, r_3, \ldots, r_N$
- Party $\mathscr{P}_2$: $r_1, r_3, \ldots, r_N$

  $\ldots$

- Party $\mathscr{P}_2$: $r_1, r_2, \ldots, r_{N-1}$

[CDI05] Cramer, Damgard, Ishai: "Share conversion, pseudorandom secret-sharing and applications to secure computation" (TCC 2005)

Step 2: Convert in a *Shamir's secret sharing* [CDI05]:

$$[\![x]\!]_i \leftarrow \sum_{j=1, j\neq i}^{N} r_j \cdot P_j(e_i)$$

where $P_j(X) := 1 - \dfrac{1}{e_j} X\,.$

TCitH-GGM: Using a Seed Tree

Step 1: Generate a *replicated secret sharing* [ISN89]:

$$r = r_1 + r_2 + \ldots + r_N$$

- Party $\mathcal{P}_1$: $r_2, r_3, \ldots, r_N$
- Party $\mathcal{P}_2$: $r_1, r_3, \ldots, r_N$

  $\ldots$

- Party $\mathcal{P}_2$: $r_1, r_2, \ldots, r_{N-1}$

[CDI05] Cramer, Damgard, Ishai: "Share conversion, pseudorandom secret-sharing and applications to secure computation" (TCC 2005)

Step 2: Convert in a *Shamir's secret sharing* [CDI05]:

$$[\![x]\!]_i \leftarrow \sum_{j=1, j\neq i}^{N} r_j \cdot P_j(e_i)$$

where $P_j(X) := 1 - \frac{1}{e_j} X$.

This process ensures that $[\![x]\!]_i$'s are the evaluations of a degree-1 polynomial.

The obtained sharing is a 1-private Shamir's secret sharing of r.

TCitH-GGM: Using a Seed Tree

Step 1: Generate a *replicated secret sharing* [ISN89]:

$$r = r_1 + r_2 + \ldots + r_N$$

- Party $\mathscr{P}_1$: $r_2, r_3, \ldots, r_N$
- Party $\mathscr{P}_2$: $r_1, r_3, \ldots, r_N$

  $\ldots$

- Party $\mathscr{P}_2$: $r_1, r_2, \ldots, r_{N-1}$

[CDI05] Cramer, Damgard, Ishai: "Share conversion, pseudorandom secret-sharing and applications to secure computation" (TCC 2005)

Step 2: Convert in a *Shamir's secret sharing* [CDI05]:

$$[\![x]\!]_i \leftarrow \sum_{j=1, j\neq i}^{N} r_j \cdot P_j(e_i)$$

where $P_j(X) := 1 - \frac{1}{e_j} X$.

This process ensures that $[\![x]\!]_i$'s are the evaluations of a degree-1 polynomial.

The obtained sharing is a 1-private Shamir's secret sharing of r.

Can be generalized for any Shamir's secret sharing (of higher degree).

Using a Seed Tree

$$r = r_1 + r_2 + r_3 + \ldots + r_{N-1} + r_N$$

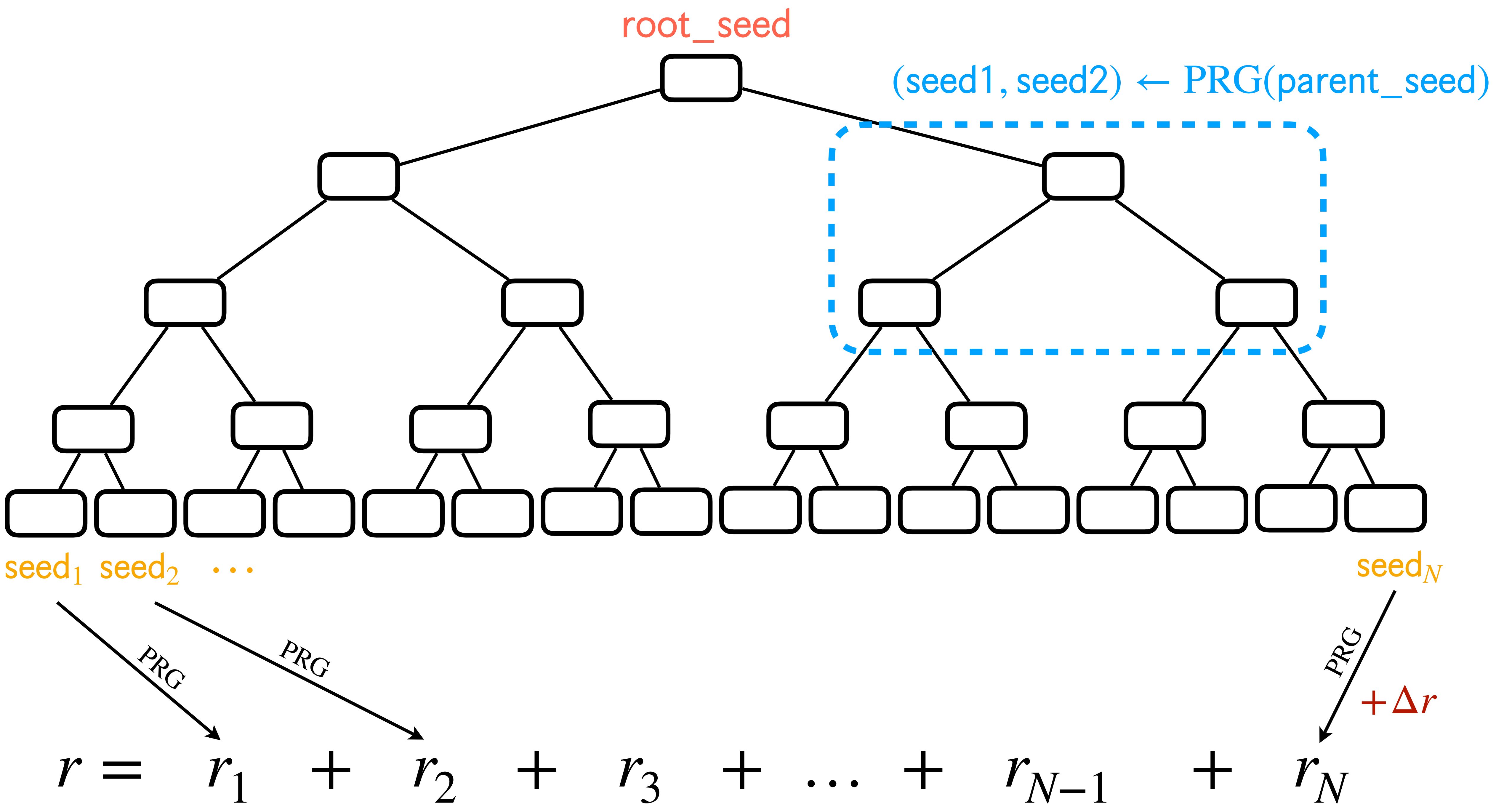
Using a Seed Tree

[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)

$$\begin{tikzcd}
 & \mathrm{seed}_1 \arrow[d, "\mathrm{PRG}"'] & & \mathrm{seed}_2 \arrow[d, "\mathrm{PRG}"'] & & \mathrm{seed}_3 \arrow[d, "\mathrm{PRG}"'] & & & & \mathrm{seed}_{N-1} \arrow[d, "\mathrm{PRG}"'] & & \mathrm{seed}_N \arrow[d, "\mathrm{PRG}"', "+\Delta r"] \\
r = & r_1 & + & r_2 & + & r_3 & + & \dots & + & r_{N-1} & + & r_N
\end{tikzcd}$$

Using a Seed Tree

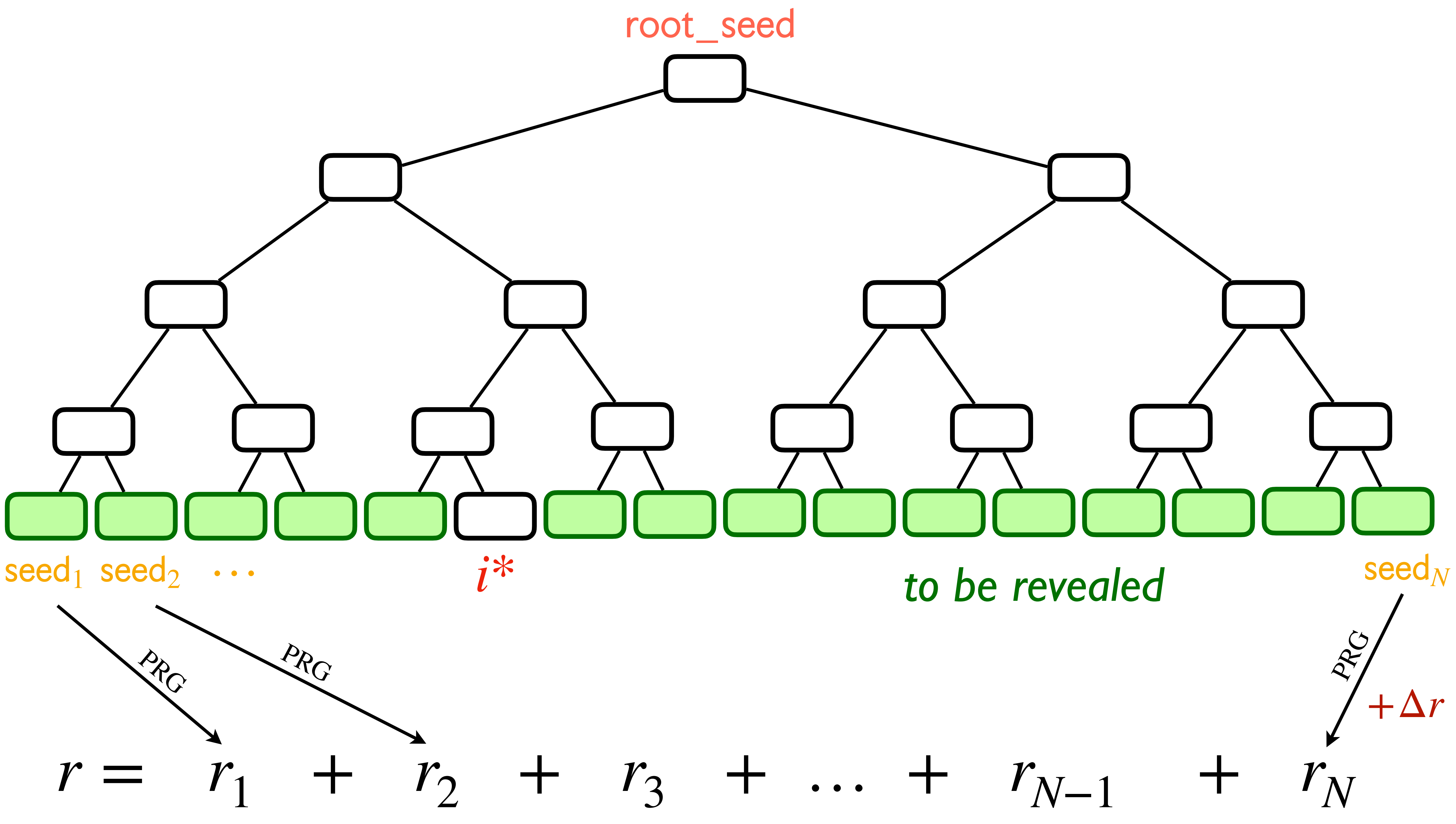
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



$$r = r_1 + r_2 + r_3 + \ldots + r_{N-1} + r_N$$

Using a Seed Tree

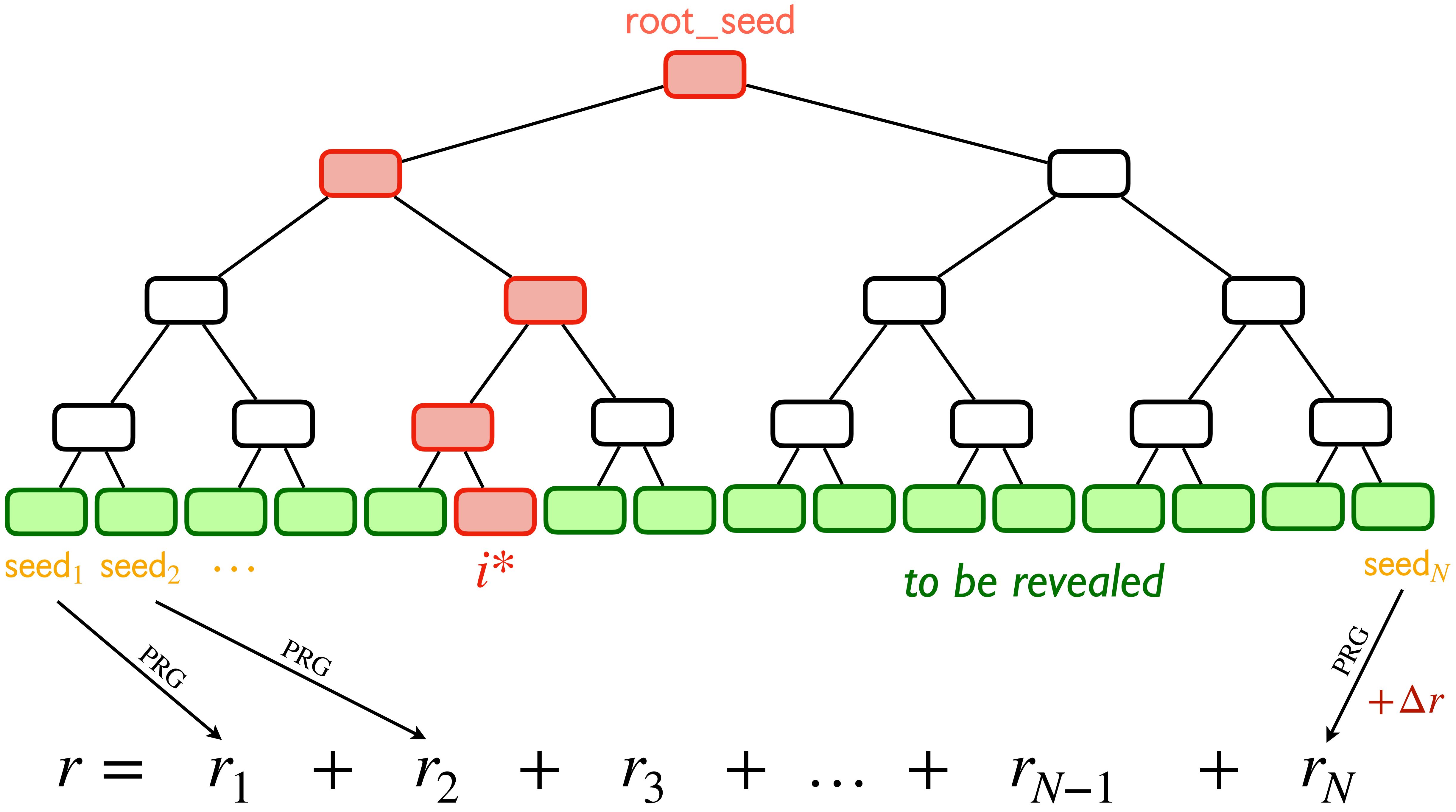
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



$$r = r_1 + r_2 + r_3 + \ldots + r_{N-1} + r_N$$

Using a Seed Tree

[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)

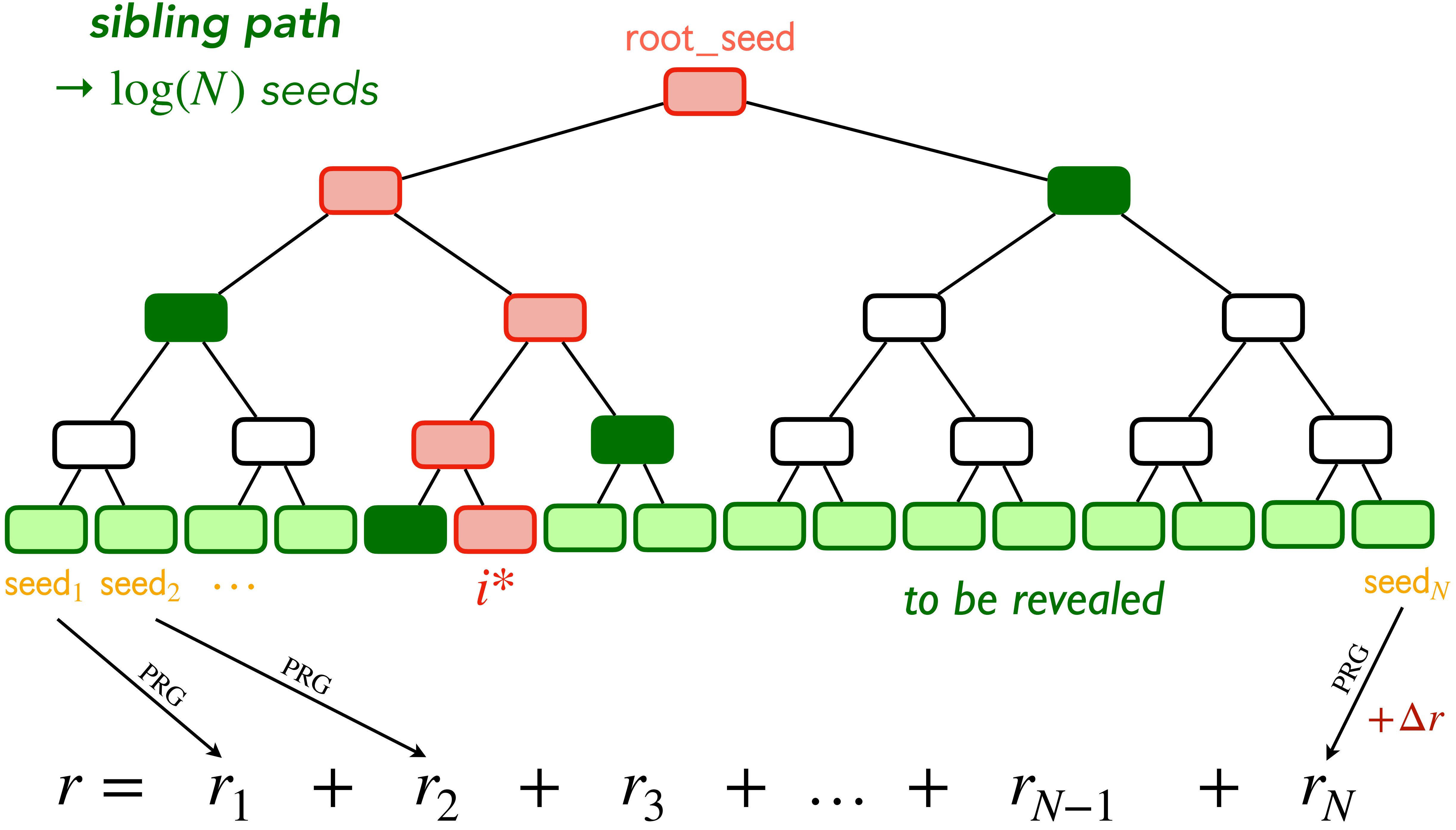


Using a Seed Tree

[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)

sibling path

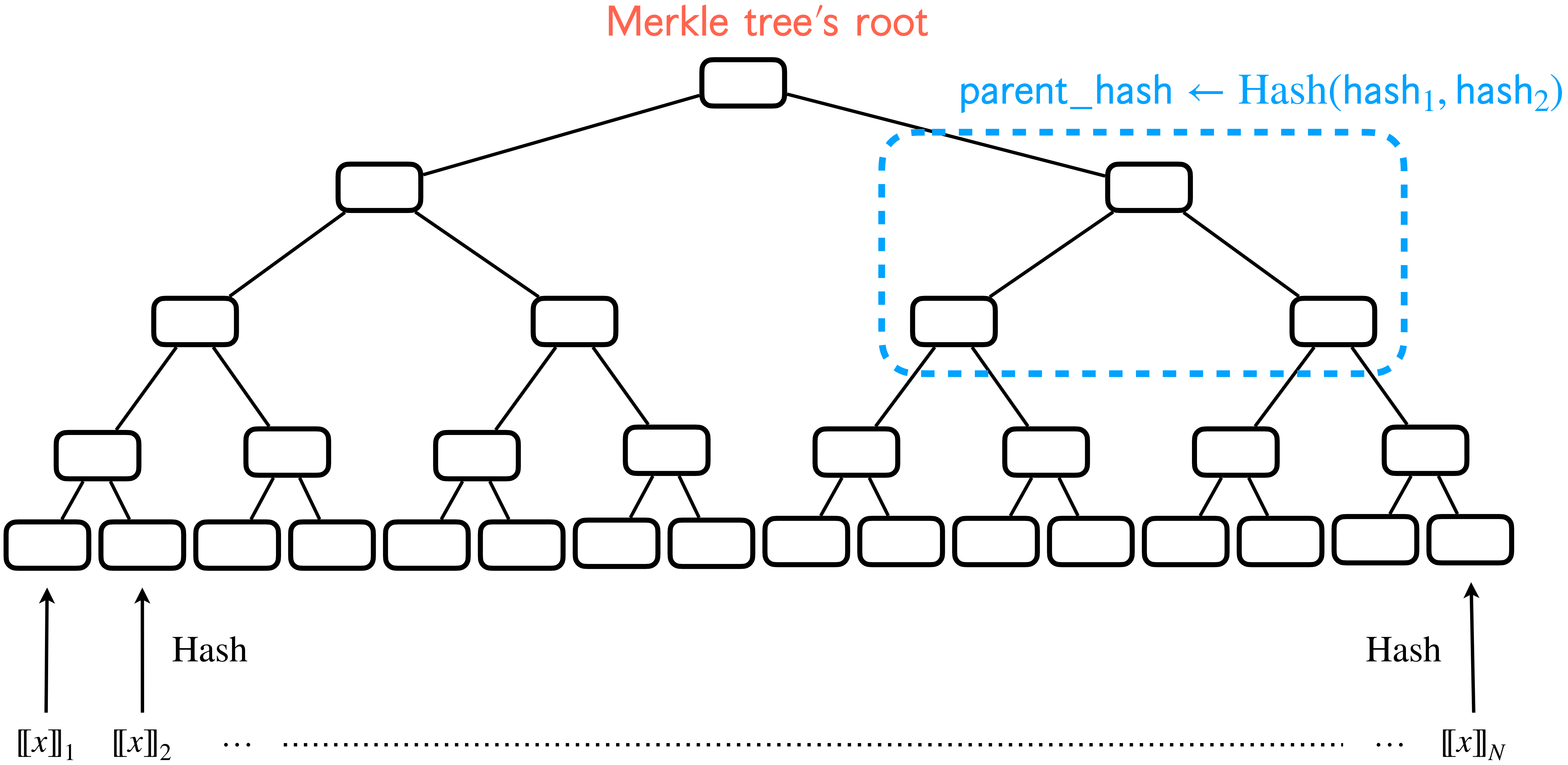
→ $\log(N)$ *seeds*



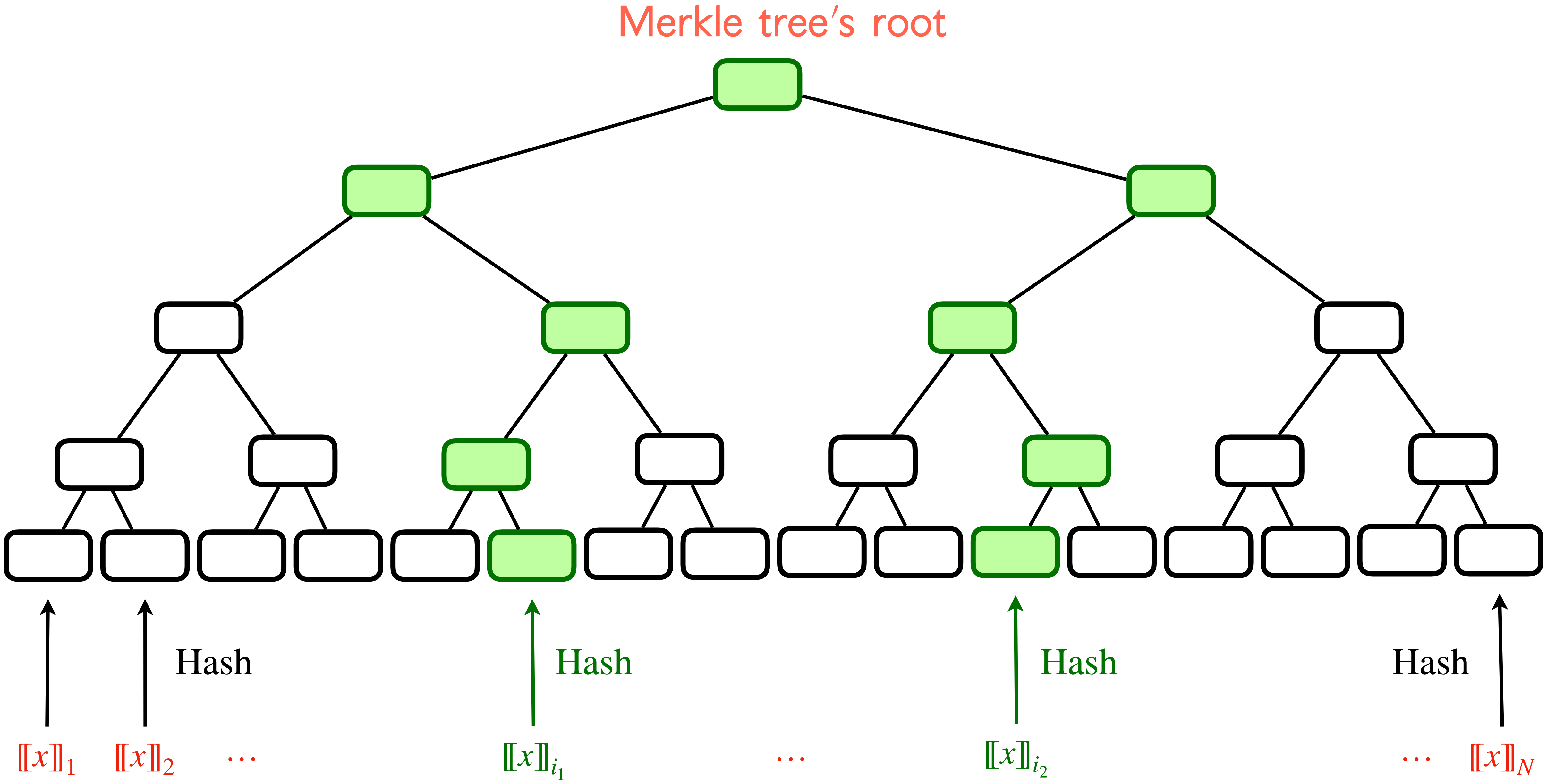
$$r = r_1 + r_2 + r_3 + \ldots + r_{N-1} + r_N$$

TCitH-MT: Using a Merkle tree

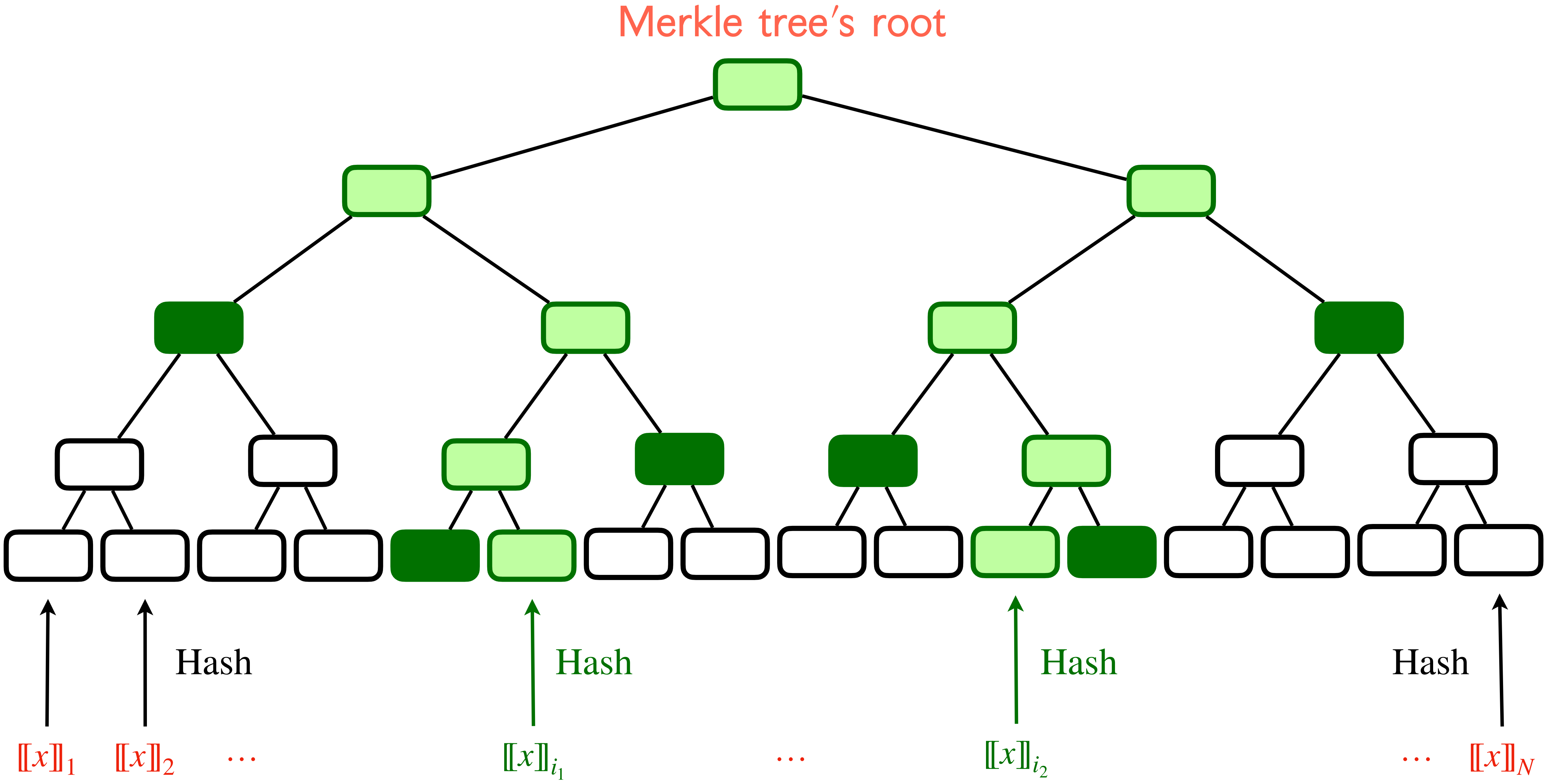
TCitH-MT: Using a Merkle tree



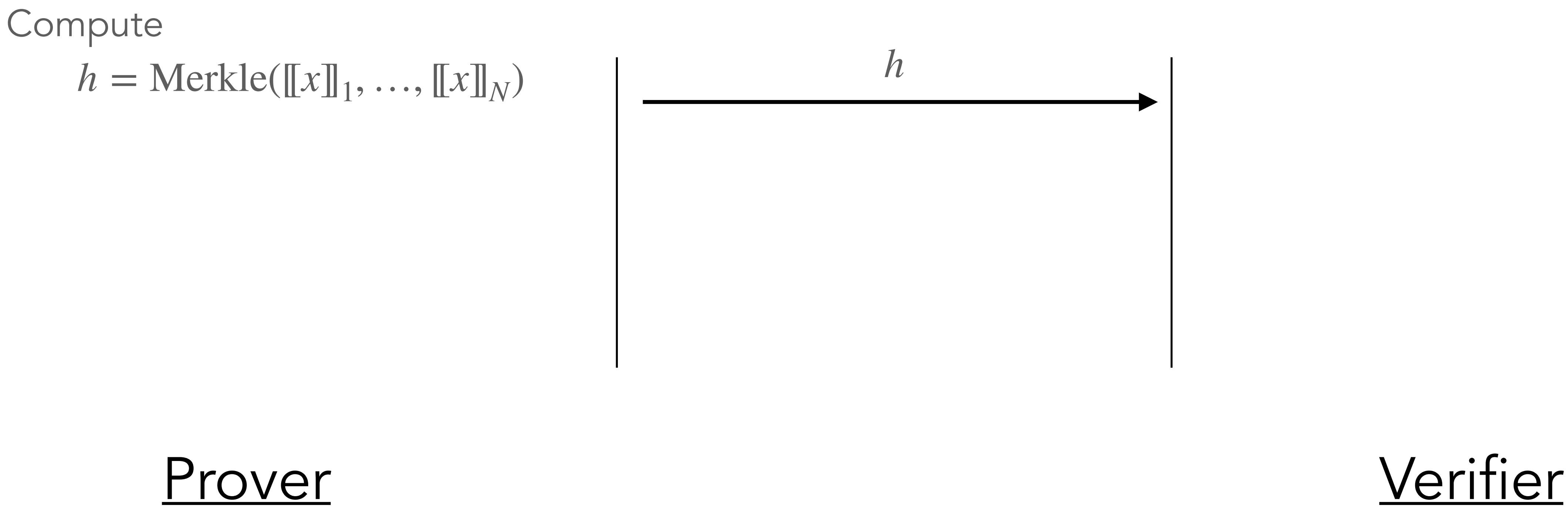
TCitH-MT: Using a Merkle tree



TCitH-MT: Using a Merkle tree



TCitH-MT: Using a Merkle tree



Compute

$h = \mathrm{Merkle}([\![x]\!]_1, \ldots, [\![x]\!]_N)$

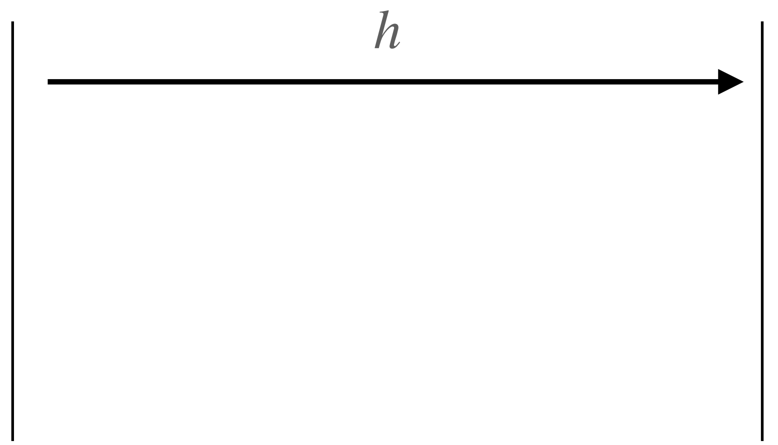
$\xrightarrow{h}$

Prover

Verifier

TCitH-MT: Using a Merkle tree

Compute
$h = \text{Merkle}(\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$



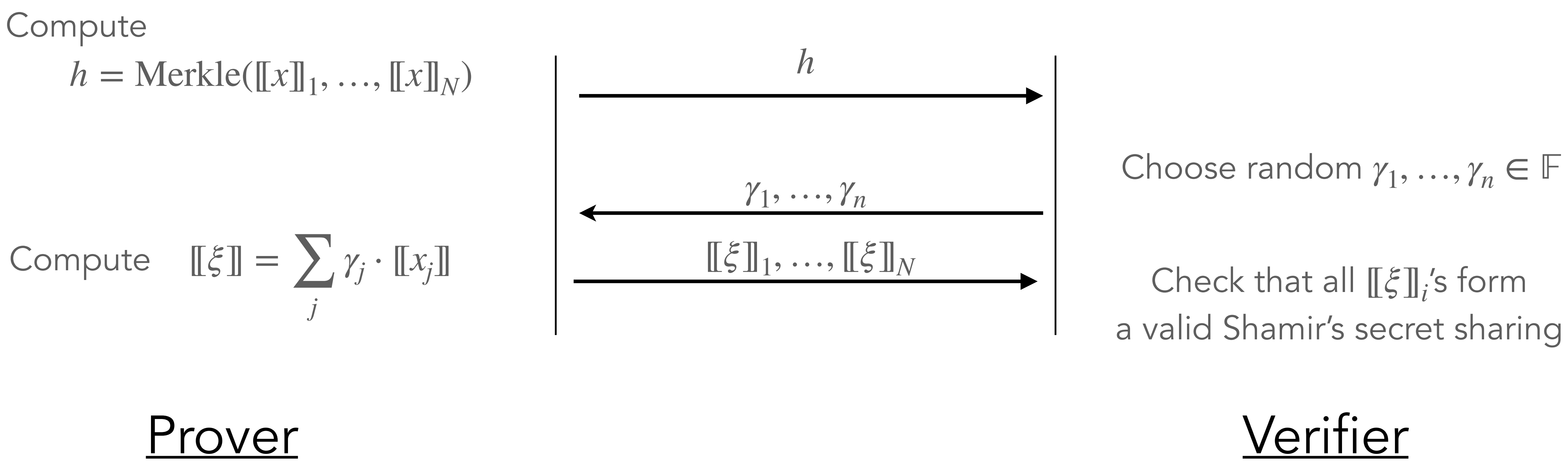
h

Prover

Verifier

🤔 **How to be sure that the committed shares correspond to a valid Shamir's secret sharing?**

TCitH-MT: Using a Merkle tree



Compute

$h = \text{Merkle}(\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$

h

Choose random $\gamma_1, \ldots, \gamma_n \in \mathbb{F}$

$\gamma_1, \ldots, \gamma_n$

Compute $\llbracket \xi \rrbracket = \sum_j \gamma_j \cdot \llbracket x_j \rrbracket$

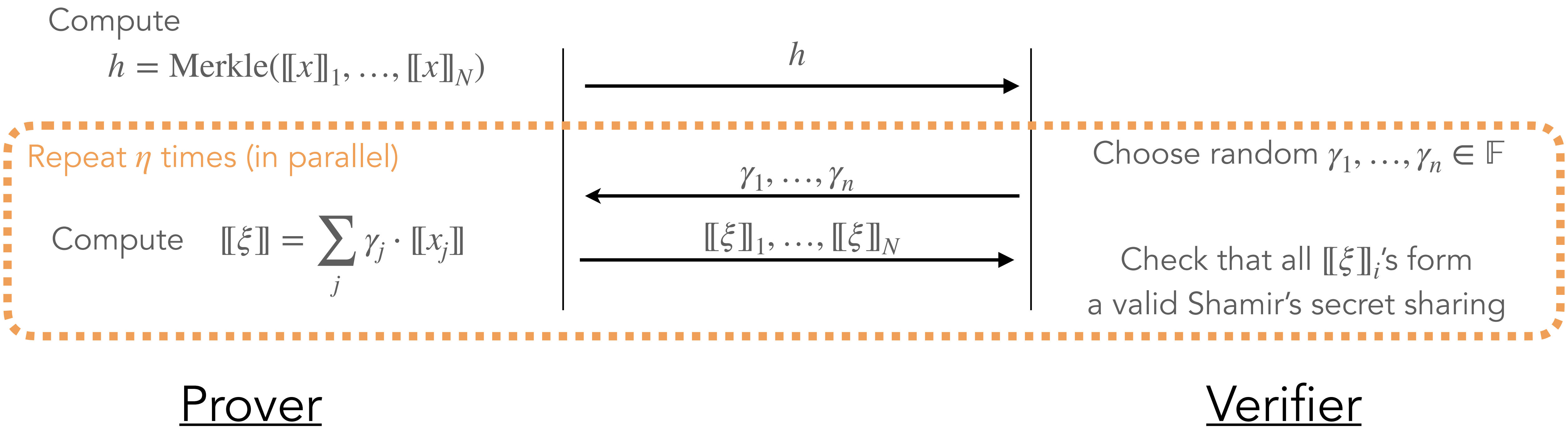
$\llbracket \xi \rrbracket_1, \ldots, \llbracket \xi \rrbracket_N$

Check that all $\llbracket \xi \rrbracket_i$'s form a valid Shamir's secret sharing

Prover

Verifier

TCitH-MT: Using a Merkle tree



Compute

$h = \text{Merkle}(\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$

h

Repeat η times (in parallel)

Choose random $\gamma_1, \ldots, \gamma_n \in \mathbb{F}$

$\gamma_1, \ldots, \gamma_n$

Compute $\llbracket \xi \rrbracket = \sum_j \gamma_j \cdot \llbracket x_j \rrbracket$

$\llbracket \xi \rrbracket_1, \ldots, \llbracket \xi \rrbracket_N$

Check that all $\llbracket \xi \rrbracket_i$'s form a valid Shamir's secret sharing

Prover

Verifier

TCitH-MT: Using a Merkle tree

Compute

$h = \text{Merkle}(\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$

$\xrightarrow{h}$

Repeat η times (in parallel)

Choose random $\gamma_1, \ldots, \gamma_n \in \mathbb{F}$

$\xleftarrow{\gamma_1, \ldots, \gamma_n}$

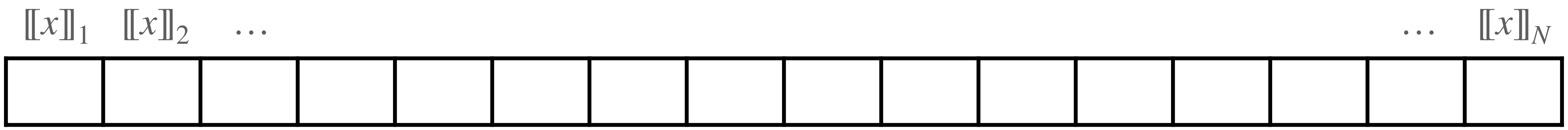
Compute $\llbracket \xi \rrbracket = \sum_j \gamma_j \cdot \llbracket x_j \rrbracket$

$\xrightarrow{\llbracket \xi \rrbracket_1, \ldots, \llbracket \xi \rrbracket_N}$

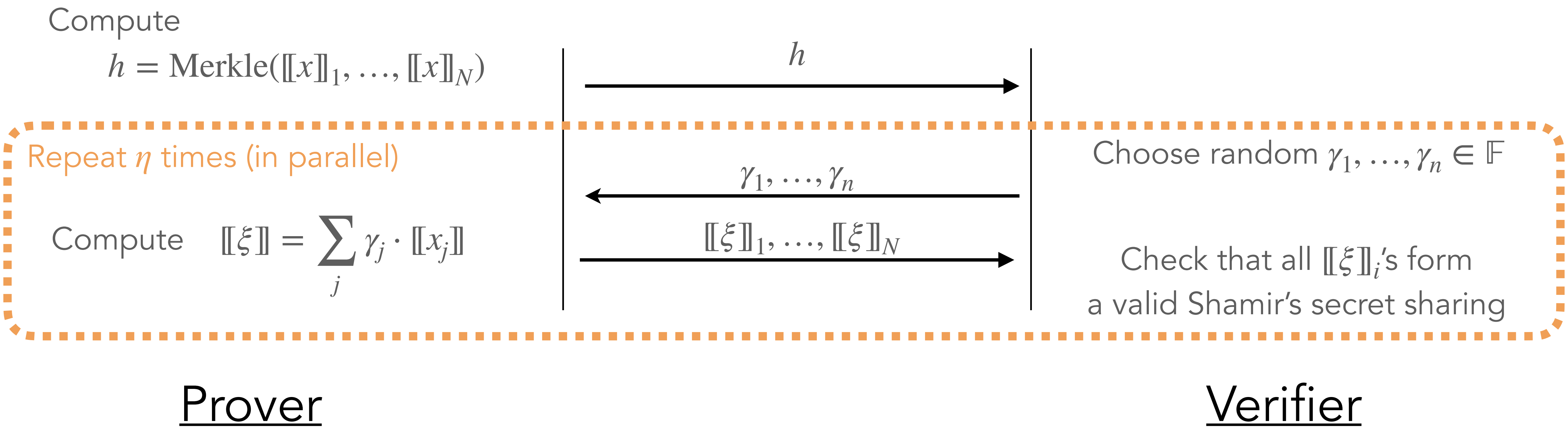
Check that all $\llbracket \xi \rrbracket_i$'s form a valid Shamir's secret sharing

Prover

Verifier


$\llbracket x \rrbracket_1$
$\llbracket x \rrbracket_2$
…
…
$\llbracket x \rrbracket_N$

TCitH-MT: Using a Merkle tree



Compute

$h = \text{Merkle}([\![x]\!]_1, \ldots, [\![x]\!]_N)$

h

Repeat η times (in parallel)

Choose random $\gamma_1, \ldots, \gamma_n \in \mathbb{F}$

$\gamma_1, \ldots, \gamma_n$

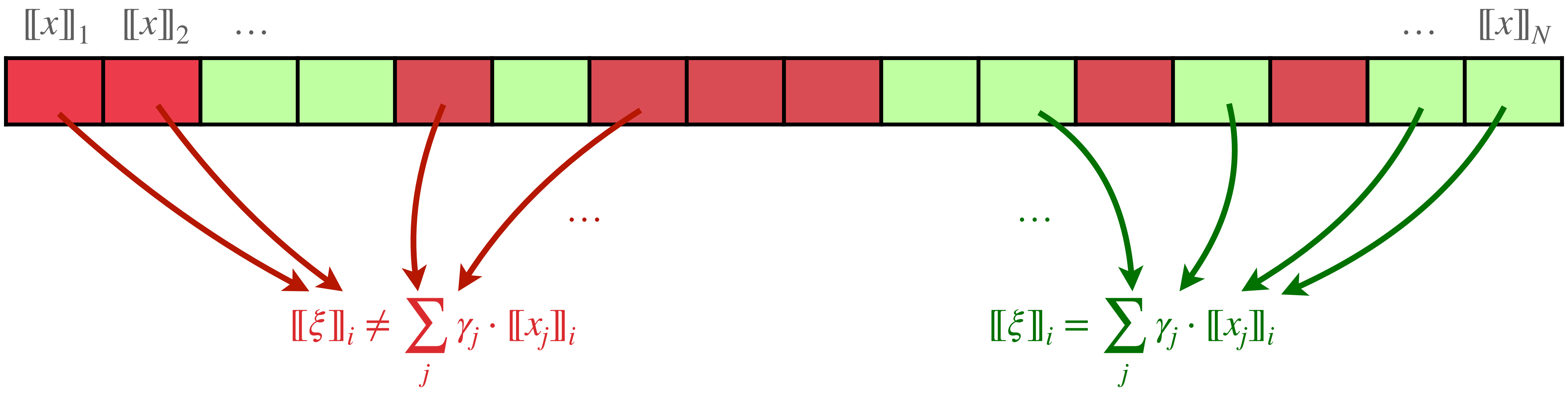
Compute $[\![\xi]\!] = \sum_j \gamma_j \cdot [\![x_j]\!]$

$[\![\xi]\!]_1, \ldots, [\![\xi]\!]_N$

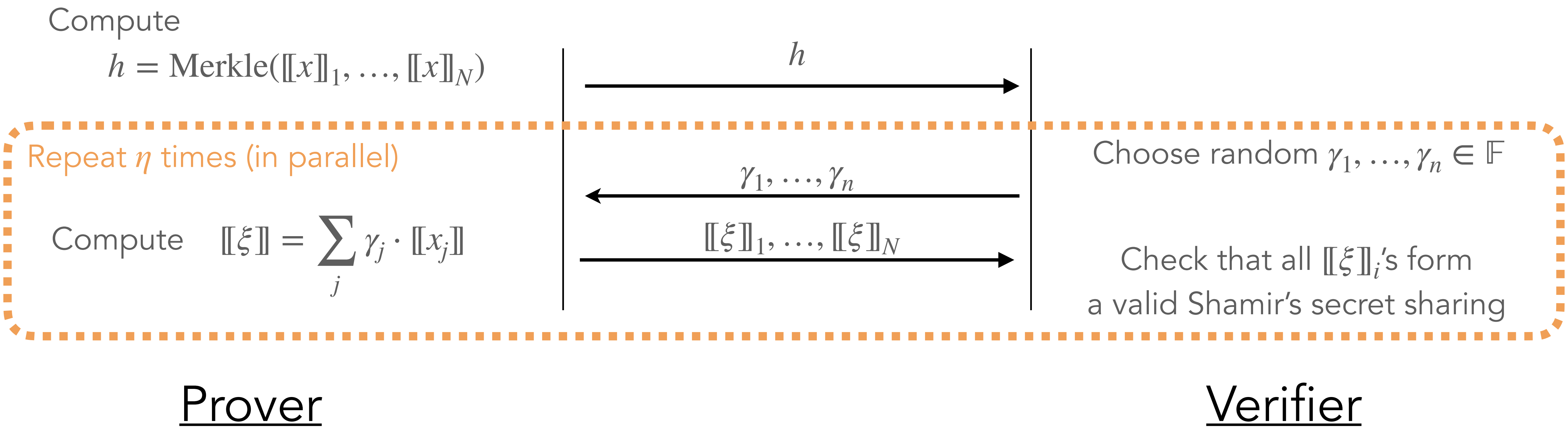
Check that all $[\![\xi]\!]_i$'s form a valid Shamir's secret sharing

Prover

Verifier


$[\![x]\!]_1$
$[\![x]\!]_2$
$\ldots$
$\ldots$
$[\![x]\!]_N$
$\ldots$
$\ldots$
$[\![\xi]\!]_i \neq \sum_j \gamma_j \cdot [\![x_j]\!]_i$
$[\![\xi]\!]_i = \sum_j \gamma_j \cdot [\![x_j]\!]_i$

TCitH-MT: Using a Merkle tree



Compute

$h = \mathrm{Merkle}([\![x]\!]_1, \dots, [\![x]\!]_N)$

h

Repeat η times (in parallel)

Choose random $\gamma_1, \dots, \gamma_n \in \mathbb{F}$

$\gamma_1, \dots, \gamma_n$

Compute $[\![\xi]\!] = \sum_j \gamma_j \cdot [\![x_j]\!]$

$[\![\xi]\!]_1, \dots, [\![\xi]\!]_N$

Check that all $[\![\xi]\!]_i$'s form a valid Shamir's secret sharing

Prover

Verifier

We can prove that

$$\mathrm{Prob}\left[\{[\![x]\!]_i\}_{i \in E} \text{ does not form a valid sharing}\right] \leq \frac{\binom{N}{d_w+1}^2}{|\mathbb{F}|^\eta}$$

where $E = \{i : [\![\xi]\!]_i = \sum_j \gamma_j \cdot [\![x_j]\!]_i$ for all repetitions$\}$.

Applications of the TCitH Framework

MPCitH-based NIST Candidates

Can rely on the TCitH Framework using the same MPC protocol:

- Number of opened parties: $\ell = 1$
- Linear MPC protocol: $d_\alpha = d_w = \ell$
- Rely on seed trees

🧐

Same soundness error

Same communication cost

MPCitH-based NIST Candidates

		Additive MPCitH		TCitH (GGM tree)	
	Size (in KB)	*Traditional*	*Hypercube*	*Threshold*	*Saving*
AIMer	4.2	4.53	3.22	3.22	-0 %
Biscuit	4.8	17.71	4.65	4.24	-16 %
MIRA	5.6	384.26	20.11	9.89	-51 %
MiRitH-Ia	5.7	54.15	6.60	5.42	-18 %
MiRitH-Ib	6.3	89.50	8.66	6.66	-23 %
MQOM-31	6.3	96.41	11.27	8.74	-21 %
MQOM-251	6.6	44.11	7.56	5.97	-21 %
RYDE	6.0	12.41	4.65	4.65	-0 %
SDitH-256	8.2	78.37	7.23	5.31	-27 %
SDitH-251	8.2	19.15	7.53	6.44	-14 %

# Party Emulations (per repetition): N $1 + \log_2 N$ $1 + \left\lceil \frac{\log_2 N}{\log_2 |\mathbb{F}|} \right\rceil$

Shorter MPCitH-based Signatures

Rely on the TCitH Framework using share-wise multiplication:

- Number of opened parties: $\ell = 1$
- Quadratic (or higher degree) MPC protocol: $d_\alpha > d_w = \ell$
- Rely on seed trees

To compute $\llbracket a \cdot b \rrbracket$ from $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$:

$$\forall i, \quad \llbracket a \cdot b \rrbracket_i \leftarrow \llbracket b \rrbracket_i \cdot \llbracket b \rrbracket_i$$

(no need for communication between parties)

Shorter MPCitH-based Signatures

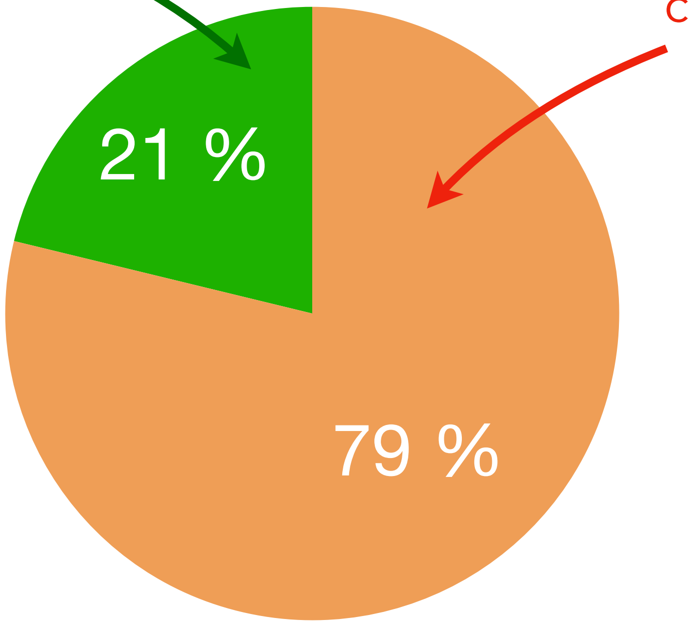
	Original Size	*Our Variant*	*Saving*
Biscuit	4 758 B	4 048 B	-15 %
MIRA	5 640 B	5 340 B	-5 %
MiRitH-Ia	5 665 B	4 694 B	-17 %
MiRitH-Ib	6 298 B	5 245 B	-17 %
MQOM-31	6 328 B	4 027 B	-37 %
MQOM-251	6 575 B	4 257 B	-35 %
RYDE	5 956 B	5 281 B	-11 %
SDitH	8 241 B	7 335 B	-27 %

	Former Size	*TCitH-GGM*	*Saving*
MQ over GF(4)	8 609 B	3 858 B	-55 %
SD over GF(2)	11 160 B	7 354 B	-34 %
6-split SD over GF(2)	12 066 B	6 974 B	-42 %

Shorter MPCitH-based Signatures

Due to the MPC protocol

(818 bytes)



Due to the sharing commitment (with GGM trees)

(3040 bytes)

Lower bound: ≥ 2048 *bytes*

Size of the signature
relying on MQ over $\mathbb{F}_4$, with 256 parties.

Other applications

- Efficient ring signatures from any one-way function
- Zero-knowledge arguments for arithmetic circuits

  Can rely on packed secret sharings.
- Exact zero-knowledge arguments for lattices

  Rely on packed secret sharings.
- …

Conclusion

Conclusion

- New generation of MPCitH-based proof systems:
  - VOLE-in-the-Head
  - TC-in-the-Head

Conclusion

- New generation of MPCitH-based proof systems:
  - VOLE-in-the-Head
  - TC-in-the-Head

- Post-quantum signatures:
  - Signature sizes below 5 KB while keeping conservative assumption
  - Bottleneck (computational and communication): symmetric parts

Conclusion

- New generation of MPCitH-based proof systems:
  - VOLE-in-the-Head
  - TC-in-the-Head
- Post-quantum signatures:
  - Signature sizes below 5 KB while keeping conservative assumption
  - Bottleneck (computational and communication): symmetric parts
- Advanced signatures:
  - Ring signatures from one-way function
  - What's next?

Conclusion

- New generation of MPCitH-based proof systems:
  - VOLE-in-the-Head
  - TC-in-the-Head
- Post-quantum signatures:
  - Signature sizes below 5 KB while keeping conservative assumption
  - Bottleneck (computational and communication): symmetric parts
- Advanced signatures:
  - Ring signatures from one-way function
  - What's next?

Thank you for your attention !