Recent Advances in MPCitH-based Post-Quantum Signatures

Thibauld Feneuil

Séminaire Crypto Rennes

March 22, 2024 — Rennes (France)

CRYPTOEXPERTS
WE INNOVATE TO SECURE YOUR BUSINESS

Table of Contents

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?

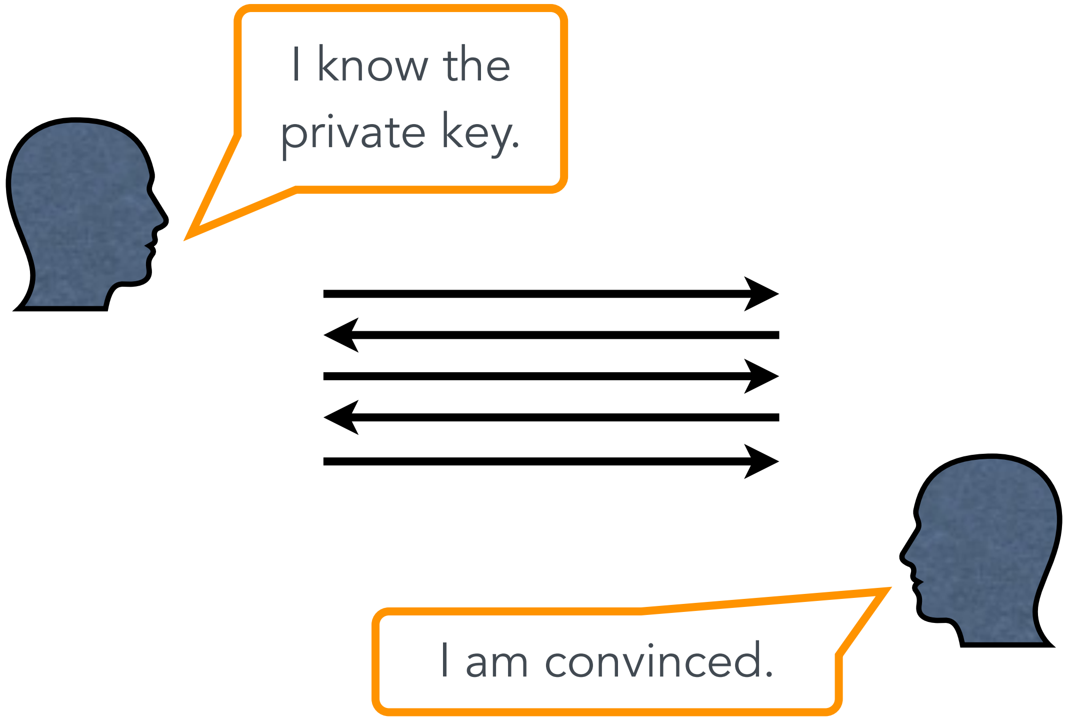
Hash & Sign

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

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?

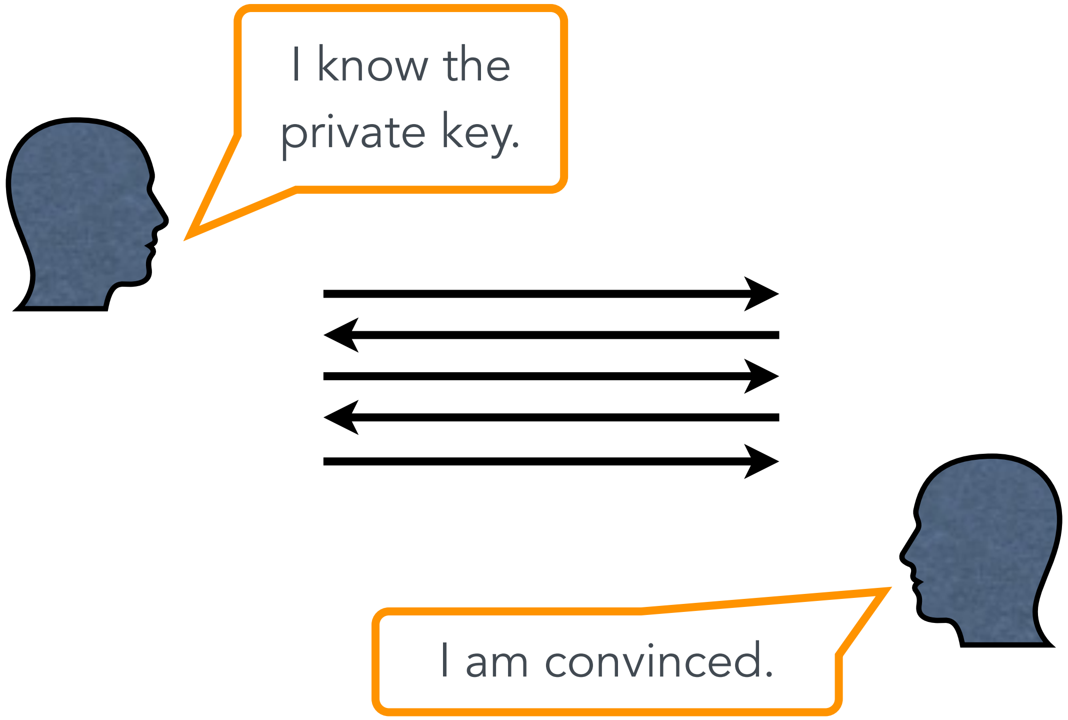
Hash & Sign

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

Very hard
to compute

- Short signatures
- “Trapdoor” in the public key

From an identification scheme



- Large(r) signatures
- Short public key

Identification Scheme

I know 🔑.

Commitment

Challenge 1

Response 1

⋮

Challenge n

Response n

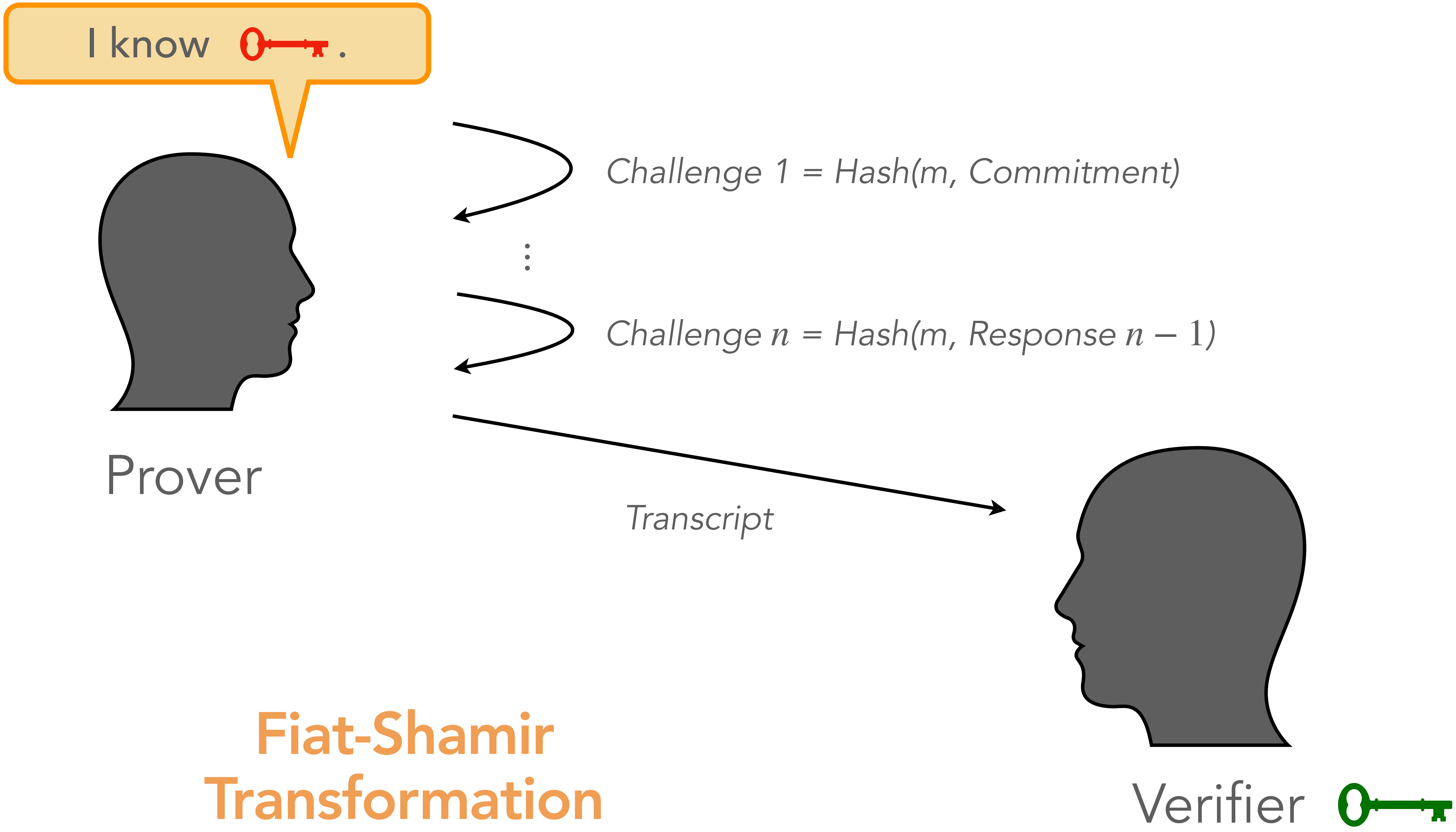
Prover

Verifier 🔑

I am convinced.

- **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 🔑

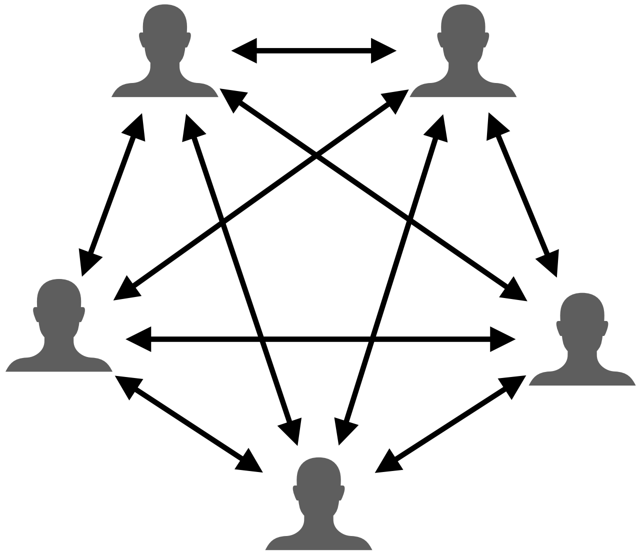
Fiat-Shamir Transformation

m: message to sign

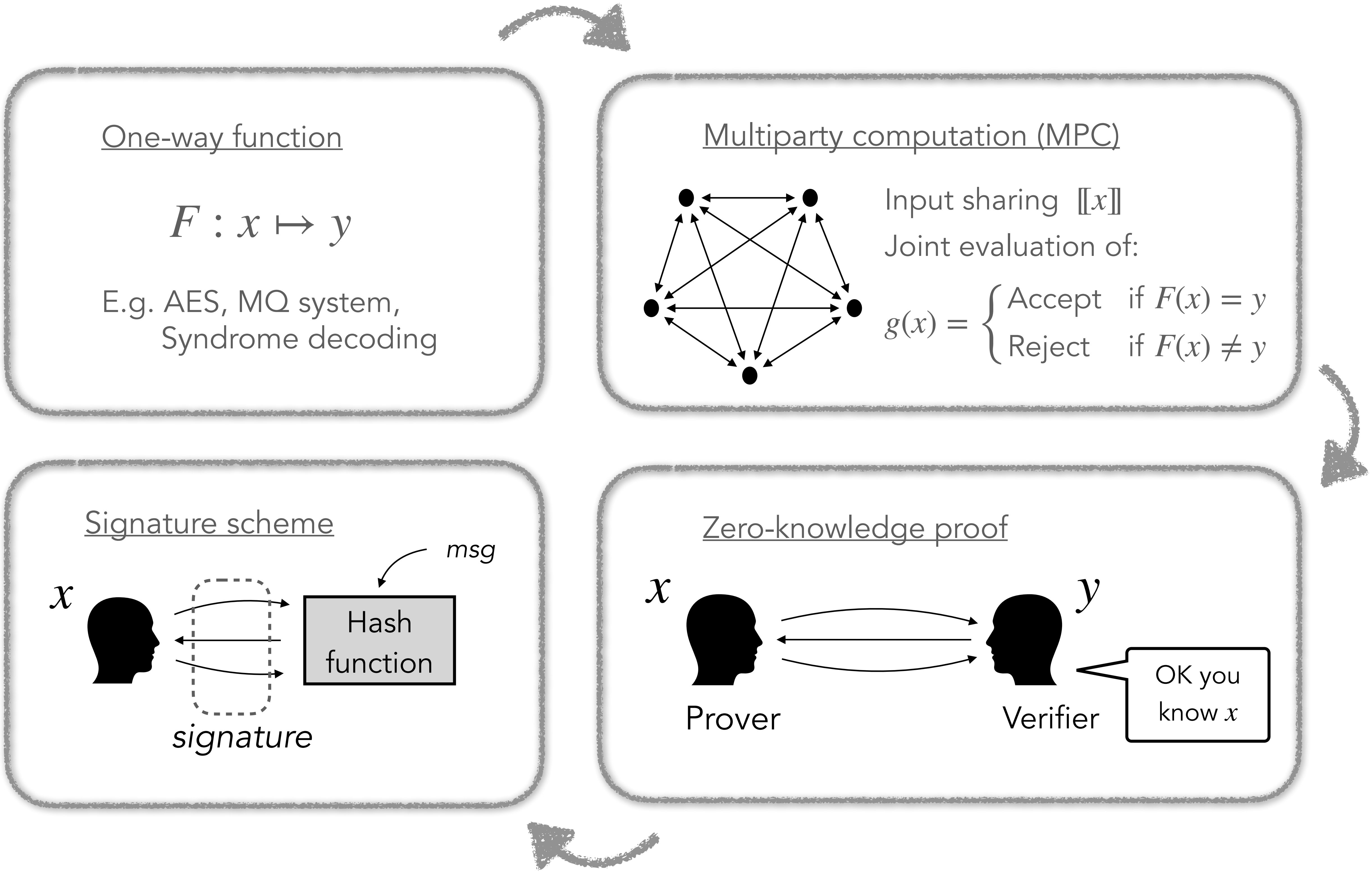
MPC in the Head

- **[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

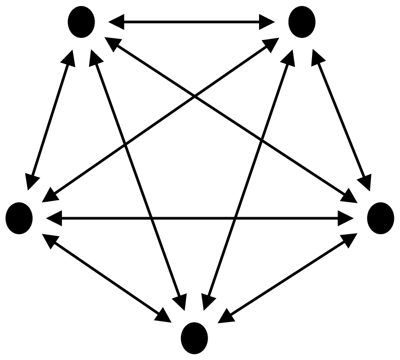
Signature scheme

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

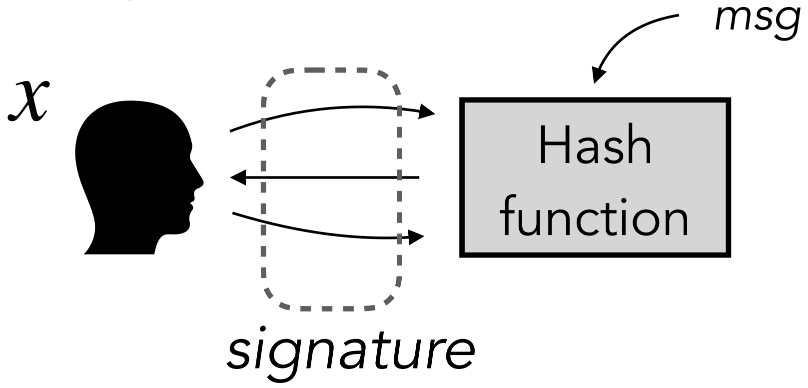


Input sharing $[\![x]\!]$

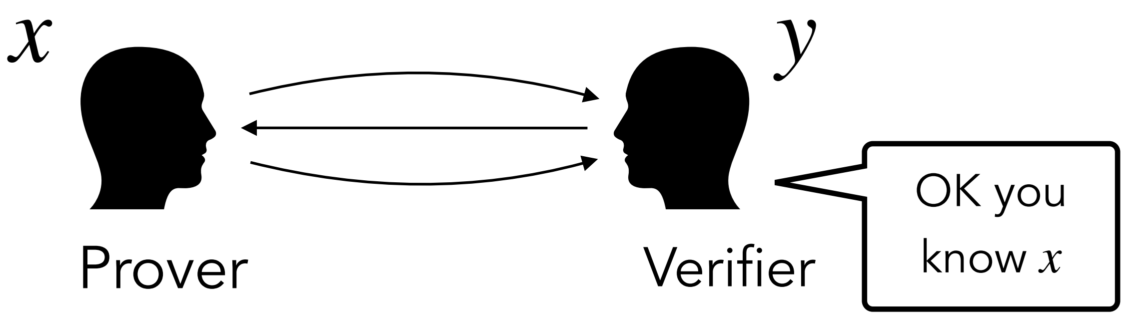
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

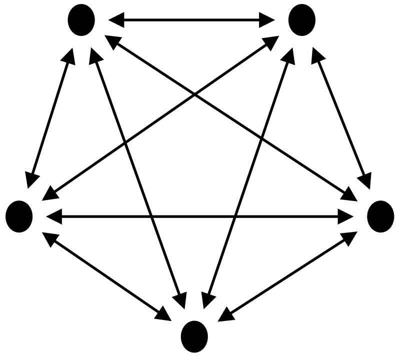


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

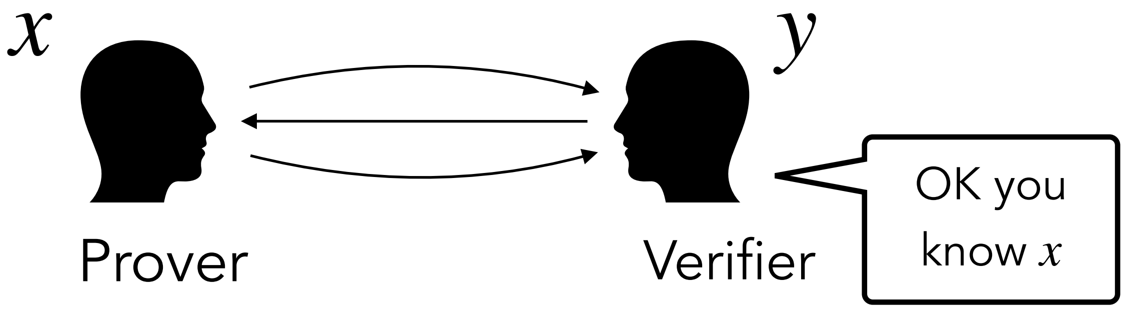


Input sharing $[\![x]\!]$

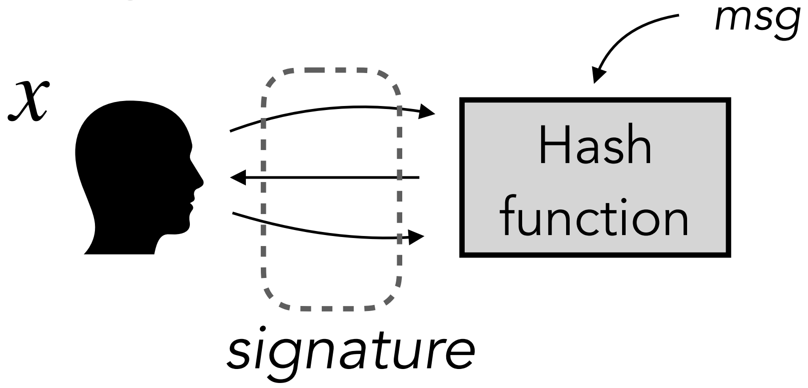
Joint evaluation of:

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

Zero-knowledge proof



Signature scheme

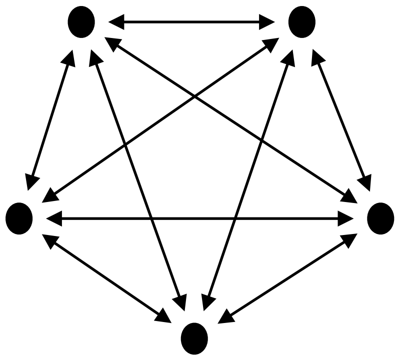


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

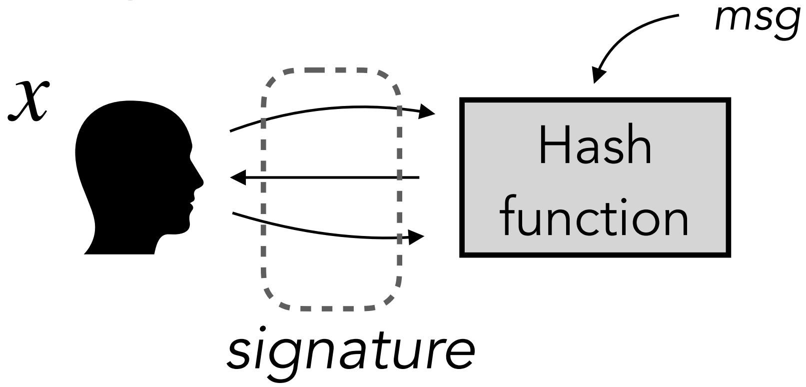


Input sharing $[\![x]\!]$

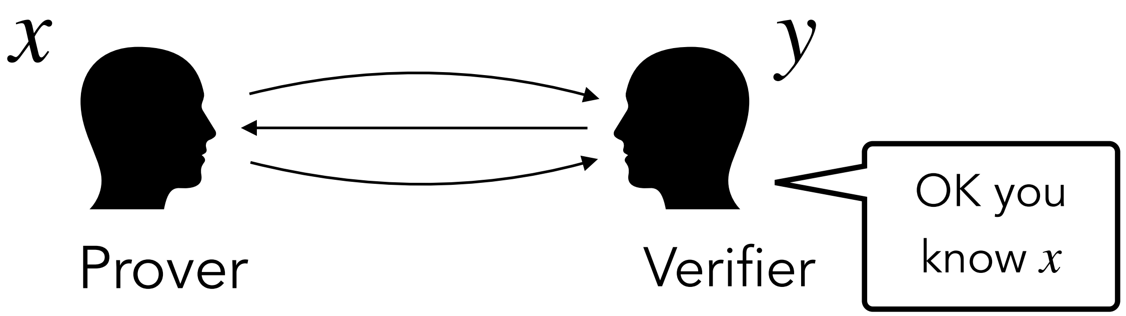
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

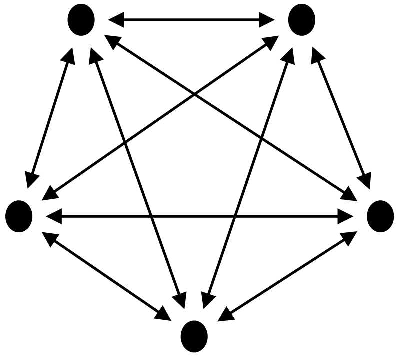


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

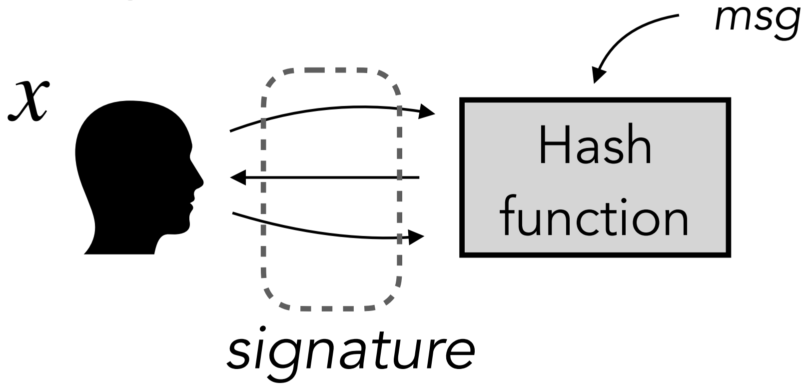


Input sharing $[\![x]\!]$

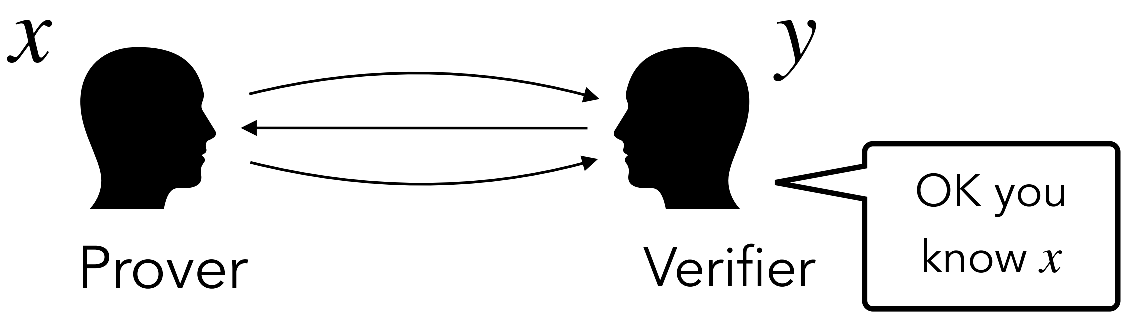
Joint evaluation of:

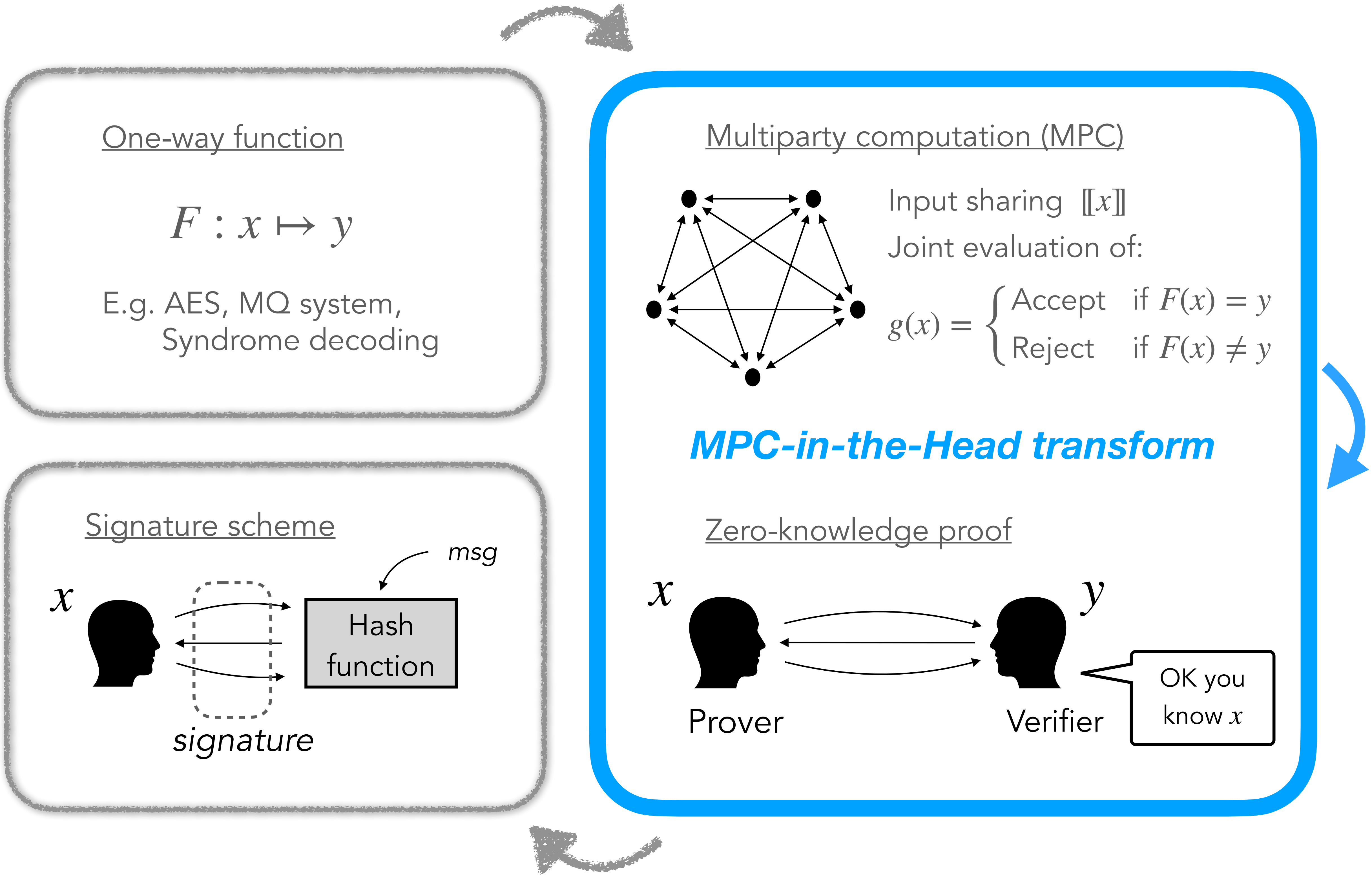
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

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:

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

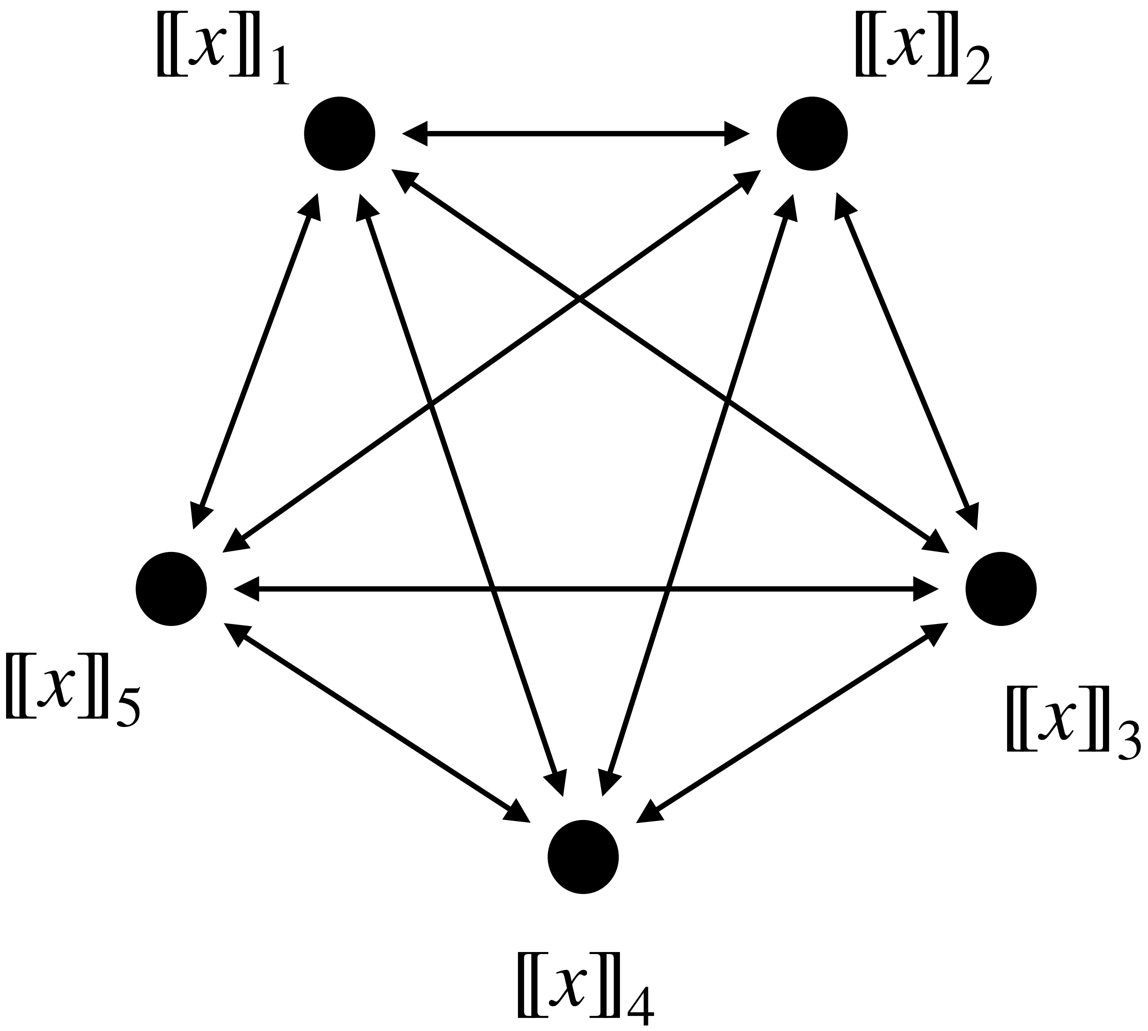
MPC-in-the-Head transform

Zero-knowledge proof

Signature scheme

MPCitH: general principle

MPC model



$[\![x]\!]$ is a linear secret sharing of x

Additive sharing:

$$x = [\![x]\!]_1 + [\![x]\!]_2 + \ldots + [\![x]\!]_N$$

Shamir's sharing:

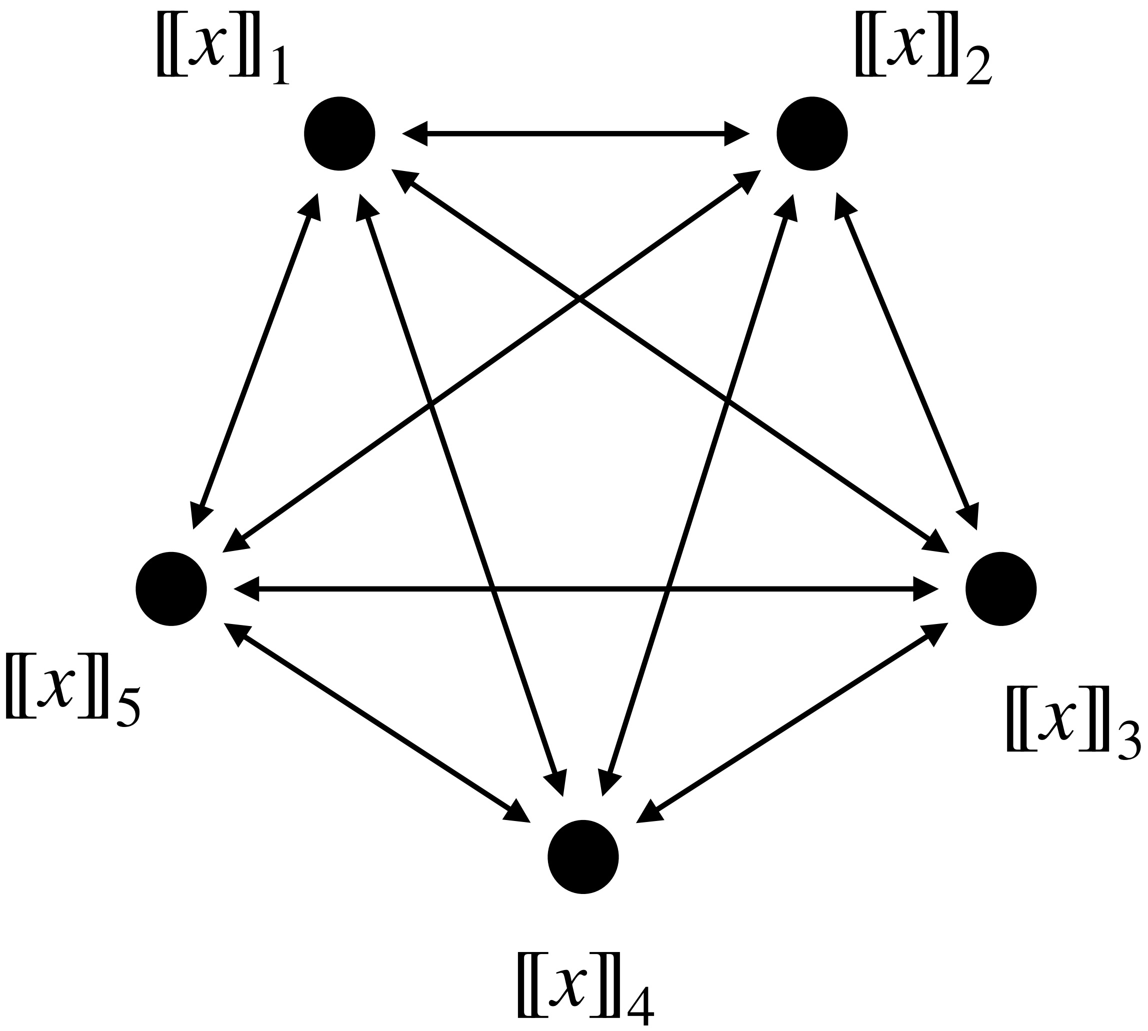
Let us build the degree-ℓ polynomial P such that

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

The shares are defined as

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

MPC model



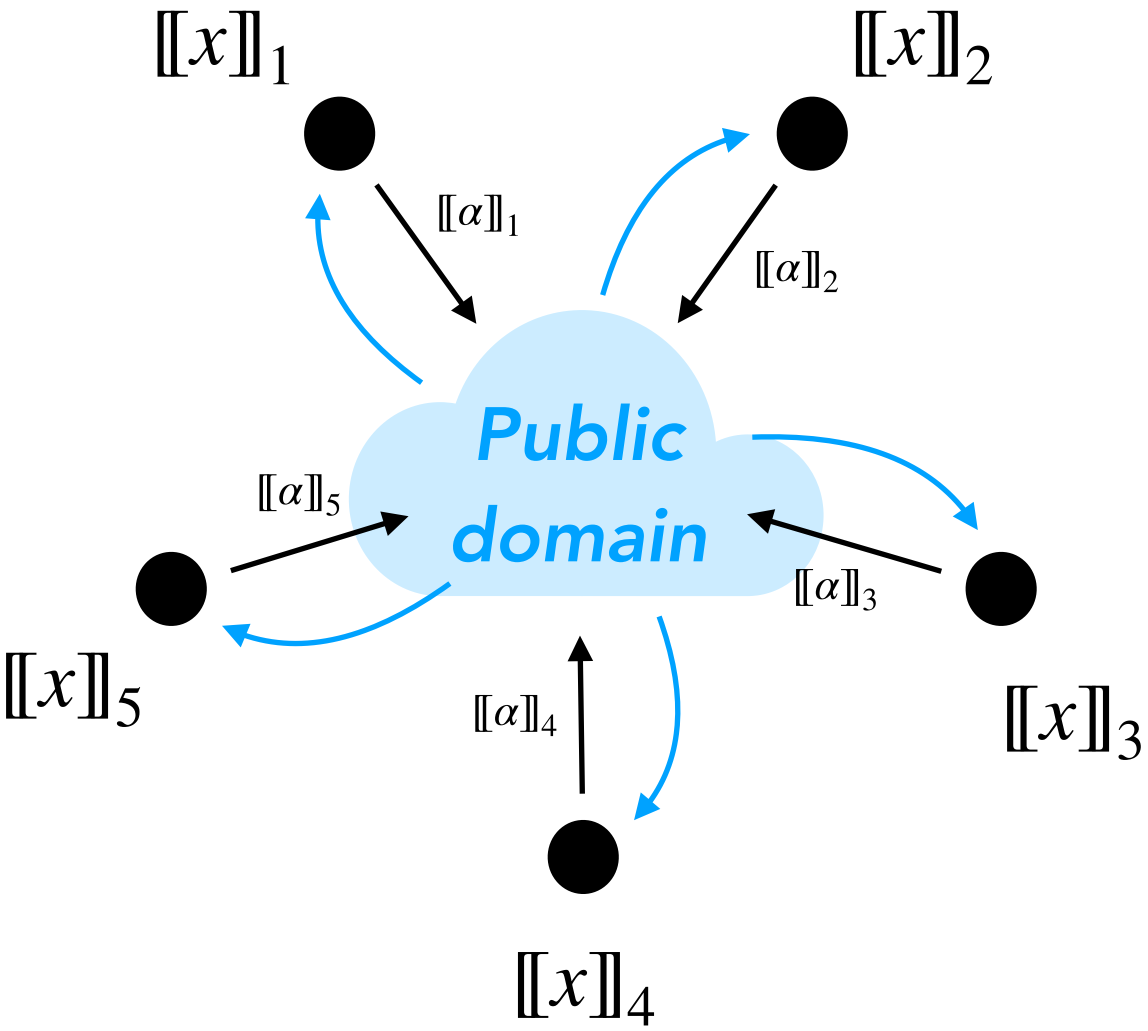
$[\![x]\!]$ is a linear secret sharing of x

- **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

MPC model



$[\![x]\!]$ is a linear secret sharing of x

- **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*)*

MPCitH transform

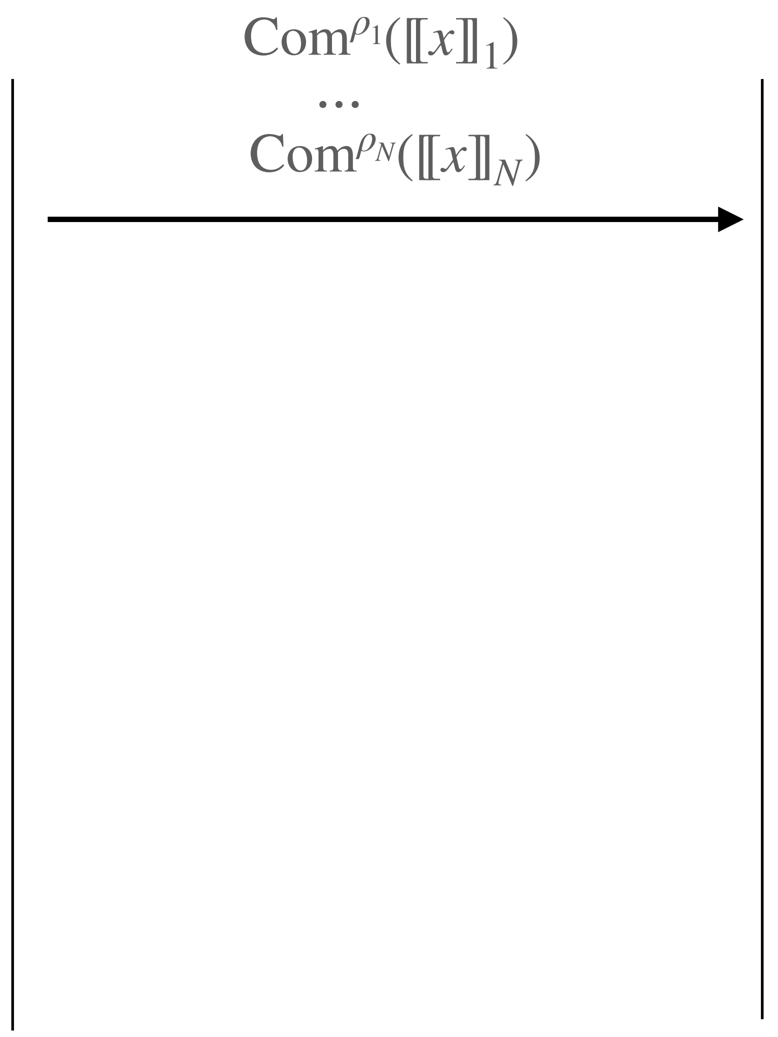
Prover

Verifier

MPCitH 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

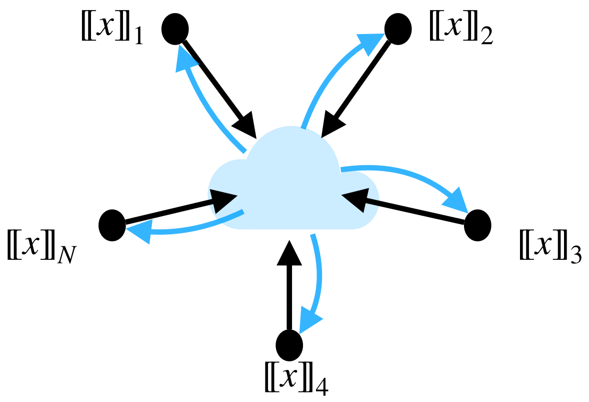
Verifier

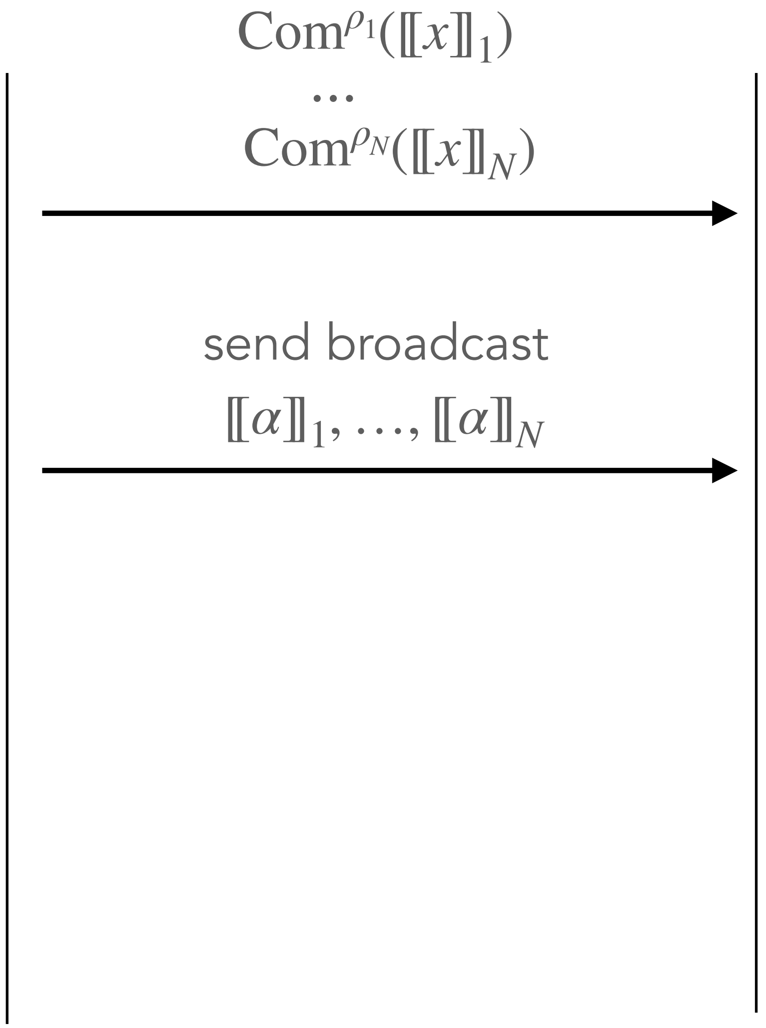
MPCitH transform

① Generate and commit shares

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

② Run MPC in their head





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

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

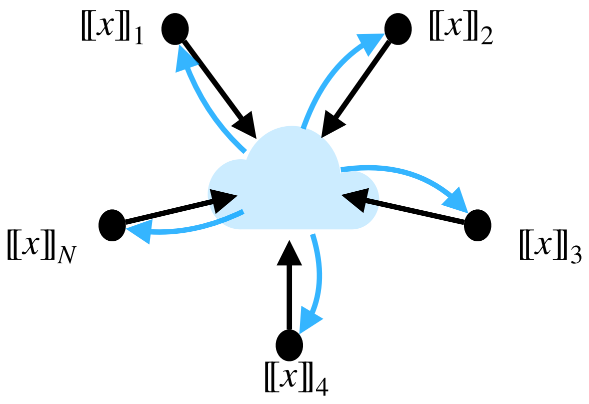
Prover

Verifier

MPCitH transform

① Generate and commit shares
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

② Run MPC in their head



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

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

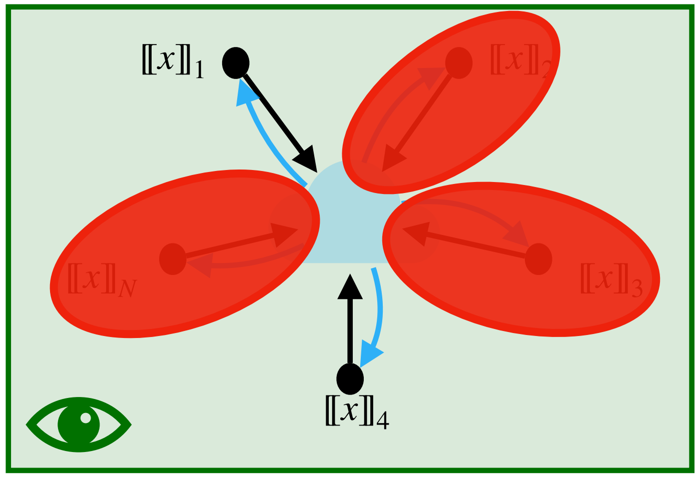
Verifier

MPCitH transform

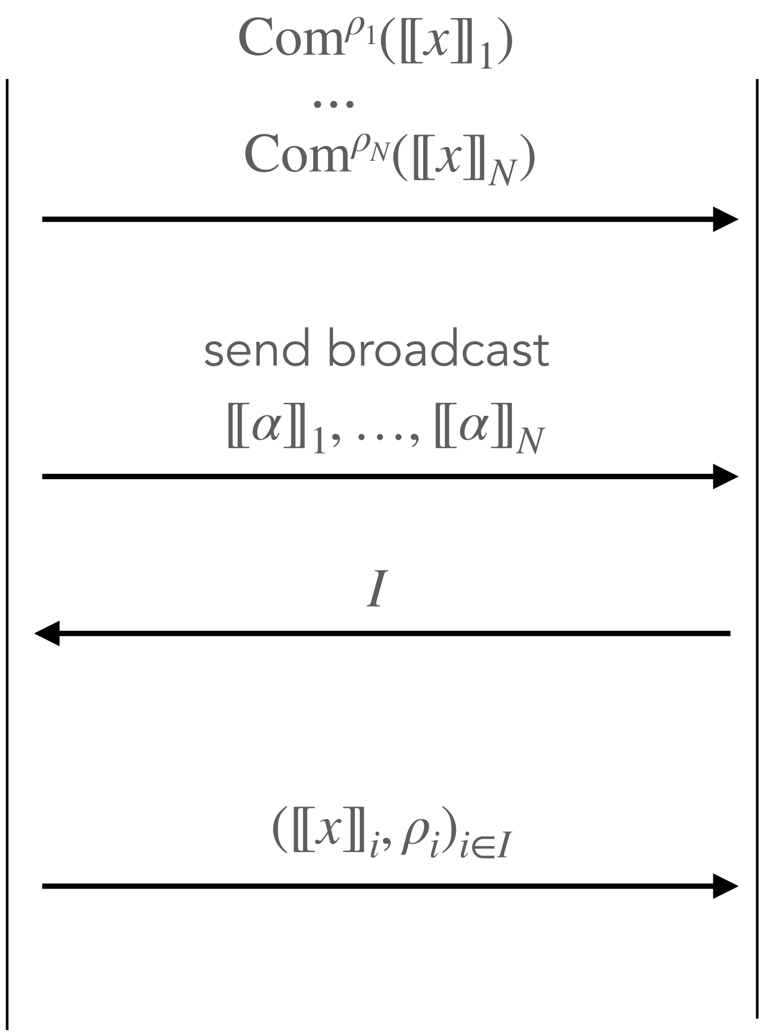
① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I



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

$\ldots$

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

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

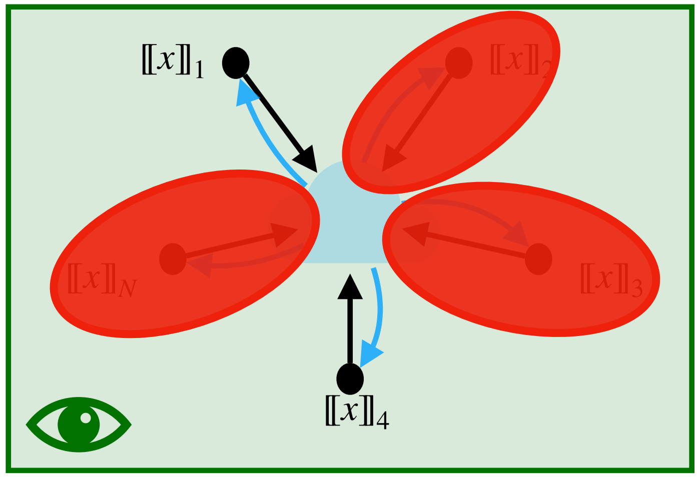
Verifier

MPCitH transform

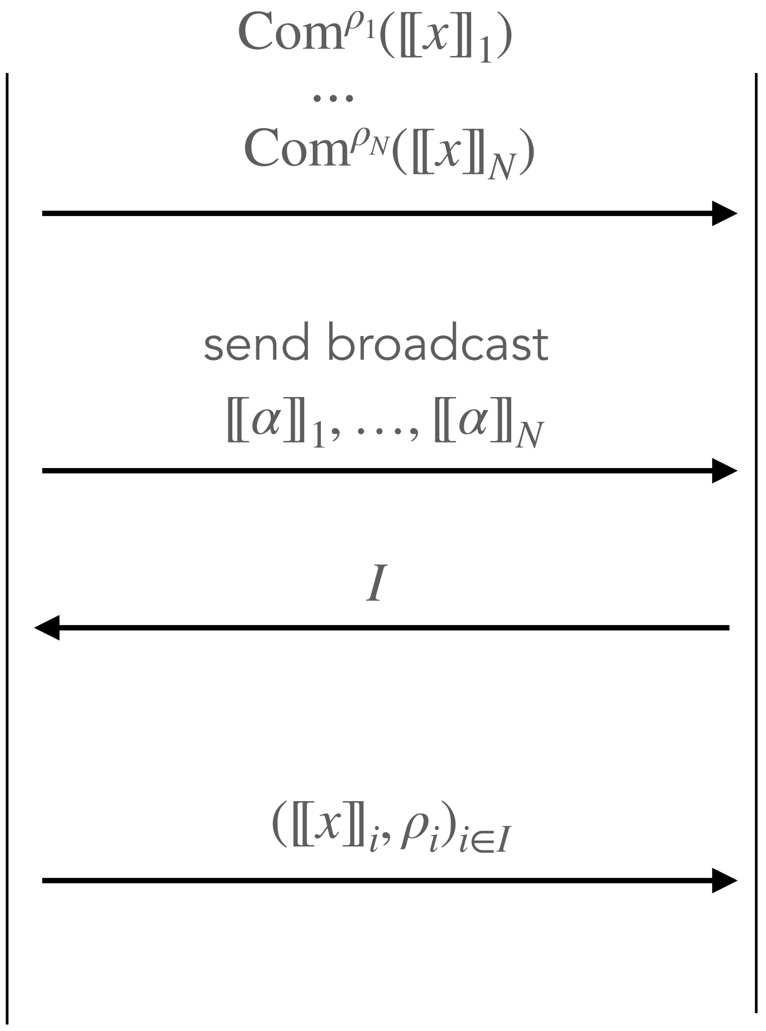
① Generate and commit shares

$\llbracket x \rrbracket = (\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$

② Run MPC in their head



④ Open parties in I



$\mathrm{Com}^{\rho_1}(\llbracket x \rrbracket_1)$

$\ldots$

$\mathrm{Com}^{\rho_N}(\llbracket x \rrbracket_N)$

send broadcast

$\llbracket \alpha \rrbracket_1, \ldots, \llbracket \alpha \rrbracket_N$

I

$(\llbracket x \rrbracket_i, \rho_i)_{i \in I}$

③ Choose a random set of parties

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

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}(\llbracket x \rrbracket_i)$
- MPC computation $\llbracket \alpha \rrbracket_i = \varphi(\llbracket x \rrbracket_i)$

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

Prover

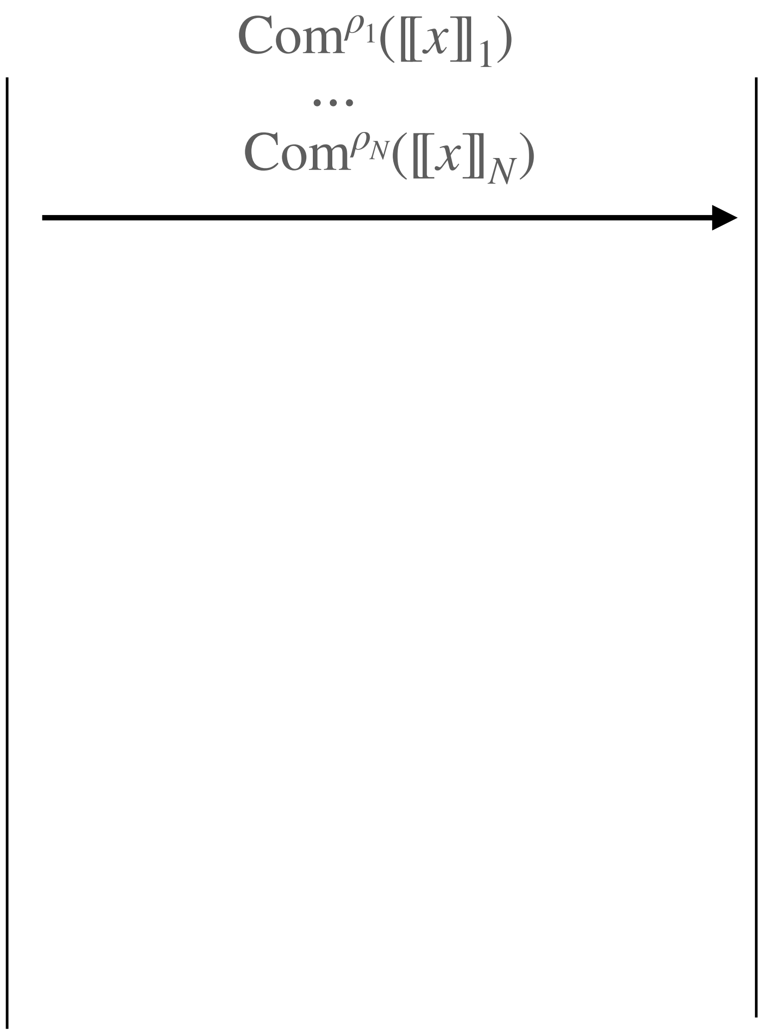
Verifier

MPCitH transform

① Generate and commit shares

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

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



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

Malicious Prover

Verifier

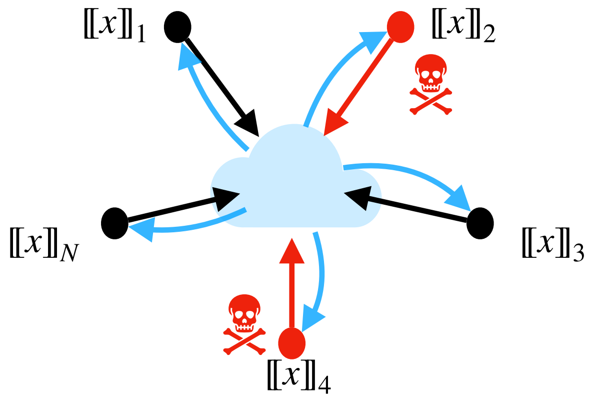
MPCitH transform

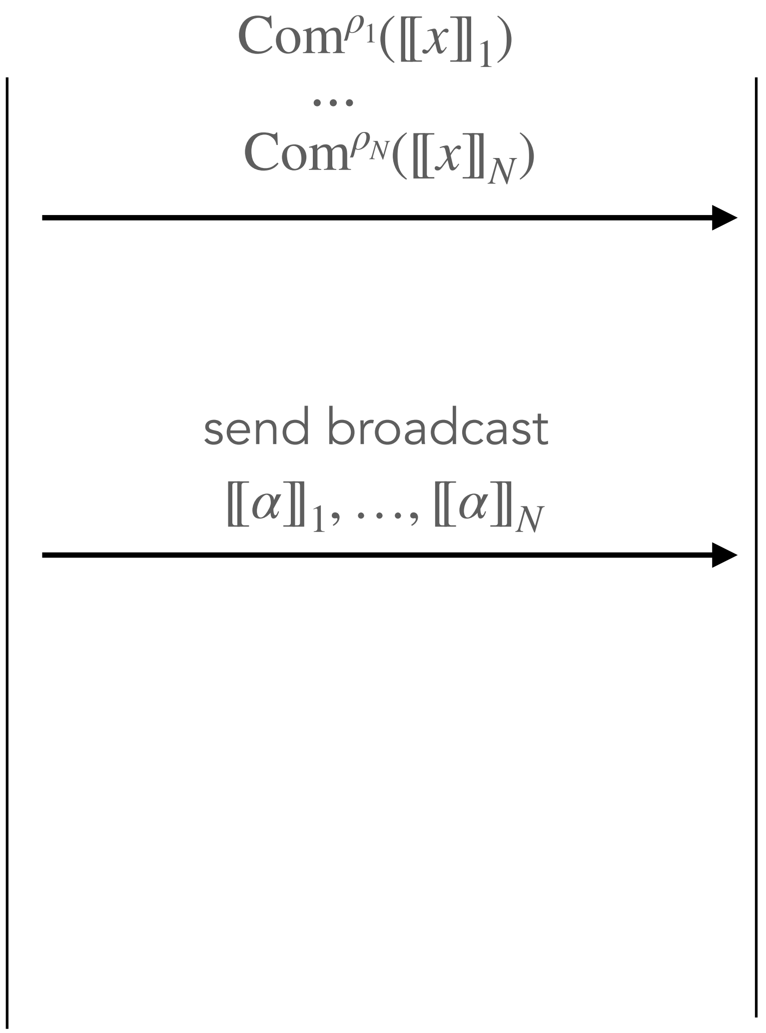
① Generate and commit shares

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

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

② Run MPC in their head





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

$\ldots$

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

send broadcast

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

Malicious Prover

Verifier

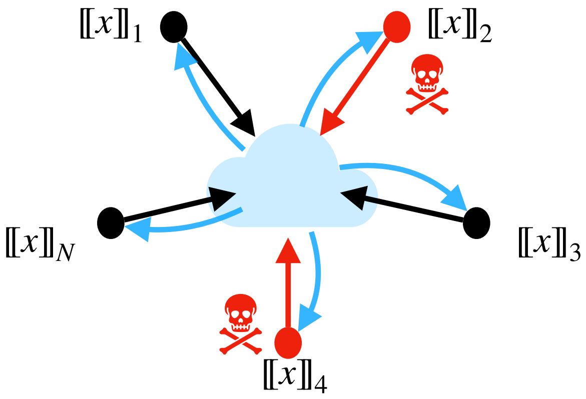
MPCitH transform

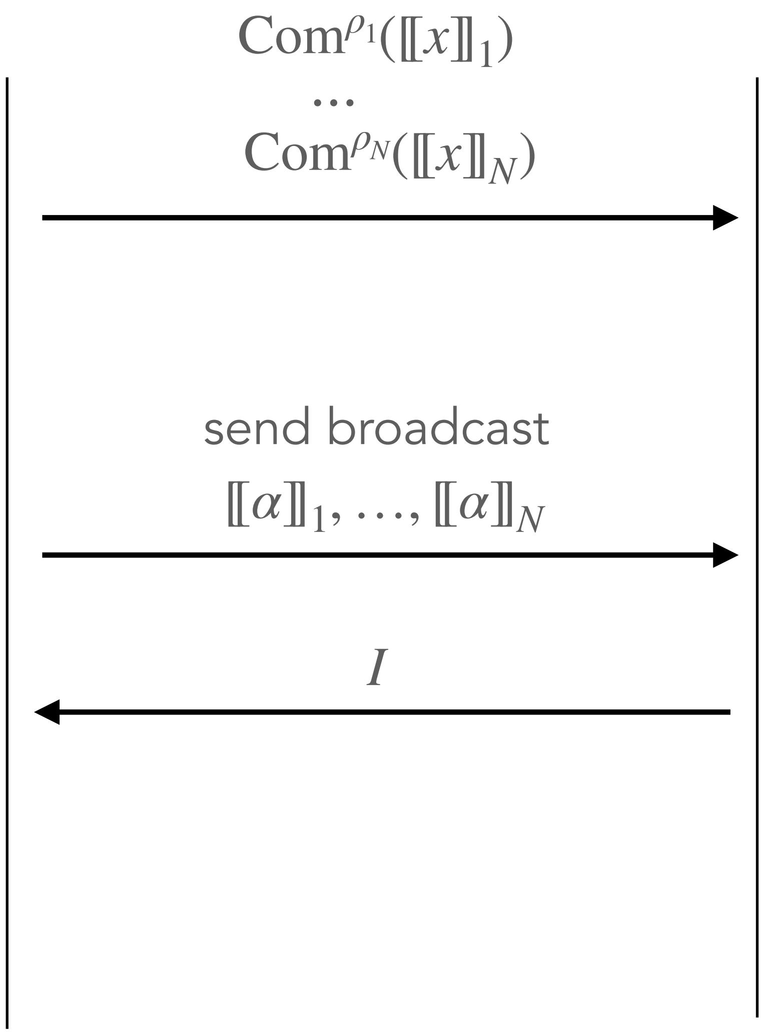
① Generate and commit shares

$$\llbracket x \rrbracket = (\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$$

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

② Run MPC in their head





$\mathrm{Com}^{\rho_1}(\llbracket x \rrbracket_1)$

$\ldots$

$\mathrm{Com}^{\rho_N}(\llbracket x \rrbracket_N)$

send broadcast

$\llbracket \alpha \rrbracket_1, \ldots, \llbracket \alpha \rrbracket_N$

I

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

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

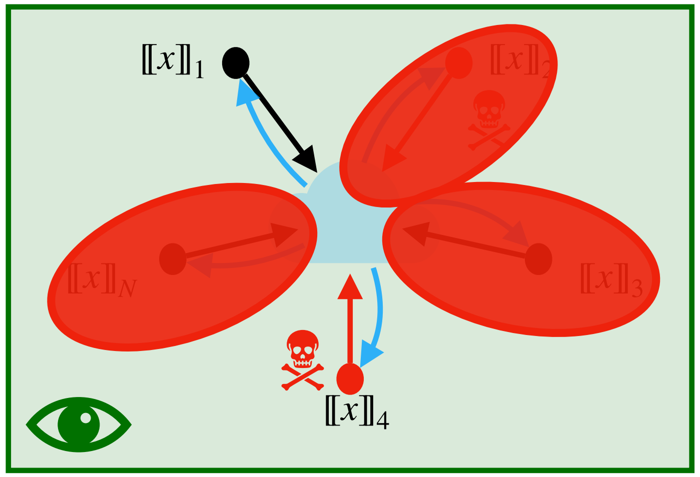
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\ldots$

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

$\longrightarrow$

② Run MPC in their head



send broadcast

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

$\longrightarrow$

③ Choose a random set of parties

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

I

$\longleftarrow$

④ Open parties in I

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

$\longrightarrow$

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

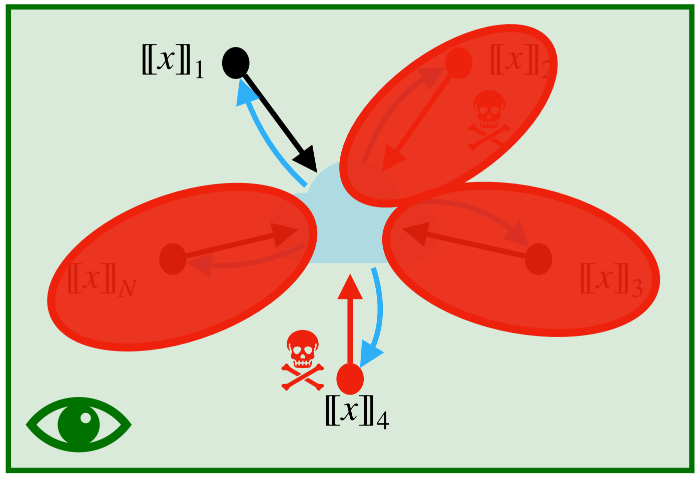
We have $F(x) \neq y$.

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

…

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

② Run MPC in their head



send broadcast

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

③ Choose a random set of parties

$I \subseteq \{1, \dots, N\}$, 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

✗ **Cheating detected!**

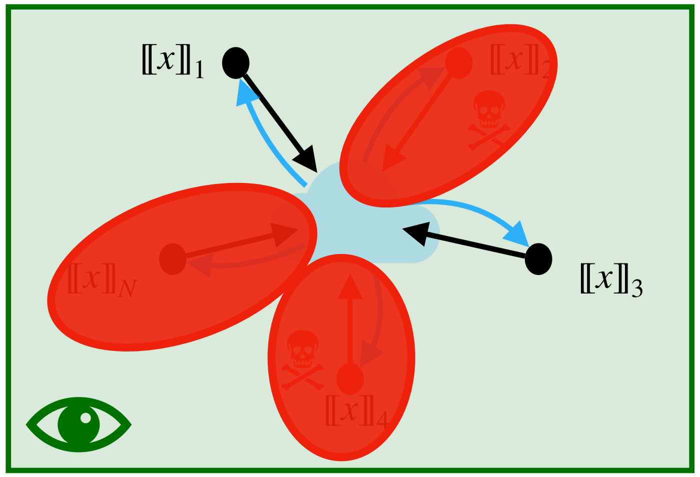
MPCitH transform

① Generate and commit shares

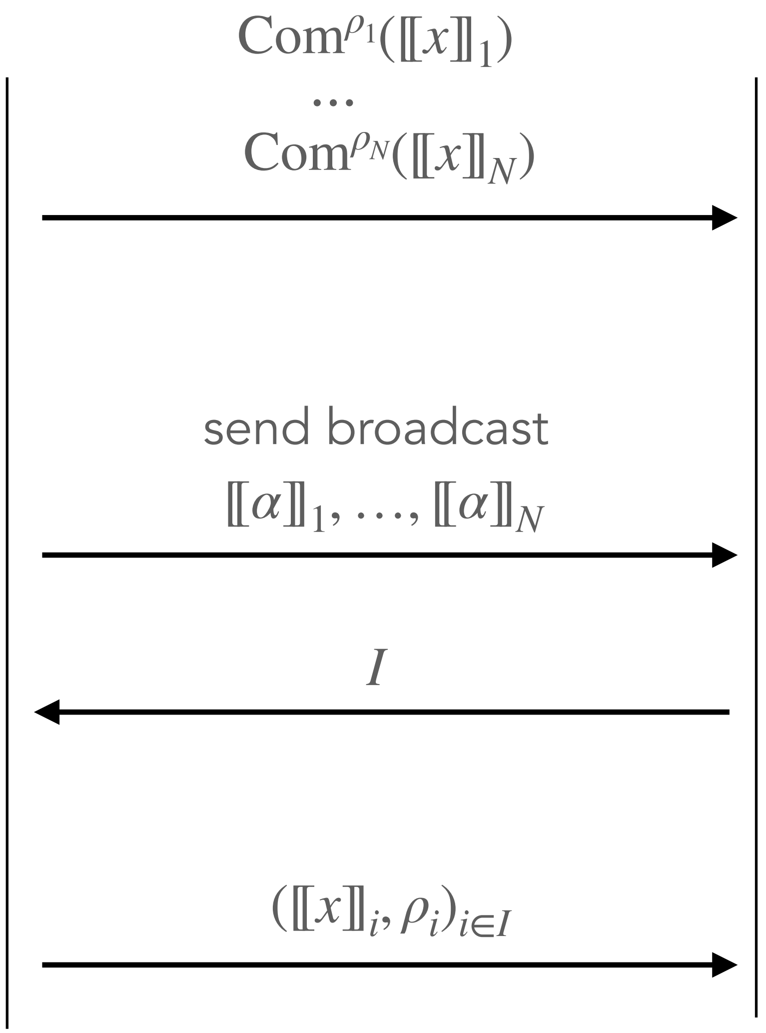
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

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

✓ **Seems OK.**

MPCitH transform

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

MPCitH transform

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

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

Number of challenges for which the corrupted parties remain hidden

Number of possible challenges

where $\#e$ is the smallest number of corrupted parties that enables a malicious prover to corrupt the MPC output.

MPCitH transform (using additive sharings)

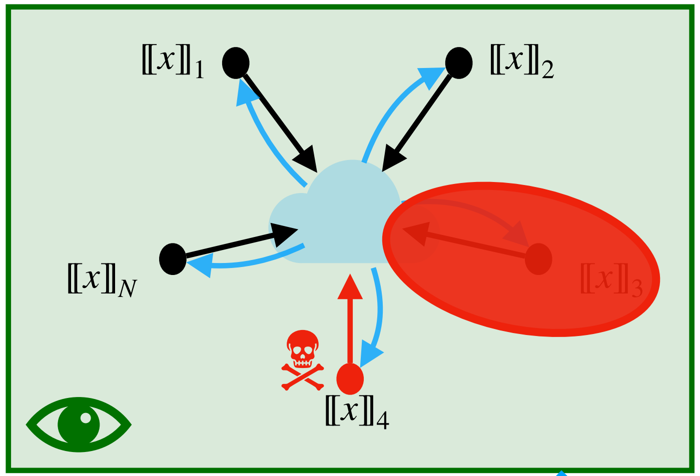
① Generate and commit shares

$\llbracket x \rrbracket = (\llbracket x \rrbracket_1, \dots, \llbracket x \rrbracket_N)$

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

$x := \llbracket x \rrbracket_1 + \dots + \llbracket x \rrbracket_N$

② Run MPC in their head



④ Open parties in I

$\mathrm{Com}^{\rho_1}(\llbracket x \rrbracket_1)$

$\dots$

$\mathrm{Com}^{\rho_N}(\llbracket x \rrbracket_N)$ →

send broadcast

$\llbracket \alpha \rrbracket_1, \dots, \llbracket \alpha \rrbracket_N$ →

← I

$(\llbracket x \rrbracket_i, \rho_i)_{i \in I}$ →

The verifier asks to reveal the views of all the parties except one ($i.e.$ $\ell := N - 1$)

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

⑤ Check $\forall i \in I$

- Commitments $\mathrm{Com}^{\rho_i}(\llbracket x \rrbracket_i)$
- MPC computation $\llbracket \alpha \rrbracket_i = \varphi(\llbracket x \rrbracket_i)$

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

The malicious prover needs to cheat for a least one party ($i.e.$ $\#e := 1$)

Malicious Prover

Verifier

MPCitH transform (using additive sharings)

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private
- **Soundness:**

$$\begin{aligned}&\mathbb{P}(\text{malicious prover convinces the verifier})\\&\qquad = \mathbb{P}(\text{all corrupted parties remain hidden})\\&\qquad = \frac{1}{N}\end{aligned}$$

MPCitH transform (using additive sharings)

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private

- **Soundness:**

$$\begin{aligned}&\mathbb{P}(\text{malicious prover convinces the verifier})\\&\qquad = \mathbb{P}(\text{all corrupted parties remain hidden})\\&\qquad = \frac{1}{N}\end{aligned}$$

- **Additional optimisations:**
  - Using *seed (GGM) trees* to save communication

MPCitH transform (using additive sharings)

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private
- **Soundness:**

$$\begin{aligned}\mathbb{P}(\text{malicious prover convinces the verifier})&\\ &= \mathbb{P}(\text{all corrupted parties remain hidden})\\ &= \frac{1}{N}\end{aligned}$$

- **Additional optimisations:**
  - Using *seed (GGM) trees* to save communication
  - Using the hypercube technique [AGH+23] to save computation

[AGH+23] Aguilar-Melchor, Gama, Howe, Hülsing, Joseph, Yue: "The Return of the SDitH" (Eurocrypt 2023)

MPCitH-based NIST Candidates

	Assumption	Size (in KB)
AIMer	AIM (MPC-friendly one-way function)	4.2
Biscuit	Structured MQ problem (PowAff2)	4.7
MIRA	MinRank problem	5.6
MiRitH	MinRank problem	5.7
RYDE	Syndrome decoding problem in rank metric	6.0
MQOM	Unstructured MQ problem	6.3
SDitH	Syndrome decoding problem in Hamming	8.2

- **Short public keys:** less than 200 bytes
- **Running times:** between $1 \sim 20$ ms to sign / verify

MPCitH transform

🤔

What about using
Shamir's secret sharings?

MPCitH transform

🤔 **What about using Shamir's secret sharings?**

[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)

Threshold-Computation-in-the-Head (TCitH) Framework

TCitH transform

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

- **Soundness:** ***if the committed sharing is valid***

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

d is the degree of the computation performed by the MPC protocol

- When considering linear MPC protocol and when $\ell = 1$:

  the soundness error is $\frac{1}{N}$.

How to commit Shamir's secret sharing?

Let us consider $\ell = 1$.

$x = (42{,}134{,}235)$ over $\mathbb{F}_{251}$

	$[\![\cdot]\!]_1$	$[\![\cdot]\!]_2$	$\dots$							$\dots$	$[\![\cdot]\!]_N$	
$[\![x_1]\!]$	124	206	37	119	201	32	114	196	27	109	191	$P_1(X) = 82 \cdot X + x_1$
$[\![x_2]\!]$	80	26	223	169	115	61	7	204	150	96	42	$P_2(X) = 197 \cdot X + x_2$
$[\![x_3]\!]$	133	31	180	78	227	125	23	172	70	219	117	$P_3(X) = 149 \cdot X + x_3$

How to commit Shamir's secret sharing?

Let us consider $\ell = 1$.

[redacted]

$[\![x_1]\!]$	[redacted]	[redacted]	[redacted]	[redacted]	201	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
$[\![x_2]\!]$	[redacted]	[redacted]	[redacted]	[redacted]	115	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
$[\![x_3]\!]$	[redacted]	[redacted]	[redacted]	[redacted]	227	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Verifier's point of view: How to be sure that the committed sharings are well-formed?

How to commit Shamir's secret sharing?

Let us consider $\ell = 1$.

😈 **Malicious Prover**

	$[\![\cdot]\!]_1$	$[\![\cdot]\!]_2$	$\dots$							$\dots$	$[\![\cdot]\!]_N$	
$[\![x_1]\!]$	103	138	166	187	201	208	208	201	187	166	138	$122 \cdot X^2 + 171 \cdot X + 61$
$[\![x_2]\!]$	241	152	185	89	115	12	31	172	184	67	72	$61 \cdot X^2 + 230 \cdot X + 201$
$[\![x_3]\!]$	182	113	223	10	227	121	194	195	124	232	17	$215 \cdot X^2 + 39 \cdot X + 179$

Verifier's point of view: How to be sure that the committed sharings are well-formed? 🧐

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

TCitH-GGM: Using a Seed Tree

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

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

- Party $\mathscr{P}_1$: $r_2, r_3, \ldots, r_N$
- 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 $P_j(X) := 1 - \dfrac{1}{e_j} X$.

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]:

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

- 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).

TCitH-GGM: Using a Seed Tree

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

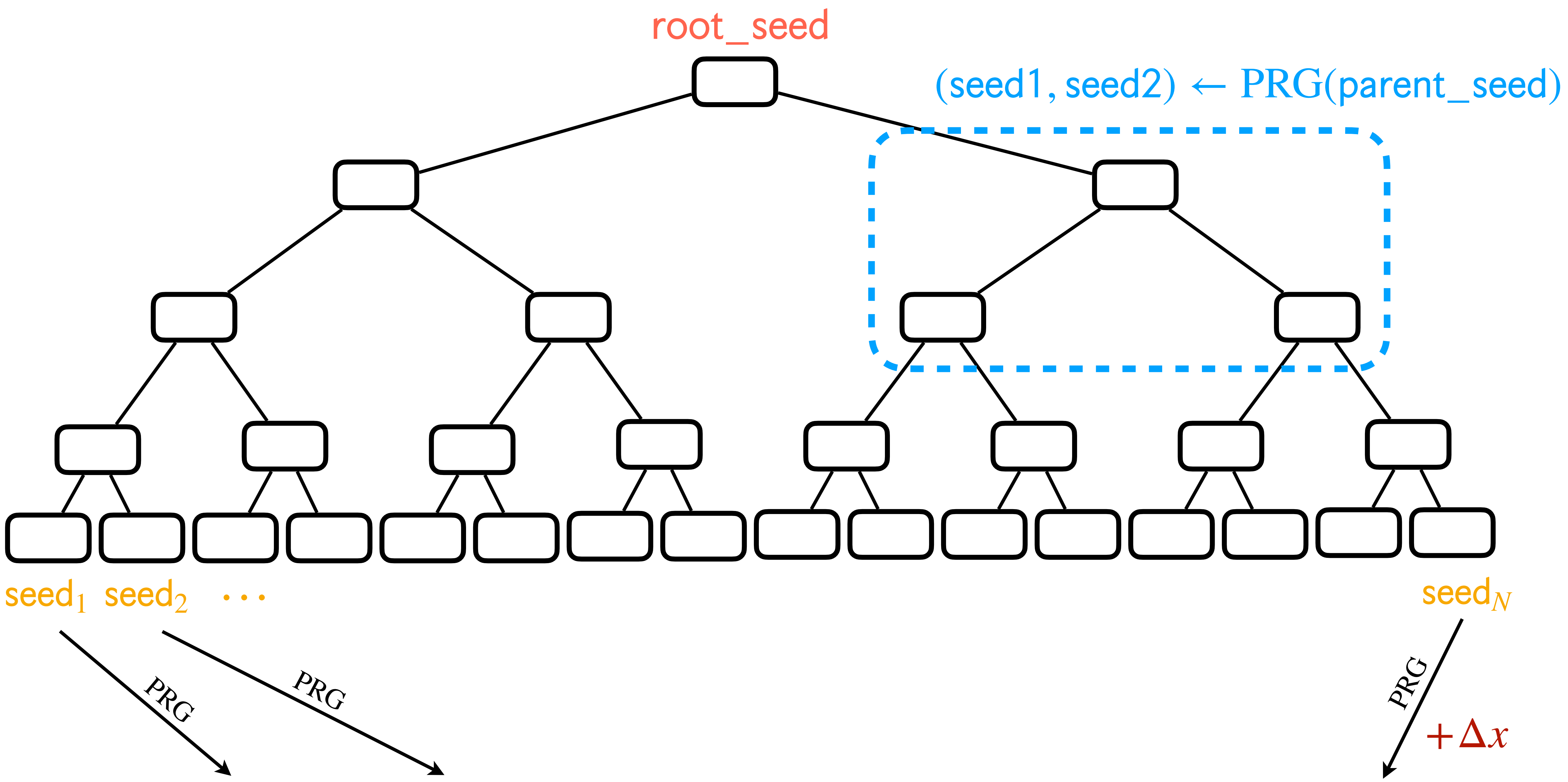
TCitH-GGM: Using a Seed Tree

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

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

TCitH-GGM: Using a Seed Tree

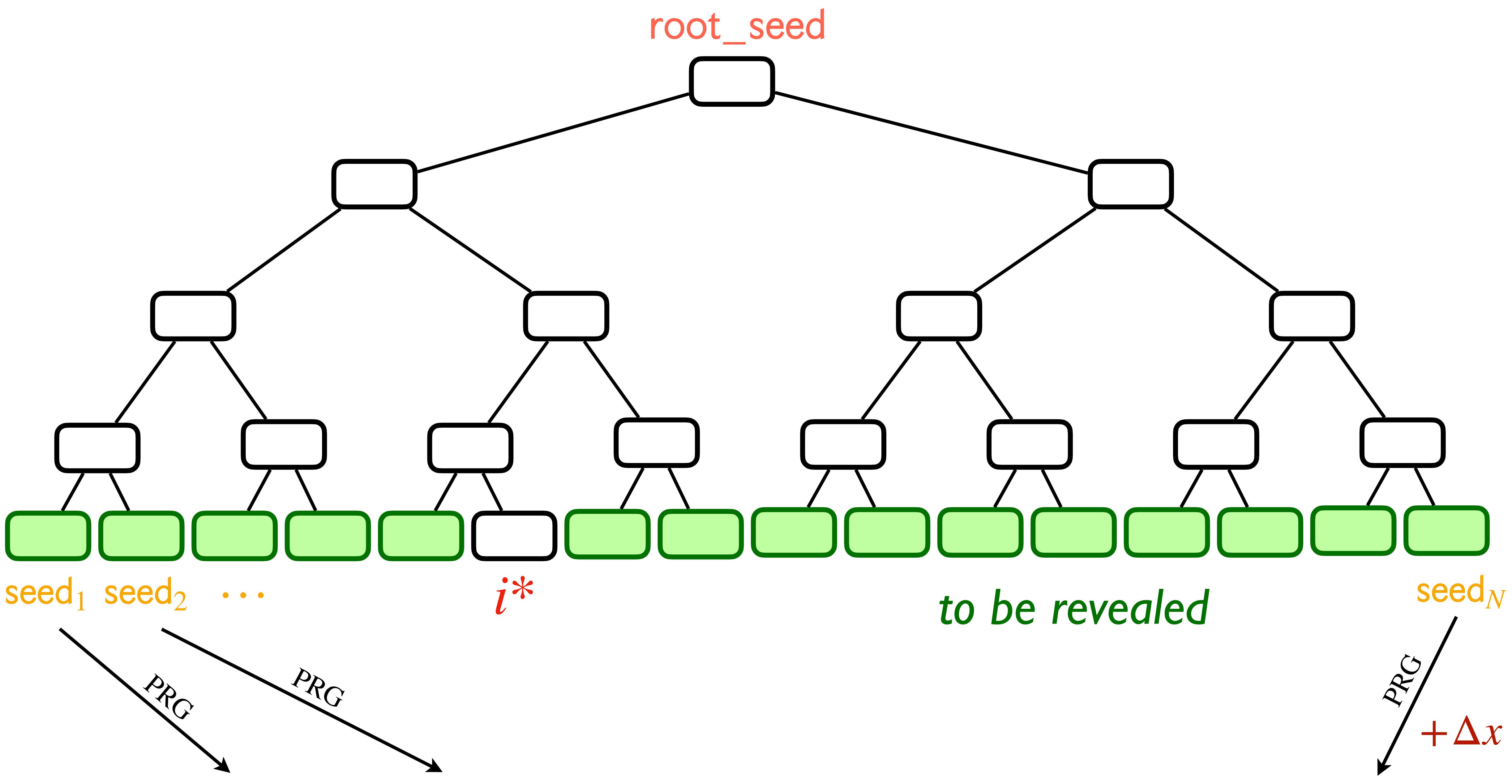
[KKW18] Katz, Kolesnikov, Wang: “Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures” (CCS 2018)



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

TCitH-GGM: Using a Seed Tree

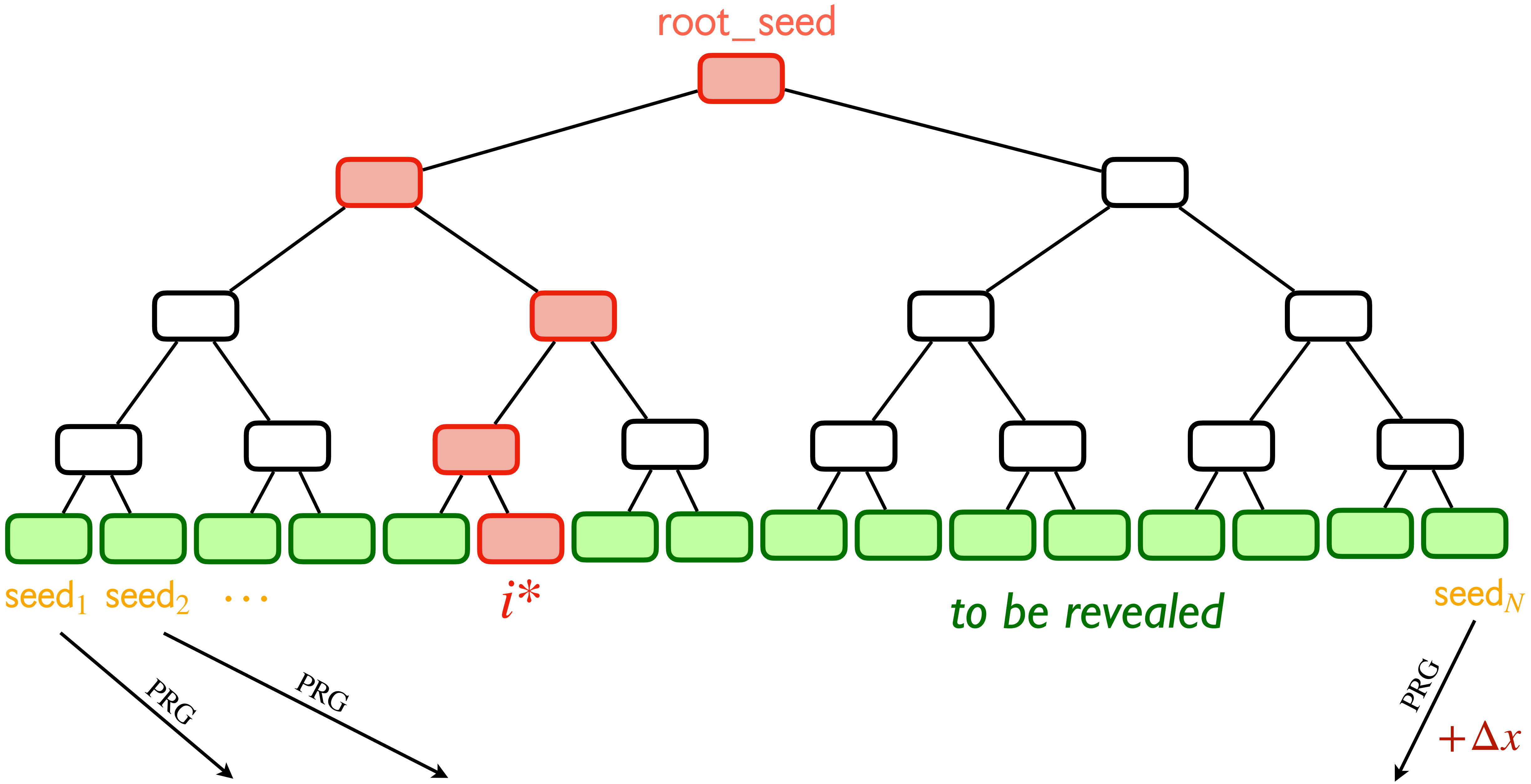
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



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

TCitH-GGM: Using a Seed Tree

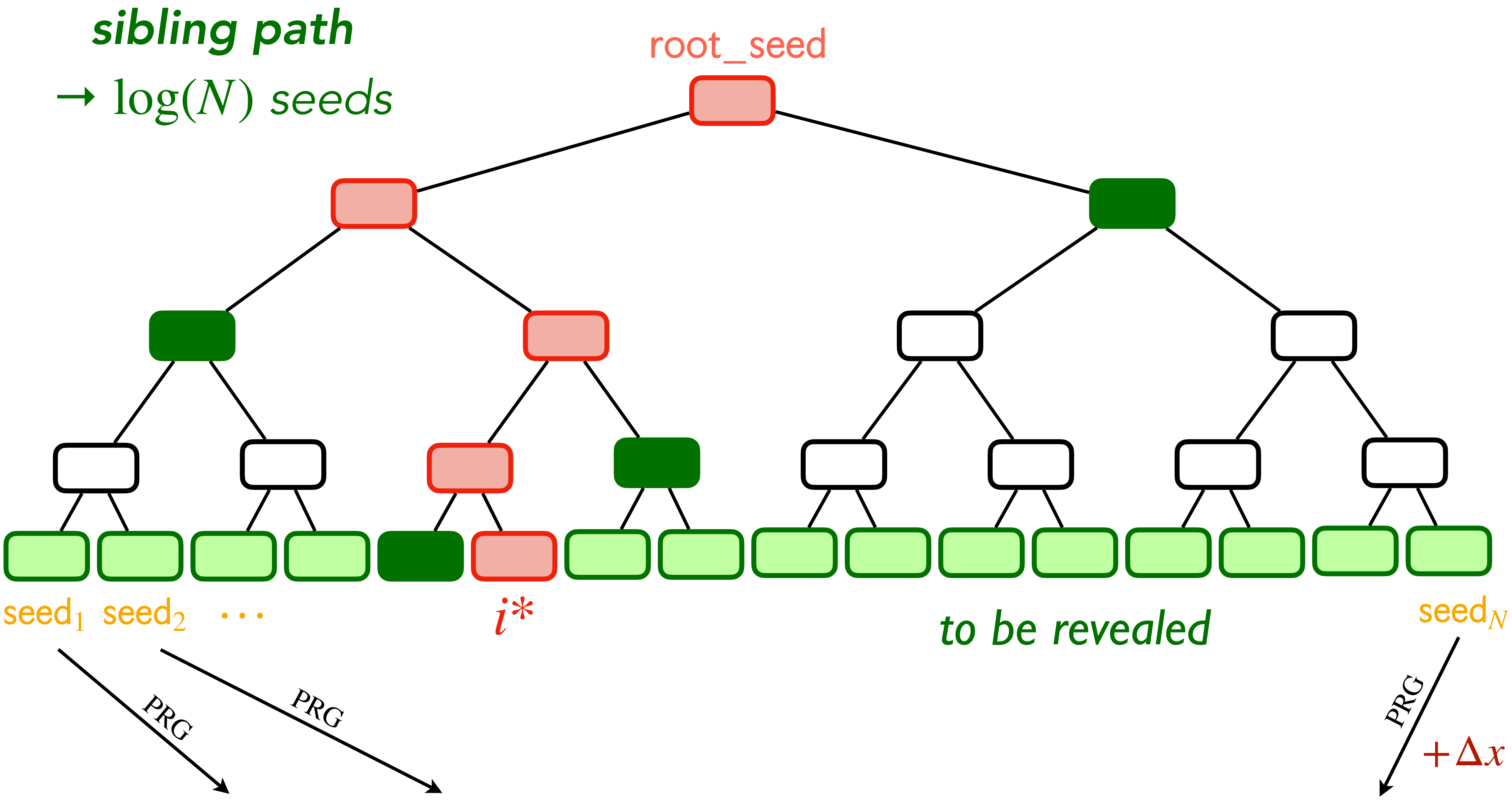
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



$$x = r_1 + r_2 + r_3 + \dots + r_{N-1} + r_N$$

TCitH-GGM: 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*

root_seed

seed$_1$ seed$_2$ $\cdots$

i^*

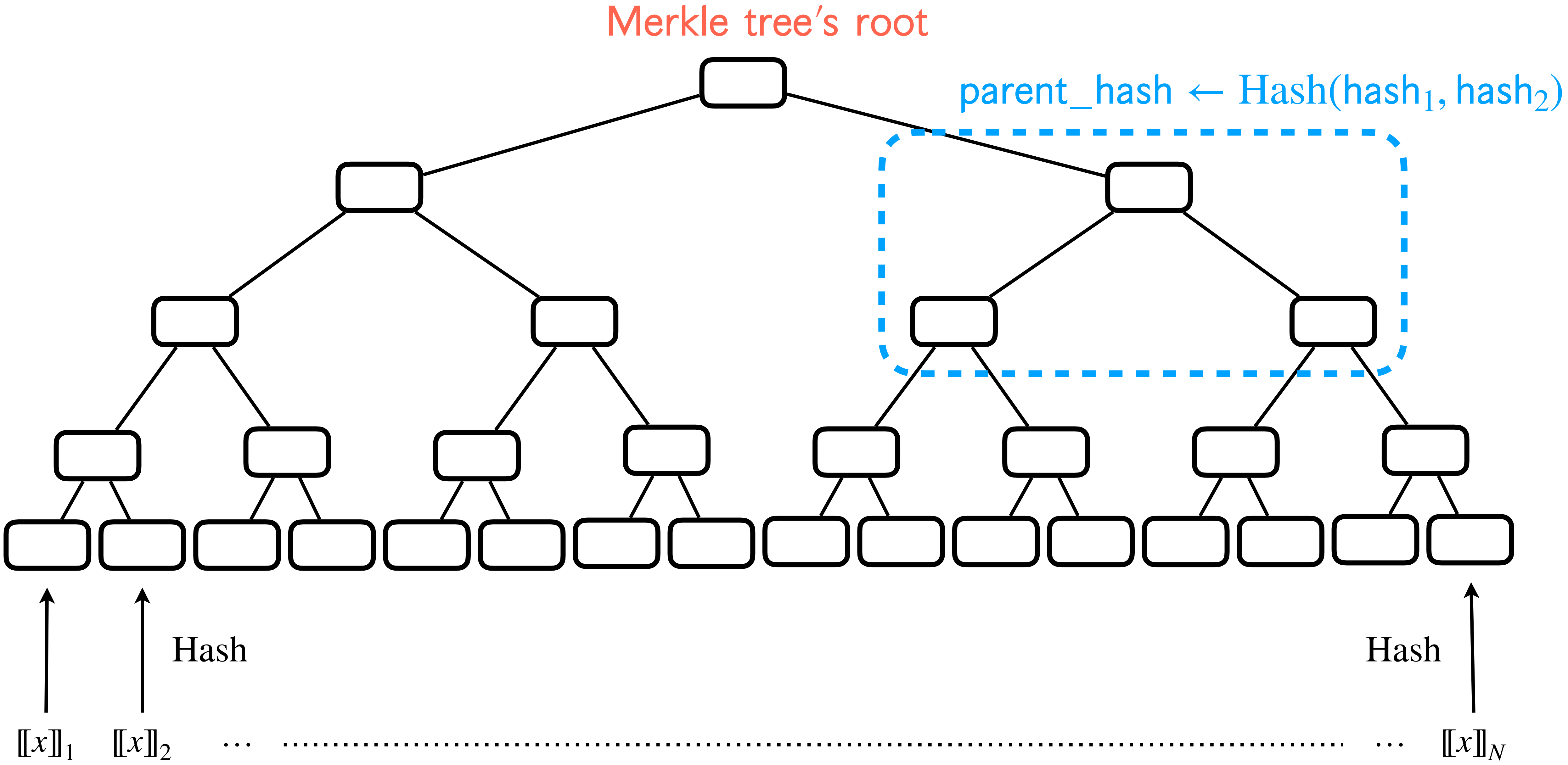
to be revealed

seed$_N$

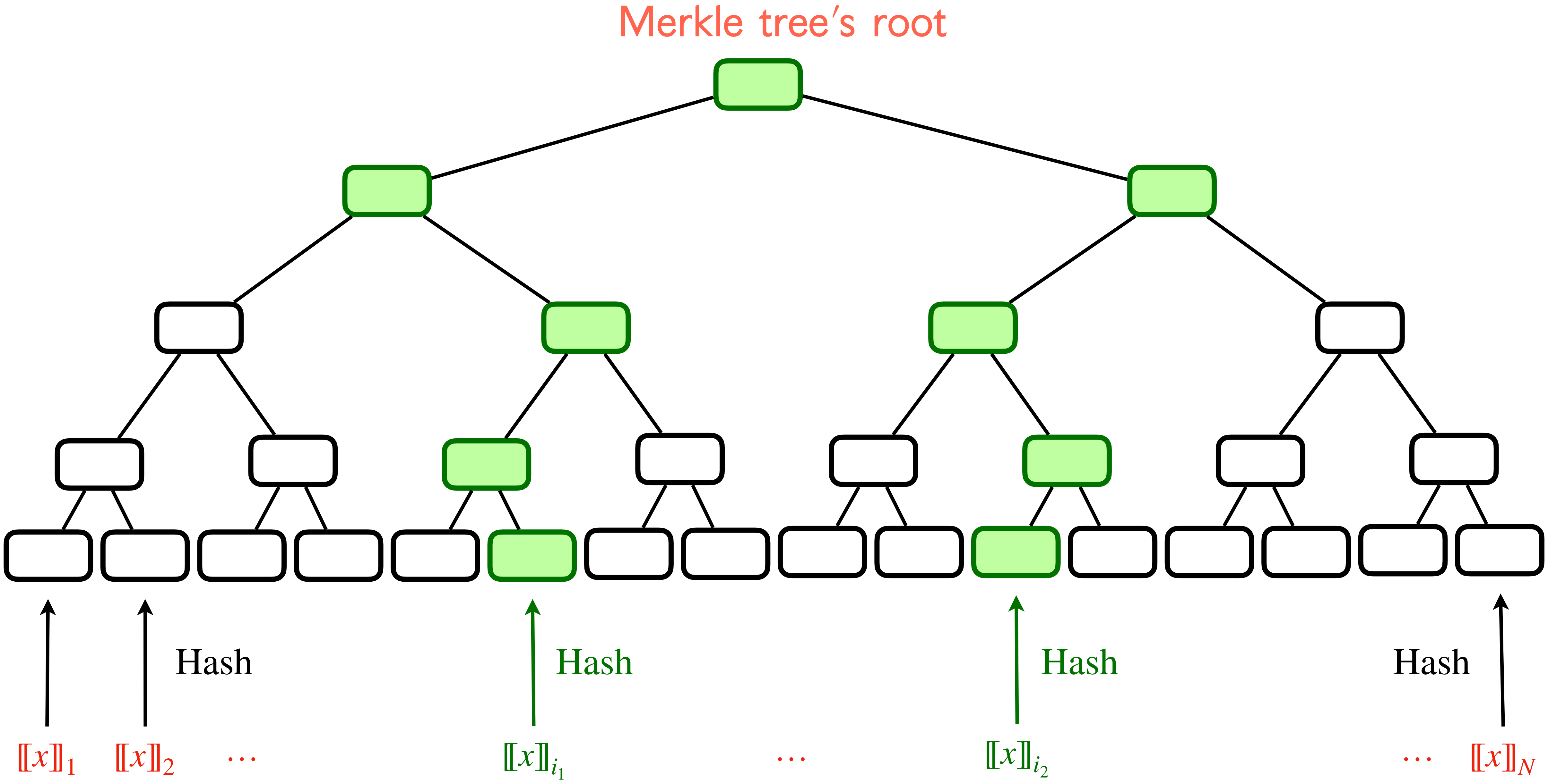
$$x = r_1 + r_2 + r_3 + \ldots + r_{N-1} + r_N$$

TCitH-MT: Using a Merkle tree

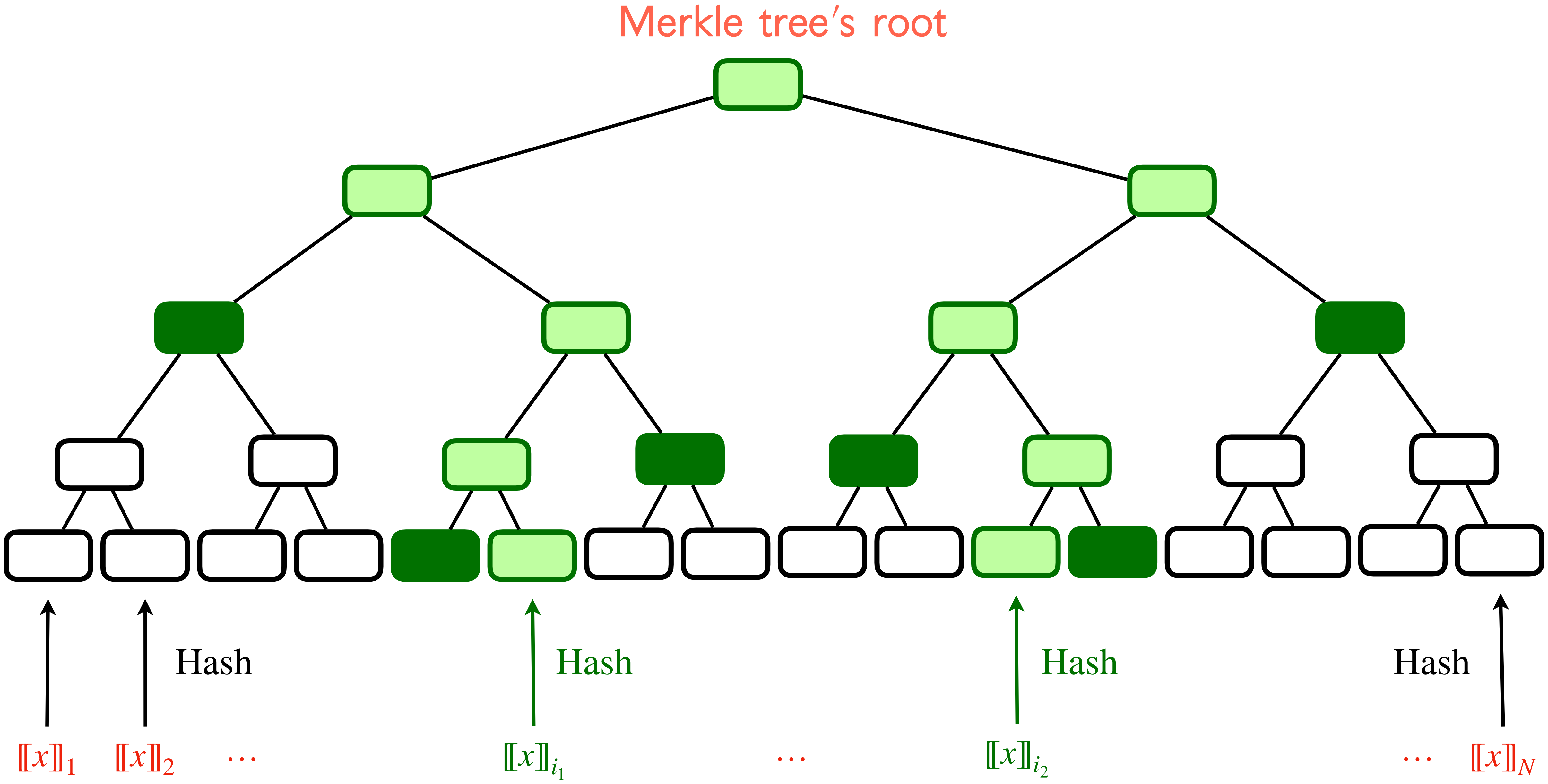
TCitH-MT: Using a Merkle tree



TCitH-MT: Using a Merkle tree



TCitH-MT: Using a Merkle tree



Merkle tree's root

Hash

Hash

Hash

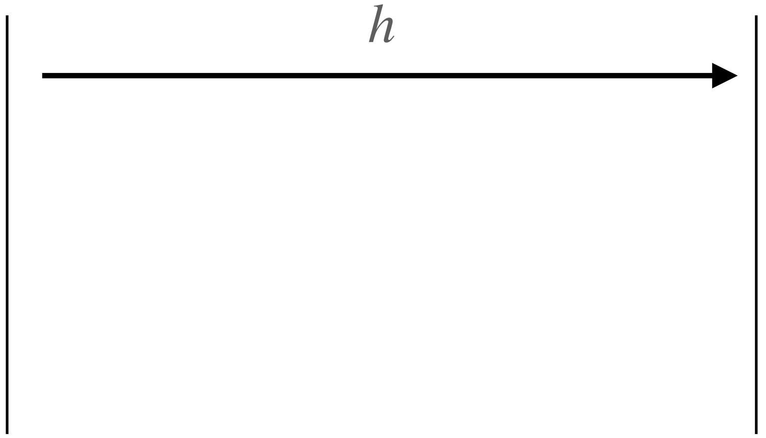
Hash

$[\![x]\!]_1$ $[\![x]\!]_2$ $\cdots$ $[\![x]\!]_{i_1}$ $\cdots$ $[\![x]\!]_{i_2}$ $\cdots$ $[\![x]\!]_N$

TCitH-MT: Using a Merkle tree

Compute

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



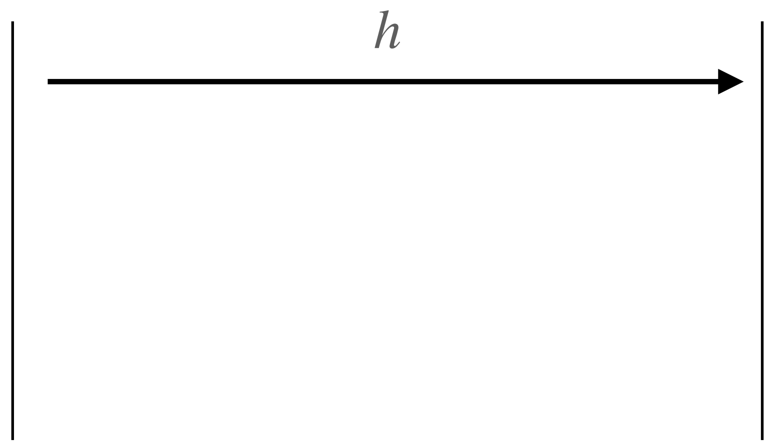
Prover

Verifier

TCitH-MT: Using a Merkle tree

Compute

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



Prover

Verifier

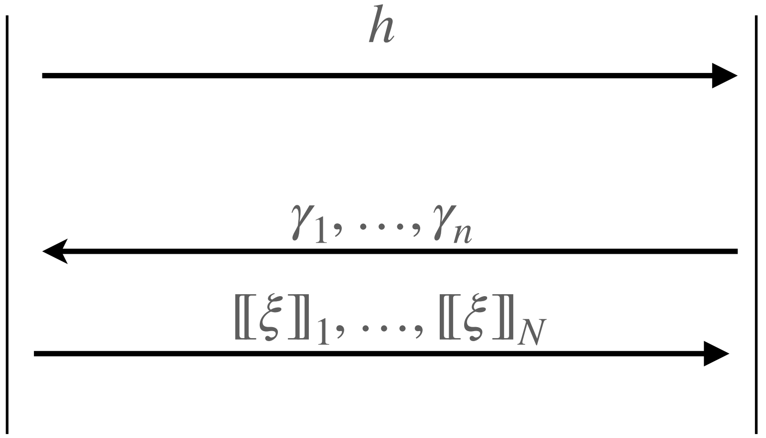
How to be sure that the committed shares correspond to a valid Shamir's secret sharing?

TCitH-MT: Using a Merkle tree

Interactive commitment scheme

Compute

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



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

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

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

<u>Prover</u>

<u>Verifier</u>

TCitH-MT: Using a Merkle tree

Interactive commitment scheme

Compute

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

$$\xrightarrow{\quad h \quad}$$

Repeat η times (in parallel)

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

$$\xleftarrow{\quad \gamma_1, \dots, \gamma_n \quad}$$

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

$$\xrightarrow{\quad [\![\xi]\!]_1, \dots, [\![\xi]\!]_N \quad}$$

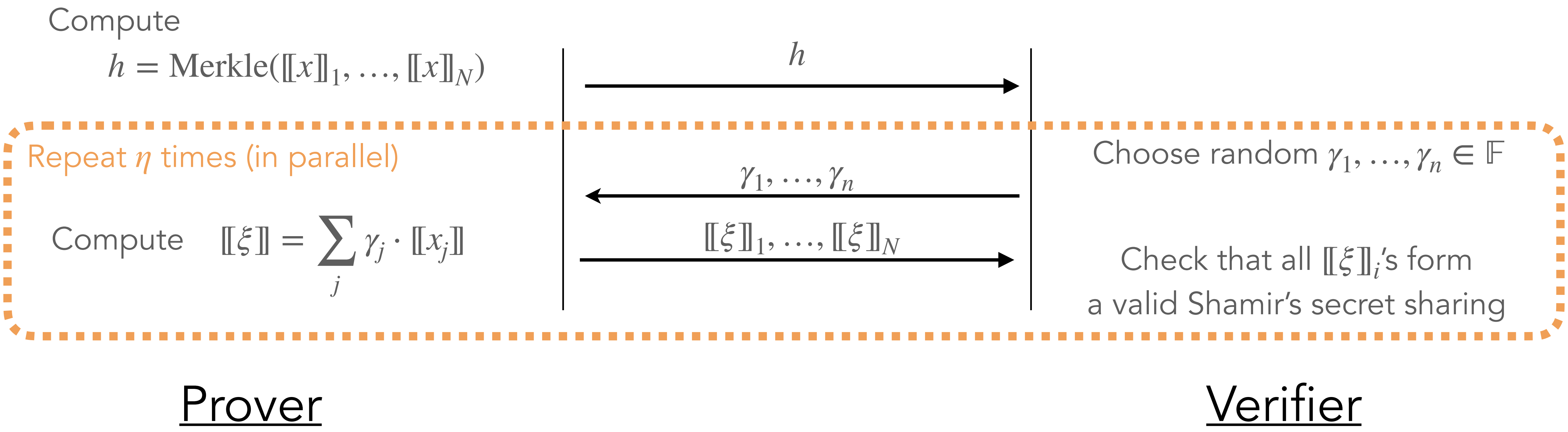
Check that all $[\![\xi]\!]_i$'s form a valid Shamir's secret sharing

Prover

Verifier

TCitH-MT: Using a Merkle tree

Interactive commitment scheme



Compute

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

h

Repeat η times (in parallel)

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

$\gamma_1, \dots, \gamma_n$

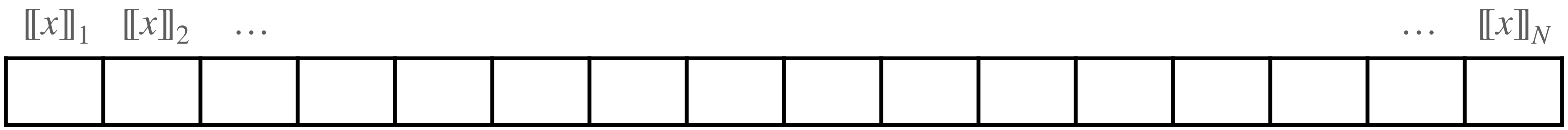
Compute $[\![\xi]\!] = \sum_j \gamma_j \cdot [\![x_j]\!]$

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

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

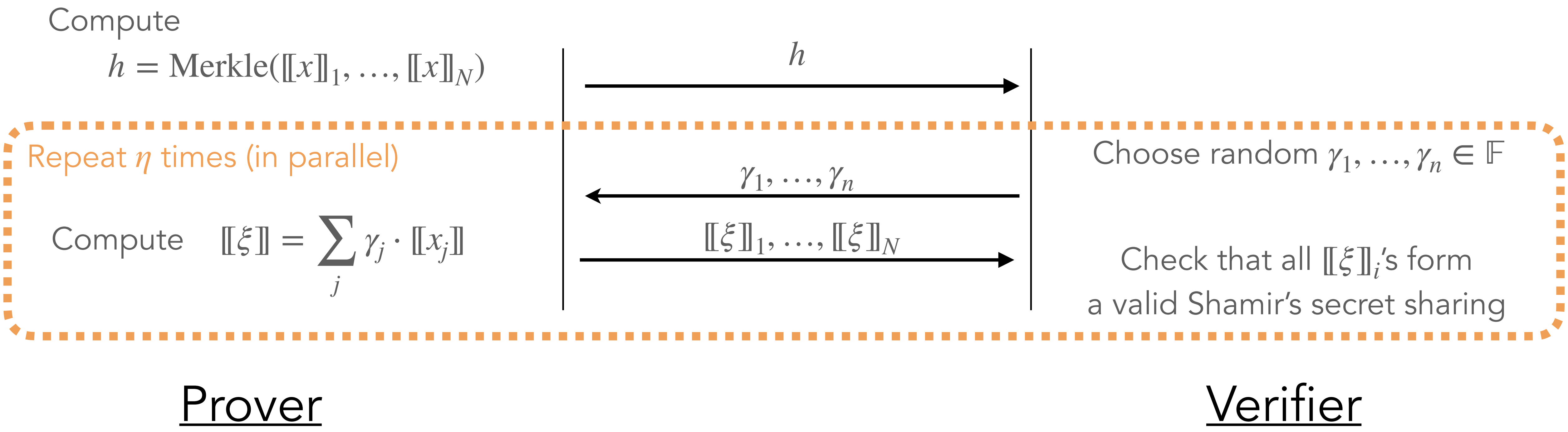
Prover

Verifier



TCitH-MT: Using a Merkle tree

Interactive commitment scheme



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

$\xrightarrow{h}$

Repeat η times (in parallel)

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

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

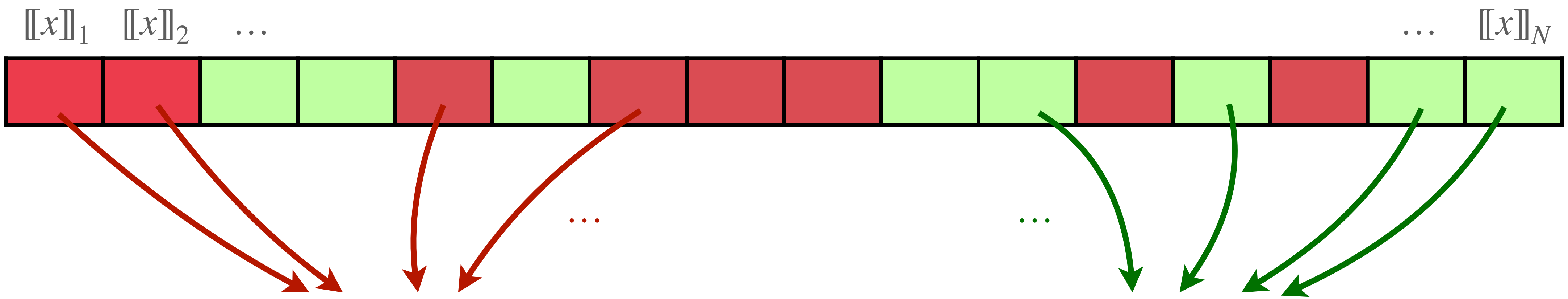
Compute $[\![\xi]\!] = \sum_j \gamma_j \cdot [\![x_j]\!]$

$\xrightarrow{[\![\xi]\!]_1, \ldots, [\![\xi]\!]_N}$

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

Prover

Verifier



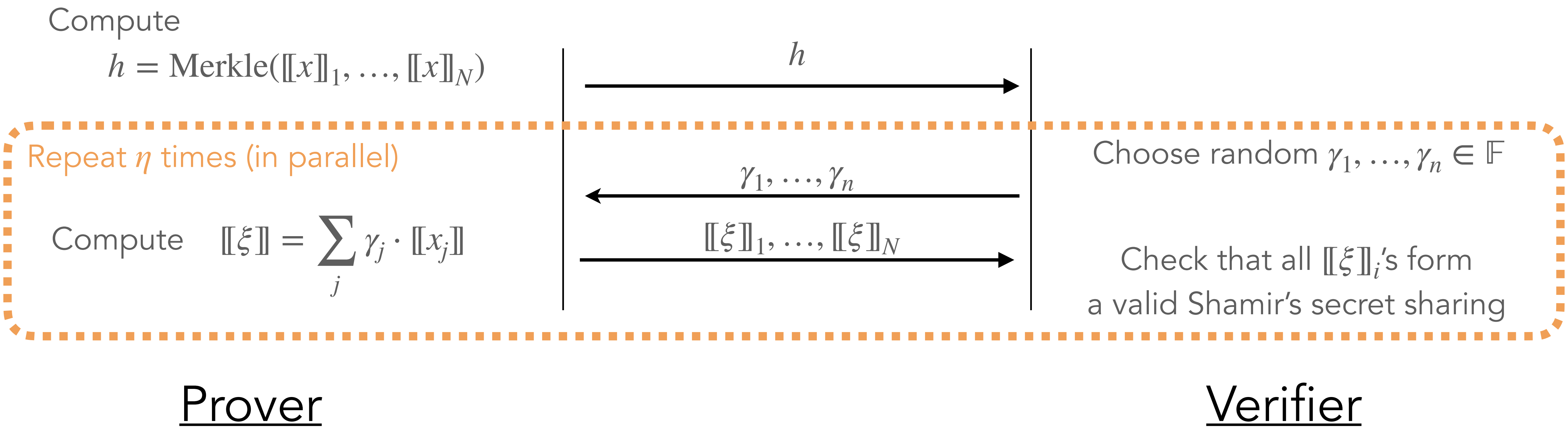
$[\![\xi]\!]_i \neq \sum_j \gamma_j \cdot [\![x_j]\!]_i$

Impossible to open!

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

TCitH-MT: Using a Merkle tree

Interactive commitment scheme



Compute

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

$\xrightarrow{h}$

Repeat η times (in parallel)

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

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

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

$\xrightarrow{[\![\xi]\!]_1, \ldots, [\![\xi]\!]_N}$

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

Prover

Verifier

We can prove that

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

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 $[\![a \cdot b]\!]$ from $[\![a]\!]$ and $[\![b]\!]$:

$$\forall i, \quad [\![a \cdot b]\!]_i \leftarrow [\![a]\!]_i \cdot [\![b]\!]_i$$

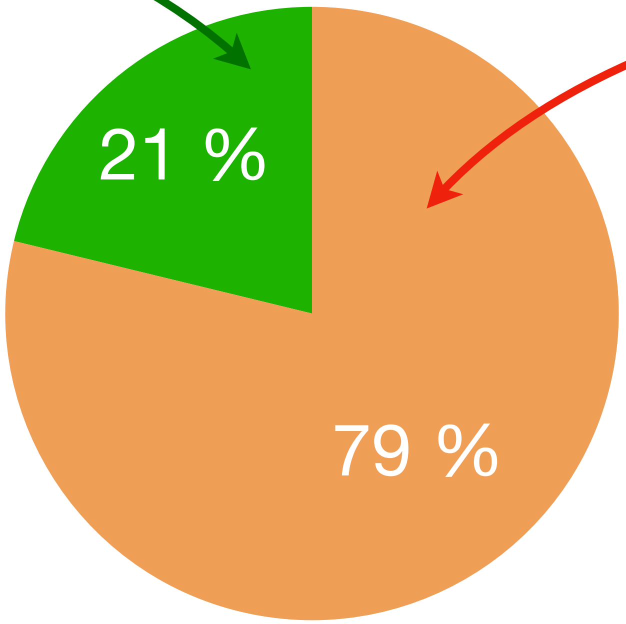
(no need for communication between parties)

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)

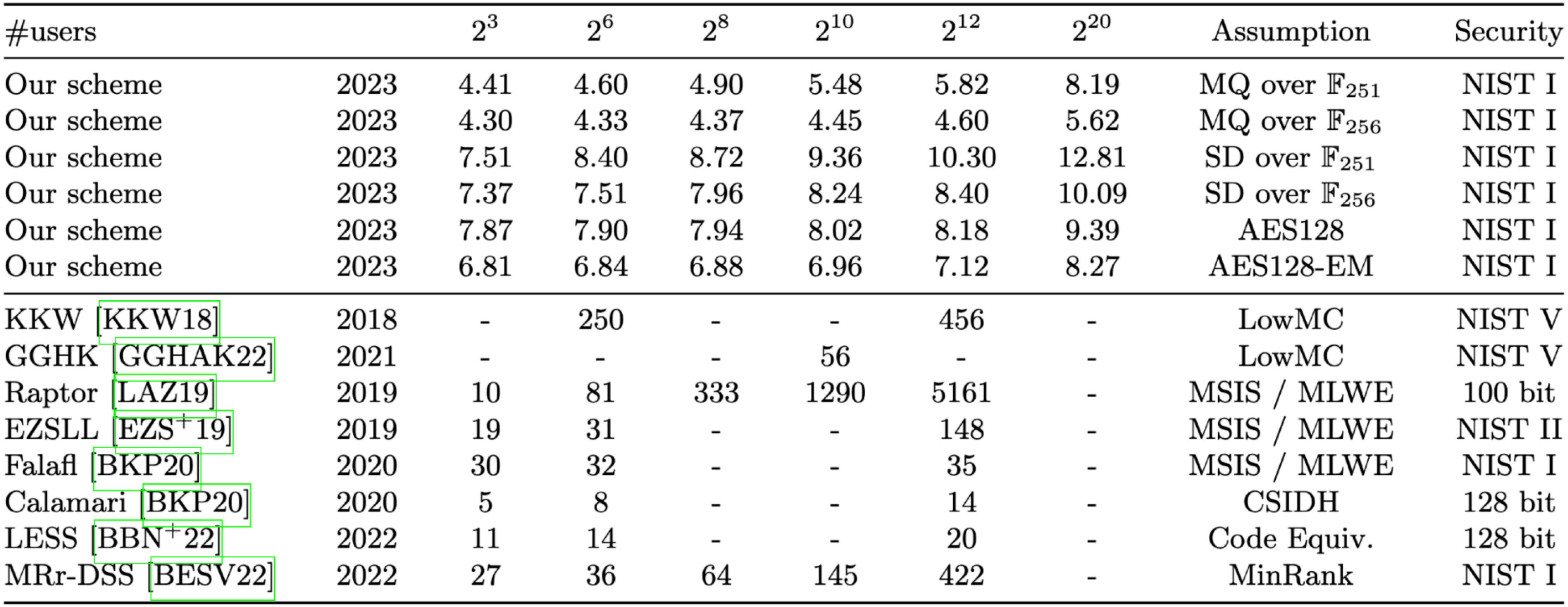
Due to the sharing commitment (with GGM trees)
(3040 bytes)

Lower bound: ≥ 2048 *bytes*

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

Efficient Ring Signatures…

… from any one-way function 🤩



#users		2^3	2^6	2^8	2^{10}	2^{12}	2^{20}	Assumption	Security
Our scheme	2023	4.41	4.60	4.90	5.48	5.82	8.19	MQ over $\mathbb{F}_{251}$	NIST I
Our scheme	2023	4.30	4.33	4.37	4.45	4.60	5.62	MQ over $\mathbb{F}_{256}$	NIST I
Our scheme	2023	7.51	8.40	8.72	9.36	10.30	12.81	SD over $\mathbb{F}_{251}$	NIST I
Our scheme	2023	7.37	7.51	7.96	8.24	8.40	10.09	SD over $\mathbb{F}_{256}$	NIST I
Our scheme	2023	7.87	7.90	7.94	8.02	8.18	9.39	AES128	NIST I
Our scheme	2023	6.81	6.84	6.88	6.96	7.12	8.27	AES128-EM	NIST I
KKW [KKW18]	2018	-	250	-	-	456	-	LowMC	NIST V
GGHK [GGHAK22]	2021	-	-	-	56	-	-	LowMC	NIST V
Raptor [LAZ19]	2019	10	81	333	1290	5161	-	MSIS / MLWE	100 bit
EZSLL [EZS$^+$19]	2019	19	31	-	-	148	-	MSIS / MLWE	NIST II
Falafl [BKP20]	2020	30	32	-	-	35	-	MSIS / MLWE	NIST I
Calamari [BKP20]	2020	5	8	-	-	14	-	CSIDH	128 bit
LESS [BBN$^+$22]	2022	11	14	-	-	20	-	Code Equiv.	128 bit
MRr-DSS [BESV22]	2022	27	36	64	145	422	-	MinRank	NIST I

Other applications

- 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 !