Post-Quantum Signatures from Threshold Computation in the Head

Matthieu Rivain

Joint work with Thibauld Feneuil

NIST 5th PQC Standardization Conference

Washington DC, 11 April, 2024

CryptoExperts

WE INNOVATE TO SECURE YOUR BUSINESS

Roadmap

- MPC-in-the-Head paradigm
- Threshold Computation in the Head
  - Original framework (Asiacrypt 2023) https://ia.cr/2022/1407
  - Improved framework (preprint) https://ia.cr/2023/1573

MPC-in-the-Head paradigm

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

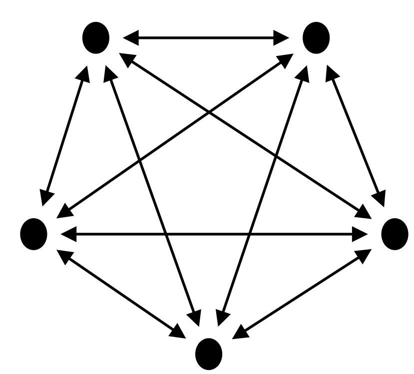
MPC-in-the-Head paradigm

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

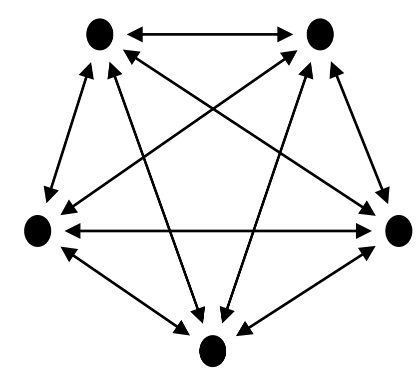
MPC-in-the-Head paradigm

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

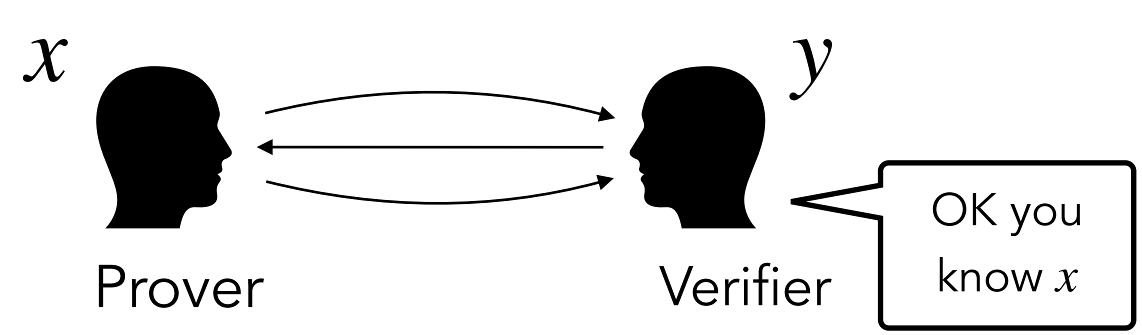


Input sharing $[\![x]\!]$

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



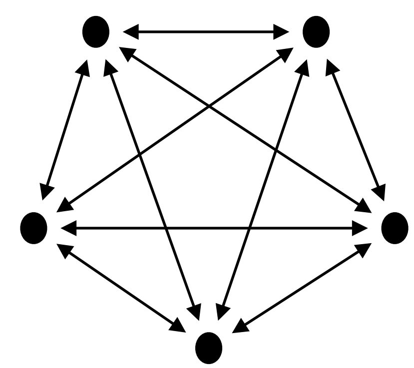
MPC-in-the-Head paradigm

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

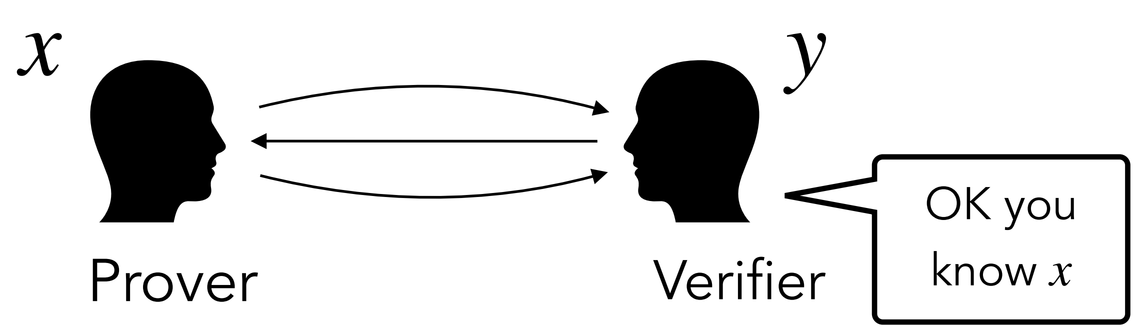


Input sharing $[\![x]\!]$

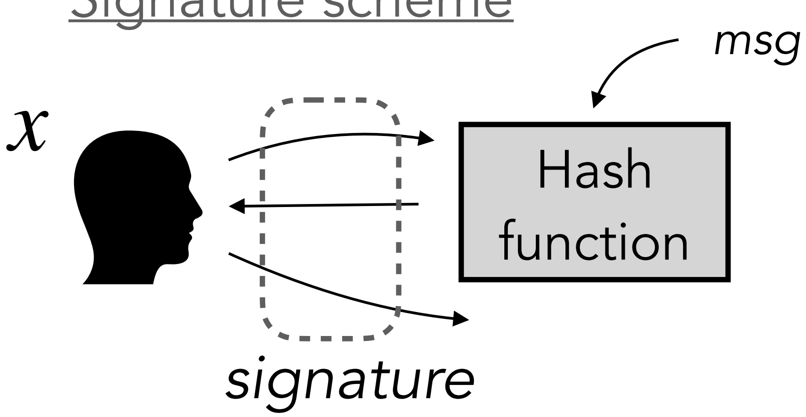
Joint evaluation of:

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

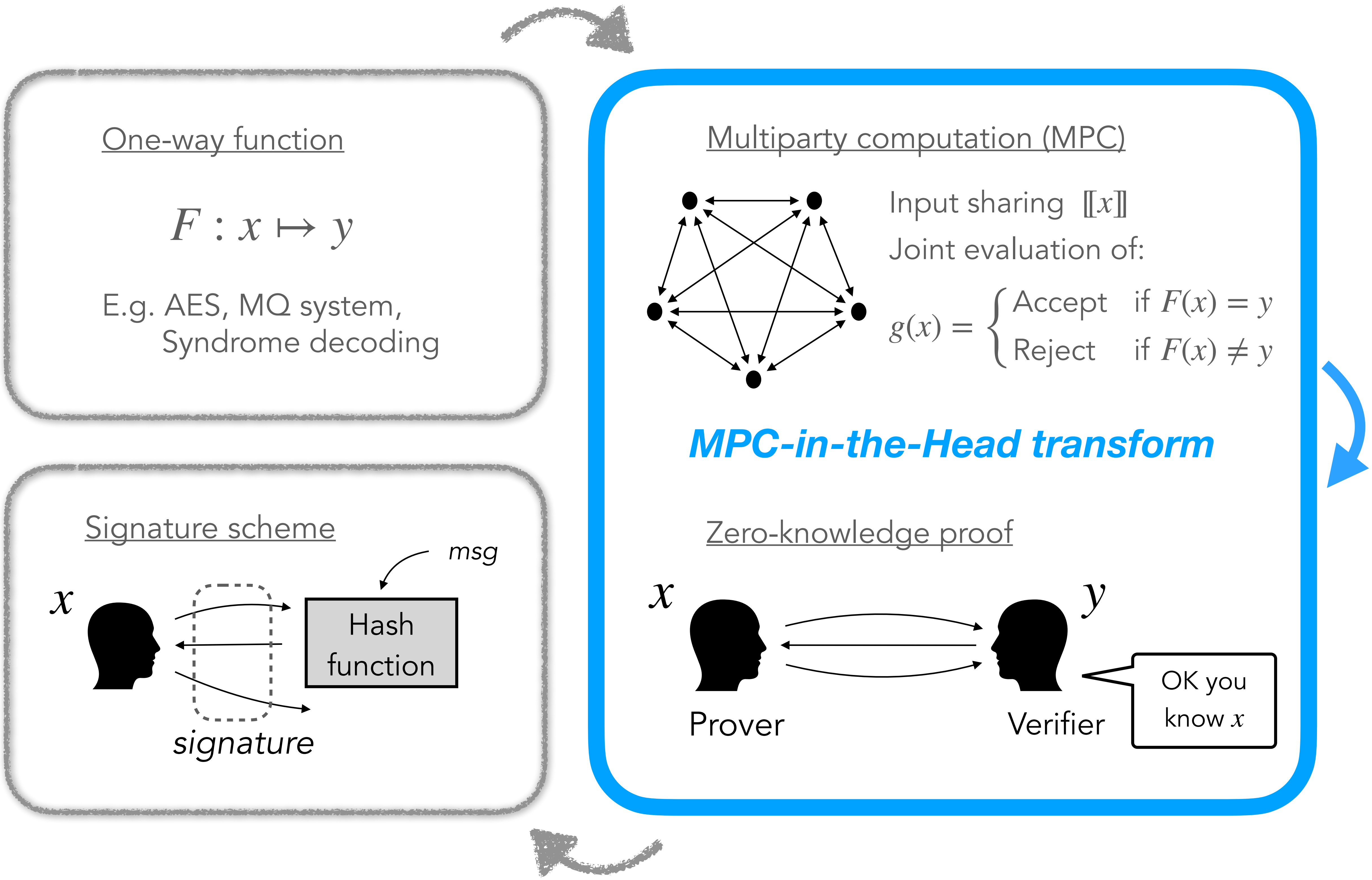
Zero-knowledge proof



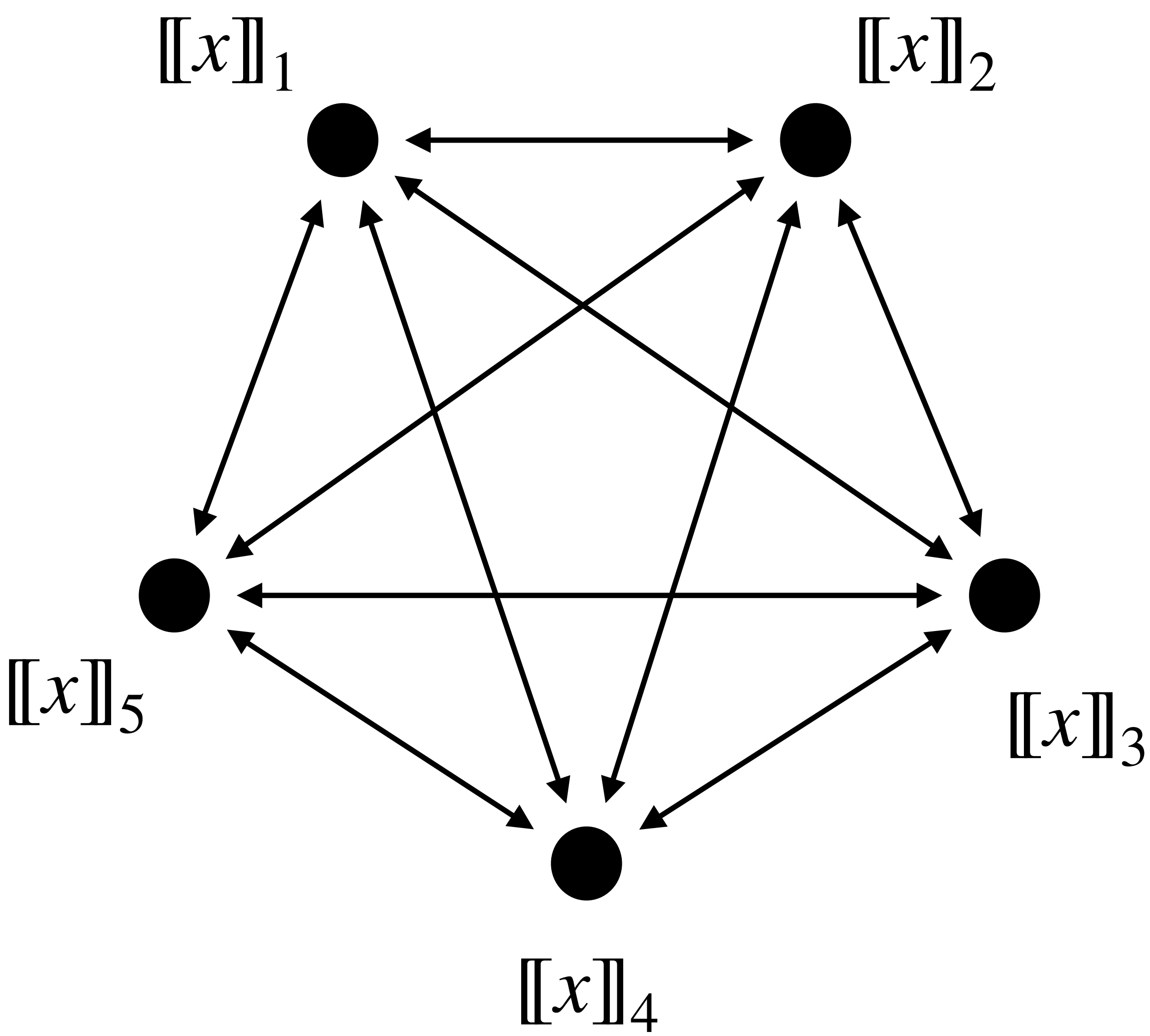
Signature scheme



MPC-in-the-Head paradigm



MPC model



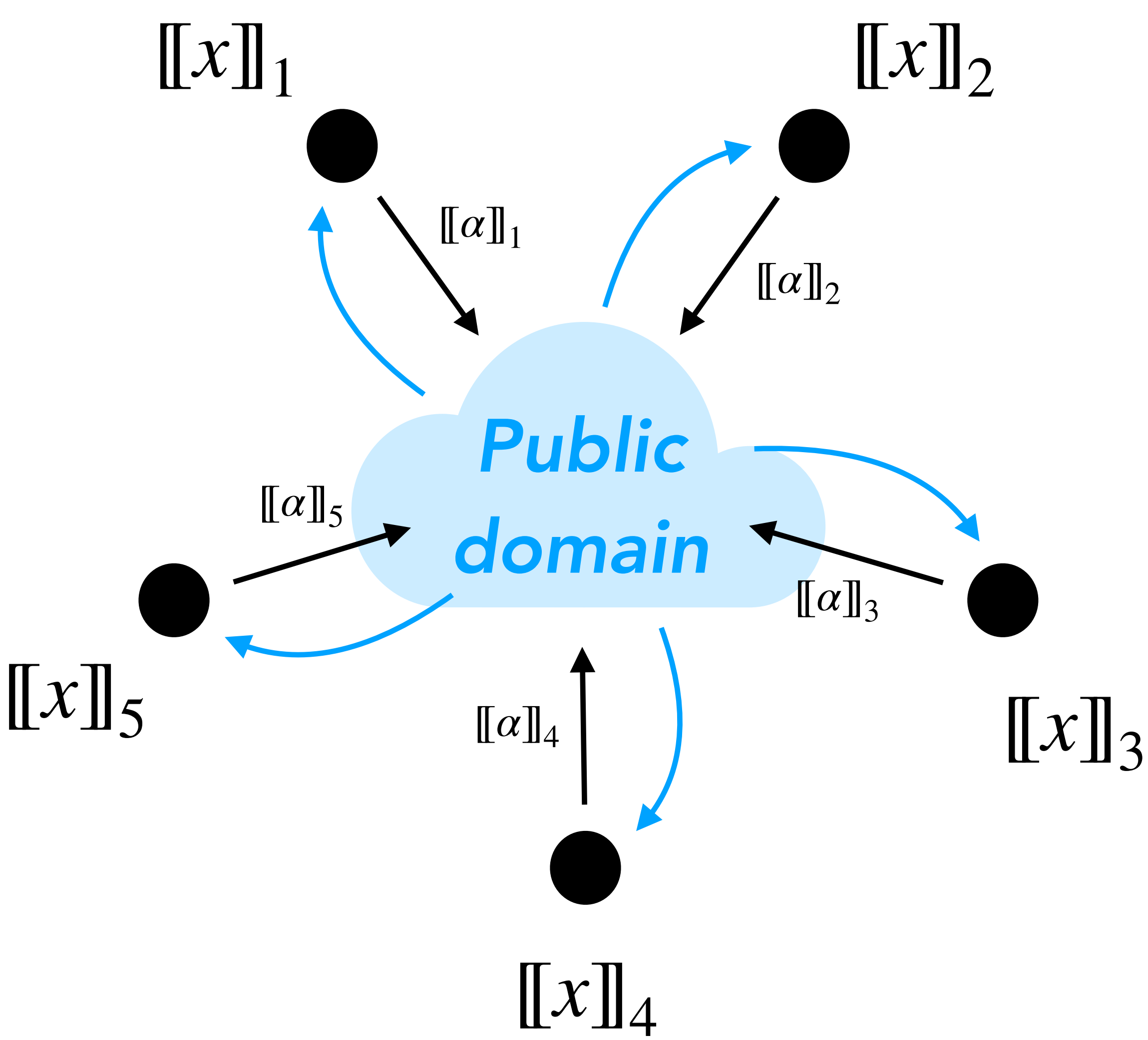
$[\![x]\!]$ is a linear secret sharing of x

- 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
- Semi-honest model

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
- Semi-honest model
- *Broadcast model*

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

MPCitH transform

Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

$\cdots$

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

Prover

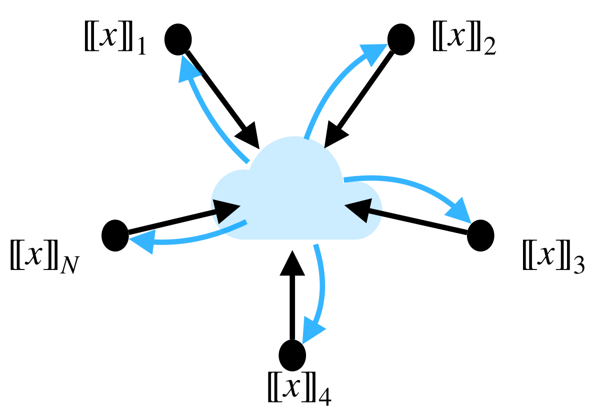
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\cdots$

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

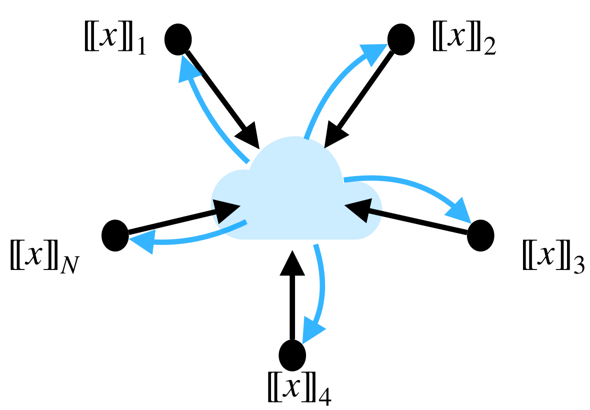
Prover | Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\ldots$

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

I

$\longleftarrow$

③ Choose a random set of parties

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

Prover

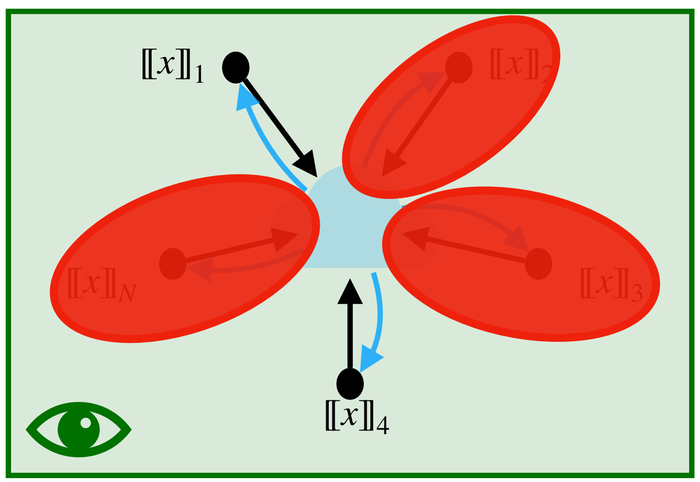
Verifier

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

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

③ Choose a random set of parties

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

← I

$([\![x]\!]_i, \rho_i)_{i \in I}$ →

Prover

Verifier

MPCitH transform

① Generate and commit shares

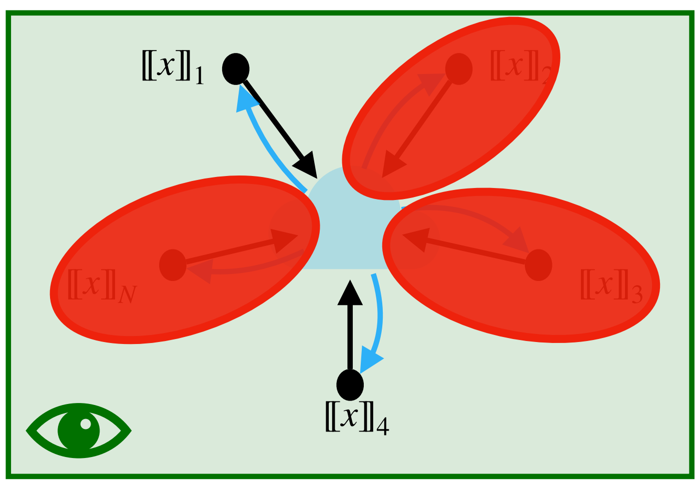
$[\![x]\!] = ([\![x]\!]_1, \dots, [\![x]\!]_N)$

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

$\cdots$

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

② Run MPC in their head



send broadcast

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

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

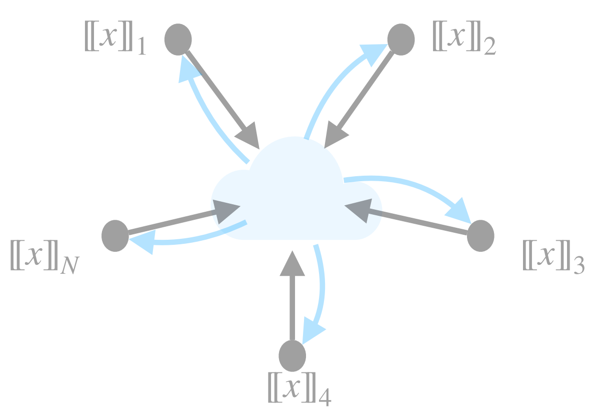
Prover

Verifier

MPCitH transform: with additive sharing

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

② Run MPC in their head



④ Open parties in I

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

send broadcast

Additive sharing:

$$x = [\![x]\!]_1 + \cdots + [\![x]\!]_N$$

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

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

Prover

Verifier

MPCitH transform: with additive sharing

① Generate and commit shares

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

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

$\ldots$

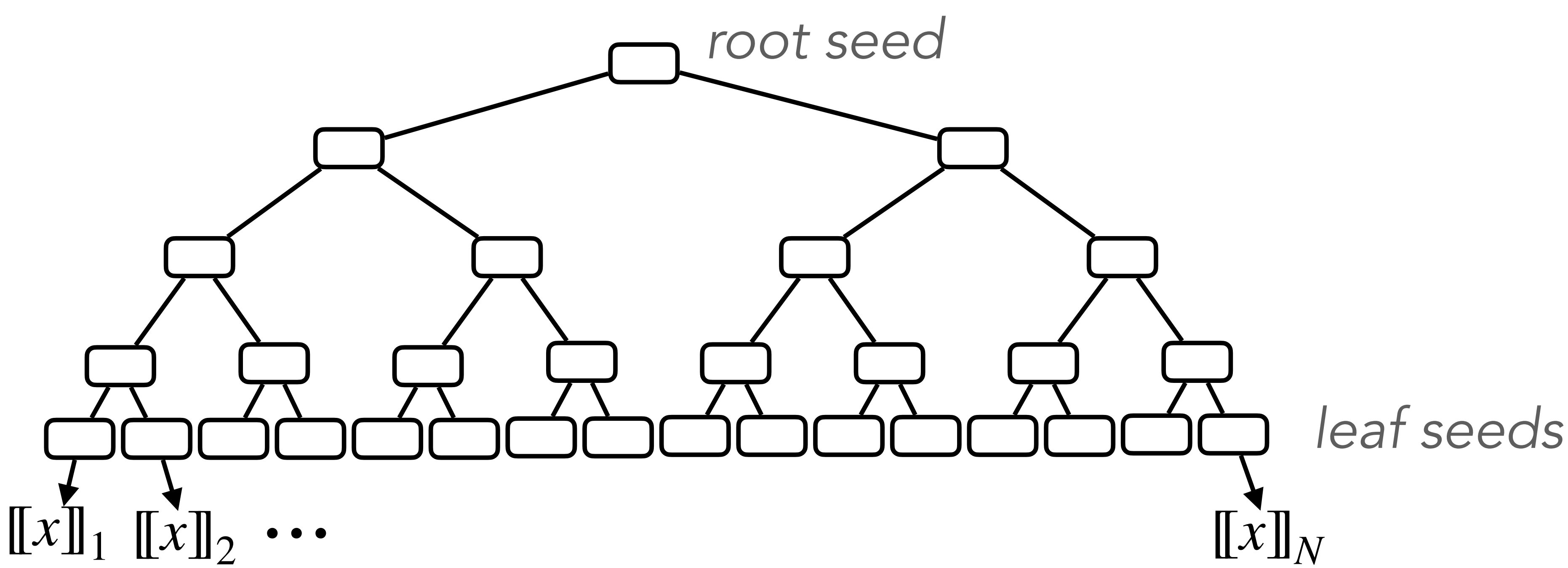
$\mathrm{Com}^{\rho_N}(\llbracket x \rrbracket_N)$

② Run MPC in their head

es

$(\llbracket x \rrbracket_i)$

Generated using a GGM seed tree [KKW18]:

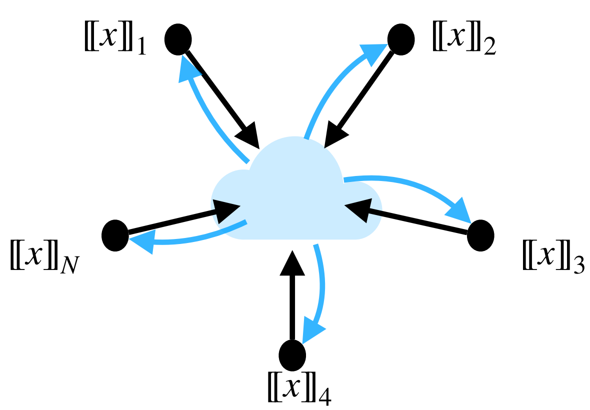


MPCitH transform: with additive sharing

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Sharing / MPC protocol

$(N-1)$-*private*

parties

ℓ.

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$

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

- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

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

Prover

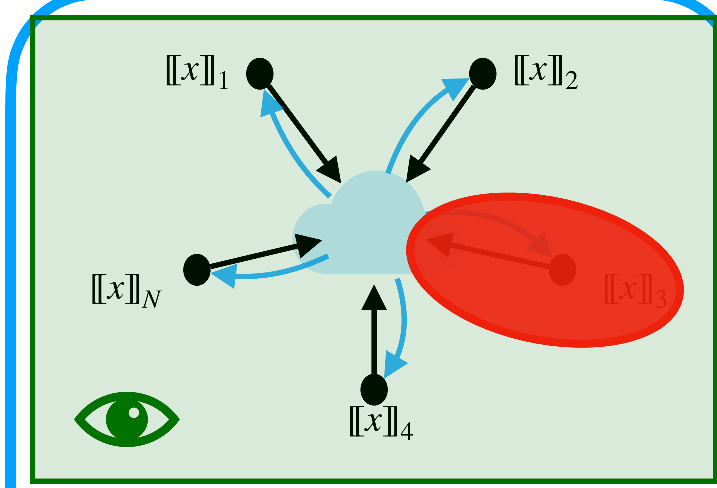
Verifier

MPCitH transform: with additive sharing

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Sharing / MPC protocol

$(N-1)$-private

parties

ℓ.

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$

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

- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

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

Prover

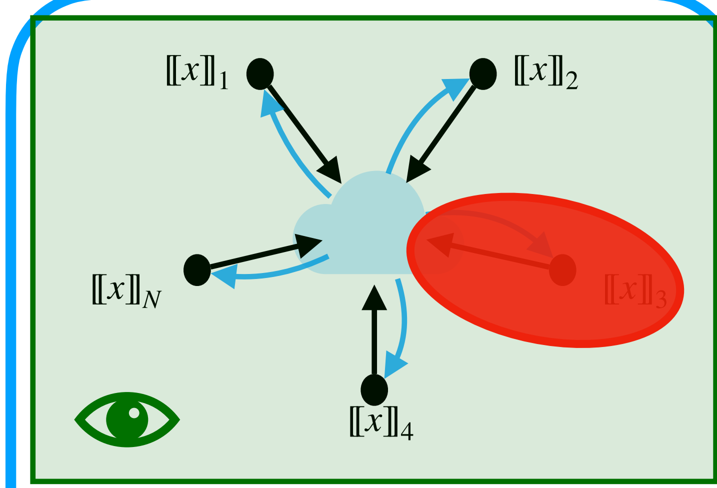
Verifier

MPCitH transform: with additive sharing

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Sharing / MPC protocol

$(N-1)$-*private*

$\Rightarrow$ soundness error $\approx \dfrac{1}{N}$

parties

ℓ.

- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$

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

- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

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

Prover

Verifier

MPCitH transform: with additive sharing

① Generate and commit shares

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

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

$\cdots$

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

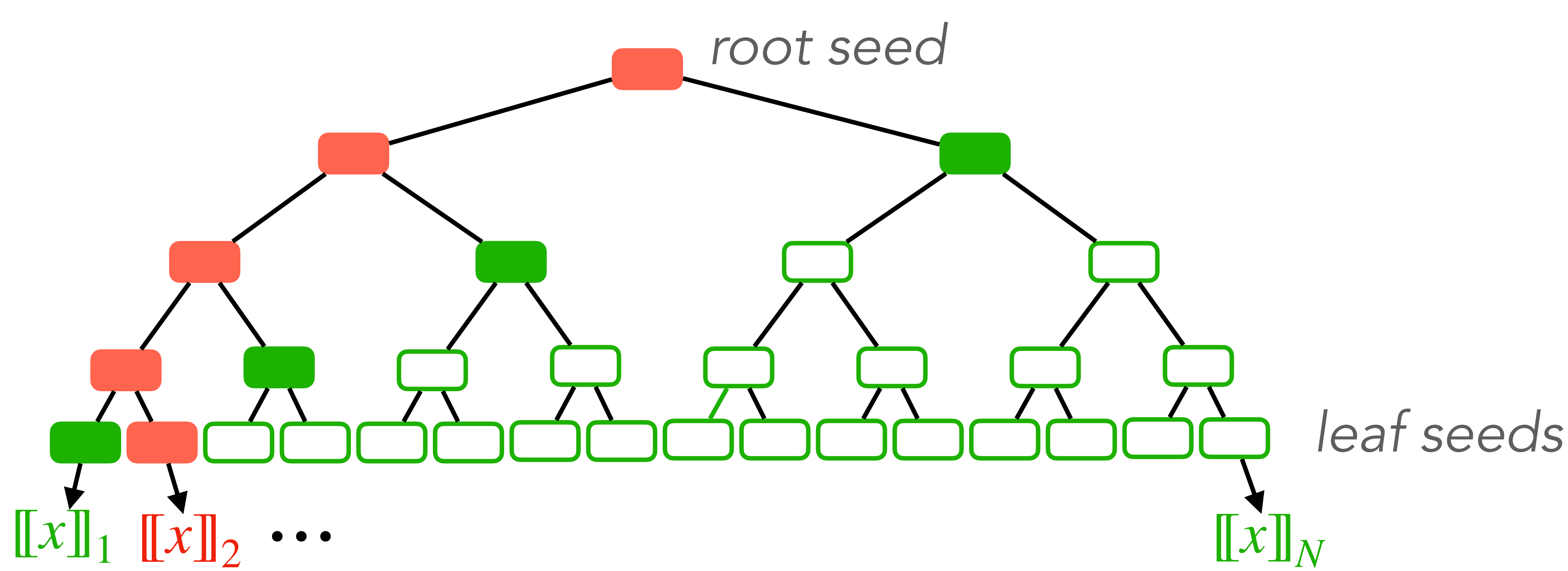
es

ties

$([\![x]\!]_i)$

$\varphi([\![x]\!]_i)$

Only $\log_2 N$ seeds to be revealed:



MPCitH transform: with threshold sharing

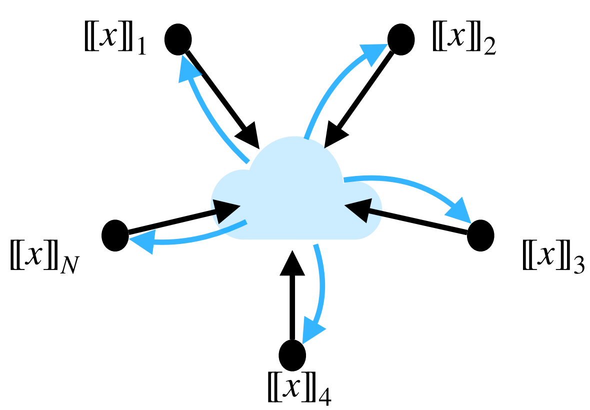
a.k.a. Threshold Computation in the Head (TCitH)

MPCitH transform: with threshold sharing

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

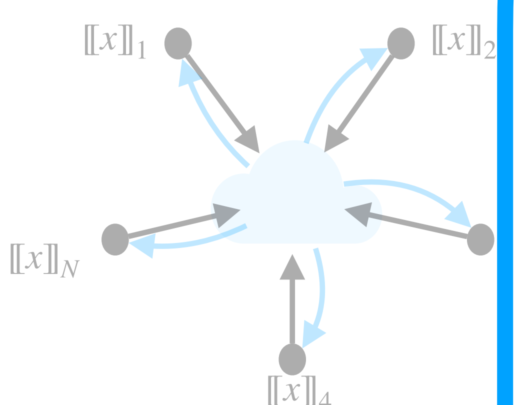
Verifier

MPCitH transform: with threshold sharing

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

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

② Run MPC in their hea



④ Open parties in I

Shamir secret sharing:

$$[\![x]\!]_i := P(e_i) \quad \forall i$$

$$\text{for } \; P(X) := x + r_1 \cdot X + \cdots + r_\ell \cdot X^\ell$$

m set of parties
t. $|I| = \ell$.

$\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
ion $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$
ccept

Prover

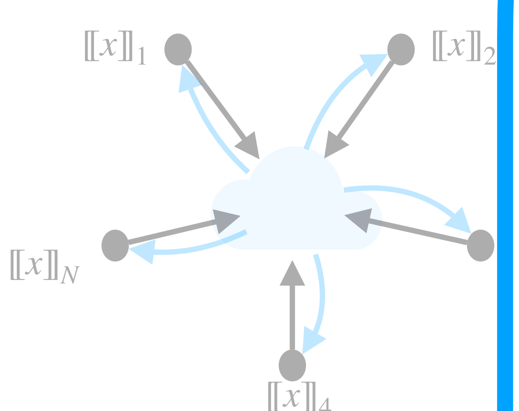
Verifier

MPCitH transform: with threshold sharing

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

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

② Run MPC in their hea



④ Open parties in I

n set of parties
t. $|I| = \ell$.

$\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
ion $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$
ccept

Shamir secret sharing:

$$[\![x]\!]_i := P(e_i) \quad \forall i$$

$$\text{for} \quad P(X) := x + r_1 \cdot X + \cdots + r_\ell \cdot X^\ell$$

$\Rightarrow \ell$-privacy

Prover

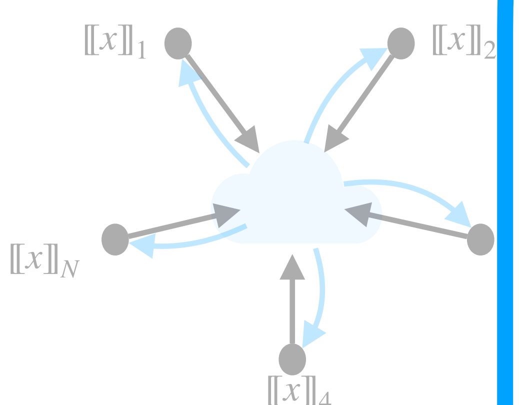
Verifier

MPCitH transform: with threshold sharing

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

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

② Run MPC in their hea



④ Open parties in I

m set of parties
t. $|I| = \ell$.

$\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
ion $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$
ccept

Shamir secret sharing:

$$[\![x]\!]_i := P(e_i) \quad \forall i$$

$$\text{for} \quad P(X) := x + r_1 \cdot X + \cdots + r_\ell \cdot X^\ell$$

$\Rightarrow \ell$-privacy

We use $\ell \ll N$ (e.g. $\ell = 1$)

Prover

Verifier

MPCitH transform: with threshold sharing

① Generate and commit shares

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

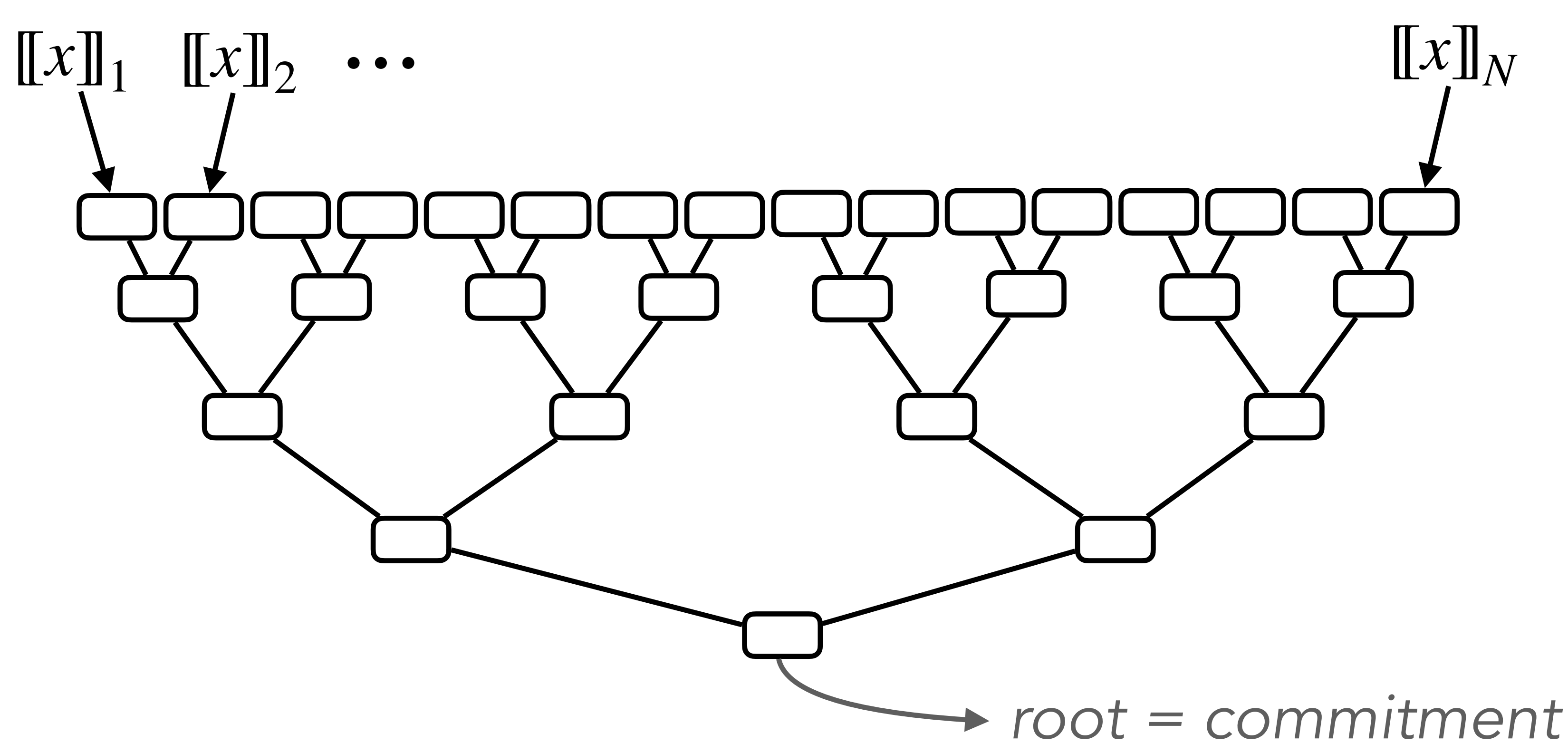
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

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

② Run MPC in their head

Committed using a Merkle tree:



es

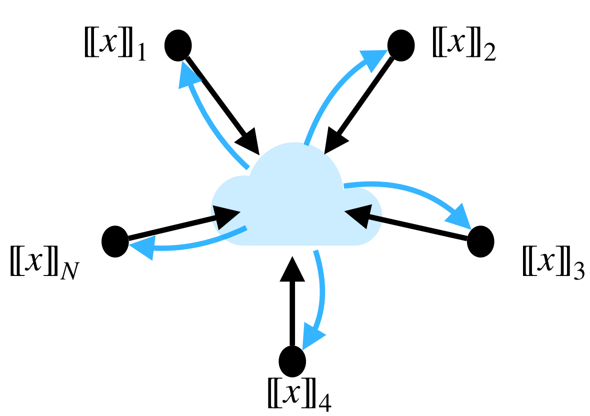
$([\![x]\!]_i)$

MPCitH transform: with threshold sharing

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Sharing / MPC protocol *ℓ-private*

parties

ℓ.

$]\!]_i)$

$= \varphi([\![x]\!]_i)$

Prover

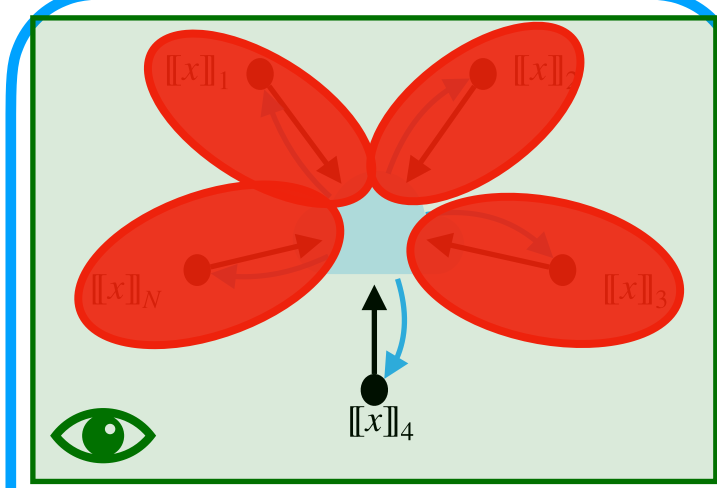
Verifier

MPCitH transform: with threshold sharing

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Prover

Sharing / MPC protocol *ℓ-private*

parties

ℓ.

$]\!]_i)$

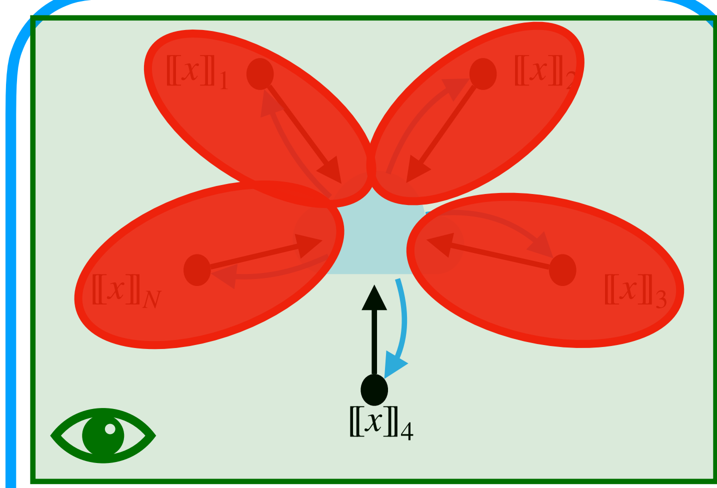
$= \varphi([\![x]\!]_i)$

MPCitH transform: with threshold sharing

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Prover

Sharing / MPC protocol *ℓ-private*

$\Rightarrow$ soundness error $\approx (N - \ell)/N$ 🤔

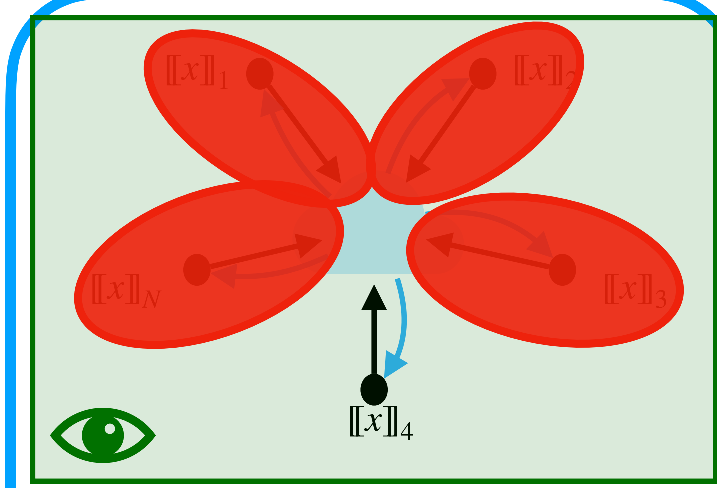
Verifier

MPCitH transform: with threshold sharing

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Sharing / MPC protocol *ℓ-private*

$\Rightarrow$ soundness error $\approx (N - \ell)/N$ 🤔

Much better! $\approx \dfrac{1}{\binom{N}{\ell}}$ 🤩

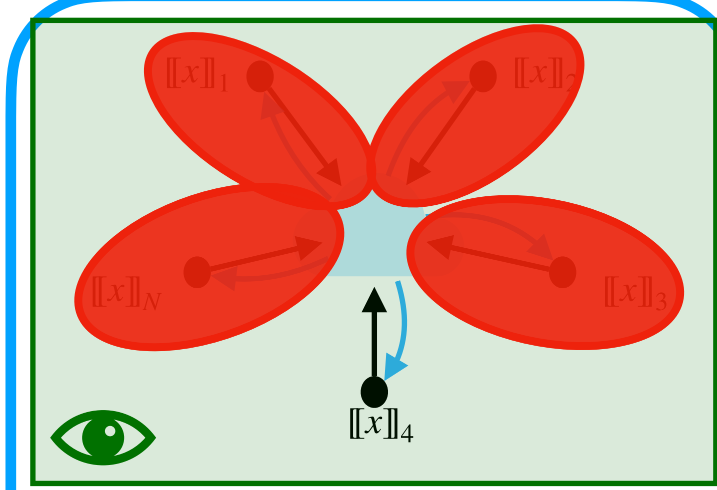
Prover

Verifier

MPCitH transform: with threshold sharing

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

② Run MPC in their head



④ Open parties in I

Prover

Sharing / MPC protocol *ℓ-private*

$\Rightarrow$ soundness error $\approx (N - \ell)/N$ 🤔

Much better! $\approx \dfrac{1}{\binom{N}{\ell}}$ 🤩

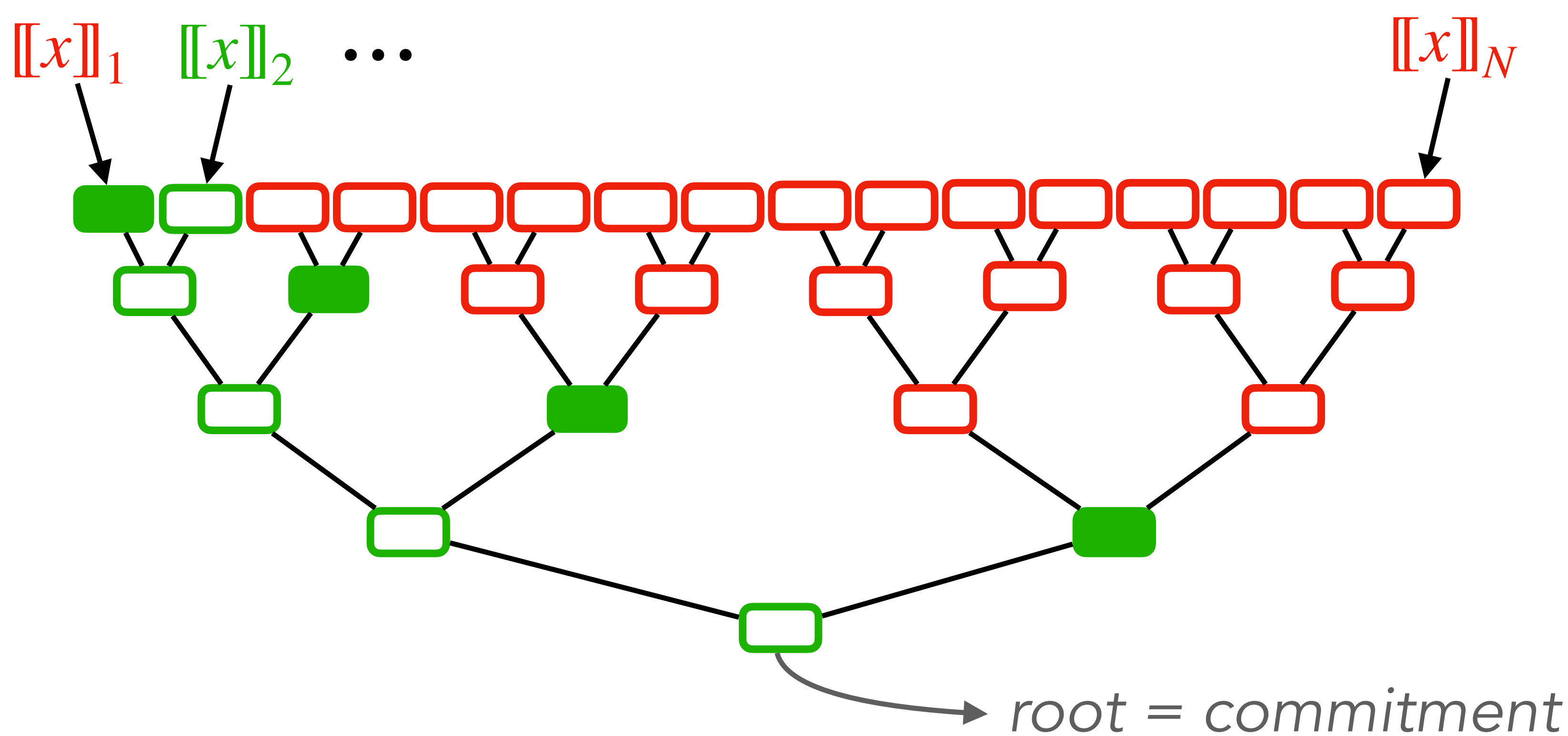
💡 broadcasted sharings = Reed-Solomon codewords

Verifier

MPCitH transform: with threshold sharing

① Generate and commit shares $\quad \mathrm{Com}^{\rho_1}([\![x]\!]_1)$

Only $\log_2 N$ labels to be revealed:



TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime		
Verifier runtime		
Size of tree		

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime		
Verifier runtime	*false positive probability*	*$\Rightarrow$ must be smaller in TCitH*
Size of tree		

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees **+ hypercube** [AGHHJY]	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime	Party emulations: $\log N + 1$ Symmetric crypto: $O(N)$	Party emulations: 2 Symmetric crypto: $O(N)$
Verifier runtime		
Size of tree		

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime	Party emulations: $\log N + 1$ Symmetric crypto: $O(N)$	Party emulations: 2 Symmetric crypto: $O(N)$
Verifier runtime		fewer party emulations
Size of tree		

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime	Party emulations: log *N* +1 Symmetric crypto: *O(N)*	Party emulations: 2 Symmetric crypto: *O(N)*
Verifier runtime	Party emulations: log *N* Symmetric crypto: *O(N)*	Party emulations: 1 Symmetric crypto: *O(log N)*
Size of tree		

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime	Party emulations: $\log N + 1$ Symmetric crypto: $O(N)$	Party emulations: 2 Symmetric crypto: $O(N)$
Verifier runtime	Party emulations: $\log N$ Symmetric crypto: $O(N)$	Party emulations: 1 Symmetric crypto: $O(\log N)$
Size of tree		*fewer party emulations*

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime	Party emulations: log N +1 Symmetric crypto: $O(N)$	Party emulations: 2 Symmetric crypto: $O(N)$
Verifier runtime	Party emulations: log N Symmetric crypto: $O(N)$	Party emulations: 1 Symmetric crypto: $O(\log N)$
Size of tree		🚀 *much less symmetric crypto*

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime	Party emulations: log N +1 Symmetric crypto: *O(N)*	Party emulations: 2 Symmetric crypto: *O(N)*
Verifier runtime	Party emulations: log N Symmetric crypto: *O(N)*	Party emulations: 1 Symmetric crypto: *O(log N)*
Size of tree	128-bit security: ~2KB 256-bit security: ~8KB	128-bit security: ~4KB 256-bit security: ~16KB

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime	Party emulations: log N +1 Symmetric crypto: $O(N)$	Party emulations: 2 Symmetric crypto: $O(N)$
Verifier runtime	Party emulations: log N Symmetric crypto: $O(N)$	Party emulations: 1 Symmetric crypto: $O(\log N)$
Size of tree	128-bit security: ~2KB 256-bit security: ~8KB	128-bit security: ~4KB 256-bit security: ~16KB

factor 2

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees + hypercube	**TCitH** (original framework) $\ell = 1$
Soundness error	$\approx \frac{1}{N} + p$	$\approx \frac{1}{N} + p \cdot \binom{N}{2}$
Prover runtime	Party emulations: $\log N + 1$ Symmetric crypto: $O(N)$	Party emulations: 2 Symmetric crypto: $O(N)$
Verifier runtime	Party emulations: $\log N$ Symmetric crypto: $O(N)$	Party emulations: 1 Symmetric crypto: $O(\log N)$
Size of tree	128-bit security: ~2KB 256-bit security: ~8KB	128-bit security: ~4KB 256-bit security: ~16KB
Number of parties		$\boxed{N \leq \lvert\mathbb{F}\rvert}$

TCitH vs. (additive-sharing) MPCitH

	MPCitH + seed trees	**TCitH** (original framework)
Soundn		$\frac{N}{2}\Big)$
Prover		2 : *O(N)*
Verifier		1 : *O(log N)*
Size c		4KB 16KB
Number of parties		$N \leq \lvert\mathbb{F}\rvert$

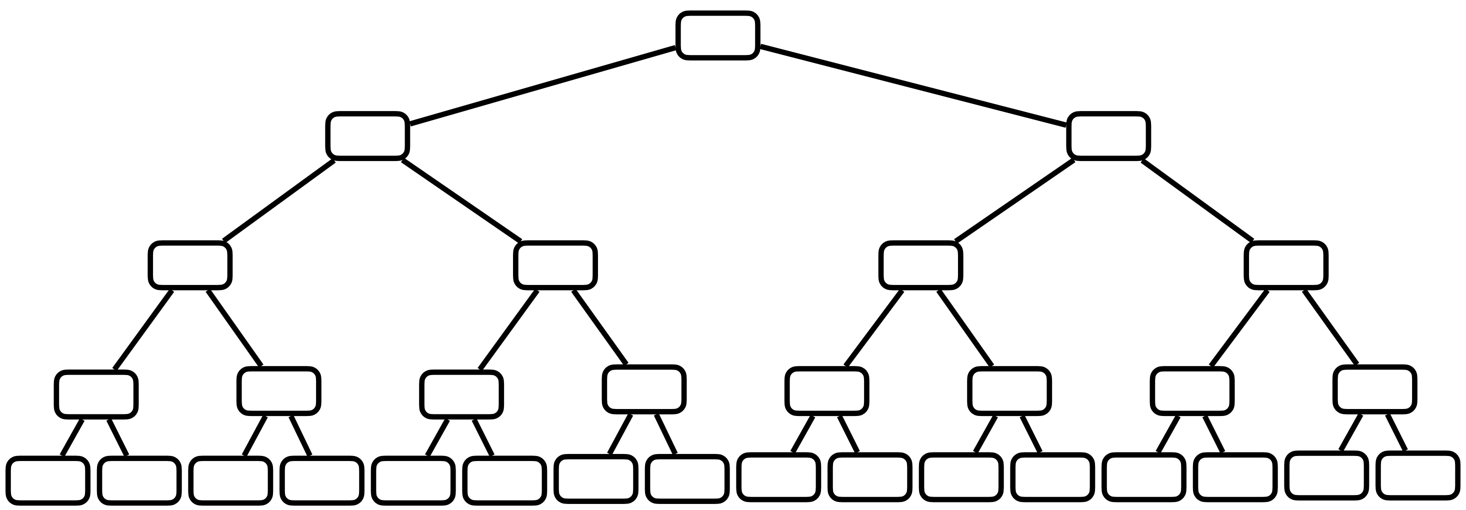
🤔 What can we do to avoid these limitations?

☝️ TCitH with GGM trees

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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

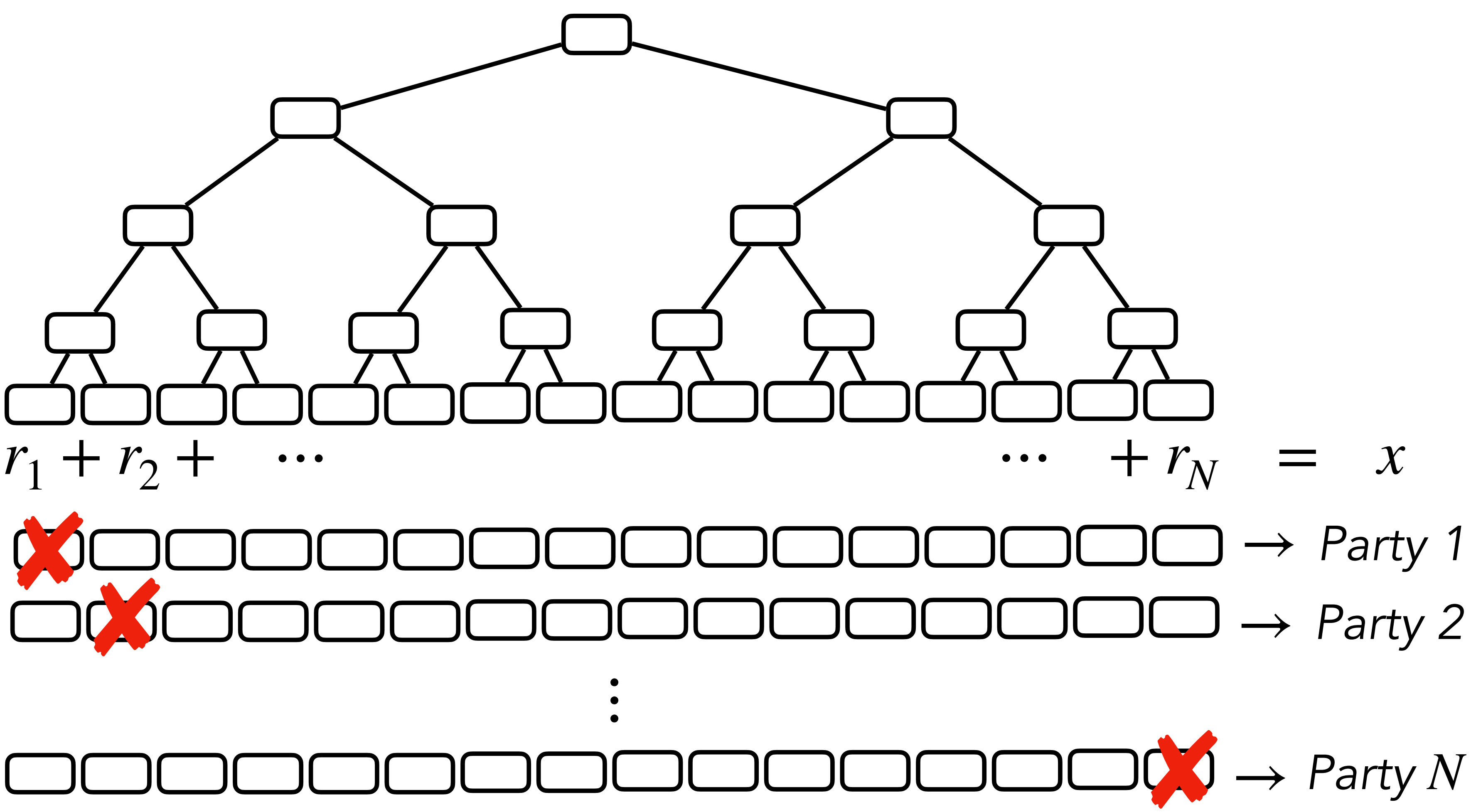


$r_1 + r_2 + \cdots \quad \cdots + r_N = x$

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



TCitH with GGM trees

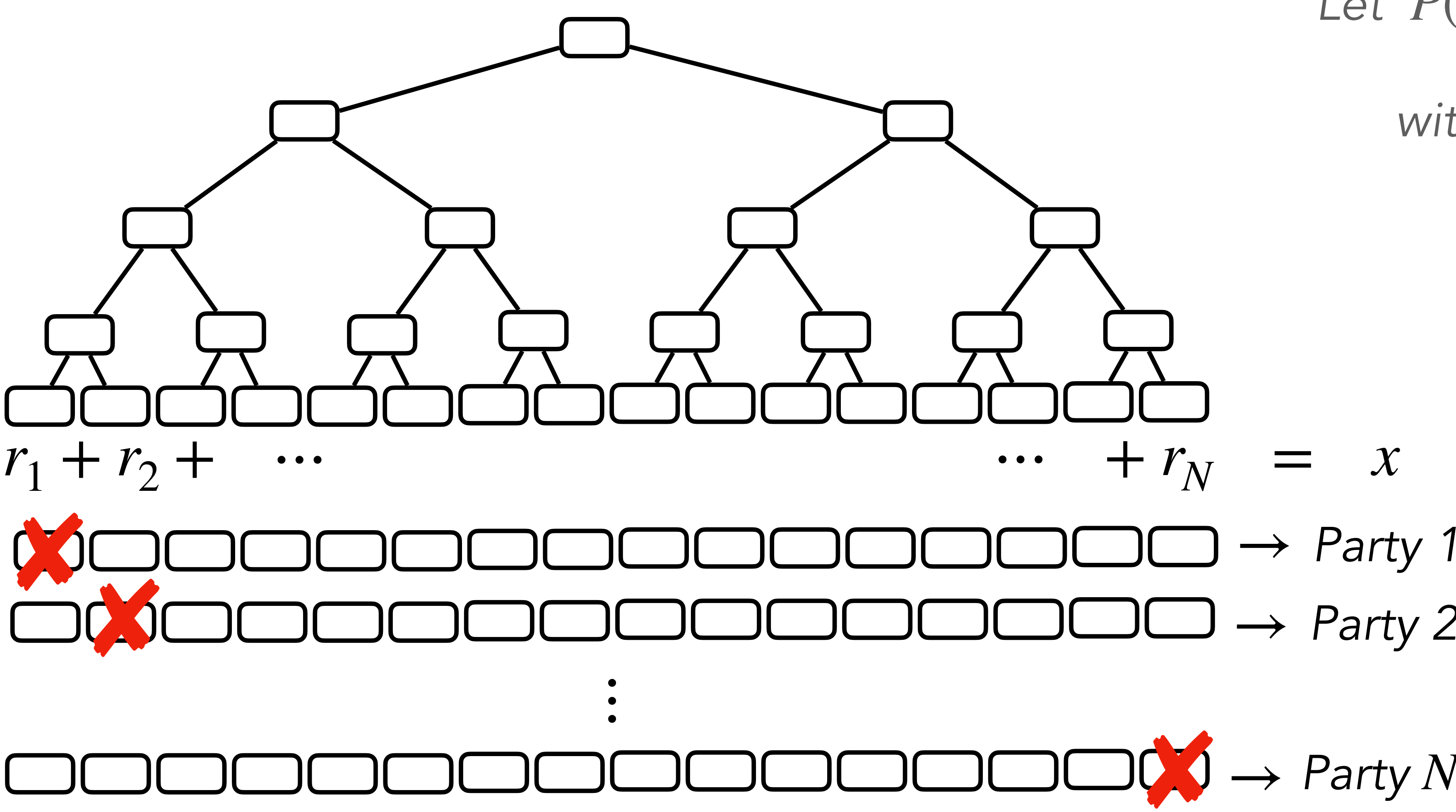
Step 1: Generate a replicated secret sharing [ISN89]

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

Step 2: Convert it into a Shamir's secret sharing [CDI05]

Let $P(X) = \sum_j r_j P_j(X)$

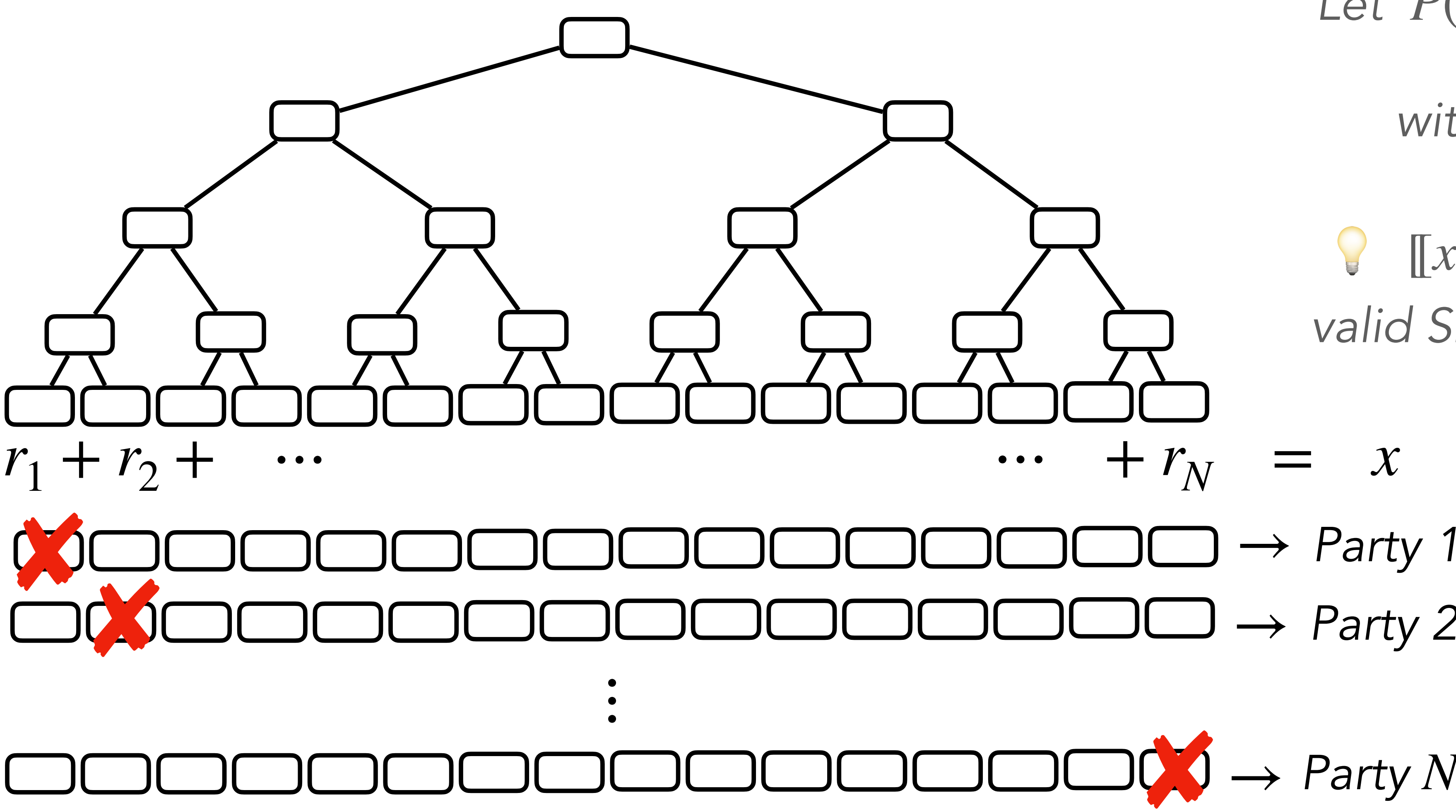
with $P_j(X) = 1 - (1/e_j) \cdot X$



TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



Step 2: Convert it into a Shamir's secret sharing [CDI05]

Let $P(X) = \sum_j r_j P_j(X)$

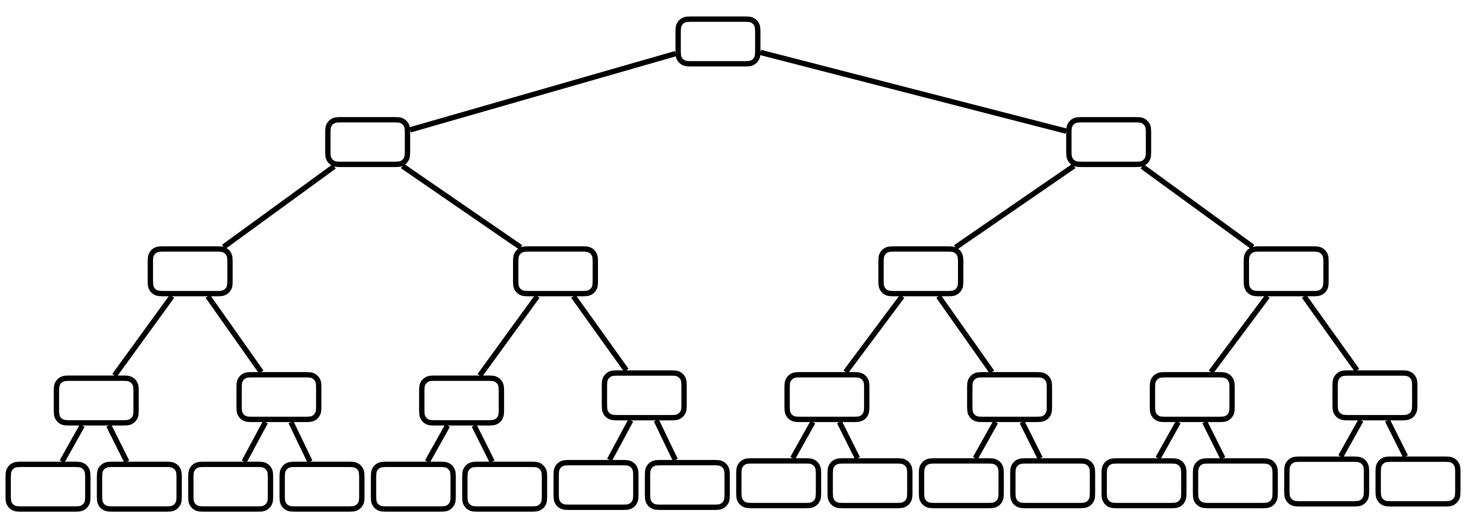
with $P_j(X) = 1 - (1/e_j) \cdot X$

💡 $[\![x]\!] = (P(e_1), \ldots, P(e_N))$ is a valid Shamir's secret sharing of x

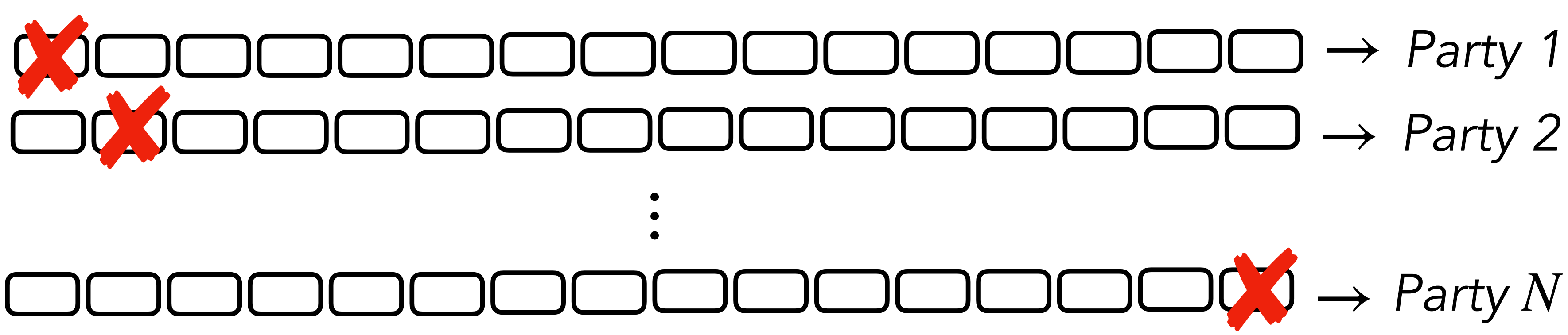
TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



$r_1 + r_2 + \quad \cdots \quad \cdots \quad + r_N \quad = \quad x$



Step 2: Convert it into a Shamir's secret sharing [CDI05]

Let $P(X) = \sum_j r_j P_j(X)$

with $P_j(X) = 1 - (1/e_j) \cdot X$

💡 $[\![x]\!] = (P(e_1), \ldots, P(e_N))$ *is a valid Shamir's secret sharing of* x

Party i *can compute*

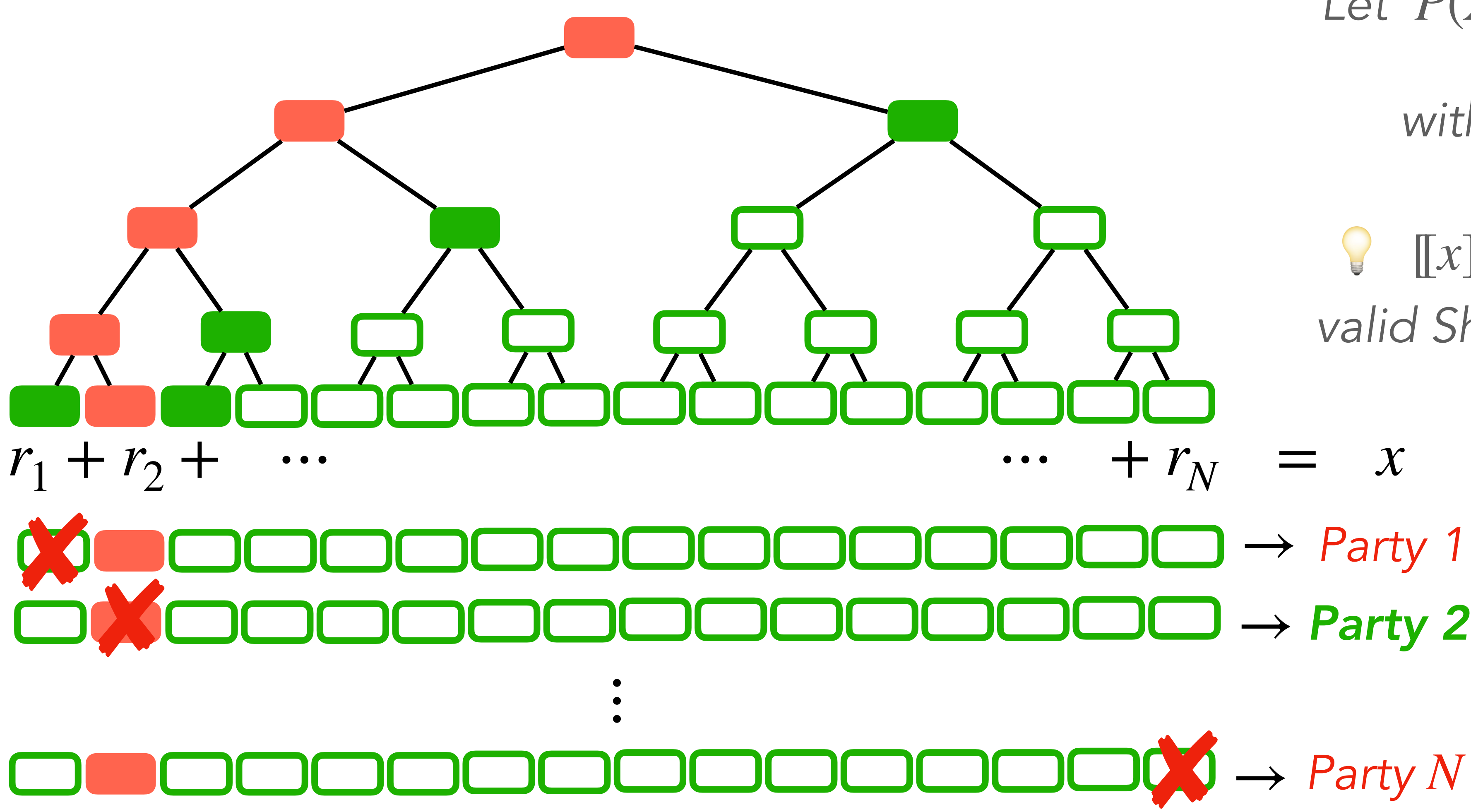
$$[\![x]\!]_i = \sum_{j \neq i} r_j P_j(e_i)$$

(since $P_i(e_i) = 0$*)*

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



Step 2: Convert it into a Shamir's secret sharing [CDI05]

Let $P(X) = \sum_j r_j P_j(X)$

with $P_j(X) = 1 - (1/e_j) \cdot X$

💡 $[\![x]\!] = (P(e_1), \ldots, P(e_N))$ *is a valid Shamir's secret sharing of* x

Party i *can compute*

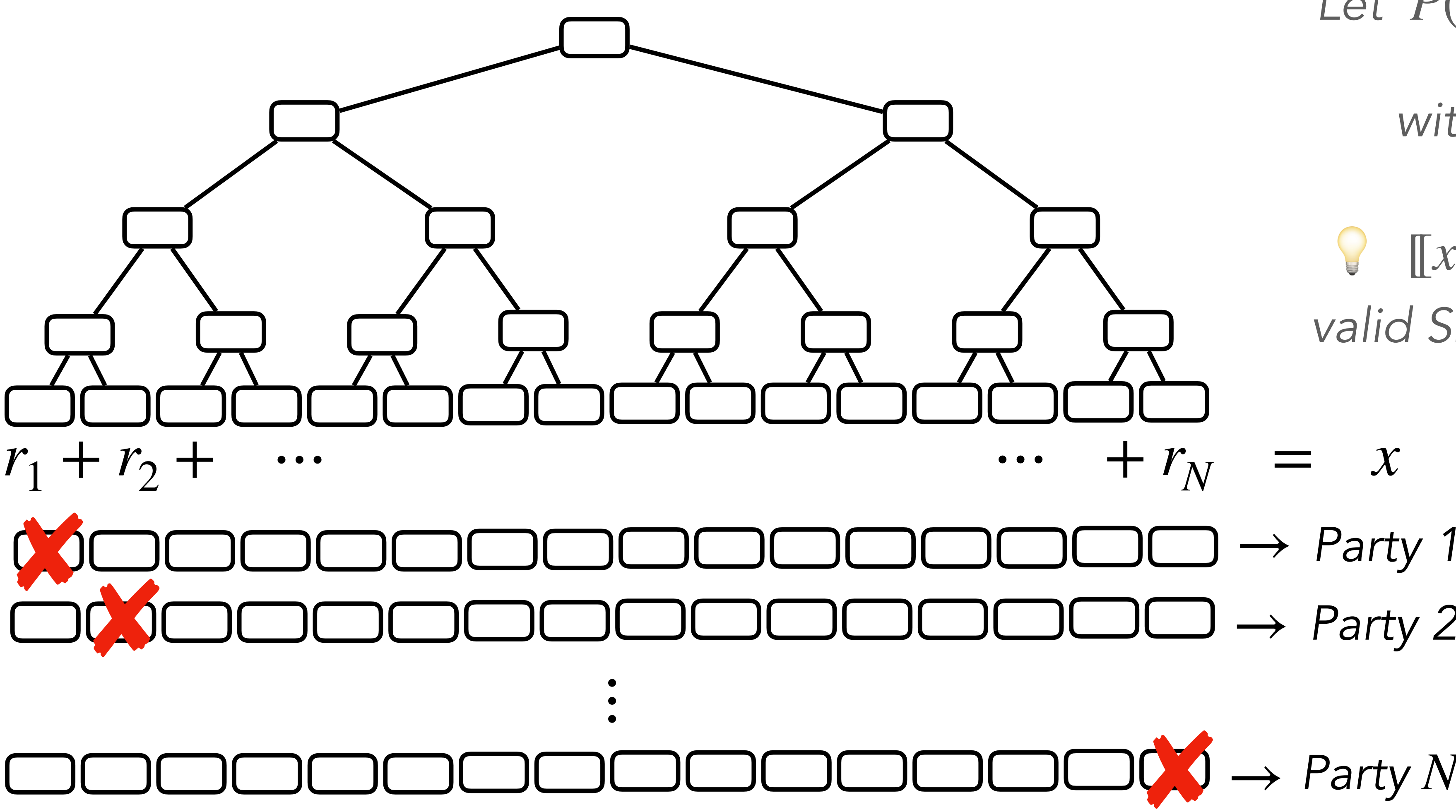
$$[\![x]\!]_i = \sum_{j \neq i} r_j P_j(e_i)$$

(since $P_i(e_i) = 0$*)*

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



Step 2: Convert it into a Shamir's secret sharing [CDI05]

Let $P(X) = \sum_j r_j P_j(X)$

with $P_j(X) = 1 - (1/e_j) \cdot X$

💡 $[\![x]\!] = (P(e_1), \ldots, P(e_N))$ *is a valid Shamir's secret sharing of* x

Party i can compute

$$[\![x]\!]_i = \sum_{j \neq i} r_j P_j(e_i)$$

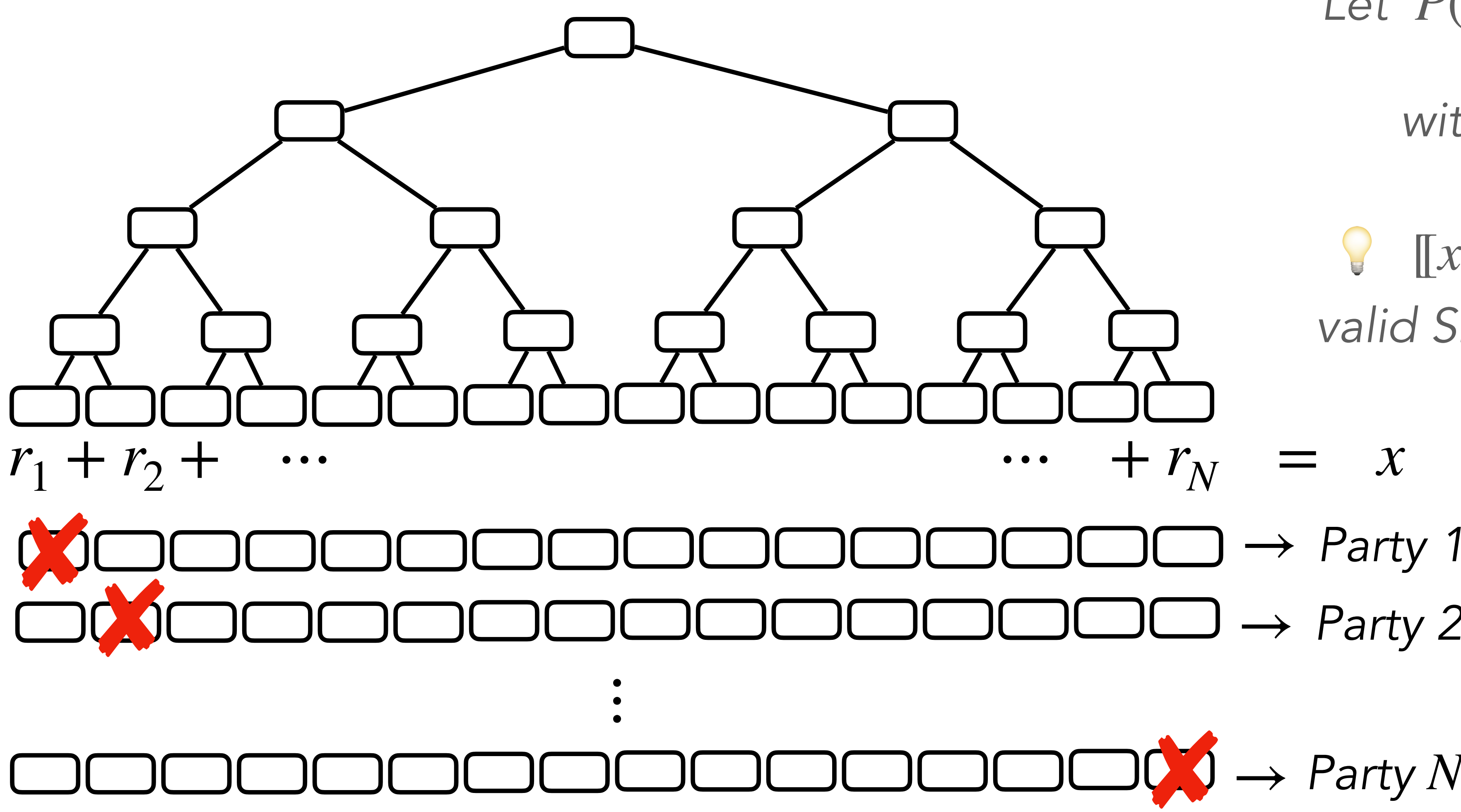
(since $P_i(e_i) = 0$*)*

🛠 Can be adapted to $\ell > 1$

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



Step 2: Convert it into a Shamir's secret sharing [CDI05]

Let $P(X) = \sum_j r_j P_j(X)$

with $P_j(X) = 1 - (1/e_j) \cdot X$

💡 $[\![x]\!] = (P(e_1), \ldots, P(e_N))$ *is a valid Shamir's secret sharing of x*

Party i can compute

$$[\![x]\!]_i = \sum_{j \neq i} r_j P_j(e_i)$$

(since $P_i(e_i) = 0$)

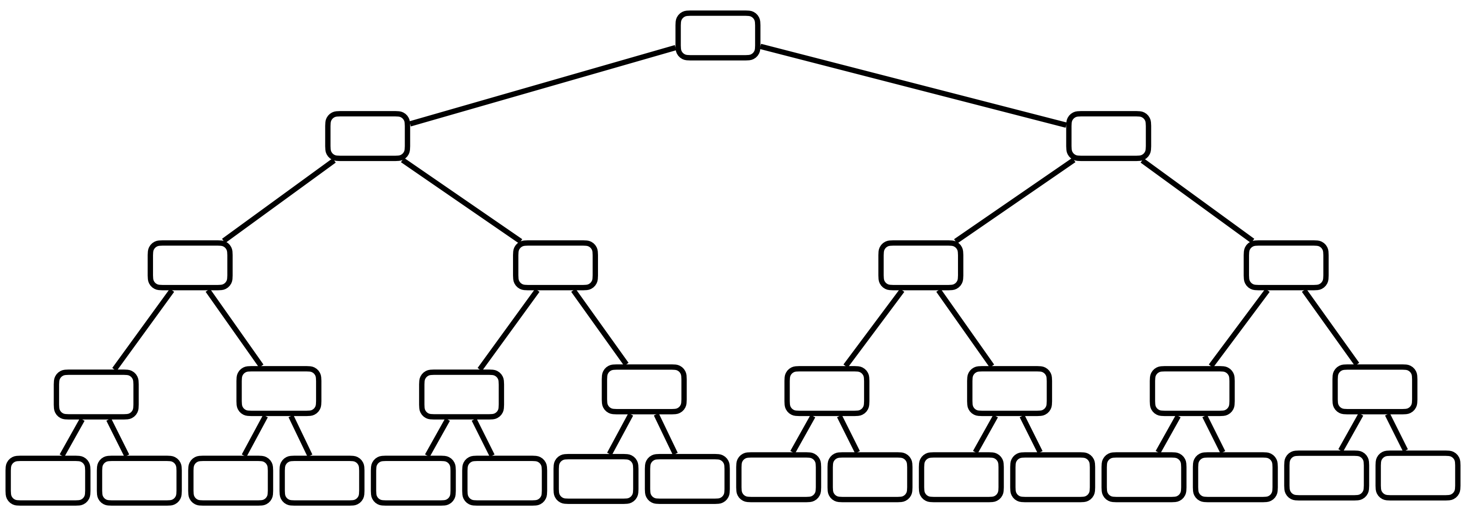
🛠 Can be adapted to $\ell > 1$

🌲 Size of GGM tree

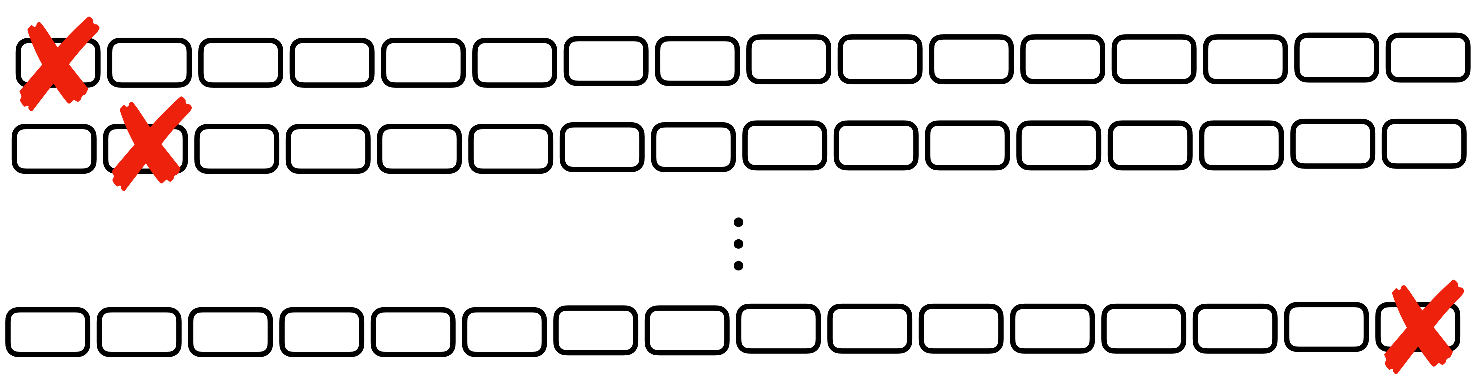
TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



$r_1 + r_2 + \quad \cdots \quad \cdots \quad + r_N \quad = \quad x$



Step 2: Convert it into a Shamir's secret sharing [CDI05]

Let $P(X) = \sum_j r_j P_j(X)$

with $P_j(X) = 1 - (1/e_j) \cdot X$

💡 $[\![x]\!] = (P(e_1), \ldots, P(e_N))$ *is a valid Shamir's secret sharing of* x

Party i *can compute*

$$[\![x]\!]_i = \sum_{j \neq i} r_j P_j(e_i)$$

(since $P_i(e_i) = 0$*)*

🛠️ Can be adapted to $\ell > 1$

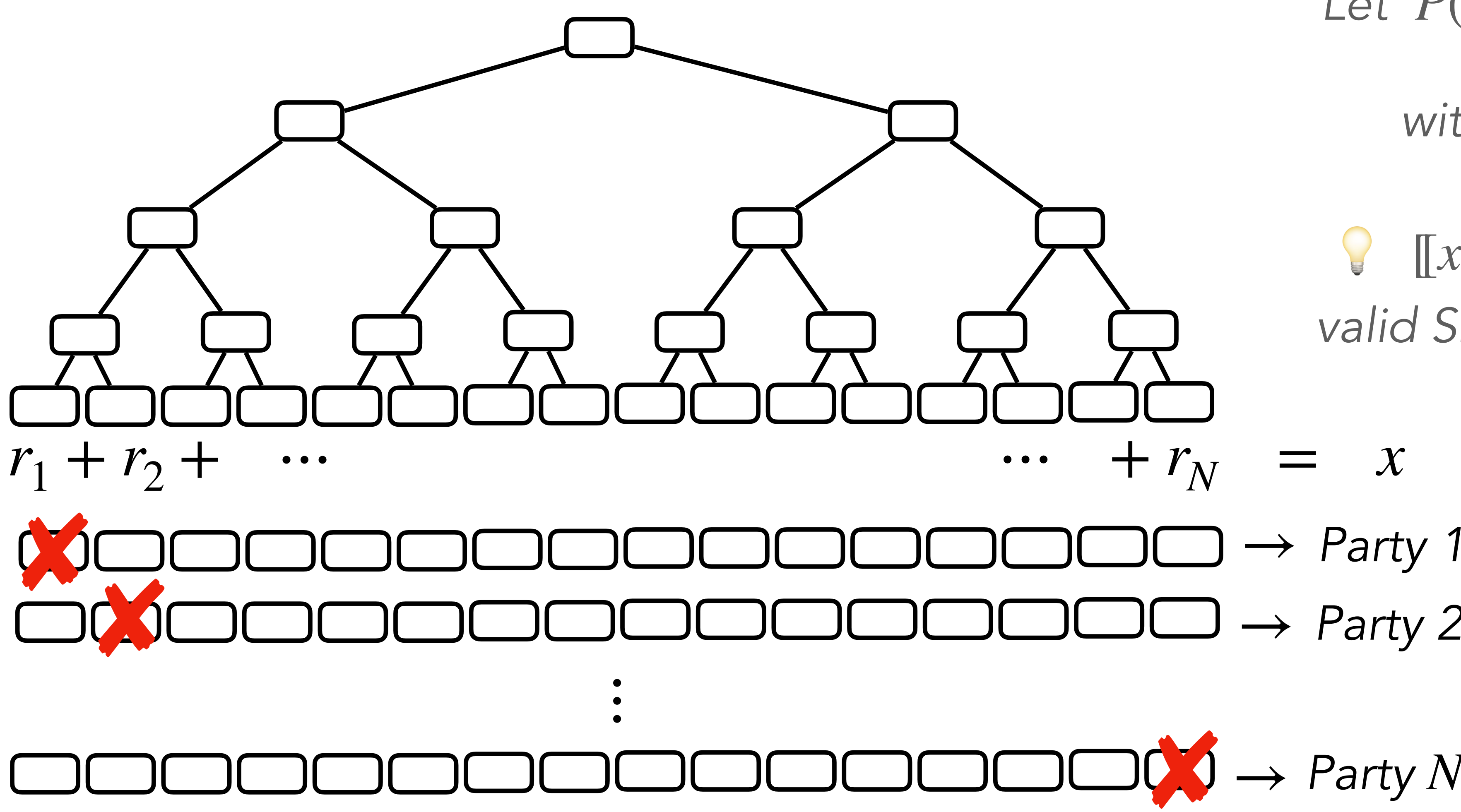
🌲 Size of GGM tree

😇 Good soundness (only valid sharings)

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



Step 2: Convert it into a Shamir's secret sharing [CDI05]

Let $P(X) = \sum_j r_j P_j(X)$

with $P_j(X) = 1 - (1/e_j) \cdot X$

💡 $[\![x]\!] = (P(e_1), \ldots, P(e_N))$ is a valid Shamir's secret sharing of x

Party i can compute

$$[\![x]\!]_i = \sum_{j \neq i} r_j P_j(e_i)$$

(since $P_i(e_i) = 0$)

🛠 Can be adapted to $\ell > 1$

🌲 Size of GGM tree

😇 Good soundness (only valid sharings)

🐌 Loose fast verification

Speedups for MPCitH candidates

	Additive MPCitH		TCitH (GGM tree)	
	Traditional (ms)	Hypercube (ms)	TCitH (ms)	Saving
Party emulations / repetition	N	$1 + \log_2 N$	2	

Speedups for MPCitH candidates

	Additive MPCitH		TCitH (GGM tree)	
	Traditional (ms)	Hypercube (ms)	TCitH (ms)	Saving
Party emulations / repetition	N	$1 + \log_2 N$	2	

⚠️ But only if $|\mathbb{F}| \geq N$

Speedups for MPCitH candidates

	Additive MPCitH		TCitH (GGM tree)	
	Traditional (ms)	Hypercube (ms)	TCitH (ms)	Saving
Party emulations / repetition	N	$1 + \log_2 N$	2	

⚠️ But only if $|\mathbb{F}| \geq N$

🛠️ Party emulations $= 1 + \left\lceil \frac{\log_2 N}{\log_2 |\mathbb{F}|} \right\rceil$

Speedups for MPCitH candidates

	Additive MPCitH		TCitH (GGM tree)	
	Traditional (ms)	Hypercube (ms)	TCitH (ms)	Saving
Party emulations / repetition	N	$1 + \log_2 N$	2	

⚠️ But only if $|\mathbb{F}| \geq N$

🛠️ Party emulations $= 1 + \left\lceil \dfrac{\log_2 N}{\log_2 |\mathbb{F}|} \right\rceil = \begin{cases} 2 & \text{if } |\mathbb{F}| \geq N \\ & \vdots \\ 1 + \log_2 N & \text{if } |\mathbb{F}| = 2 \end{cases}$

Speedups for MPCitH candidates

	Additive MPCitH		TCitH (GGM tree)	
	Traditional (ms)	Hypercube (ms)	TCitH (ms)	Saving
Party emulations / repetition	N	$1 + \log_2 N$	$1 + \left\lceil \frac{\log_2 N}{\log_2 \lvert \mathbb{F} \rvert} \right\rceil$	
AIMer	4.53	3.22	3.22	-0 %
Biscuit	17.71	4.65	4.24	-16 %
MIRA	384.26	20.11	9.89	-51 %
MiRitH-Ia	54.15	6.60	5.42	-18 %
MiRitH-Ib	89.50	8.66	6.66	-23 %
MQOM-31	96.41	11.27	8.74	-21 %
MQOM-251	44.11	7.56	5.97	-21 %
RYDE	12.41	4.65	4.65	-0 %
SDitH-256	78.37	7.23	5.31	-27 %
SDitH-251	19.15	7.53	6.44	-14 %

- Comparison based on a generic MPCitH library (libmpcith)
- Code for MPC protocols fetched from the submission packages

Speedups for MPCitH candidates

	A... MPCitH		TCitH (GGM tree)	
				...ving
Party em... / repe...				
AIM...				... %
Bisc...				...5 %
MIR...				...1 %
MiRit...				...3 %
MiRit...				...3 %
MQOI...				...1 %
MQOM...				...1 %
RYD...				... %
SDitH-...				...7 %
SDitH-251	19.15	7.53	6.44	-14 %

🤔 Nice speedups, but could we also improve the sizes?

☝️ Let's use the multiplication homomorphism

- Comparison based on a generic MPCitH library (libmpcith)
- Code for MPC protocols fetched from the submission packages

Using multiplication homomorphism

- Shamir's secret sharing satisfies:

$$[\![x]\!]^{(d)} \cdot [\![y]\!]^{(d)} = [\![x \cdot y]\!]^{(2d)}$$

Using multiplication homomorphism

- Shamir's secret sharing satisfies:

$$[\![x]\!]^{(d)} \cdot [\![y]\!]^{(d)} = [\![x \cdot y]\!]^{(2d)}$$

- Simple protocol to verify polynomial constraints
  - w valid $\Leftrightarrow f_1(w) = 0, \ \ldots, f_m(w) = 0$
  - parties locally compute

$$[\![\alpha]\!] = [\![v]\!] + \sum_{j=1}^{m} \gamma_j \cdot f_j([\![w]\!])$$

Using multiplication homomorphism

- Shamir's secret sharing satisfies:

$$[\![x]\!]^{(d)} \cdot [\![y]\!]^{(d)} = [\![x \cdot y]\!]^{(2d)}$$

- Simple protocol to verify polynomial constraints
  - w valid $\Leftrightarrow f_1(w) = 0, \ \ldots, f_m(w)$
  - parties locally compute

$$[\![\alpha]\!] = [\![v]\!] + \sum_{j=1}^{m} \gamma_j \, f_j([\![w]\!])$$

randomness
from the verifier

Using multiplication homomorphism

- Shamir's secret sharing satisfies:

$$[\![x]\!]^{(d)} \cdot [\![y]\!]^{(d)} = [\![x \cdot y]\!]^{(2d)}$$

- Simple protocol to verify polynomial constraints
  - w valid $\Leftrightarrow f_1(w) = 0, \ \ldots, f_m(w)$
  - parties locally compute

$$[\![\alpha]\!] = [\![v]\!] + \sum_{j=1}^{m} \gamma_j \, f_j([\![w]\!])$$

pre-committed sharing of 0

randomness from the verifier

Using multiplication homomorphism

- Shamir's secret sharing satisfies:

$$[\![x]\!]^{(d)} \cdot [\![y]\!]^{(d)} = [\![x \cdot y]\!]^{(2d)}$$

- Simple protocol to verify polynomial constraints
  - w valid $\Leftrightarrow f_1(w) = 0, \ \ldots, f_m(w)$
  - parties locally compute

$$[\![\alpha]\!] = [\![v]\!] + \sum_{j=1}^{m} \gamma_j \cdot f_j([\![w]\!])$$

check $\alpha = 0$
false positive proba $1/|\mathbb{F}|$

pre-committed
sharing of 0

randomness
from the verifier

Using multiplication homomorphism

- Shamir's secret sharing satisfies:

$$[\![x]\!]^{(d)} \cdot [\![y]\!]^{(d)} = [\![x \cdot y]\!]^{(2d)}$$

- Simple protocol to verify polynomial constraints
  - w valid $\Leftrightarrow f_1(w) = 0, \ \ldots, f_m(w)$
  - parties locally compute

$$[\![\alpha]\!] = [\![v]\!] + \sum_{j=1}^{m} \gamma_j \cdot f_j([\![w]\!])$$

- Tweaking MPCitH-based candidates $\Rightarrow$ smaller signatures

Shorter signatures for MPCitH-based candidates

	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 %

MQ over GF(4)	8 609 B	3 858 B	-55 %
SD over GF(2)	11 160 B	7 354 B	-34 %
SD over GF(2)	12 066 B	6 974 B	-42 %

* $N = 256$

Shorter signatures for MPCitH-based candidates

	Original Size	*Our Variant*	*Saving*
Biscuit	4 758 B	3 431 B	
MIRA	5 640 B	4 314 B	
MiRitH-Ia	5 665 B	3 873 B	
MiRitH-Ib	6 298 B	4 250 B	
MQOM-31	6 328 B	3 567 B	
MQOM-251	6 575 B	3 418 B	
RYDE	5 956 B	4 274 B	
SDitH	8 241 B	5 673 B	
MQ over GF(4)	8 609 B	3 301 B	
SD over GF(2)	11 160 B	7 354 B	-34 %
SD over GF(2)	12 066 B	6 974 B	-42 %

* $N = 256$ * $N = 2048$

Shorter signatures for MPCitH-based candidates

Two very recent works :

- Baum, Beullens, Mukherjee, Orsini, Ramacher, Rechberger, Roy, Scholl. *One Tree to Rule Them All: Optimizing GGM Trees and OWFs for Post-Quantum Signatures.* https://ia.cr/2024/490
  - General techniques to reduce the size of GGM trees
  - **Apply to TCitH-GGM** (gain of ~500 B at 128-bit security)

- Bidoux, Feneuil, Gaborit, Neveu, Rivain. *Dual Support Decomposition in the Head: Shorter Signatures from Rank SD and MinRank.* https://ia.cr/2024/541
  - New MPC protocols for TCitH / VOLEitH signatures based on **MinRank & Rank SD**

Other results

- Improvements for TCitH-MT
    - Degree-enforcing commitment scheme
    - Packed secret sharing
- Other applications
    - Post-quantum ring signatures
        - For any one-way function
        - $|\sigma| \leq$ 10 kB (~ 5 kB with MQ) for $|\text{ring}| = 2^{20}$
    - ZKP for lattices
        - Smallest with MPCitH paradigm
        - Competitive to lattice-based ZKP
    - Improvement of Ligero for general arithmetic circuits
- Connections to VOLEitH and Ligero proof systems

Thank you for listening 🙏

Original TCitH framework (Asiacrypt'23)

Improved TCitH framework (preprint)

References

[AGHJY23] Aguilar Melchor, Gama, Howe, Hülsing, Joseph, Yue: “The Return of the SDitH” (EUROCRYPT 2023)

[BBMORRRS24] Baum, Beullens, Mukherjee, Orsini, Ramacher, Rechberger, Roy, Scholl: “One Tree to Rule Them All: Optimizing GGM Trees and OWFs for Post-Quantum Signatures” https://ia.cr/2024/490

[BFGNR24] Bidoux, Feneuil, Gaborit, Neveu, Rivain. "Dual Support Decomposition in the Head: Shorter Signatures from Rank SD and MinRank” https://ia.cr/2024/541

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

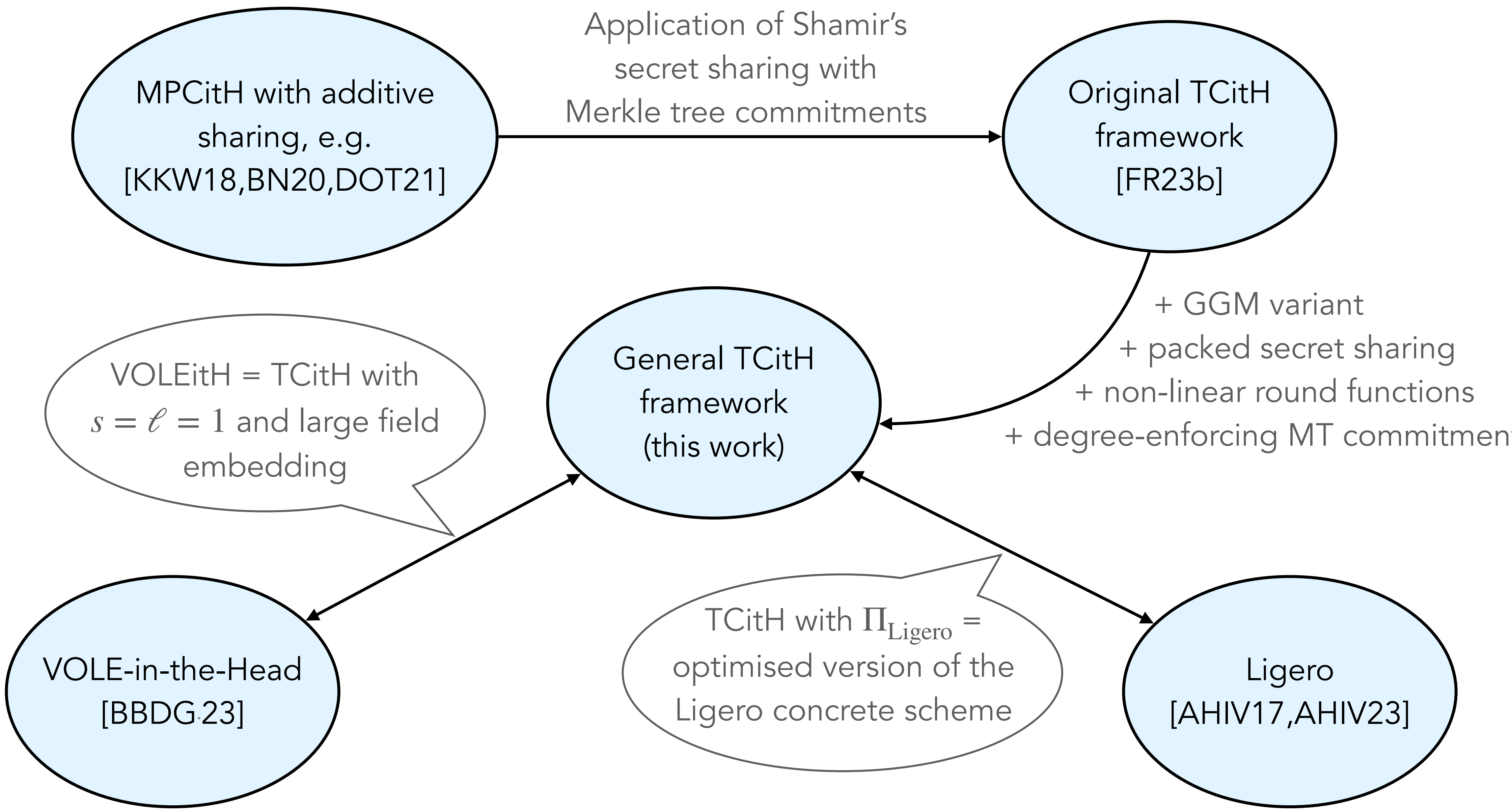
[FR22] Thibauld Feneuil, Matthieu Rivain: "Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head" https://ia.cr/2022/1407 (ASIACRYPT 2023)

[FR23] Thibauld Feneuil, Matthieu Rivain: "Threshold Computation in the Head: Improved Framework for Post-Quantum Signatures and Zero-Knowledge Arguments" https://ia.cr/2023/1573

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

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

Connections to other proof systems



$N = 256$

	TCitH-GGM		VOLEitH	
	Size	Comput. Field	Size	Computat. Field
AIMer [CCH+23]	4 352 B	19× $GF(2^8)$	3 938 B	$GF(2^{128})$
Biscuit [BKPV23]	4 048 B	19× $GF(16^2)$	3 682 B	$GF(16^{2\times16})$
MIRA [ABB+23d]	5 340 B	19× $GF(16^2)$	4 770 B	$GF(16^{2\times16})$
MiRitH-Ia [ABB+23b]	4 694 B	19× $GF(16^2)$	4 226 B	$GF(16^{2\times16})$
MiRitH-Ib [ABB+23b]	5 245 B	19× $GF(16^2)$	4 690 B	$GF(16^{2\times16})$
MQOM (over $\mathbb{F}_{251}$) [FR23a]	4 257 B	19× $GF(251)$	3 858 B	$GF(251^{16})$
MQOM (over $\mathbb{F}_{31}$) [FR23a]	4 027 B	19× $GF(31^2)$	3 660 B	$GF(31^{2\times16})$
RYDE [ABB+23c]	5 281 B	19× $GF(2^8)$ 19× $GF(2^{31})$	4 720 B	$GF(2^{128})$
SDitH (over $\mathbb{F}_{251}$) [AFG+23]	7 335 B	19× $GF(251)$	6 450 B	$GF(251^{16})$
SDitH (over $\mathbb{F}_{256}$) [AFG+23]	7 335 B	19× $GF(256)$	6 450 B	$GF(256^{16})$

$N = 2048$

	TCitH-GGM		VOLEitH	
	Size	Comput. Field	Size	Computat. Field
AIMer [CCH+23]	3 639 B	13× $GF(2^{11})$	3 546 B	$GF(2^{128})$
Biscuit [BKPV23]	3 431 B	13× $GF(16^3)$	3 354 B	$GF(16^{3\times12})$
MIRA [ABB+23d]	4 314 B	13× $GF(16^3)$	4 170 B	$GF(16^{3\times12})$
MiRitH-Ia [ABB+23b]	3 873 B	13× $GF(16^3)$	3 762 B	$GF(16^{3\times12})$
MiRitH-Ib [ABB+23b]	4 250 B	13× $GF(16^3)$	4 110 B	$GF(16^{3\times12})$
MQOM (over $\mathbb{F}_{251}$) [FR23a]	3 567 B	13× $GF(251^2)$	3 486 B	$GF(251^{2\times12})$
MQOM (over $\mathbb{F}_{31}$) [FR23a]	3 418 B	13× $GF(31^3)$	3 338 B	$GF(31^{3\times12})$
RYDE [ABB+23c]	4 274 B	13× $GF(2^{11})$ 13× $GF(2^{31})$	4 133 B	$GF(2^{128})$
SDitH (over $\mathbb{F}_{251}$) [AFG+23]	5 673 B	13× $GF(251^2)$	5 430 B	$GF(251^{2\times12})$
SDitH (over $\mathbb{F}_{256}$) [AFG+23]	5 673 B	13× $GF(256^2)$	5 430 B	$GF(256^{2\times12})$