Threshold Computation in the Head

Matthieu Rivain

New Trends in PQC Workshop

Oxford, 11 June, 2024

CryptoExperts

WE INNOVATE TO SECURE YOUR BUSINESS

Threshold Computation in the Head

Joint work with Thibauld Feneuil

https://ia.cr/2022/1407

Original TCitH
framework
(Asiacrypt'23)

https://ia.cr/2023/1573

Improved TCitH
framework
(preprint)

Roadmap

- MPC-in-the-Head paradigm
- TC-in-the-Head framework (and application to PQ signatures)
    - 🌲 TCitH with Merkle trees
    - 🌲 TCitH with GGM trees
    - ✖ TCitH using multiplication homomorphism
    - 📦 TCitH using packed secret sharing
- Application: post-quantum ring signatures
- Relation to other proof systems

MPC-in-the-Head paradigm

MPC-in-the-Head paradigm

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

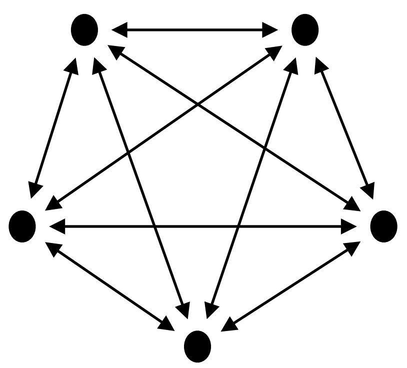
MPC-in-the-Head paradigm

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

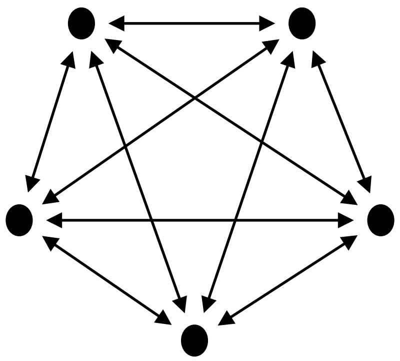
MPC-in-the-Head paradigm

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

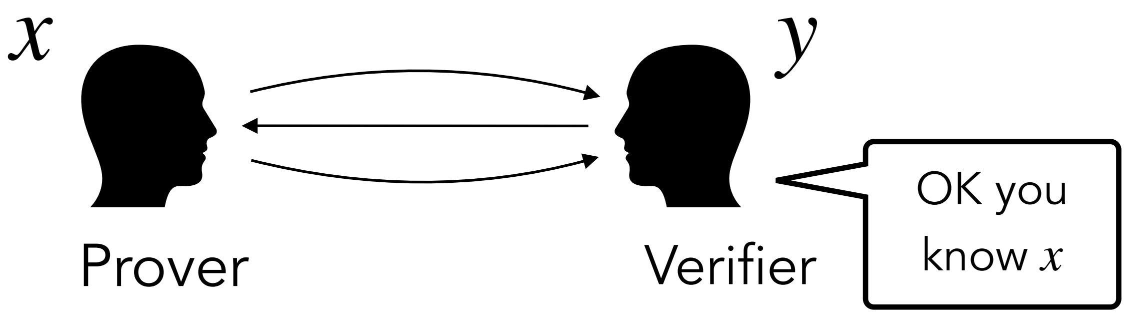


Input sharing $[\![x]\!]$

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



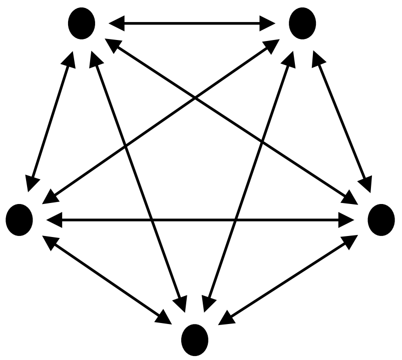
MPC-in-the-Head paradigm

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

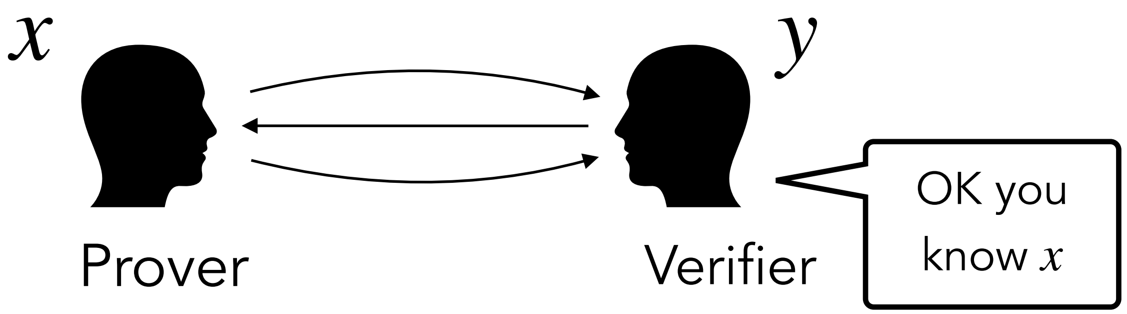


Input sharing $[\![x]\!]$

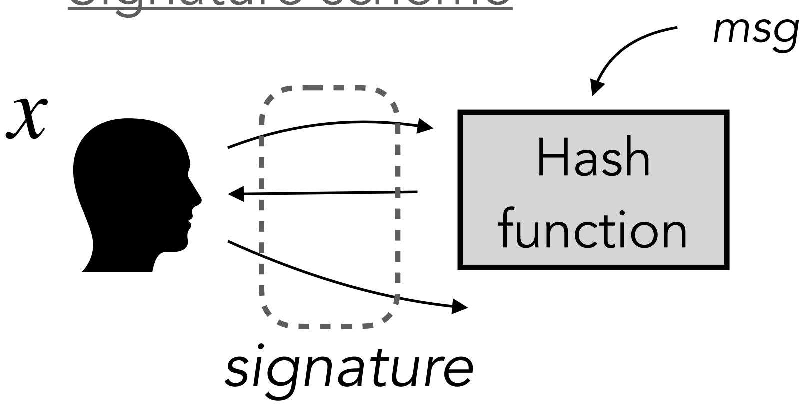
Joint evaluation of:

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

Zero-knowledge proof



Signature scheme



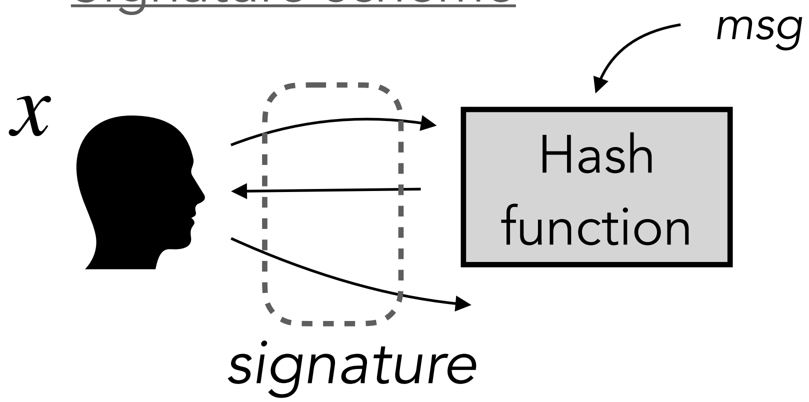
MPC-in-the-Head paradigm

One-way function

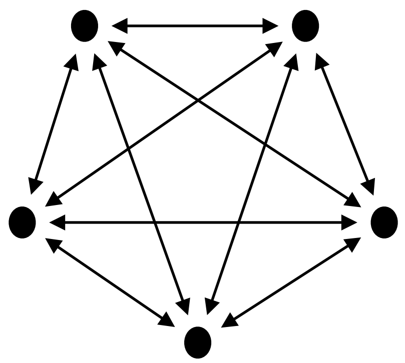
$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Signature scheme



Multiparty computation (MPC)



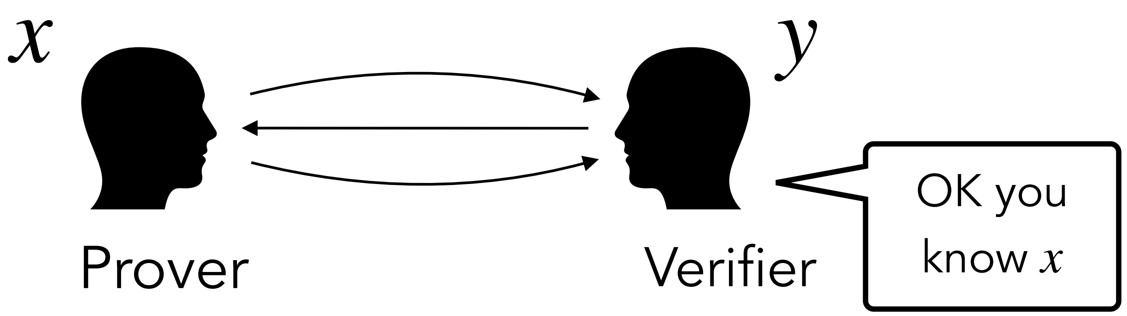
Input sharing $[\![x]\!]$

Joint evaluation of:

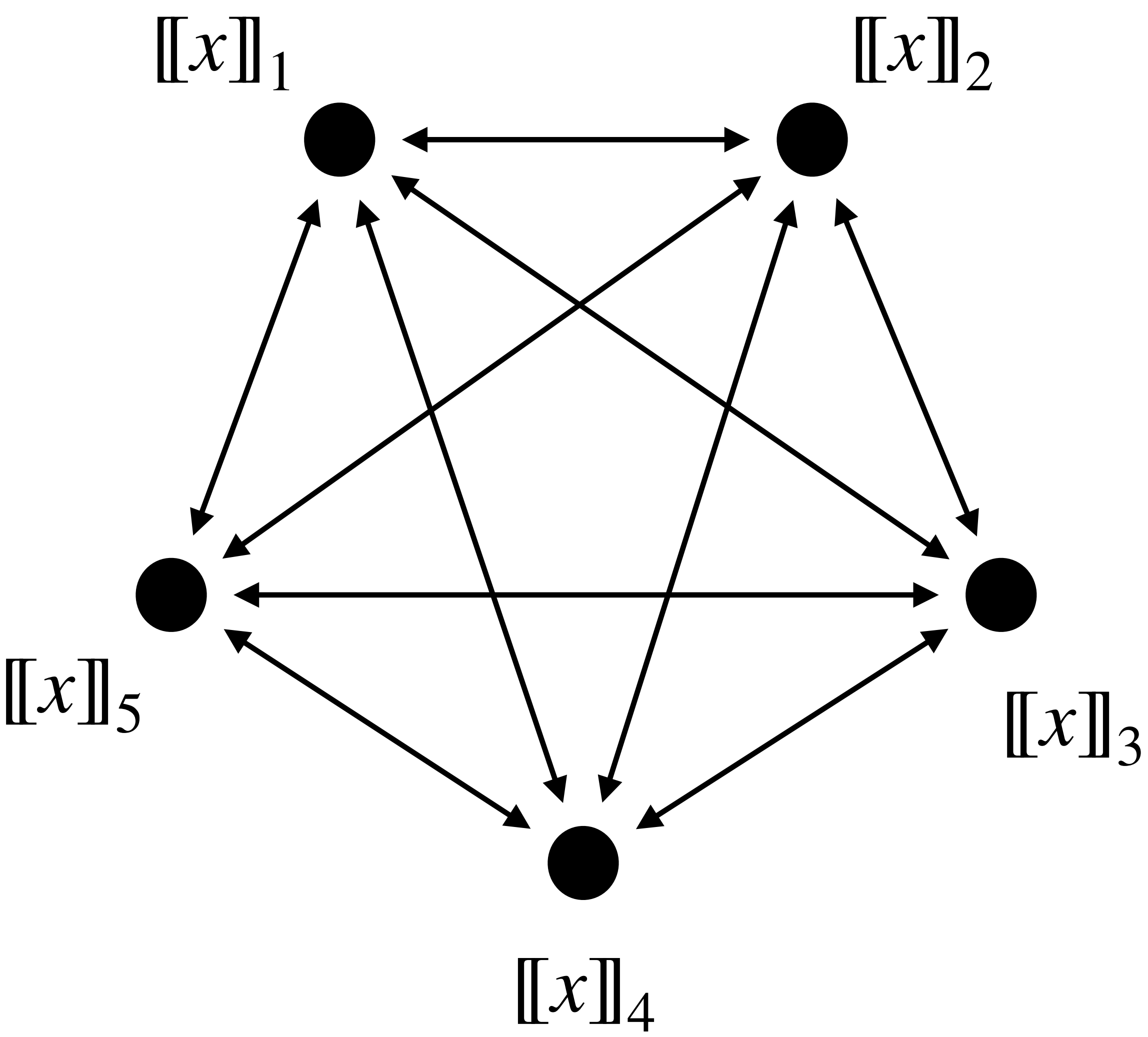
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

MPC-in-the-Head transform

Zero-knowledge proof



MPC model



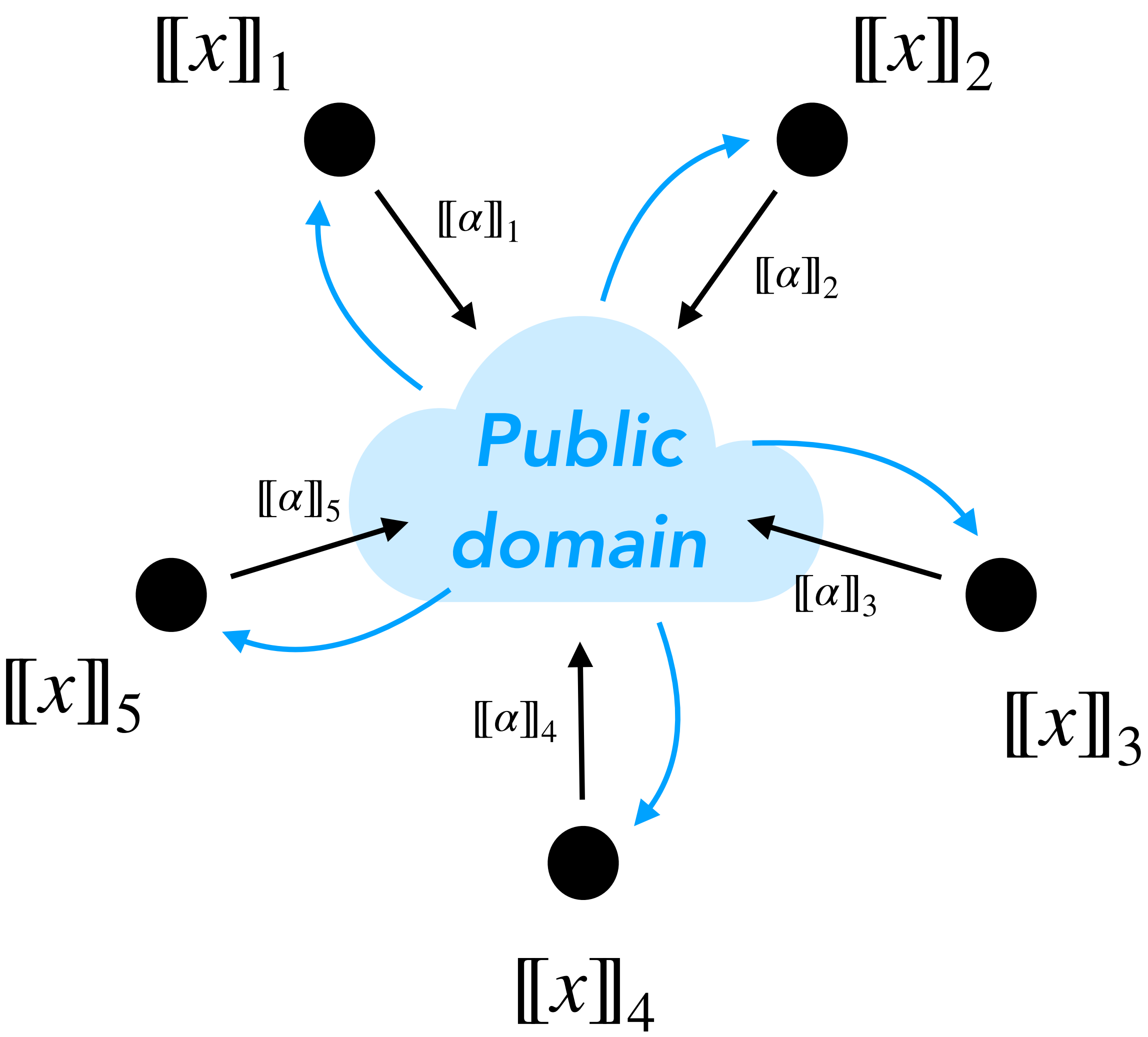
$[\![x]\!]$ is a linear secret sharing of x

- 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



$[\![x]\!]$ 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
- *Broadcast model*

MPCitH transform

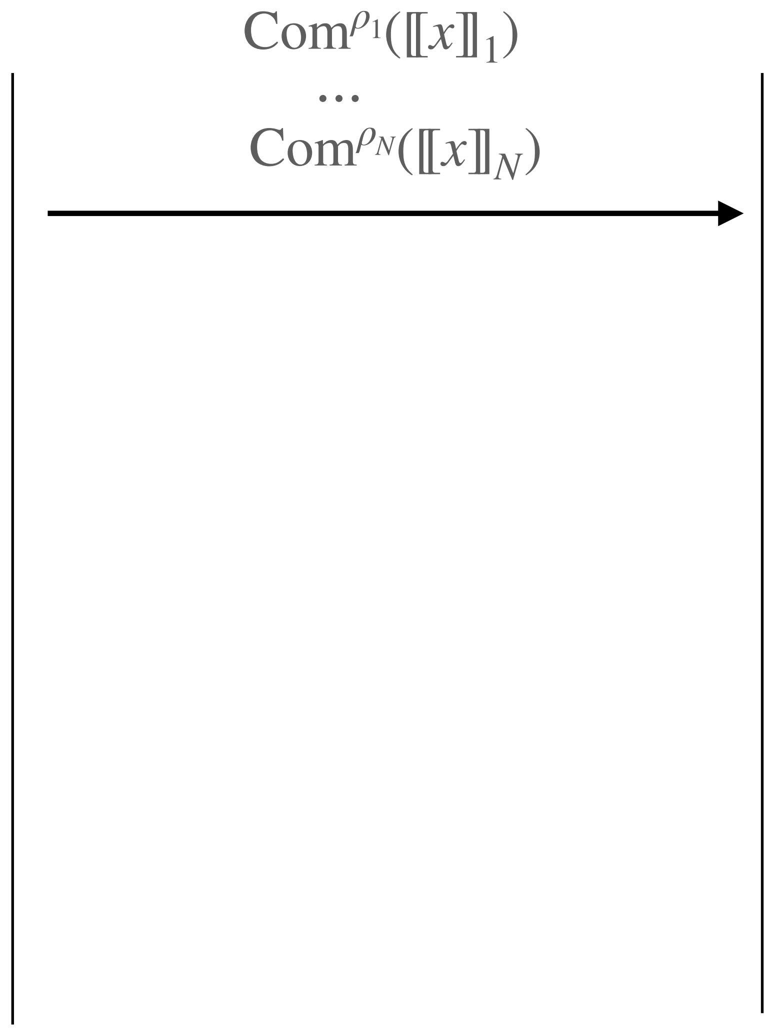
Prover

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

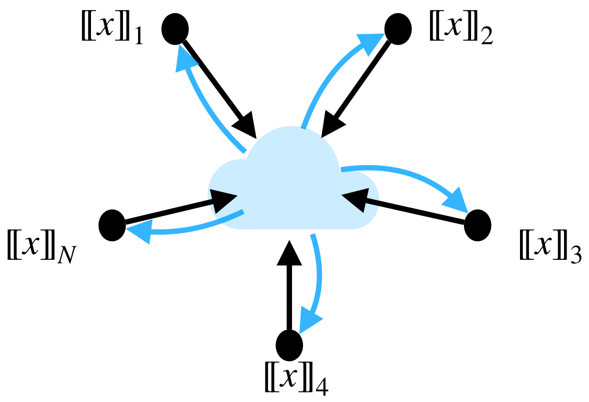
Verifier

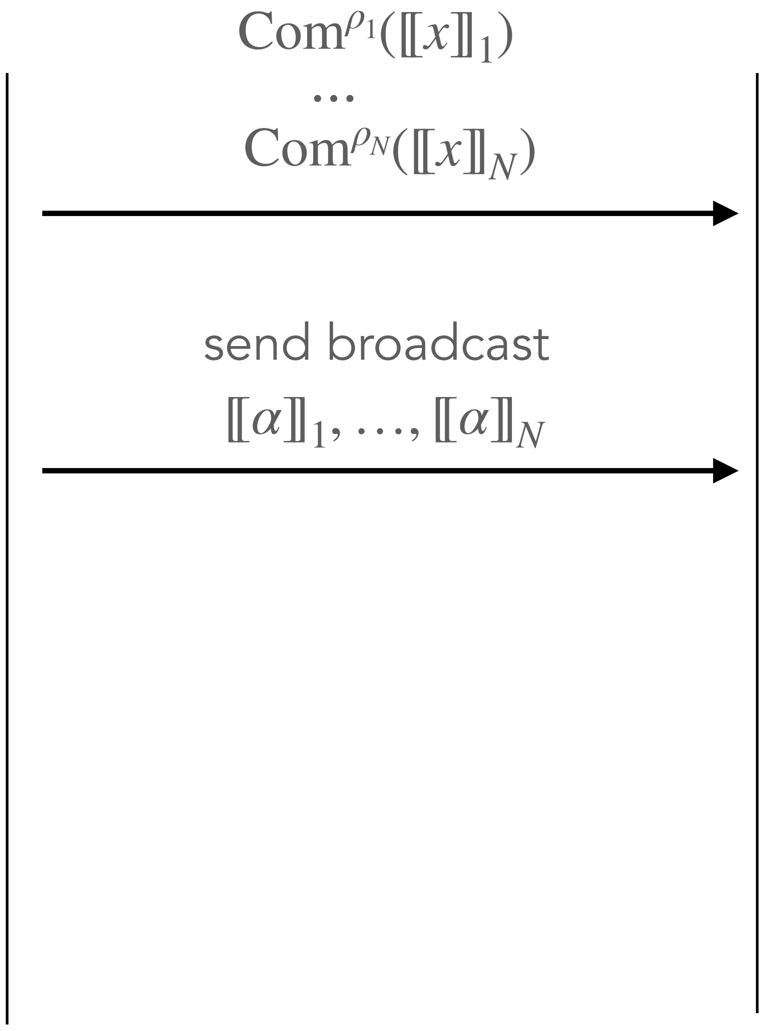
MPCitH transform

① Generate and commit shares

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

② Run MPC in their head





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

…

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

send broadcast

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

Prover

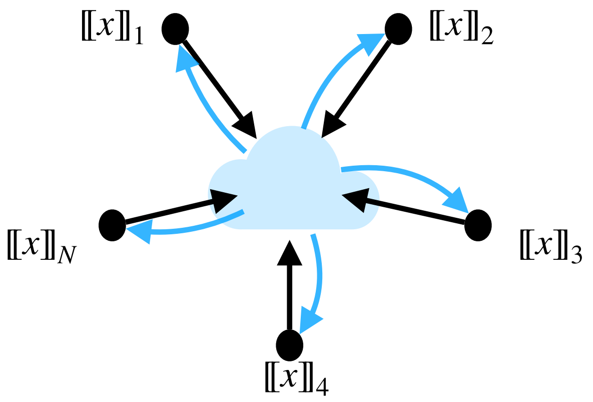
Verifier

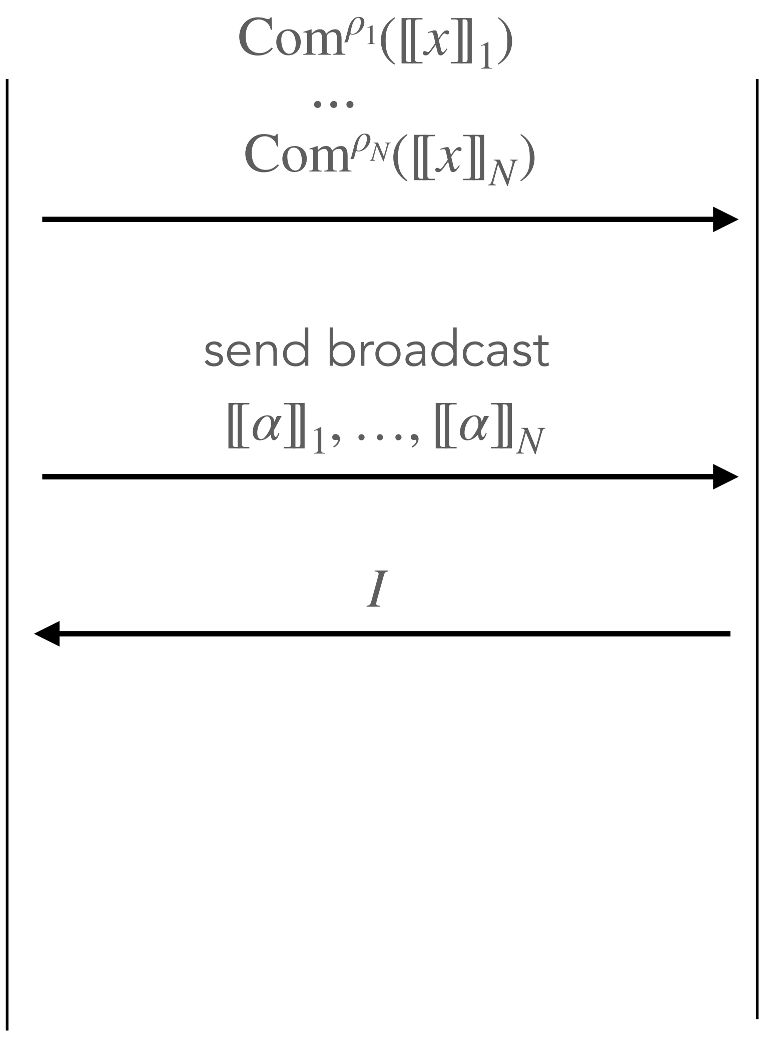
MPCitH transform

① Generate and commit shares

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

② Run MPC in their head





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

$\cdots$

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

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

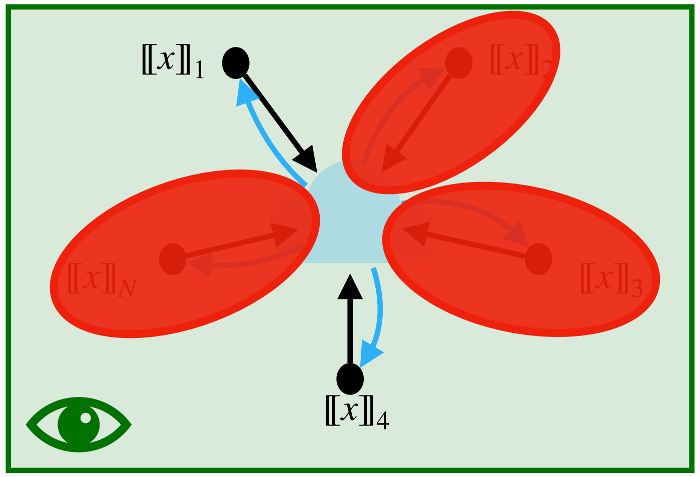
Verifier

MPCitH transform

① Generate and commit shares

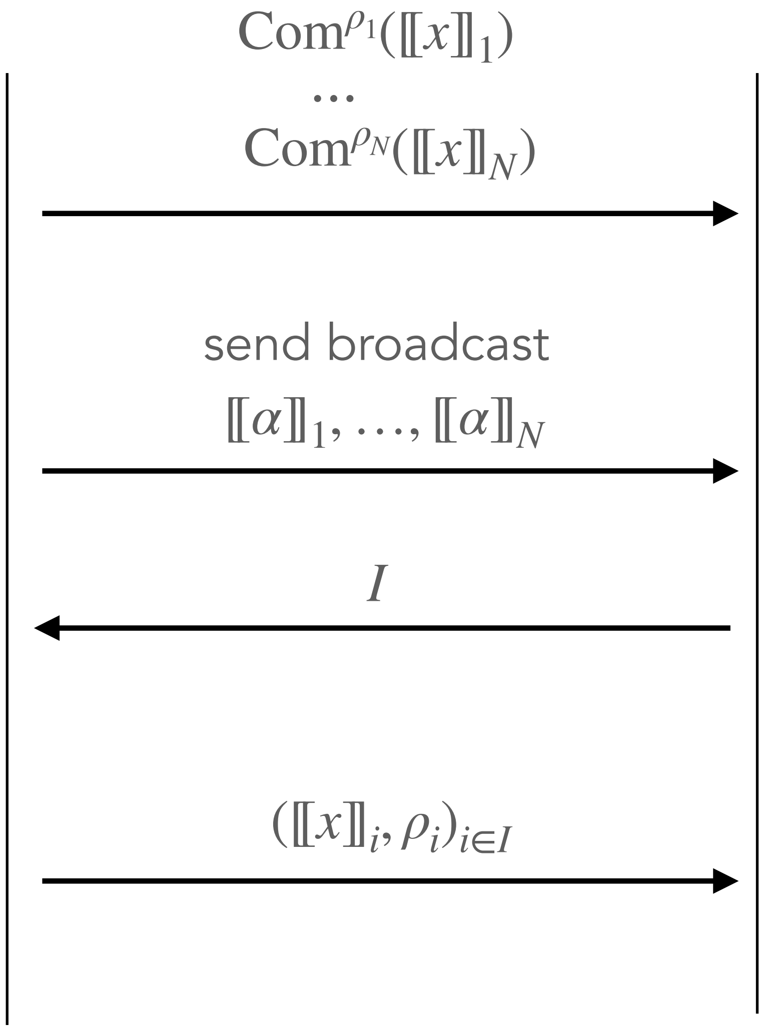
$\llbracket x \rrbracket = (\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$

② Run MPC in their head



$\llbracket x \rrbracket_1$
$\llbracket x \rrbracket_2$
$\llbracket x \rrbracket_N$
$\llbracket x \rrbracket_3$
$\llbracket x \rrbracket_4$

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. $|I| = \ell$.

Prover

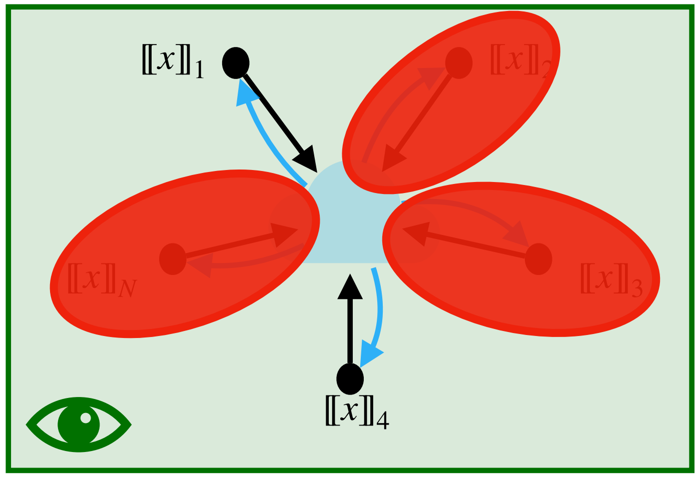
Verifier

MPCitH transform

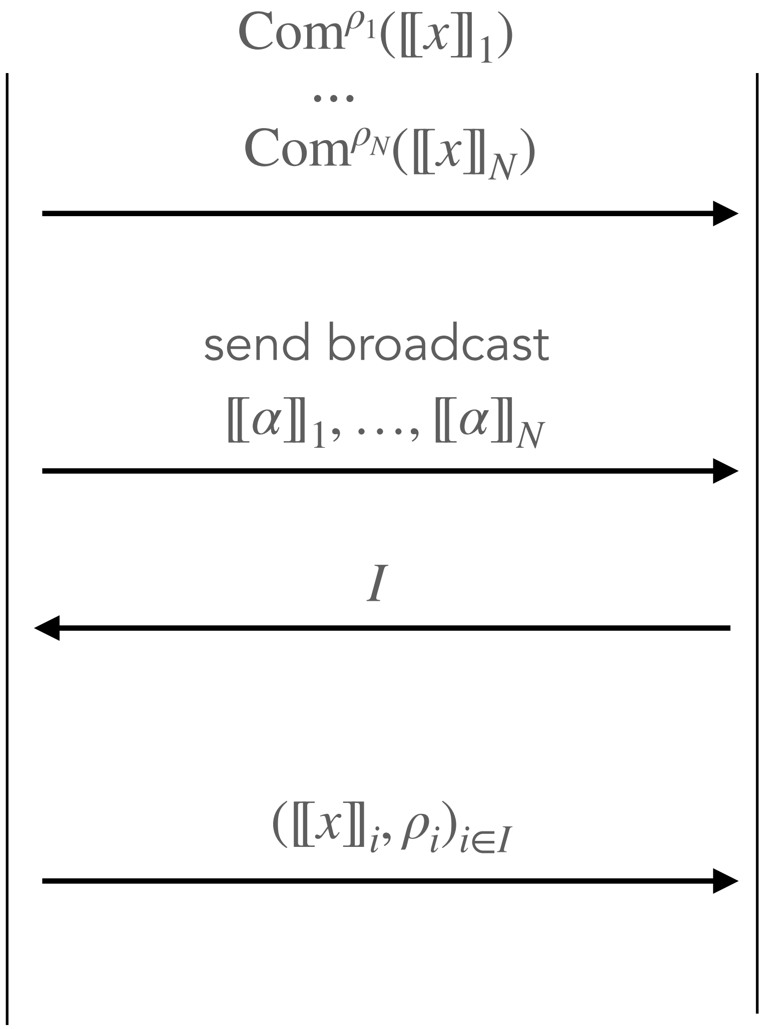
① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I



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

…

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

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 additive sharing

① Generate and commit shares

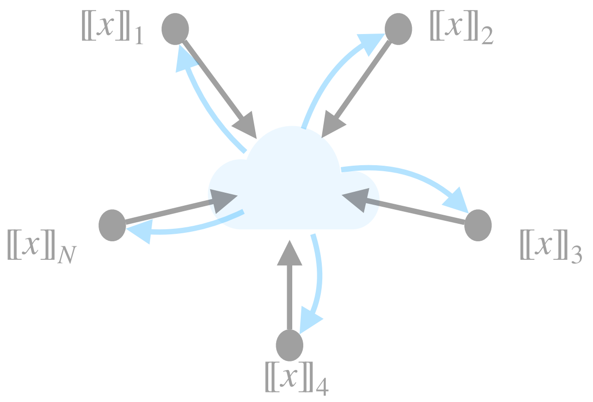
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

$\cdots$

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

② Run MPC in their head



send broadcast

Additive sharing:

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

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

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

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

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

$\dots$

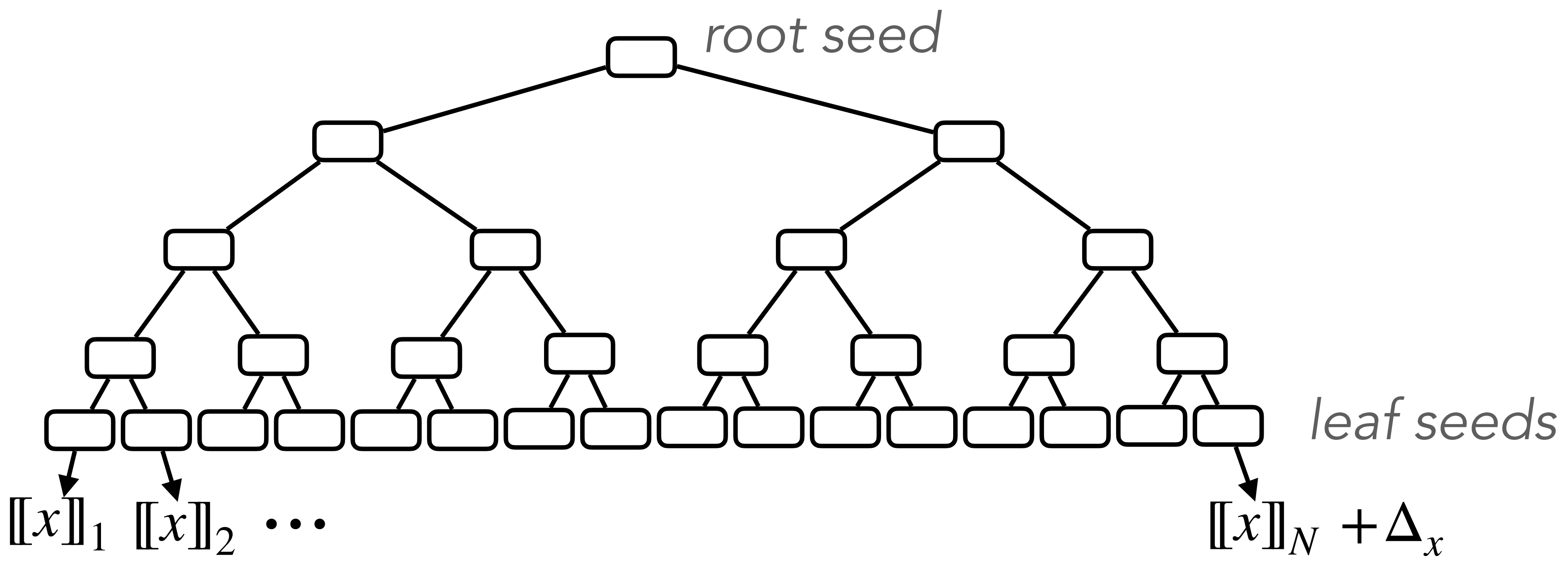
$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

② Run MPC in their head

es

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

Generated using a GGM seed tree [KKW18]:

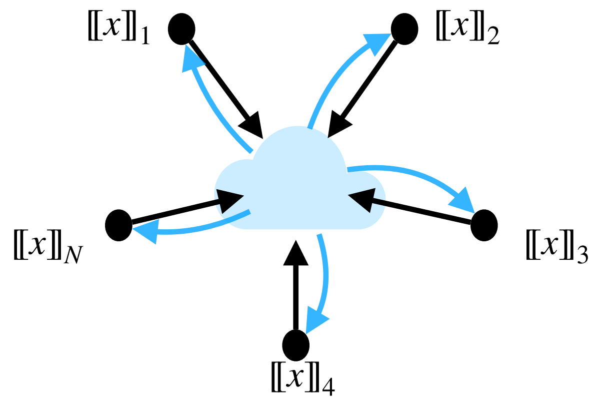


MPCitH transform: with additive sharing

① Generate and commit shares

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

② Run MPC in thei 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

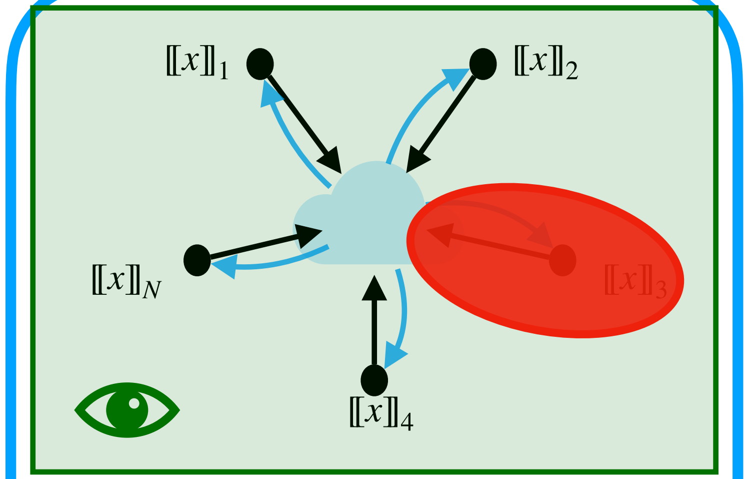
Verifier

MPCitH transform: with additive sharing

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Sharing / MPC protocol

$(N-1)$-*private*

parties

ℓ.

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

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

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

Prover

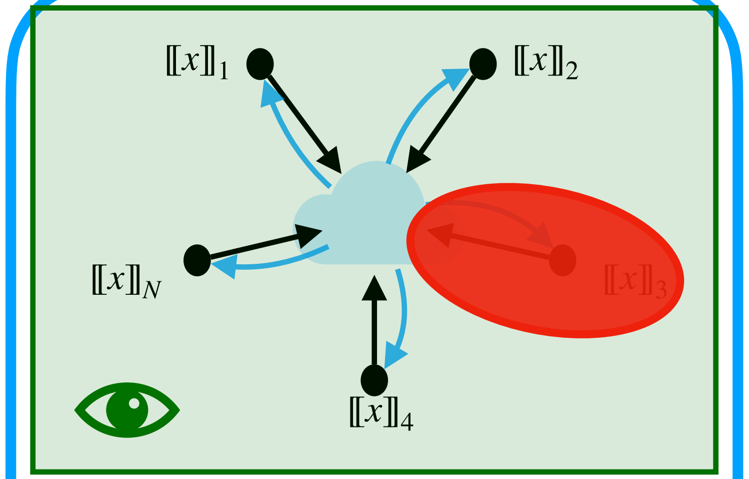
Verifier

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

Sharing / MPC protocol

$(N-1)$-*private*

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

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

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

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

$\cdots$

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

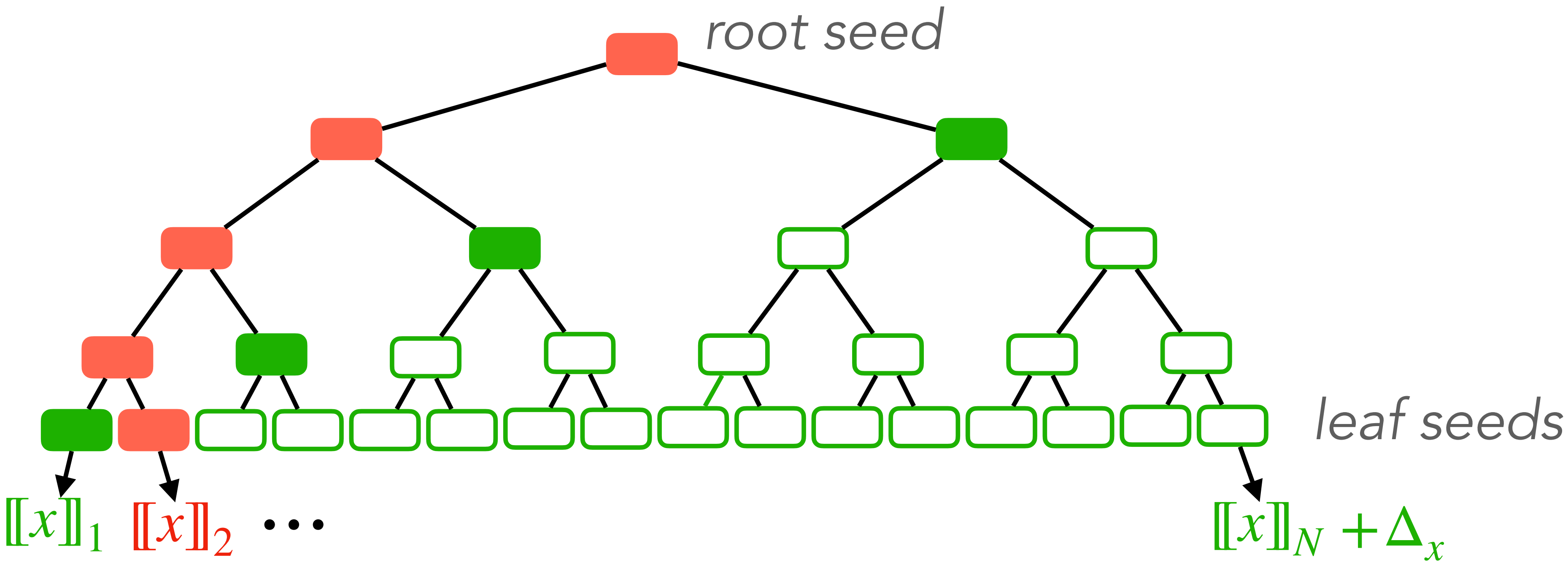
es

ties

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

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

Only $\log_2 N$ seeds to be revealed:



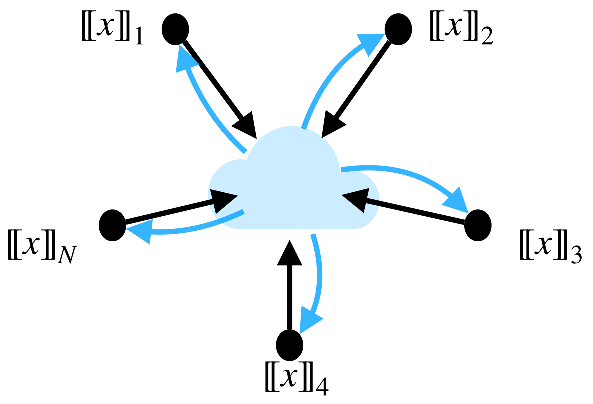
TC-in-the-Head framework (with Merkle trees)

Threshold Computation in the Head

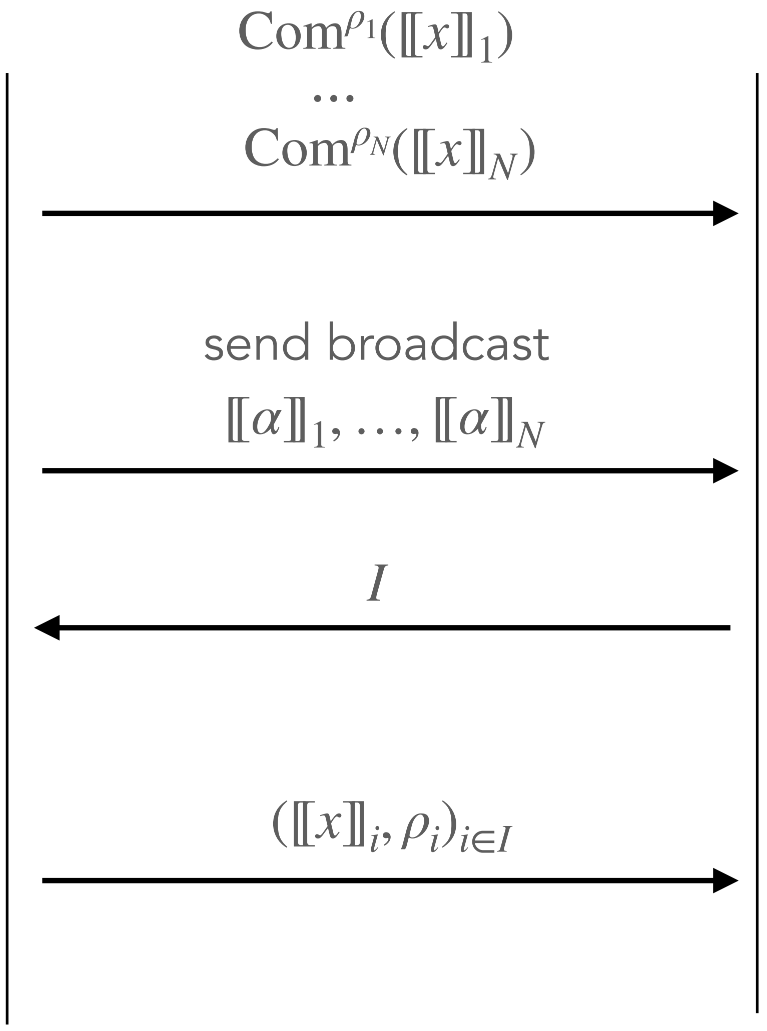
① Generate and commit shares

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

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

Threshold Computation in the Head

① Generate and commit shares

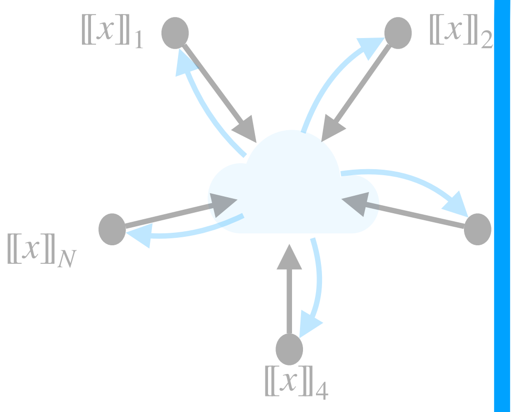
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

$\cdots$

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

② Run MPC in their hea



Shamir secret sharing:

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

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

n set of parties

t. $|I| = \ell$.

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

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

ccept

④ Open parties in I

Prover

Verifier

Threshold Computation in the Head

① Generate and commit shares

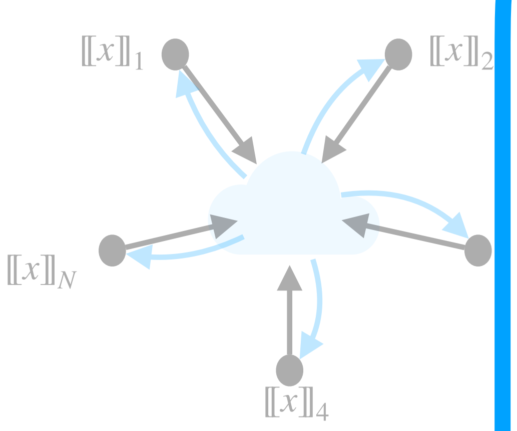
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

$\cdots$

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

② Run MPC in their hea



④ Open parties in I

Shamir secret sharing:

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

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

$\Rightarrow \ell$-privacy

n set of parties

t. $|I| = \ell$.

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

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

ccept

Prover

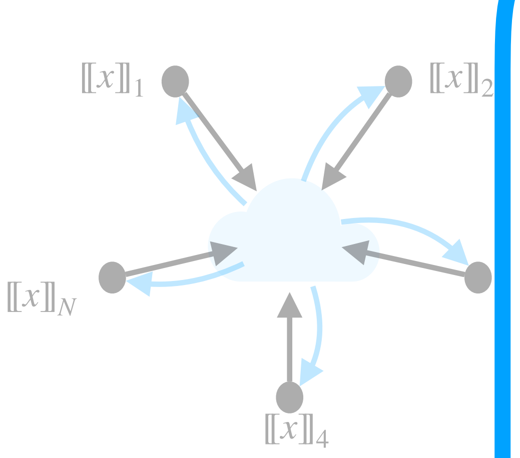
Verifier

Threshold Computation in the Head

① Generate and commit shares
$\llbracket x \rrbracket = (\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$

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

② Run MPC in their hea



$\llbracket x \rrbracket_1$
$\llbracket x \rrbracket_2$
$\llbracket x \rrbracket_N$
$\llbracket x \rrbracket_4$

④ Open parties in I

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

$\mathrm{Com}^{\rho_i}(\llbracket x \rrbracket_i)$
ion $\llbracket \alpha \rrbracket_i = \varphi(\llbracket x \rrbracket_i)$
ccept

Shamir secret sharing:

$$\llbracket x \rrbracket_i := P(e_i) \quad \forall i$$

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

$\Rightarrow \ell$-privacy

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

Prover

Verifier

Threshold Computation in the Head

① Generate and commit shares

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

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

$\ldots$

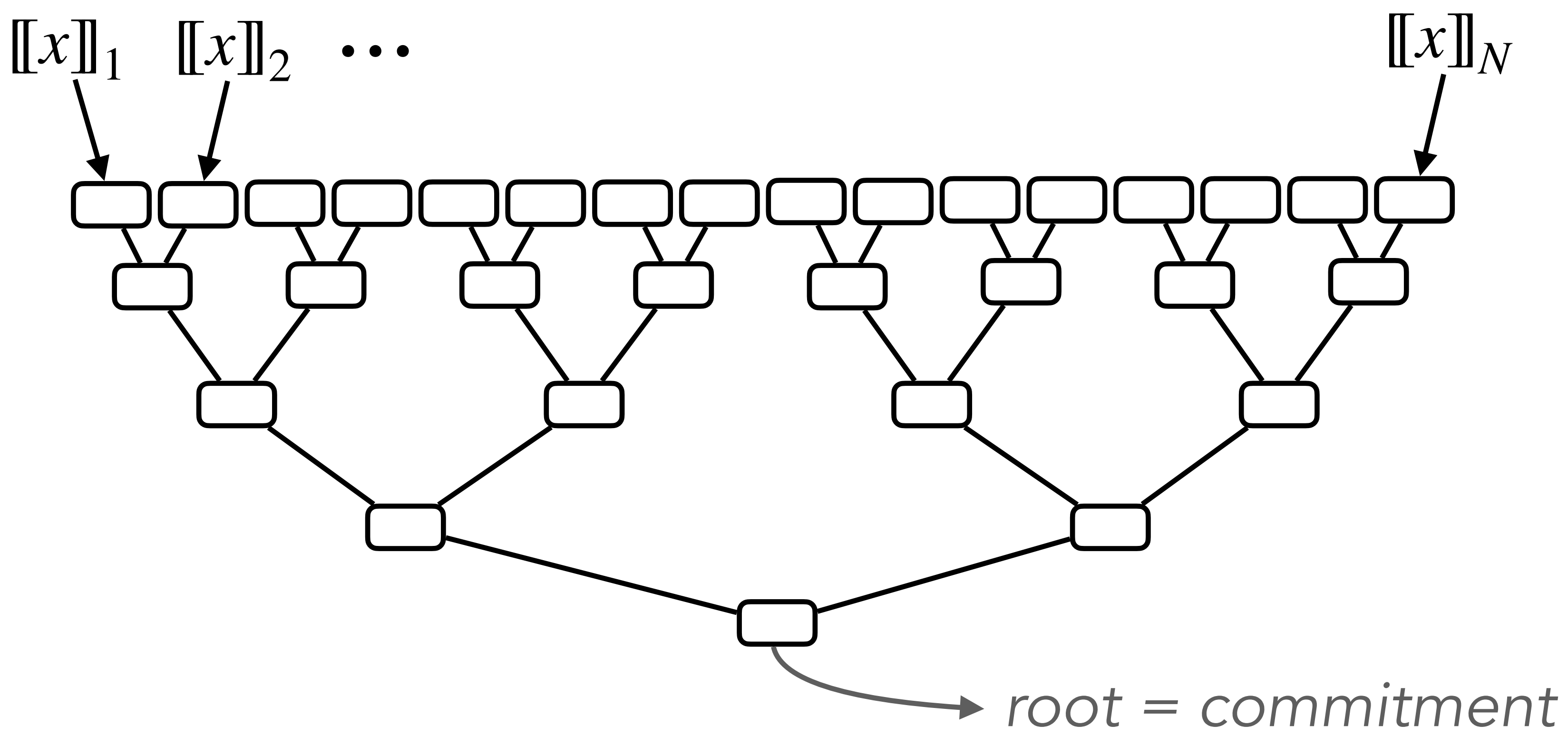
$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$

② Run MPC in their head

es

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

Committed using a Merkle tree:

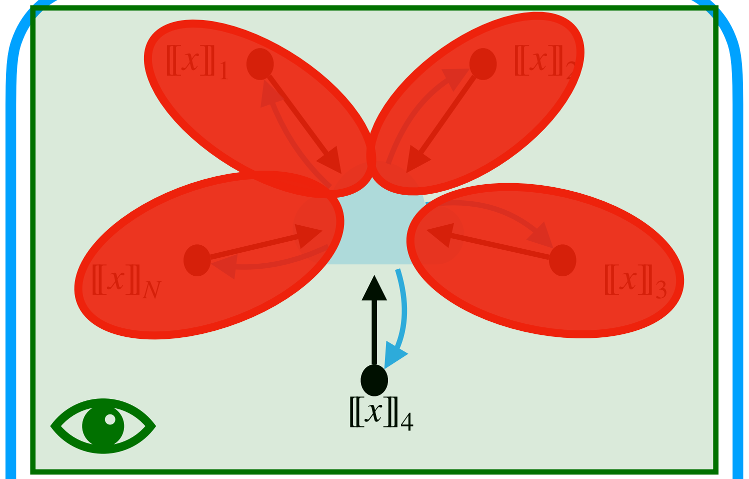


Threshold Computation in the Head

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*

parties

ℓ.

$]\!]_i)$

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

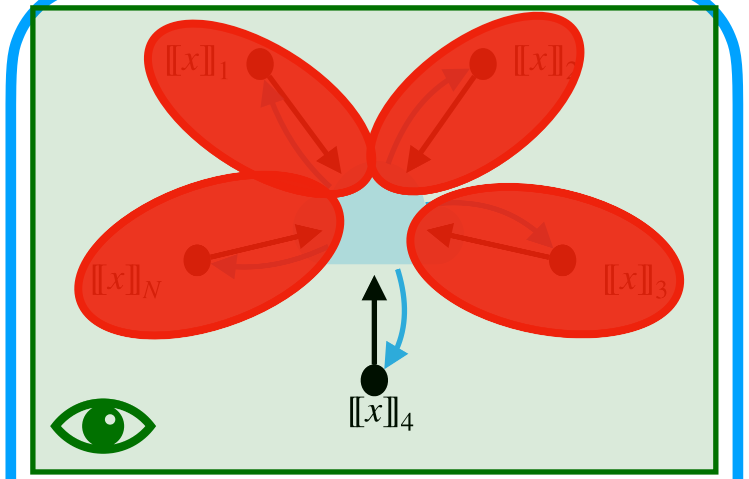
Verifier

Threshold Computation in the Head

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*

⇒ soundness error = $(N - \ell)/N$ 🤔

parties

ℓ.

$]\!]_i)$

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

Prover

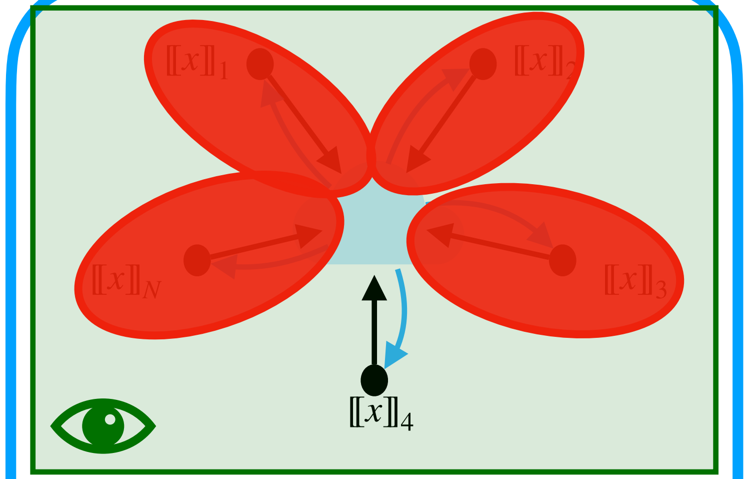
Verifier

Threshold Computation in the Head

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*

⇒ soundness error = $(N - \ell)/N$ 🤔

💡 broadcast messages must be valid Shamir's sharings

parties

ℓ.

$]\!]_i)$

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

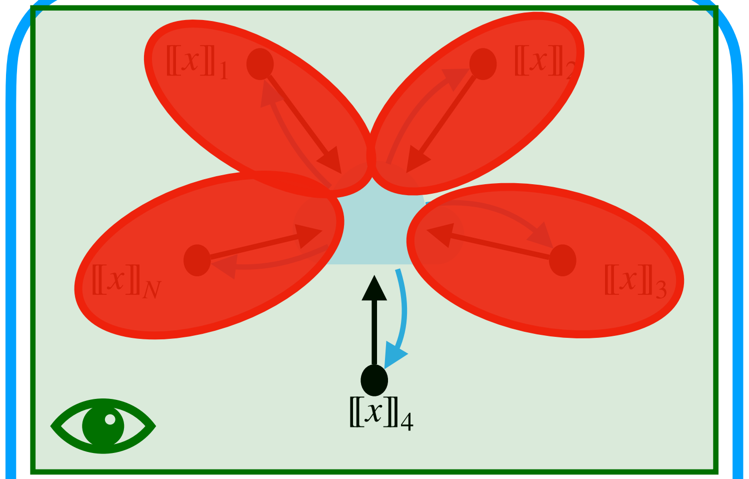
Verifier

Threshold Computation in the Head

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*

~~⇒ soundness error = $(N-\ell)/N$~~ 🤔

💡 broadcast messages must be valid Shamir's sharings

⇒ soundness error = $\dfrac{1}{\binom{N}{\ell}}$ 🤩

parties

ℓ.

$]\!]_i)$

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

Verifier

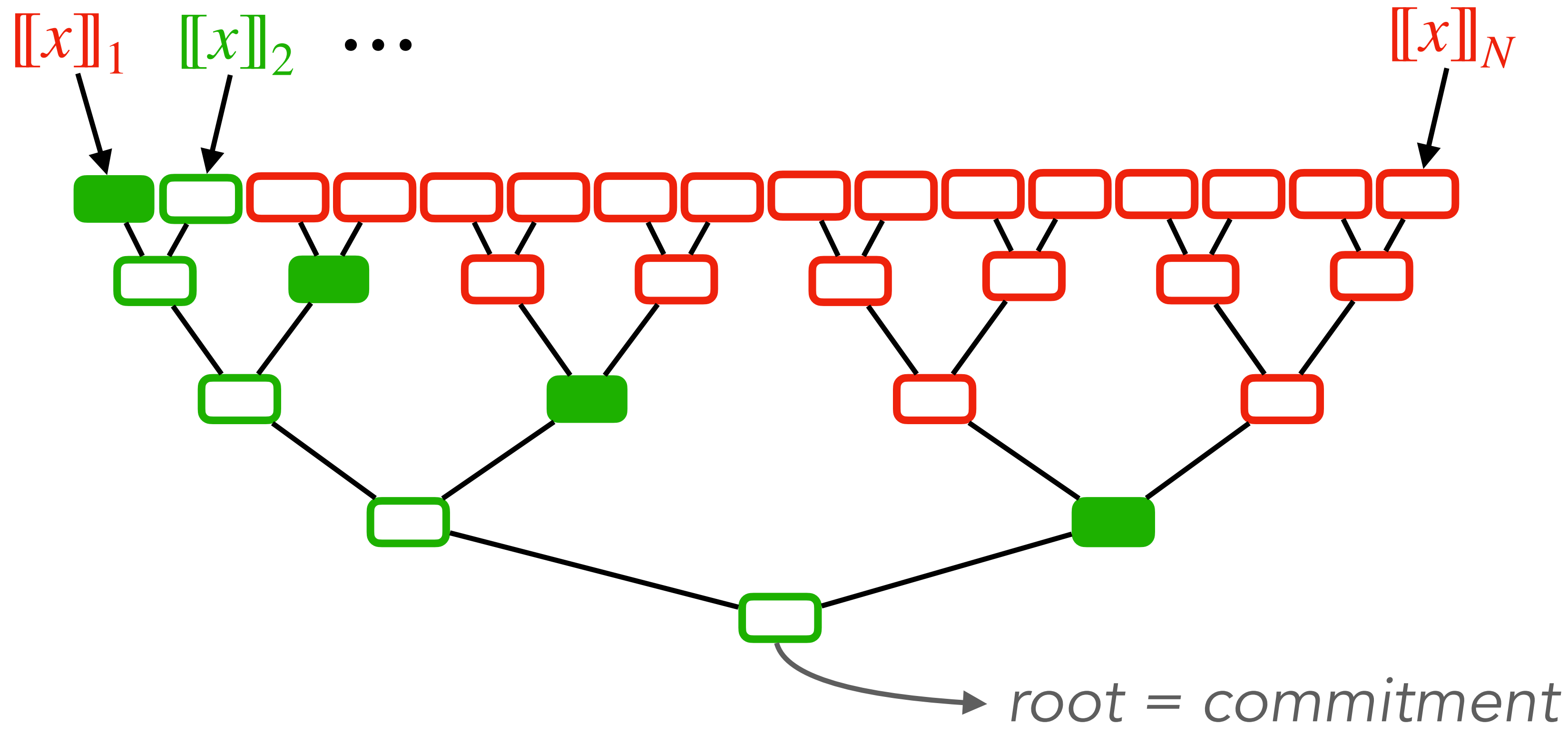
Threshold Computation in the Head

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

ties

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

Only $\log_2 N$ labels to be revealed:



Soundness

$$\begin{aligned} p &= \text{“false positive probability”} \\ &= P\left[\text{MPC protocol accepts } [\![x]\!] \text{ while } f(x) \neq y\right] \end{aligned}$$

Soundness

$$p = \text{“false positive probability”}$$

$$= P\big[\text{MPC protocol accepts } [\![x]\!] \text{ while } f(x) \neq y\big]$$

$$\boxed{\frac{1}{N} + p}$$

Soundness error of standard MPCitH

Soundness

$$\begin{aligned} p &= \text{“false positive probability”} \\ &= P\big[\text{MPC protocol accepts } [\![x]\!] \text{ while } f(x) \neq y\big] \end{aligned}$$

$$\begin{tikzcd} \boxed{\dfrac{1}{N} + p} \arrow[r, bend left, "\text{hope 🙏}"] & \boxed{\dfrac{1}{\binom{N}{\ell}} + p} \end{tikzcd}$$

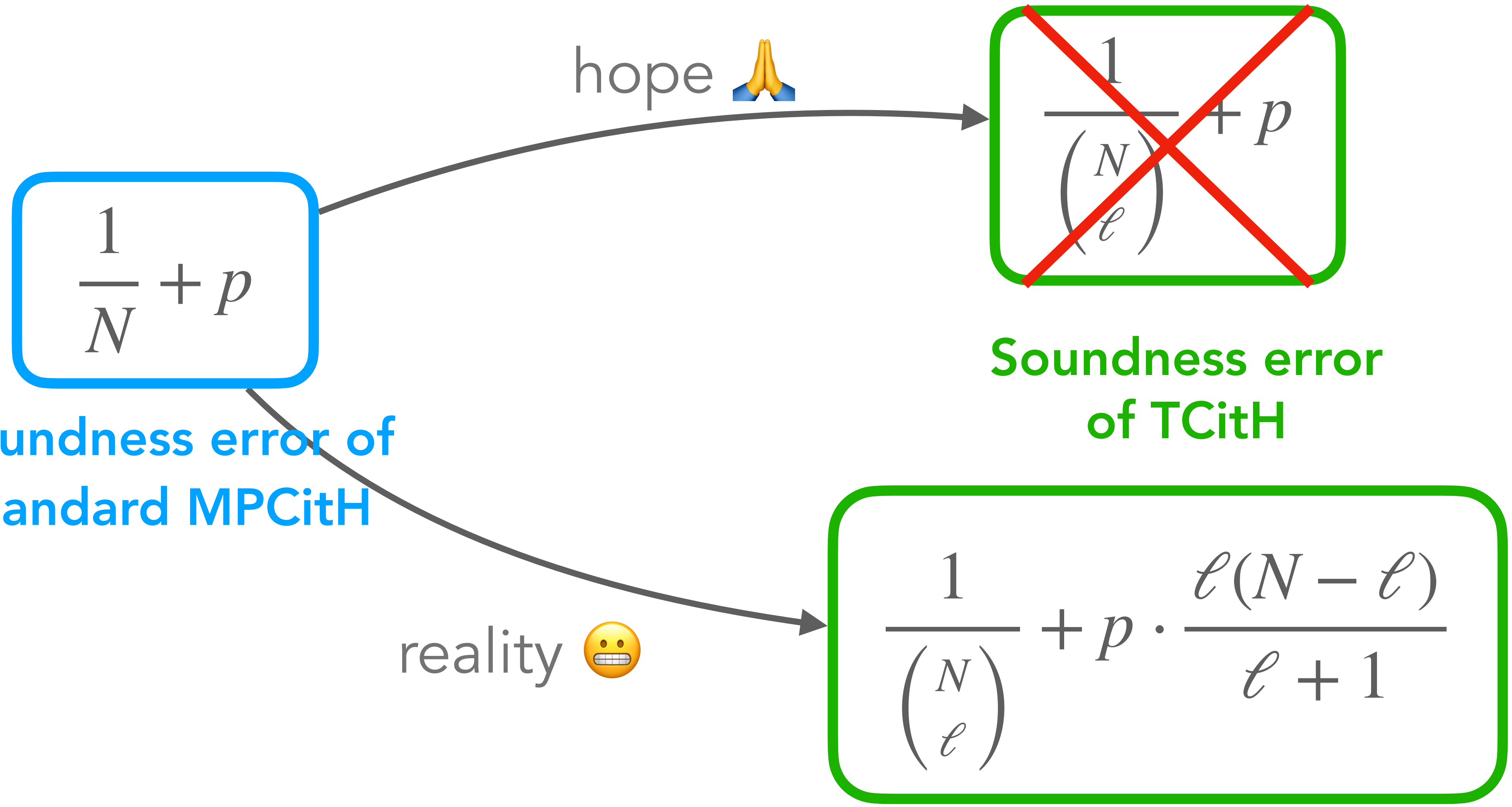
Soundness error of standard MPCitH

Soundness error of TCitH

Soundness

$$p = \text{“false positive probability”}$$

$$= P\left[\text{MPC protocol accepts } [\![x]\!] \text{ while } f(x) \neq y\right]$$



$$\frac{1}{N} + p$$

Soundness error of standard MPCitH

hope 🙏

$$\xcancel{\frac{1}{\binom{N}{\ell}} + p}$$

Soundness error of TCitH

reality 😬

$$\frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell(N-\ell)}{\ell+1}$$

Soundness

$$\frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell(N-\ell)}{\ell+1}$$

Why?

🤔

Soundness

$$\frac{1}{\binom{N}{\ell}} + p \cdot \boxed{\frac{\ell(N-\ell)}{\ell+1}}$$

Why?

- Prover can commit invalid sharings
- Let $[\![x]\!]^{(J)}$ = sharing interpolating $\left([\![x]\!]_i\right)_{i \in J}$
- Many different $[\![x]\!]^{(J)} \Rightarrow$ many possible false positives

Soundness

$$\frac{1}{\binom{N}{\ell}} + p \cdot \boxed{\frac{\ell(N-\ell)}{\ell+1}}$$

Why? 🤔

💡

- Prover can commit invalid sharings
- Let $[\![x]\!]^{(J)}$ = sharing interpolating $\left([\![x]\!]_i\right)_{i \in J}$
- Many different $[\![x]\!]^{(J)} \Rightarrow$ many possible false positives

🛠

- "Degree-enforcing commitment scheme"
- Verifier → Prover : random $\{\gamma_j\}$
- Prover → Verifier : $[\![\xi]\!] = \Sigma_j \, \gamma_i \cdot [\![x_j]\!]$
- Before MPC computation

Soundness

$$\frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell(N-\ell)}{\ell+1}$$

Why?

- Prover can commit invalid sharings
- Let $\llbracket x \rrbracket^{(J)}$ = sharing interpolating $\left(\llbracket x \rrbracket_i\right)_{i \in J}$
- Many different $\llbracket x \rrbracket^{(J)} \Rightarrow$ many possible false positives

- “Degree-enforcing commitment scheme”
- Verifier → Prover : random $\{\gamma_j\}$
- Prover → Verifier : $\llbracket \xi \rrbracket = \Sigma_j \, \gamma_i \cdot \llbracket x_j \rrbracket$
- Before MPC computation

$$\Longrightarrow \boxed{\frac{1}{\binom{N}{\ell}} + p}$$

TCitH vs. standard MPCitH

$\ell = 1 \;\Rightarrow\;$ Similar soundness: $\dfrac{1}{N} + p$ 🤝

TCitH vs. standard MPCitH

$\ell = 1 \;\Rightarrow\;$ Similar soundness: $\dfrac{1}{N} + p$ 🤝

	MPCitH + seed trees + hypercube [AGHHJY23]	**TCitH** $\ell = 1$

TCitH vs. standard MPCitH

$\ell = 1 \;\Rightarrow$ Similar soundness: $\dfrac{1}{N} + p$ 🤝

	MPCitH + seed trees + hypercube [AGHHJY23]	**TCitH** $\ell = 1$
Prover runtime	Party emulations: $\log N + 1$ Symmetric crypto: $O(N)$	Party emulations: 2 Symmetric crypto: $O(N)$

😀

fewer party emulations

TCitH vs. standard MPCitH

$\ell = 1 \Rightarrow$ Similar soundness: $\dfrac{1}{N} + p$ 🤝

	MPCitH + seed trees + hypercube [AGHHJY23]	**TCitH** $\ell = 1$	
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)*	😀

fewer party emulations

TCitH vs. standard MPCitH

$\ell = 1 \Rightarrow$ Similar soundness: $\frac{1}{N} + p$ 🤝

	MPCitH + seed trees + hypercube [AGHHJY23]	**TCitH** $\ell = 1$	
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)*	😀

🚀 *much less symmetric crypto*

TCitH vs. standard MPCitH

$\ell = 1 \;\Rightarrow\;$ Similar soundness: $\dfrac{1}{N} + p$ 🤝

	MPCitH + seed trees + hypercube [AGHHJY23]	**TCitH** $\ell = 1$	
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	🙁

×2

TCitH vs. standard MPCitH

$\ell = 1$ $\Rightarrow$ Similar soundness: $\frac{1}{N} + p$ 🤝

	MPCitH + seed trees + hypercube [AGHHJY23]	**TCitH** $\ell = 1$	
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		$N \leq \lvert\mathbb{F}\rvert$	🙁

TCitH vs. standard MPCitH

$\ell = 1 \Rightarrow$ Similar soundness: $\frac{1}{N} + p$ 🤝

	MPCitH + seed trees + hypercube [AGHHJY23]	**TCitH** $\ell = 1$	
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		$N \leq \|\mathbb{F}\|$	😀

Getting rid of these limitations

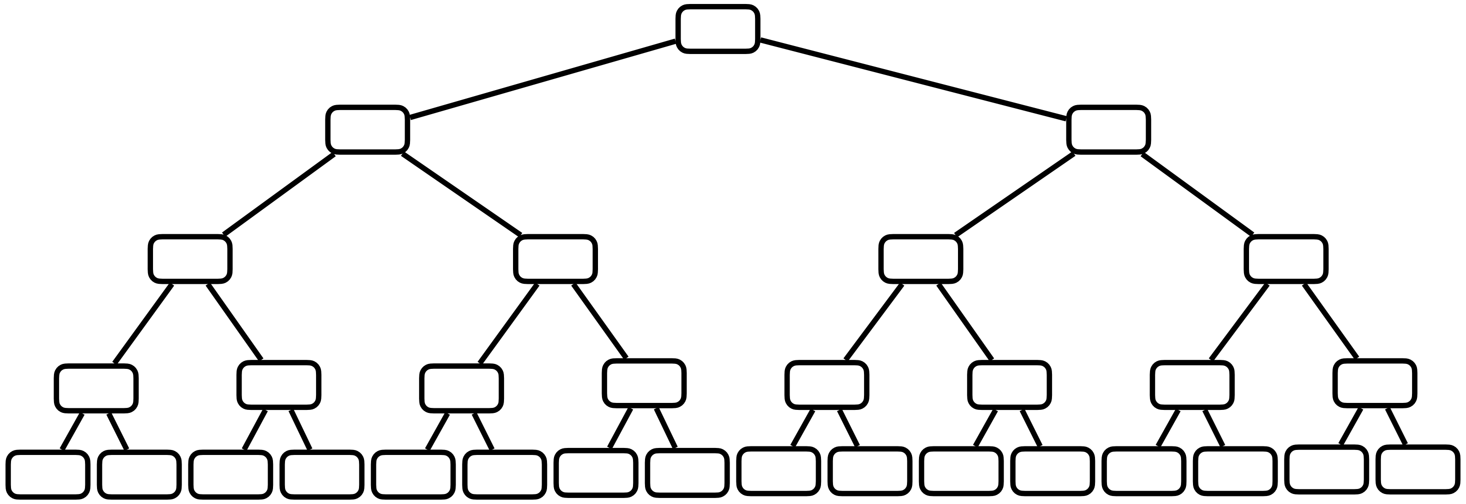
→ TCitH with GGM tree

TC-in-the-Head framework with GGM trees

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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

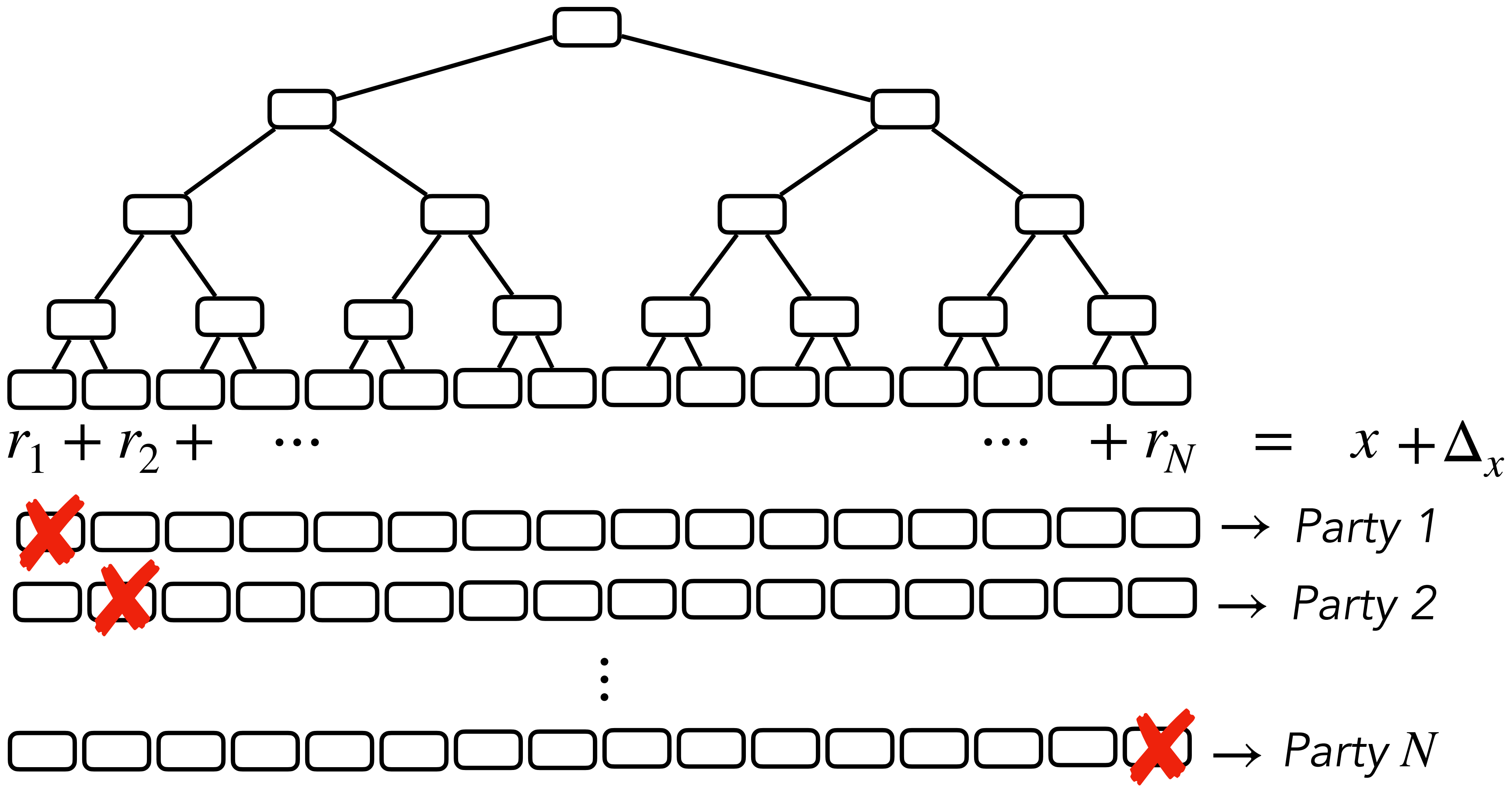


$r_1 + r_2 + \quad \cdots \qquad \cdots \quad + r_N \quad = \quad x + \Delta_x$

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



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

→ *Party 1*

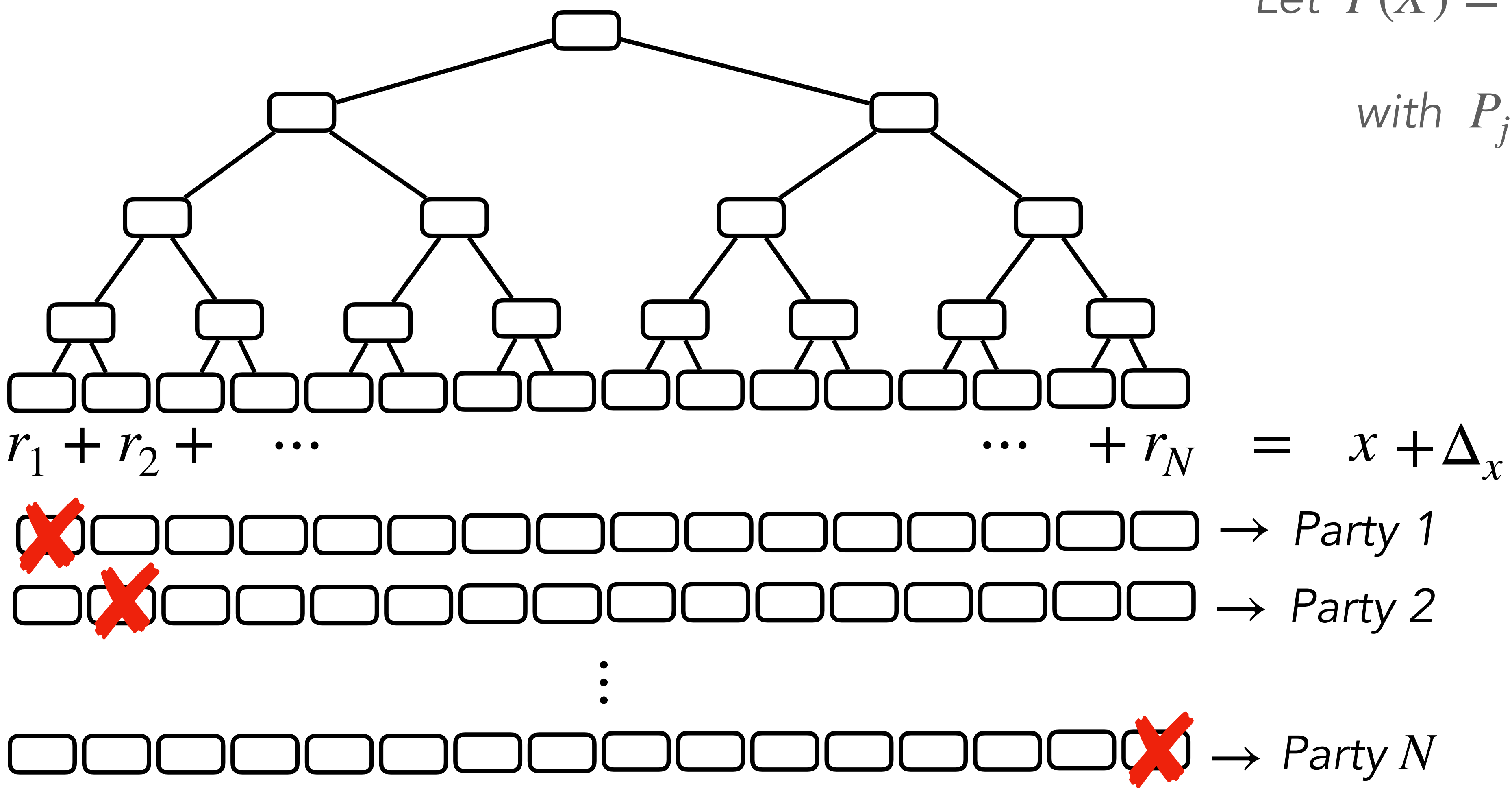
→ *Party 2*

→ *Party 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]

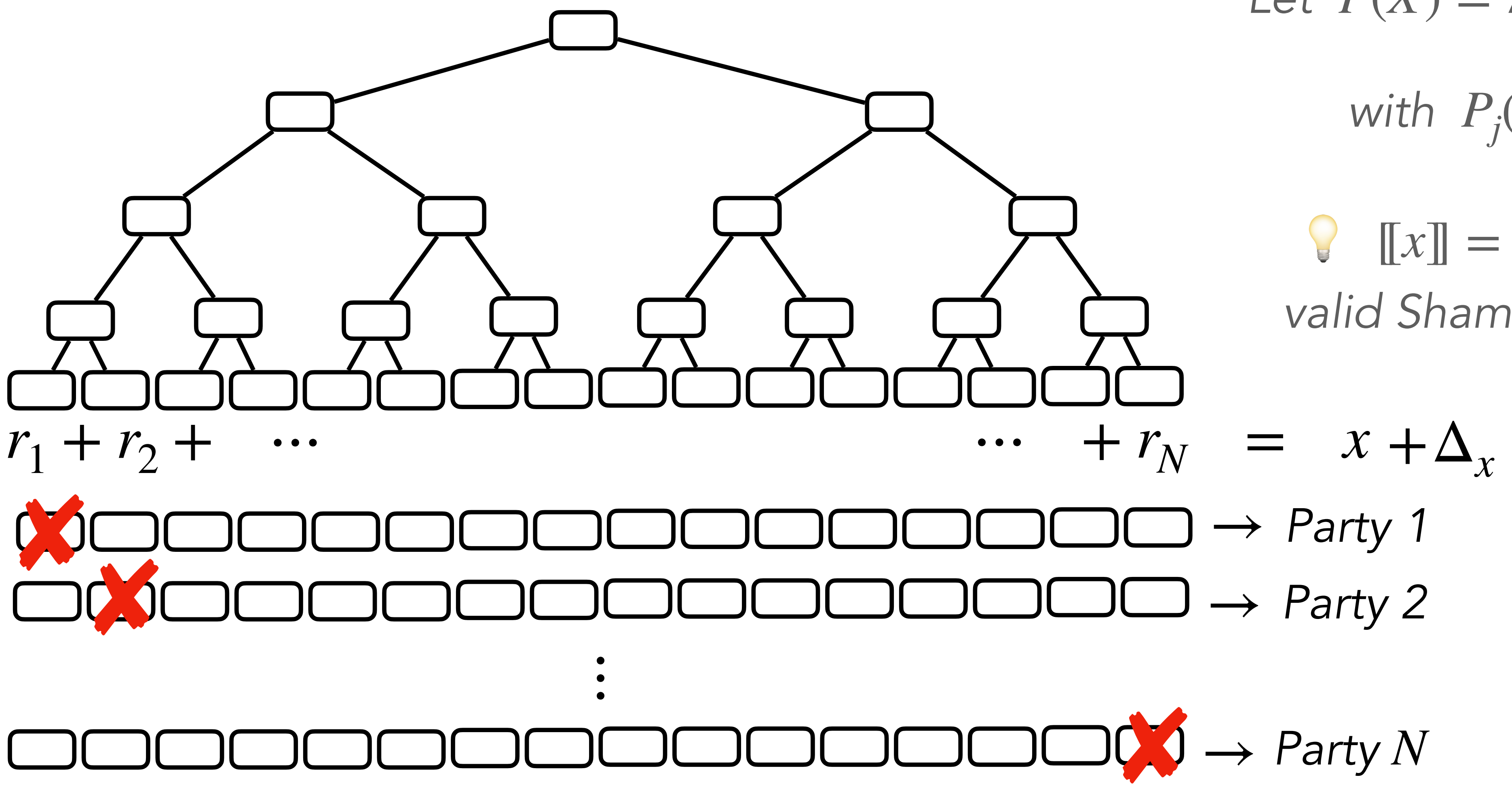
Let $P(X) = \Delta_x + \sum_j r_j P_j(X)$

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

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



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

→ *Party 1*

→ *Party 2*

⋮

→ *Party N*

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

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

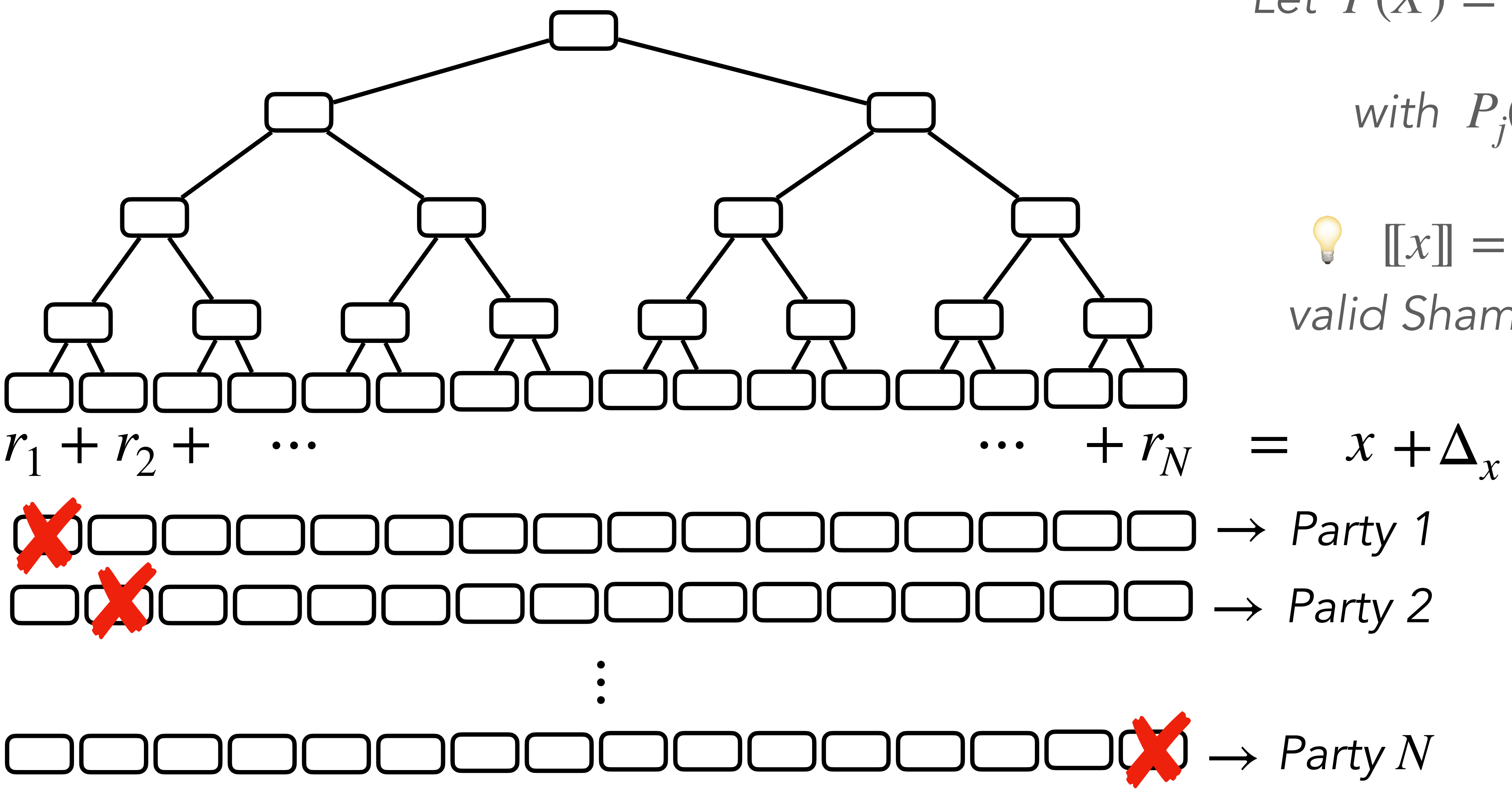
with $P_j(X) = 1 - (1/e_j) \cdot X$

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

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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


$r_1 + r_2 + \quad \cdots \quad \cdots \quad + r_N \quad = \quad x + \Delta_x$
→ Party 1
→ Party 2
→ Party N

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

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

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

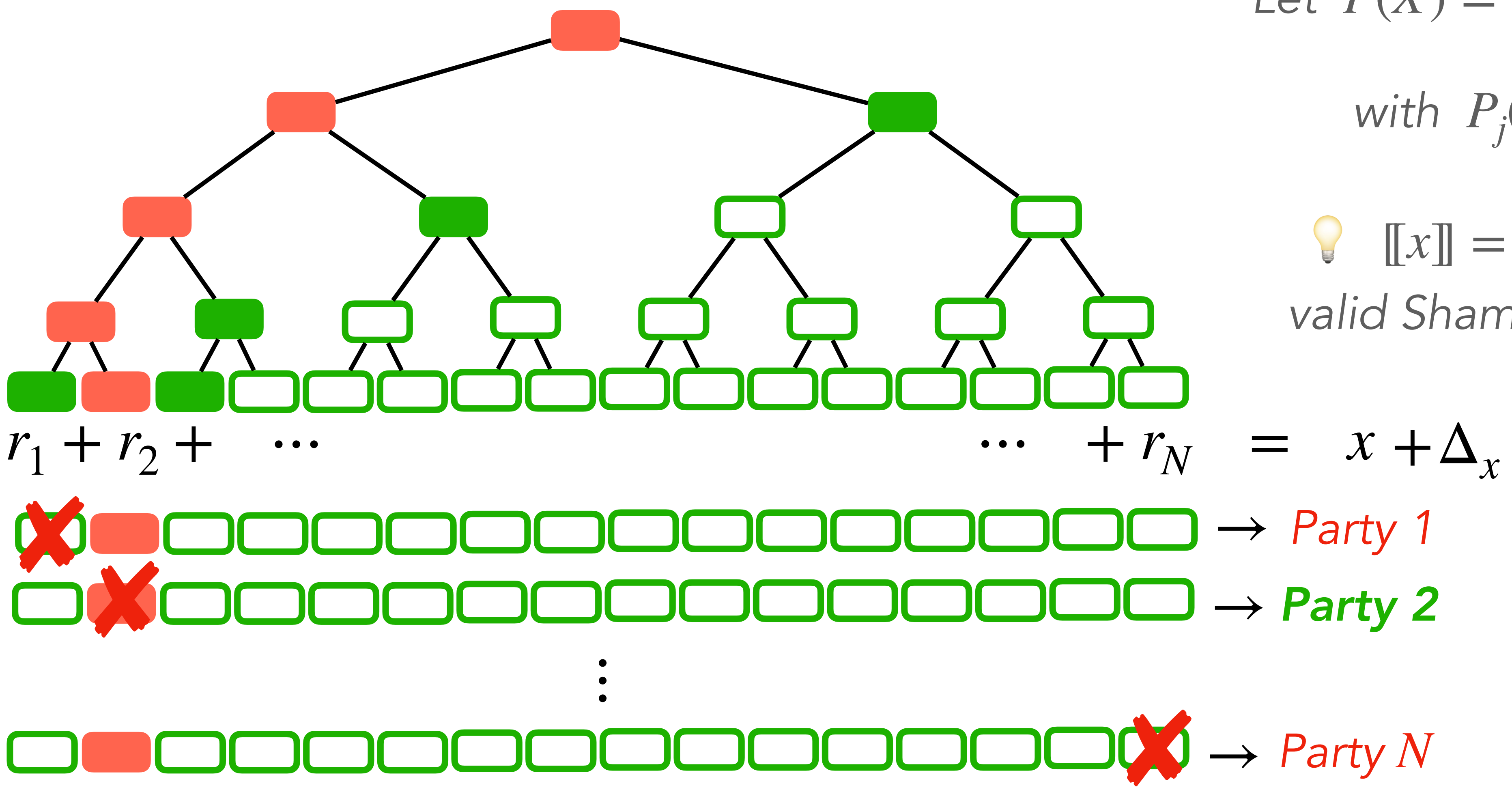
$$[\![x]\!]_i = \sum_{j \neq i} r_j P_j(e_i)$$

(since $P_i(e_i) = 0$)

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



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

→ Party 1

→ **Party 2**

⋮

→ Party N

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

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

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

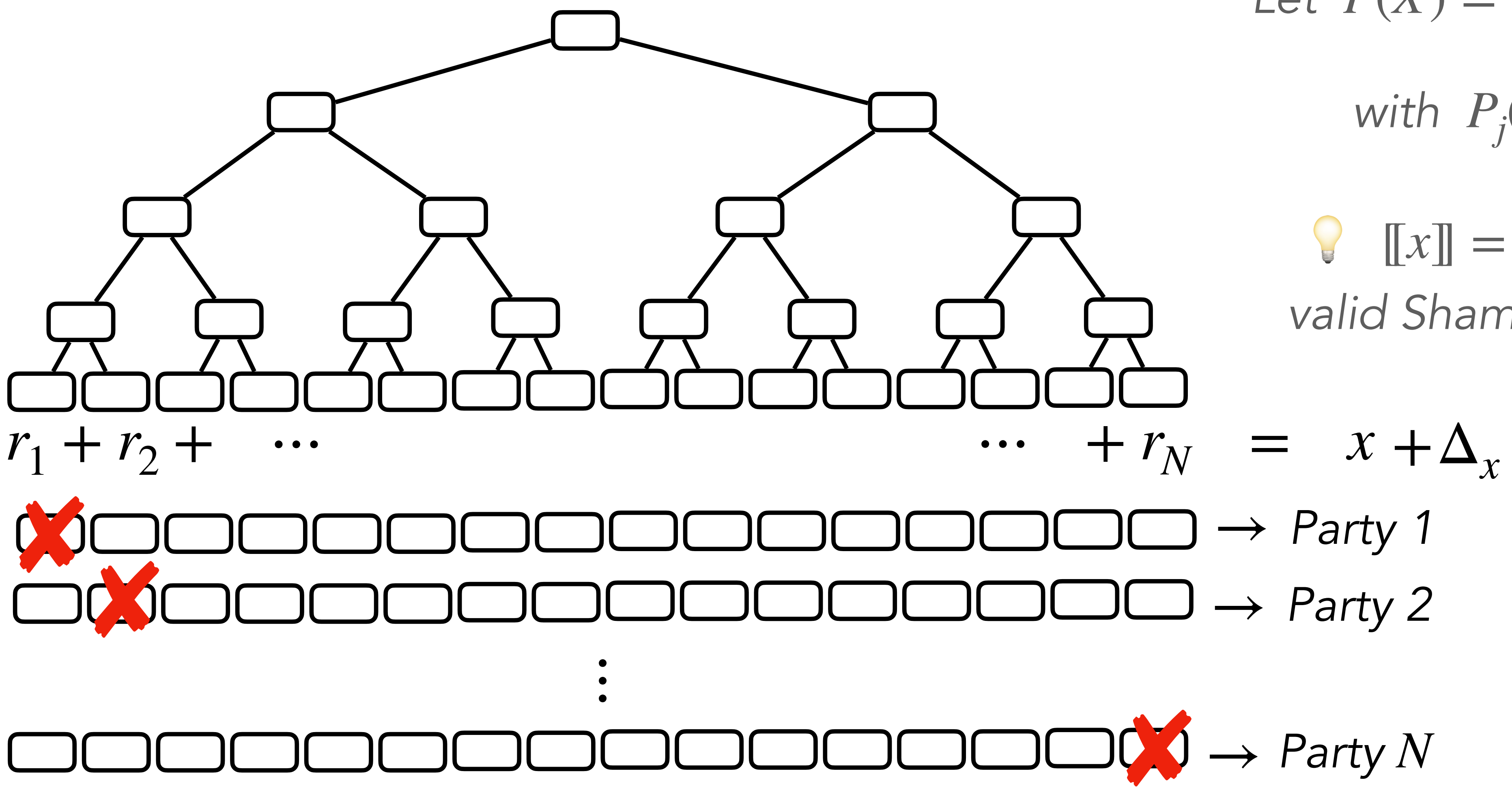
$$[\![x]\!]_i = \sum_{j \neq i} r_j P_j(e_i)$$

(since $P_i(e_i) = 0$)

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



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

→ *Party 1*

→ *Party 2*

⋮

→ *Party N*

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

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

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

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

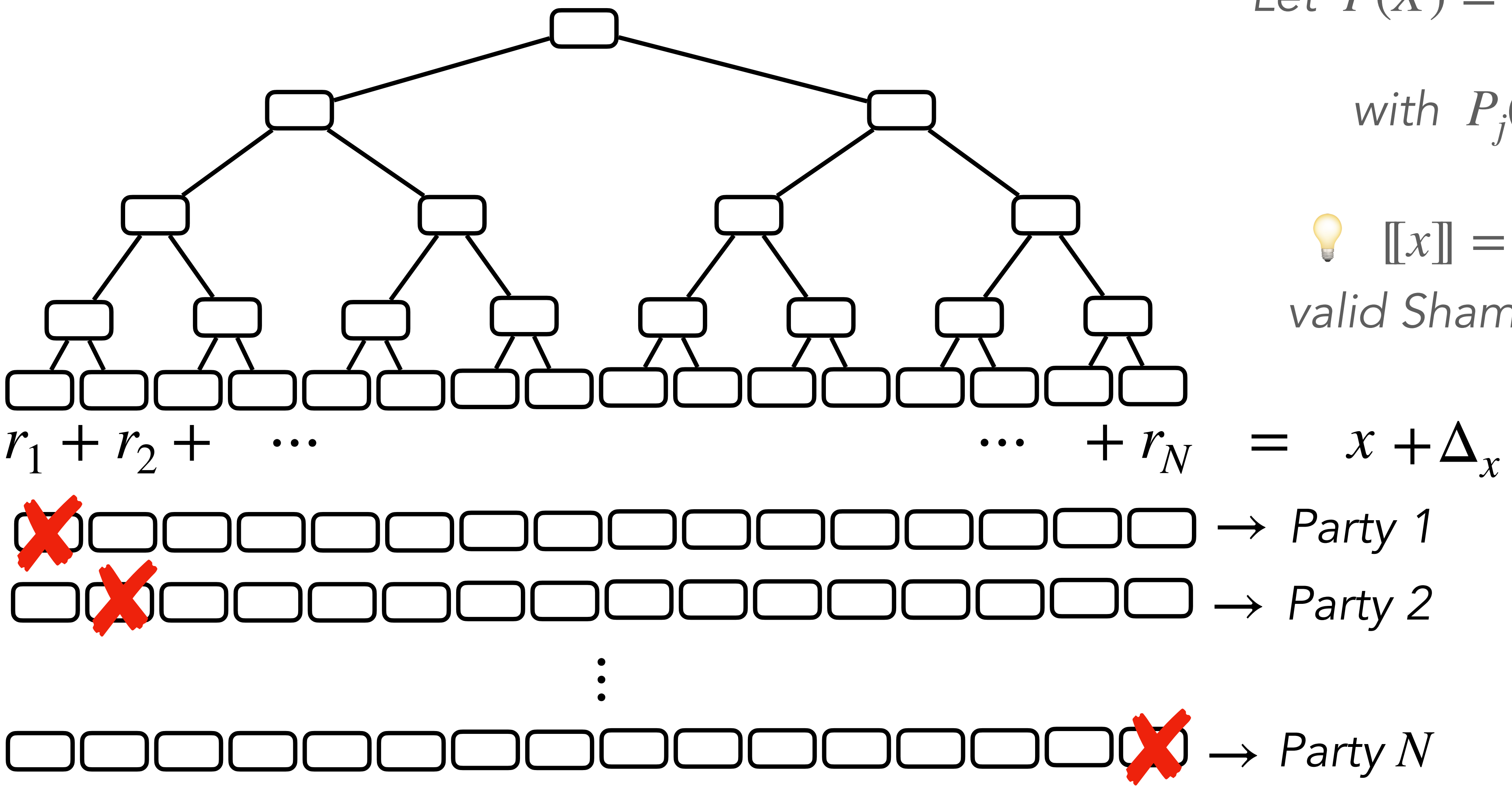
(since $P_i(e_i) = 0$*)*

⚒️ Can be adapted to $\ell > 1$

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



$r_1 + r_2 + \quad \cdots \qquad \cdots \quad + r_N \;=\; x + \Delta_x$

→ *Party 1*

→ *Party 2*

$\vdots$

→ *Party N*

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

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

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 $P_i(e_i) = 0$*)*

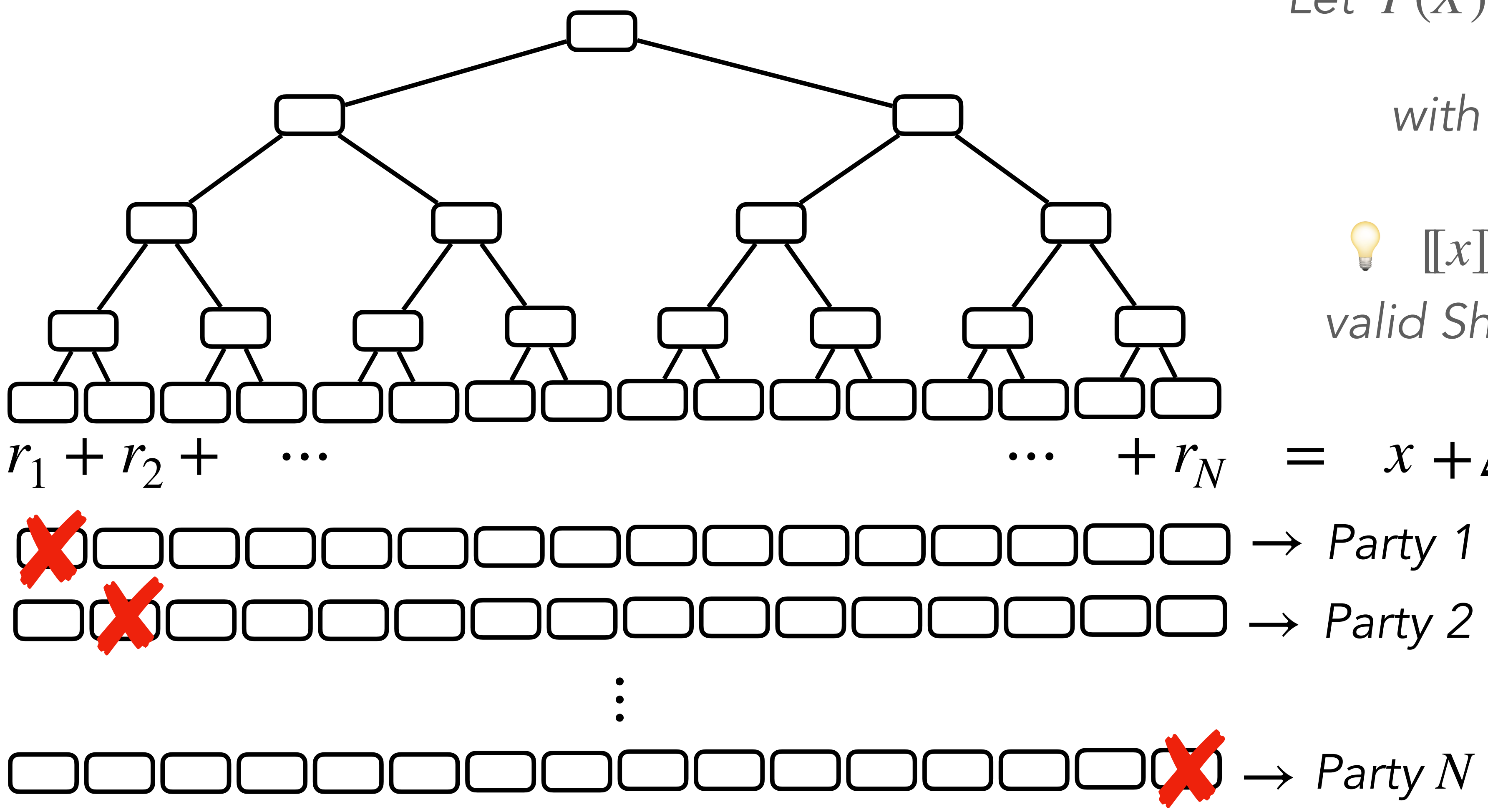
🛠️ Can be adapted to $\ell > 1$

🌲 Size of GGM tree

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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


$r_1 + r_2 + \quad \cdots \quad \cdots \quad + r_N \quad = \quad x + \Delta_x$
→ Party 1
→ Party 2
⋮
→ Party N

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

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

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 $\ell > 1$

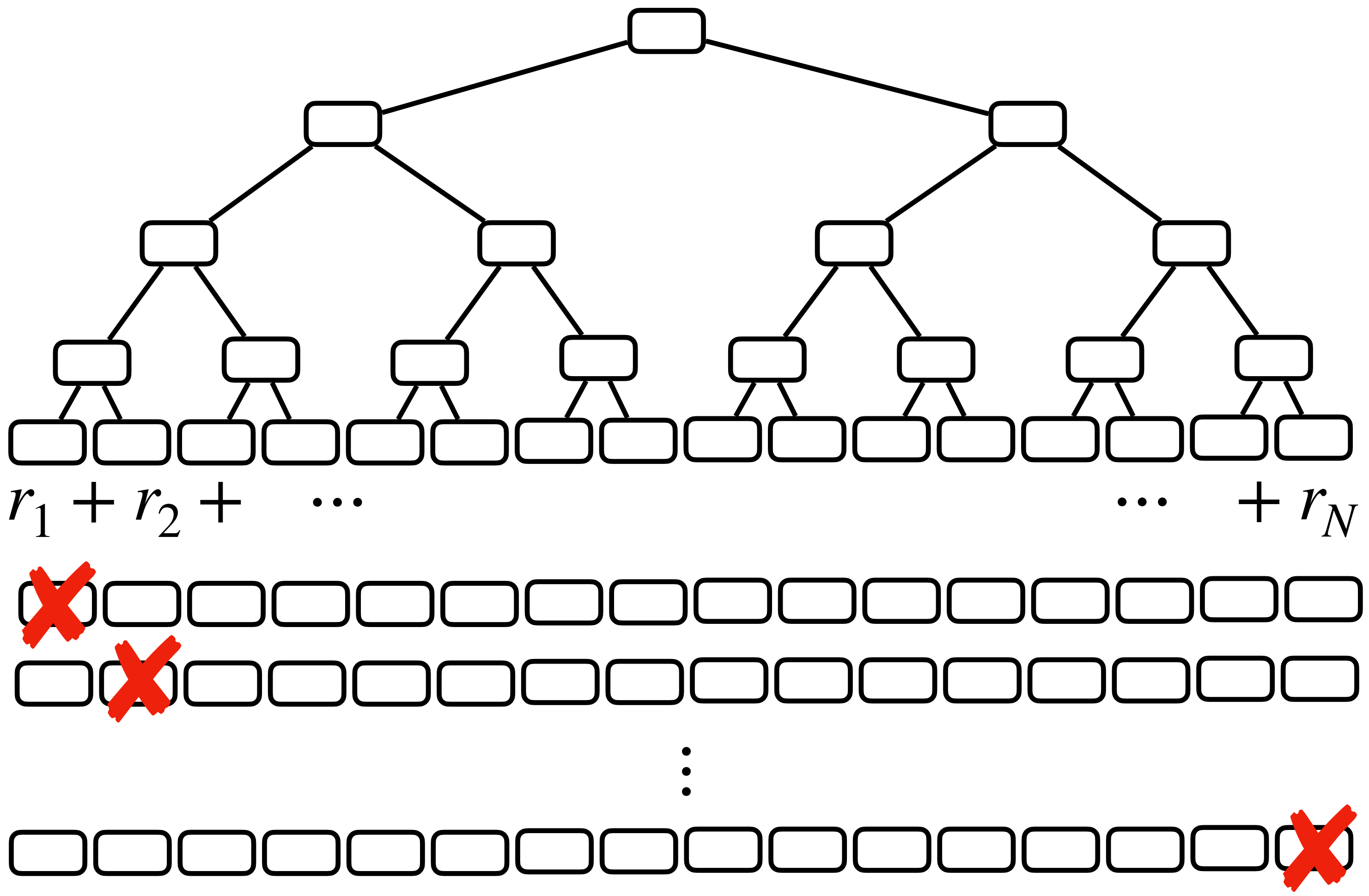
🌲 Size of GGM tree

😇 Good soundness (only valid sharings)

TCitH with GGM trees

Step 1: Generate a replicated secret sharing [ISN89]

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



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

→ *Party 1*

→ *Party 2*

$\vdots$

→ *Party N*

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

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

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 = $1 + \left\lceil \dfrac{\log_2 N}{\log_2 |\mathbb{F}|} \right\rceil$

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

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

Using multiplication homomorphism & packed secret sharing

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 $\Leftrightarrow f_1(w) = 0, \ \ldots, f_m(w) = 0$

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:

$$\llbracket x \rrbracket^{(d)} \cdot \llbracket y \rrbracket^{(d)} = \llbracket x \cdot y \rrbracket^{(2d)}$$

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

$$\llbracket \alpha \rrbracket = \llbracket v \rrbracket + \sum_{j=1}^{m} \boxed{\gamma_j} \cdot f_j(\llbracket w \rrbracket)$$

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 $\Leftrightarrow f_1(w) = 0, \ \ldots, f_m(w) = 0$
  - parties locally compute

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

pre-committed
sharing of 0

randomness
from the verifier

Using multiplication homomorphism

- Shamir's secret sharing satisfies:

$$\llbracket x \rrbracket^{(d)} \cdot \llbracket y \rrbracket^{(d)} = \llbracket x \cdot y \rrbracket^{(2d)}$$

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

$$\boxed{\llbracket \alpha \rrbracket} = \boxed{\llbracket v \rrbracket} + \sum_{j=1}^{m} \boxed{\gamma_j} \cdot f_j(\llbracket w \rrbracket)$$

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 $\Leftrightarrow f_1(w) = 0, \ \ldots, f_m(w) = 0$
  - parties locally compute

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

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

pre-committed
sharing of 0

randomness
from the verifier

$$\boxed{\frac{\binom{d_\alpha}{\ell}}{\binom{N}{\ell}} + p}$$

Soundness error

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 $\Leftrightarrow f_1(w) = 0, \; \dots, \; f_m(w) = 0$
  - parties locally compute

$$\underbrace{[\![\alpha]\!]}_{\substack{\text{check } \alpha = 0 \\ \text{false positive proba } 1/|\mathbb{F}|}} = \underbrace{[\![v]\!]}_{\substack{\text{pre-committed} \\ \text{sharing of 0}}} + \sum_{j=1}^{m} \underbrace{\gamma_j}_{\substack{\text{randomness} \\ \text{from the verifier}}} \cdot f_j([\![w]\!])$$

Here:

$$\boxed{\frac{\dbinom{\overset{\ell \cdot \deg f_j}{d_\alpha}}{\ell}}{\dbinom{N}{\ell}} + \overset{\left(\frac{1}{|\mathbb{F}|}\right)^{\#\alpha}}{p}}$$

Soundness error

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**

Using packed secret sharing

- Shamir's secret sharing can be packed
  - $P(\omega_1) = x_1 , \quad \ldots , \; P(\omega_s) = x_s$
  - $P(\omega_{s+1}) = r_1 , \; \ldots , P(\omega_{s+\ell}) = r_\ell$
  - $[\![x]\!]_1 = P(e_1) , \; \ldots , [\![x]\!]_N = P(e_N)$

Using packed secret sharing

- Shamir's secret sharing can be packed
  - $P(\omega_1) = x_1 \,, \quad \ldots \,, \quad P(\omega_s) = x_s$
  - $P(\omega_{s+1}) = r_1 \,, \quad \ldots \,, P(\omega_{s+\ell}) = r_\ell$
  - $[\![x]\!]_1 = P(e_1) \,, \quad \ldots \,, [\![x]\!]_N = P(e_N)$
- $[\![x]\!] + [\![y]\!] =$ sharing of $(x_1, \ldots, x_s) + (y_1, \ldots, y_s)$
- $[\![x]\!] \cdot [\![y]\!] =$ sharing of $(x_1, \ldots, x_s) \circ (y_1, \ldots, y_s)$

Using packed secret sharing

- Shamir's secret sharing can be packed
    - $P(\omega_1) = x_1 , \quad \ldots , \quad P(\omega_s) = x_s$
    - $P(\omega_{s+1}) = r_1 , \quad \ldots , P(\omega_{s+\ell}) = r_\ell$
    - $[\![x]\!]_1 = P(e_1) , \quad \ldots , [\![x]\!]_N = P(e_N)$
- $[\![x]\!] + [\![y]\!] =$ sharing of $(x_1, \ldots, x_s) + (y_1, \ldots, y_s)$
- $[\![x]\!] \cdot [\![y]\!] =$ sharing of $(x_1, \ldots, x_s) \circ (y_1, \ldots, y_s)$

$$\boxed{\frac{\binom{d_\alpha}{\ell}}{\binom{N}{\ell}} + p}$$

Soundness error

Using packed secret sharing

- Shamir's secret sharing can be packed
  - $P(\omega_1) = x_1 , \quad \ldots , \quad P(\omega_s) = x_s$
  - $P(\omega_{s+1}) = r_1 , \quad \ldots , P(\omega_{s+\ell}) = r_\ell$
  - $[\![x]\!]_1 = P(e_1) , \quad \ldots , [\![x]\!]_N = P(e_N)$
- $[\![x]\!] + [\![y]\!] =$ sharing of $(x_1, \ldots, x_s) + (y_1, \ldots, y_s)$
- $[\![x]\!] \cdot [\![y]\!] =$ sharing of $(x_1, \ldots, x_s) \circ (y_1, \ldots, y_s)$

Here: $(\ell + s - 1) \cdot \deg f_j$

$$\boxed{\frac{\binom{d_\alpha}{\ell}}{\binom{N}{\ell}} + p}$$

Soundness error

Using packed secret sharing

- Shamir's secret sharing can be packed
  - $P(\omega_1) = x_1 , \quad \dots , \quad P(\omega_s) = x_s$
  - $P(\omega_{s+1}) = r_1 , \quad \dots , P(\omega_{s+\ell}) = r_\ell$
  - $[\![x]\!]_1 = P(e_1) , \quad \dots , [\![x]\!]_N = P(e_N)$
- $[\![x]\!] + [\![y]\!] =$ sharing of $(x_1, \dots, x_s) + (y_1, \dots, y_s)$
- $[\![x]\!] \cdot [\![y]\!] =$ sharing of $(x_1, \dots, x_s) \circ (y_1, \dots, y_s)$
- Packed sharing & Merkle trees ≈ ÷ witness size by s

  ⇒ interesting for statements with "medium size" witness

Here: $(\ell + s - 1) \cdot \deg f_j$

$$\boxed{\frac{\binom{d_\alpha}{\ell}}{\binom{N}{\ell}} + p}$$

Soundness error

Using packed secret sharing

- Shamir's secret sharing can be packed
  - $P(\omega_1) = x_1$, … , $P(\omega_s) = x_s$
  - $P(\omega_{s+1}) = r_1$, … , $P(\omega_{s+\ell}) = r_\ell$
  - $[\![x]\!]_1 = P(e_1)$, … , $[\![x]\!]_N = P(e_N)$
- $[\![x]\!] + [\![y]\!] =$ sharing of $(x_1, \ldots, x_s) + (y_1, \ldots, y_s)$
- $[\![x]\!] \cdot [\![y]\!] =$ sharing of $(x_1, \ldots, x_s) \circ (y_1, \ldots, y_s)$
- Packed sharing & Merkle trees ≈ ÷ witness size by s

  ⇒ interesting for statements with "medium size" witness
- E.g. an ISIS statement $\vec{t} = A \cdot \vec{e}$ with $\| \vec{e} \|_\infty \leq \beta$

Here: $(\ell + s - 1) \cdot \deg f_j$

$$\boxed{\frac{\binom{\boxed{d_\alpha}}{\ell}}{\binom{N}{\ell}} + p}$$

Soundness error

TCitH-GGM vs. TCitH-MT

TCitH-GGM	*TCitH-MT*
🎄 Smaller tree	🌲 Larger tree (~x2)

TCitH-GGM vs. TCitH-MT

TCitH-GGM	***TCitH-MT***
🎄 Smaller tree	🌲 Larger tree (~x2)
❌📦 No advantage of packed sharing	📦 Takes advantage of packed sharing

TCitH-GGM vs. TCitH-MT

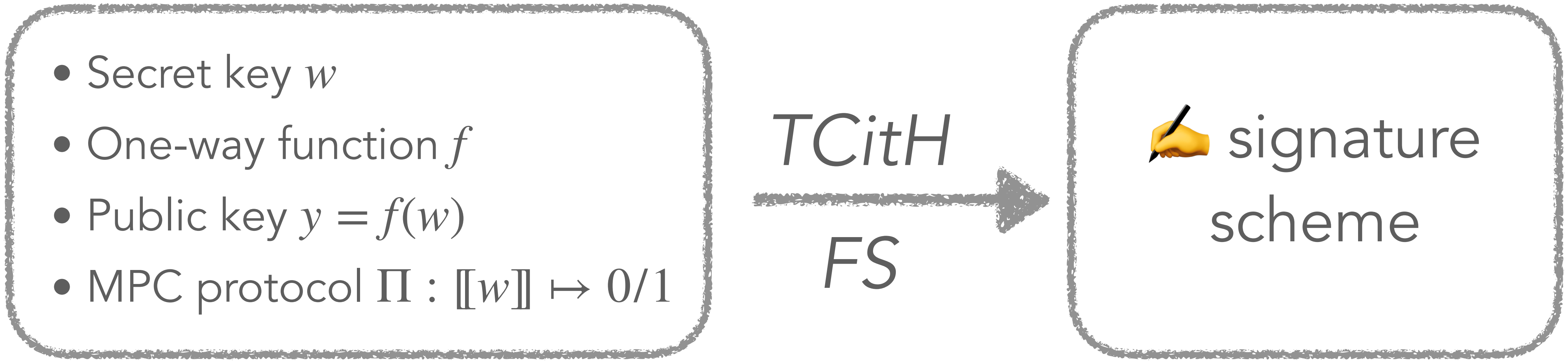
TCitH-GGM	***TCitH-MT***
🎄 Smaller tree	🌲 Larger tree (~x2)
❌📦 No advantage of packed sharing	📦 Takes advantage of packed sharing
☑ Naturally enforce degree of committed sharings	🔁 Need degree enforcing commitment (+1 round)

TCitH-GGM vs. TCitH-MT

TCitH-GGM	***TCitH-MT***
🎄 Smaller tree	🌲 Larger tree (~x2)
❌📦 No advantage of packed sharing	📦 Takes advantage of packed sharing
☑️ Naturally enforce degree of committed sharings	🔁 Need degree enforcing commitment (+1 round)
🎯 Better for "small-size" statements	🎯 Better for "medium-size" statements

Application: post-quantum ring signatures

Post-quantum ring signatures

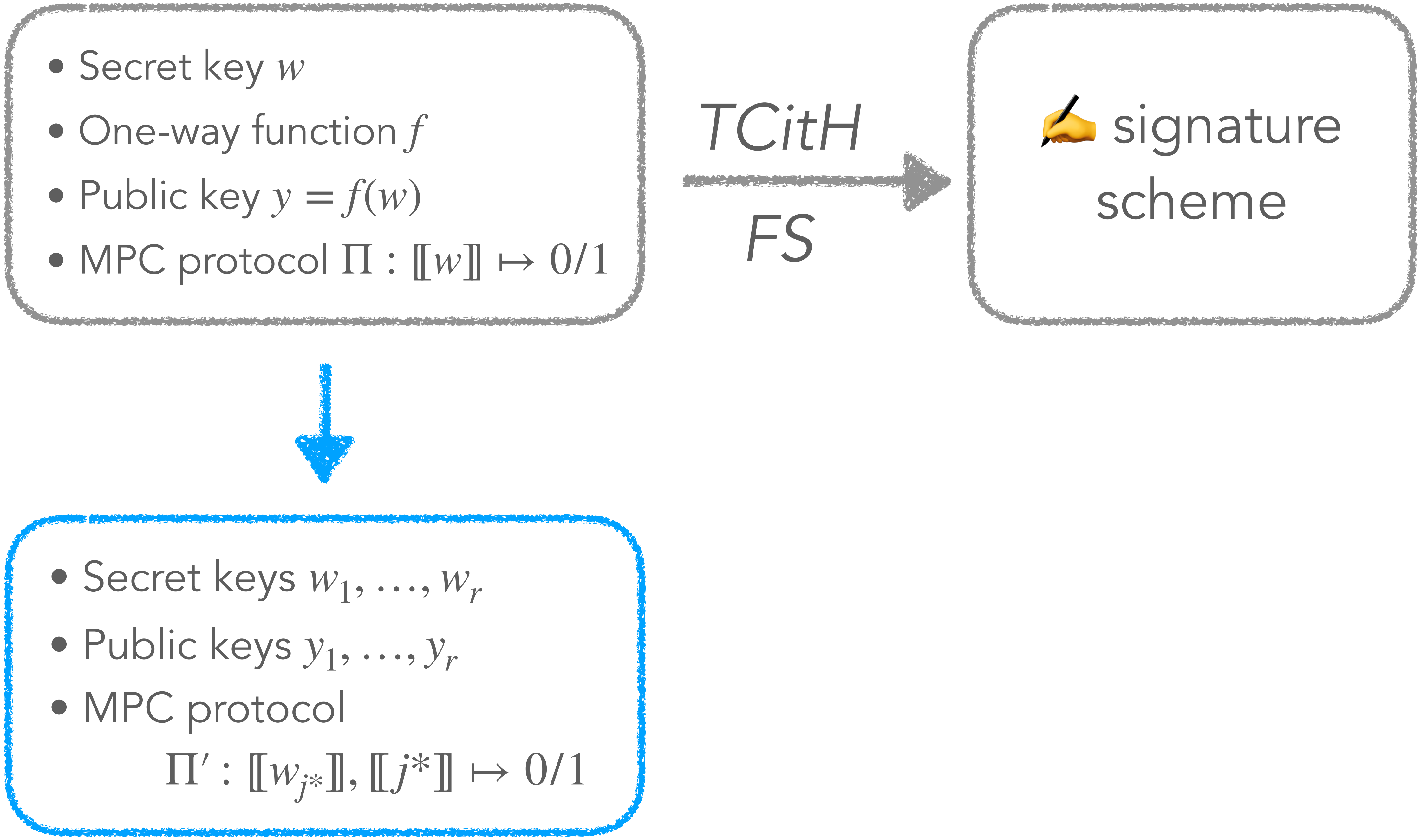


- Secret key w
- One-way function f
- Public key $y = f(w)$
- MPC protocol $\Pi : [\![w]\!] \mapsto 0/1$

$\xrightarrow[FS]{TCitH}$

✍️ signature scheme

Post-quantum ring signatures



- Secret key w
- One-way function f
- Public key $y = f(w)$
- MPC protocol $\Pi : [\![w]\!] \mapsto 0/1$

TCitH

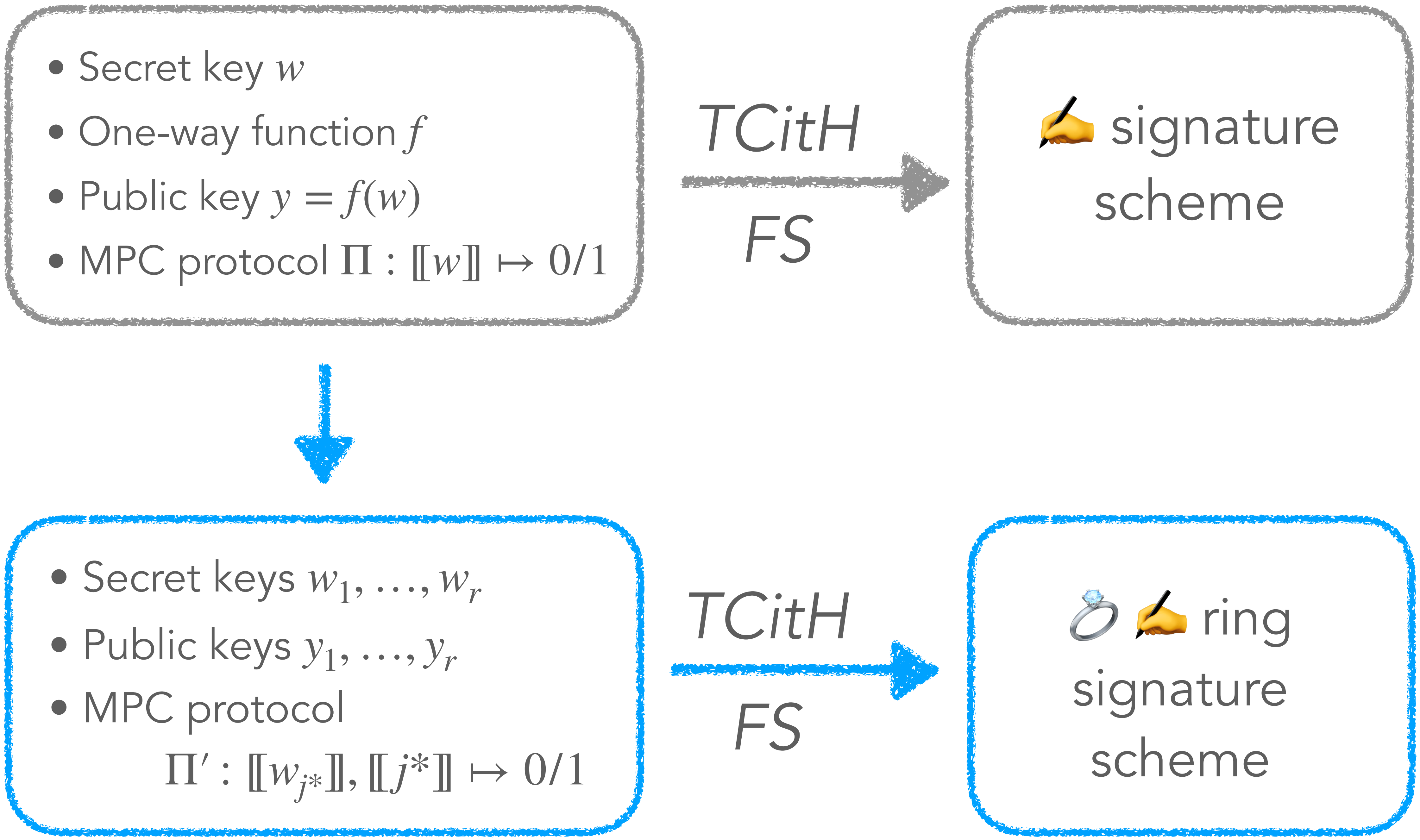
FS

✍ signature scheme

- Secret keys $w_1, \ldots, w_r$
- Public keys $y_1, \ldots, y_r$
- MPC protocol

$$\Pi' : [\![w_{j^*}]\!], [\![j^*]\!] \mapsto 0/1$$

Post-quantum ring signatures



- Secret key w
- One-way function f
- Public key $y = f(w)$
- MPC protocol $\Pi : [\![w]\!] \mapsto 0/1$

TCitH

FS

✍ signature scheme

- Secret keys $w_1, \ldots, w_r$
- Public keys $y_1, \ldots, y_r$
- MPC protocol

$$\Pi' : [\![w_{j^*}]\!], [\![j^*]\!] \mapsto 0/1$$

TCitH

FS

💍 ✍ ring signature scheme

Post-quantum ring signatures

💡 Idea:

- One-hot encoding of $j*$

$$s = (0,\ldots,0,\, s_{j*} := 1,\, 0,\ldots,0)$$

Post-quantum ring signatures

💡 Idea:

- One-hot encoding of $j*$

$$s = (0,\ldots,0, s_{j*} := 1, 0,\ldots,0)$$

- Π' computes $[\![y_{j*}]\!] = \sum_{j=1}^{r} [\![s_j]\!] \cdot y_j$

Post-quantum ring signatures

💡 Idea:

- One-hot encoding of j^*

$$s = (0,\ldots,0, s_{j^*} := 1, 0,\ldots,0)$$

- Π' computes $\llbracket y_{j^*} \rrbracket = \sum_{j=1}^{r} \llbracket s_j \rrbracket \cdot y_j$

🤔 Problem: including $\llbracket s \rrbracket$ to the witness ⇒ $\mathcal{O}(r)$ signature size

Post-quantum ring signatures

💡 Idea:

- One-hot encoding of j^*

$$s = (0,\ldots,0,\, s_{j^*} := 1,\, 0,\ldots,0)$$

- Π' computes $\llbracket y_{j^*} \rrbracket = \sum_{j=1}^{r} \llbracket s_j \rrbracket \cdot y_j$

🤔 Problem: including $\llbracket s \rrbracket$ to the witness $\Rightarrow \mathcal{O}(r)$ signature size

🛠️ Solution: $\llbracket s^{(1)} \rrbracket, \ldots, \llbracket s^{(d)} \rrbracket$ s.t. $s = s^{(1)} \otimes \cdots \otimes s^{(d)}$

$\Rightarrow \mathcal{O}\left(d \sqrt[d]{r}\,\right)$ signature size $\Rightarrow \mathcal{O}(\log r)$

Post-quantum ring signatures

Protocol Π'

Input: $[\![w]\!], [\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

Post-quantum ring signatures

Protocol Π'

Input: $[\![w]\!]$, $[\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

1. Locally compute $[\![s]\!] = [\![s_1]\!] \otimes \cdots \otimes [\![s_d]\!]$

Post-quantum ring signatures

Protocol Π'

Input: $[\![w]\!]$, $[\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

1. Locally compute $[\![s]\!] = [\![s_1]\!] \otimes \cdots \otimes [\![s_d]\!]$
2. Locally compute $[\![y_{j*}]\!] = \sum_{j=1}^{r} [\![s_j]\!] \cdot y_j$

Post-quantum ring signatures

Protocol Π'

Input: $[\![w]\!]$, $[\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

1. Locally compute $[\![s]\!] = [\![s_1]\!] \otimes \cdots \otimes [\![s_d]\!]$
2. Locally compute $[\![y_{j*}]\!] = \sum_{j=1}^{r} [\![s_j]\!] \cdot y_j$
3. Check that $[\![w]\!]$, $[\![y_{j*}]\!]$ satisfy $f(w) = y_{j*}$ using Π

Post-quantum ring signatures

Protocol Π'

Input: $[\![w]\!], [\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

1. Locally compute $[\![s]\!] = [\![s_1]\!] \otimes \cdots \otimes [\![s_d]\!]$
2. Locally compute $[\![y_{j*}]\!] = \sum_{j=1}^{r} [\![s_j]\!] \cdot y_j$
3. Check that $[\![w]\!]$, $[\![y_{j*}]\!]$ satisfy $f(w) = y_{j*}$ using Π
4. Check that $[\![s]\!]$ is the sharing of a one-hot encoding

Post-quantum ring signatures

Protocol Π'

Input: $[\![w]\!], [\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

1. Locally compute $[\![s]\!] = [\![s_1]\!] \otimes \cdots \otimes [\![s_d]\!]$
2. Locally compute $[\![y_{j^*}]\!] = \sum_{j=1}^{r} [\![s_j]\!] \cdot y_j$
3. Check that $[\![w]\!]$, $[\![y_{j^*}]\!]$ satisfy $f(w) = y_{j^*}$ using Π
4. Check that $[\![s]\!]$ is the sharing of a one-hot encoding

🛠️ Simple
MPC protocol

Post-quantum ring signatures

Protocol Π'

Input: $[\![w]\!], [\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

1. Locally compute $[\![s]\!] = [\![s_1]\!] \otimes \cdots \otimes [\![s_d]\!]$
2. Locally compute $[\![y_{j^*}]\!] = \sum_{j=1}^{r} [\![s_j]\!] \cdot y_j$
3. Check that $[\![w]\!]$, $[\![y_{j^*}]\!]$ satisfy $f(w) = y_{j^*}$ using Π
4. Check that $[\![s]\!]$ is the sharing of a one-hot encoding

🛠 Simple MPC protocol

⚠ Π must be adapted to use $[\![y_{j^*}]\!]$ instead of y_{j^*}

Post-quantum ring signatures

Protocol Π'

Input: $[\![w]\!], [\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

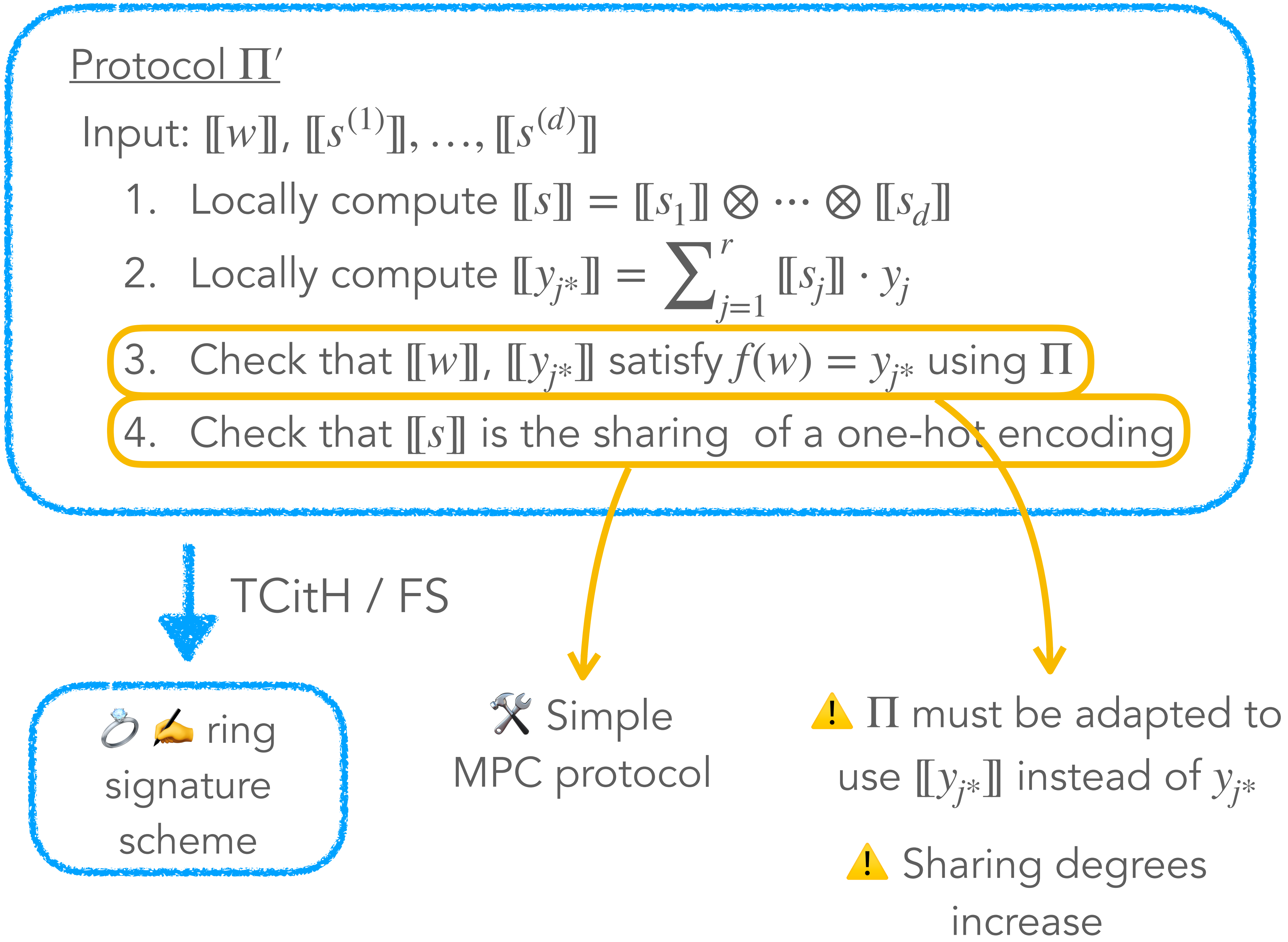
1. Locally compute $[\![s]\!] = [\![s_1]\!] \otimes \cdots \otimes [\![s_d]\!]$
2. Locally compute $[\![y_{j^*}]\!] = \sum_{j=1}^{r} [\![s_j]\!] \cdot y_j$
3. Check that $[\![w]\!], [\![y_{j^*}]\!]$ satisfy $f(w) = y_{j^*}$ using Π
4. Check that $[\![s]\!]$ is the sharing of a one-hot encoding

🛠 Simple MPC protocol

⚠️ Π must be adapted to use $[\![y_{j^*}]\!]$ instead of y_{j^*}

⚠️ Sharing degrees increase

Post-quantum ring signatures



Protocol Π'

Input: $[\![w]\!], [\![s^{(1)}]\!], \ldots, [\![s^{(d)}]\!]$

1. Locally compute $[\![s]\!] = [\![s_1]\!] \otimes \cdots \otimes [\![s_d]\!]$
2. Locally compute $[\![y_{j^*}]\!] = \sum_{j=1}^{r} [\![s_j]\!] \cdot y_j$
3. Check that $[\![w]\!]$, $[\![y_{j^*}]\!]$ satisfy $f(w) = y_{j^*}$ using Π
4. Check that $[\![s]\!]$ is the sharing of a one-hot encoding

TCitH / FS

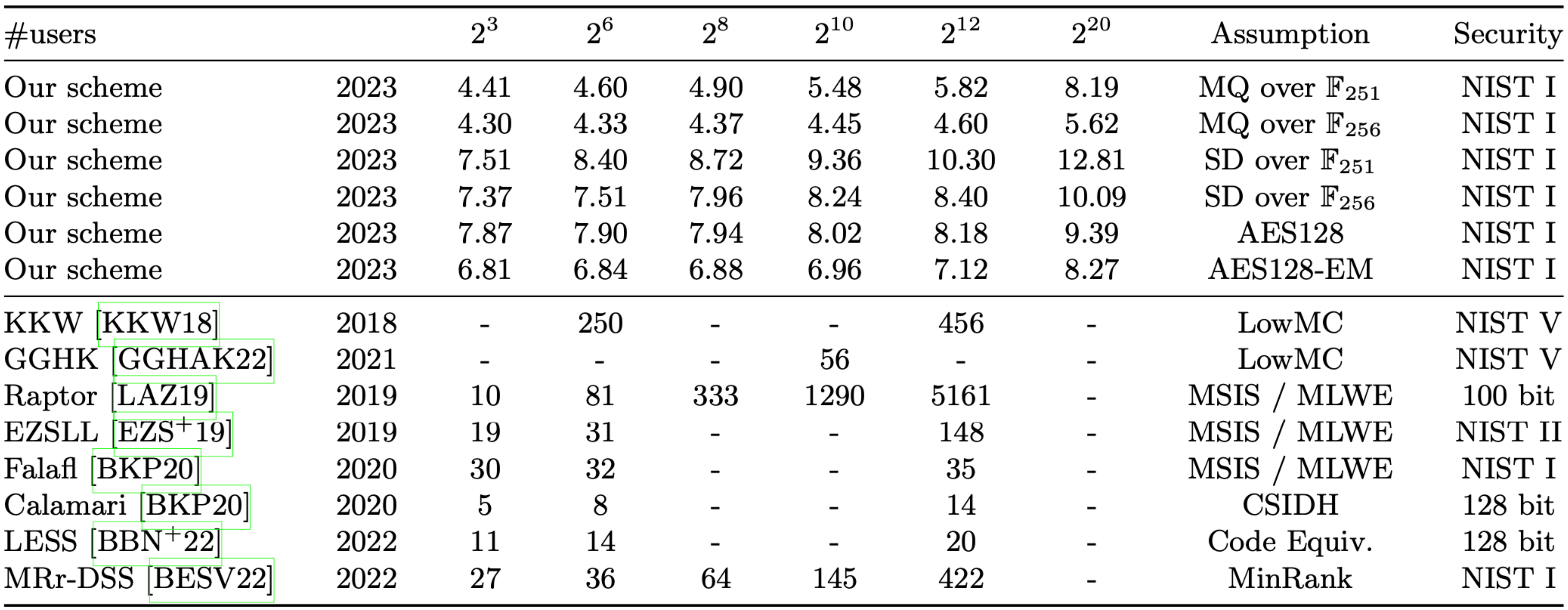
💍 ✍️ ring signature scheme

🛠️ Simple MPC protocol

⚠️ Π must be adapted to use $[\![y_{j^*}]\!]$ instead of y_{j^*}

⚠️ Sharing degrees increase

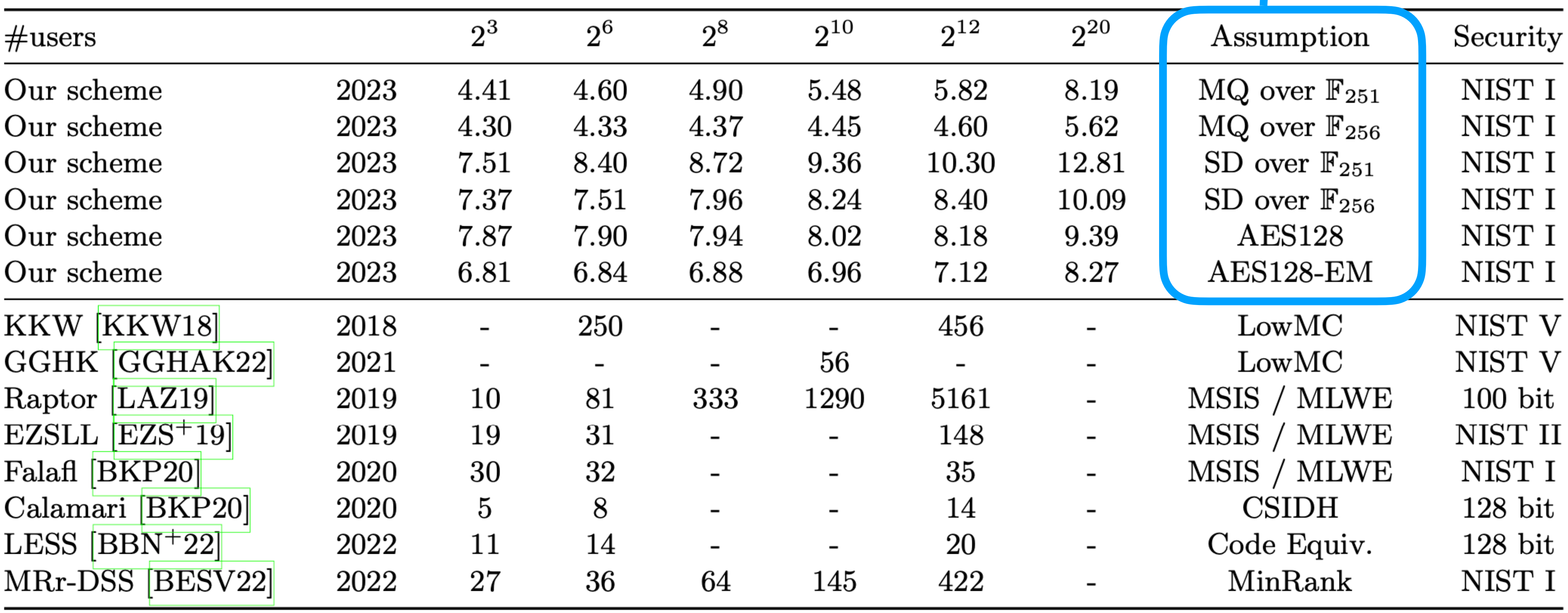
Post-quantum ring signatures



#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

Post-quantum ring signatures

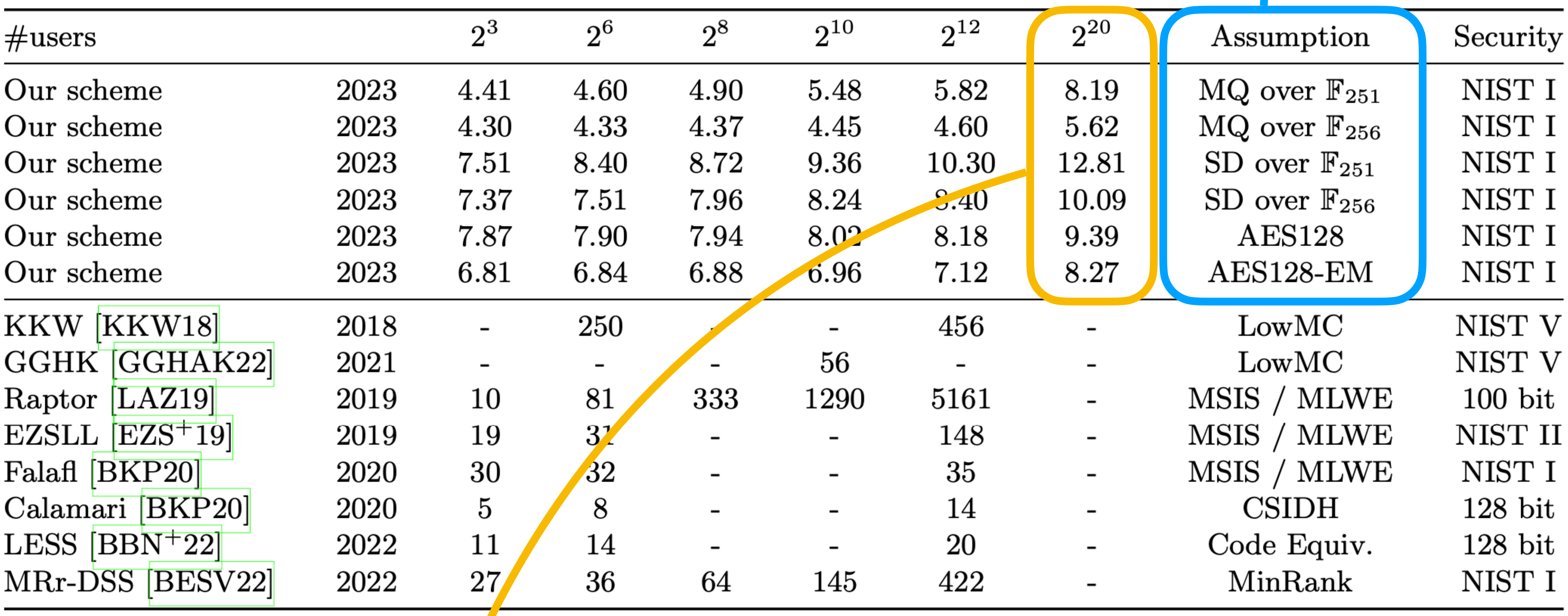
Application to MQ, SD, AES



#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

Post-quantum ring signatures

Application to MQ, SD, AES

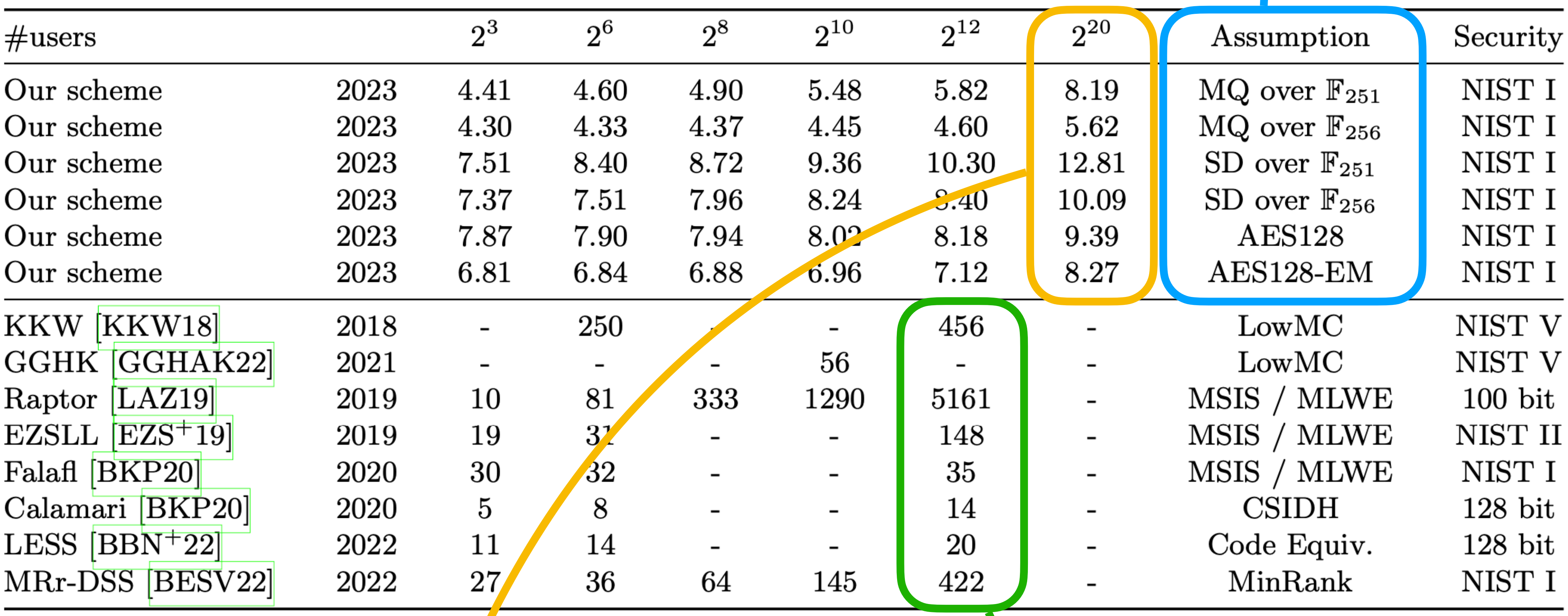


#users		2^3	2^6	2^8	2^{10}	2^{12}	2^{20}	Assumption	Security
Our scheme	2023	4.41	4.60	4.90	5.48	5.82	8.19	MQ over $\mathbb{F}_{251}$	NIST I
Our scheme	2023	4.30	4.33	4.37	4.45	4.60	5.62	MQ over $\mathbb{F}_{256}$	NIST I
Our scheme	2023	7.51	8.40	8.72	9.36	10.30	12.81	SD over $\mathbb{F}_{251}$	NIST 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

Size range: 5–13 kB for |ring|=2^{20}

Post-quantum ring signatures

Application to MQ, SD, AES



#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

Size range: 5–13 kB

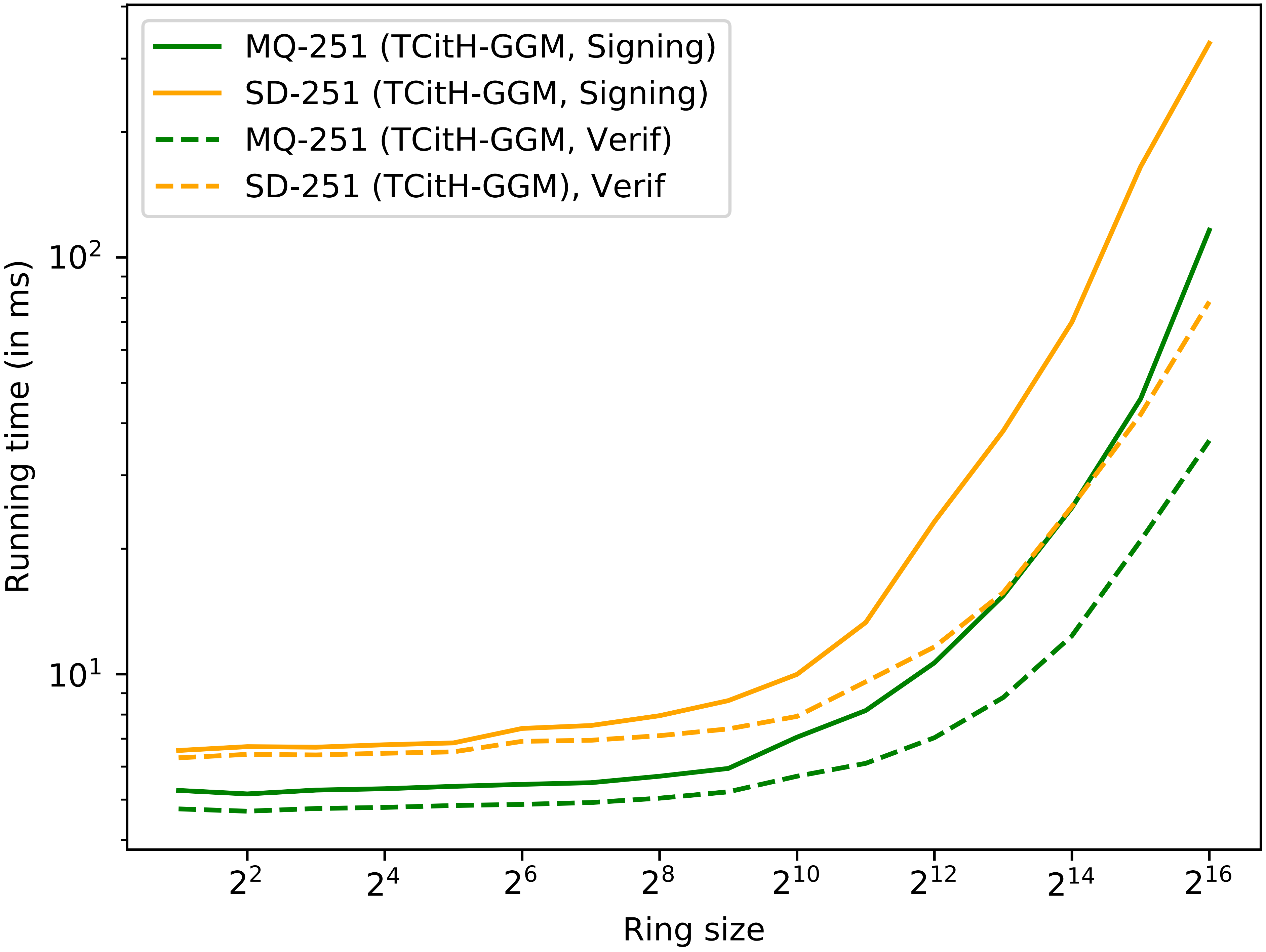
for |ring|=2^{20}

Previous works:

$\geq$ 14 kB for |ring|=2^{10}

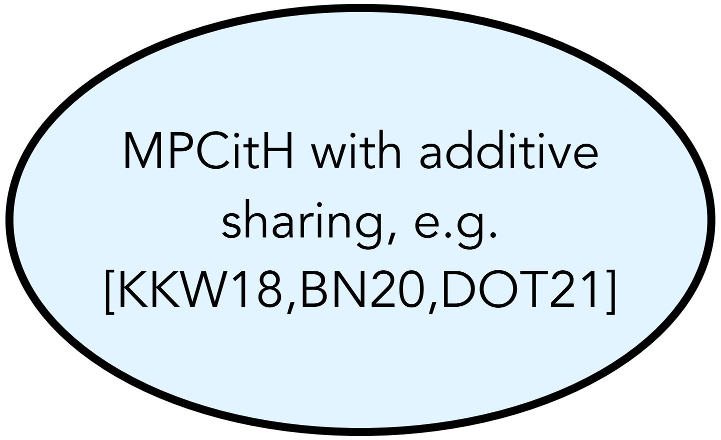
no / slow implementations

Post-quantum ring signatures



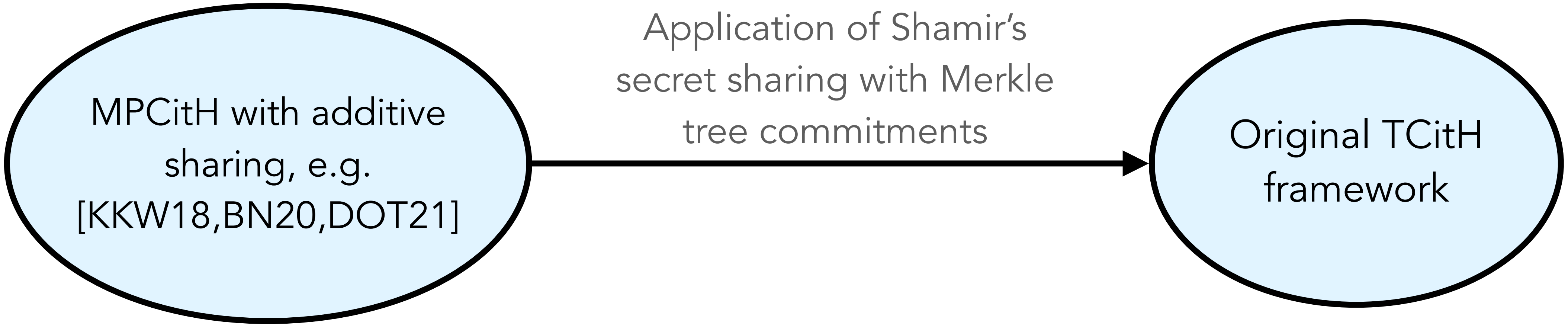
Relation to other proof systems

Connections to other proof systems

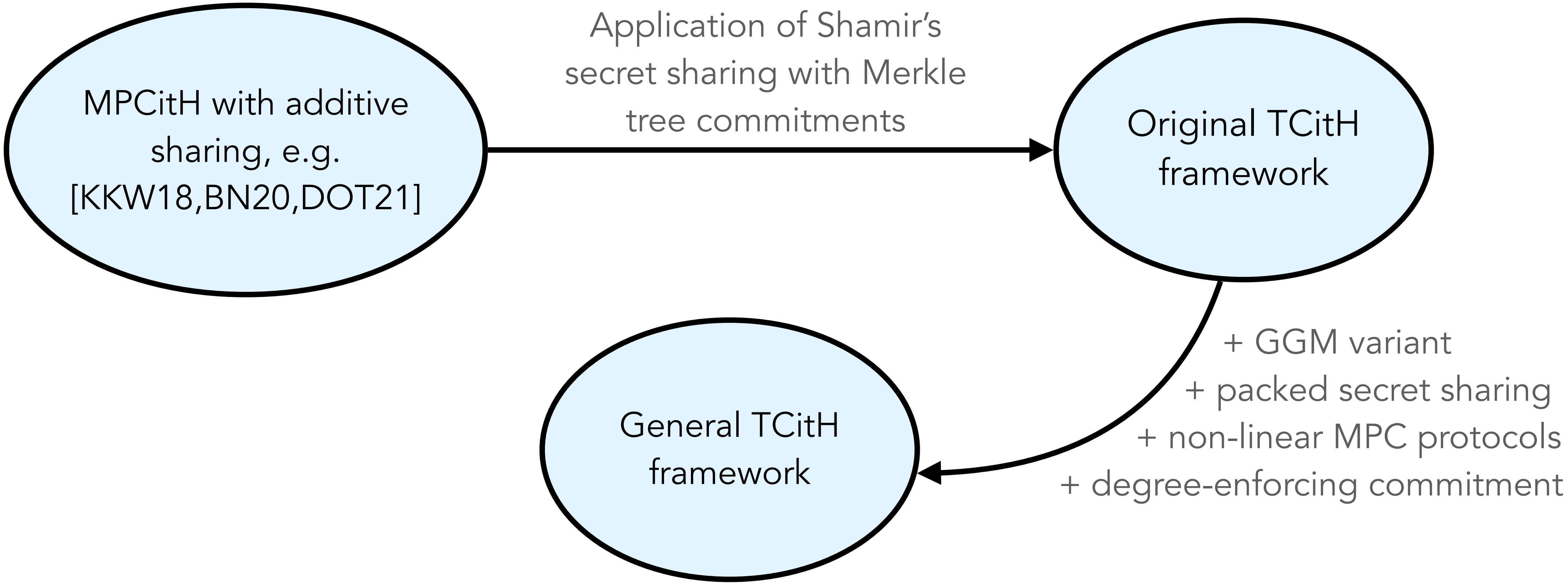


MPCitH with additive sharing, e.g. [KKW18,BN20,DOT21]

Connections to other proof systems


MPCitH with additive sharing, e.g. [KKW18,BN20,DOT21]
Application of Shamir's secret sharing with Merkle tree commitments
Original TCitH framework

Connections to other proof systems



MPCitH with additive sharing, e.g. [KKW18,BN20,DOT21]

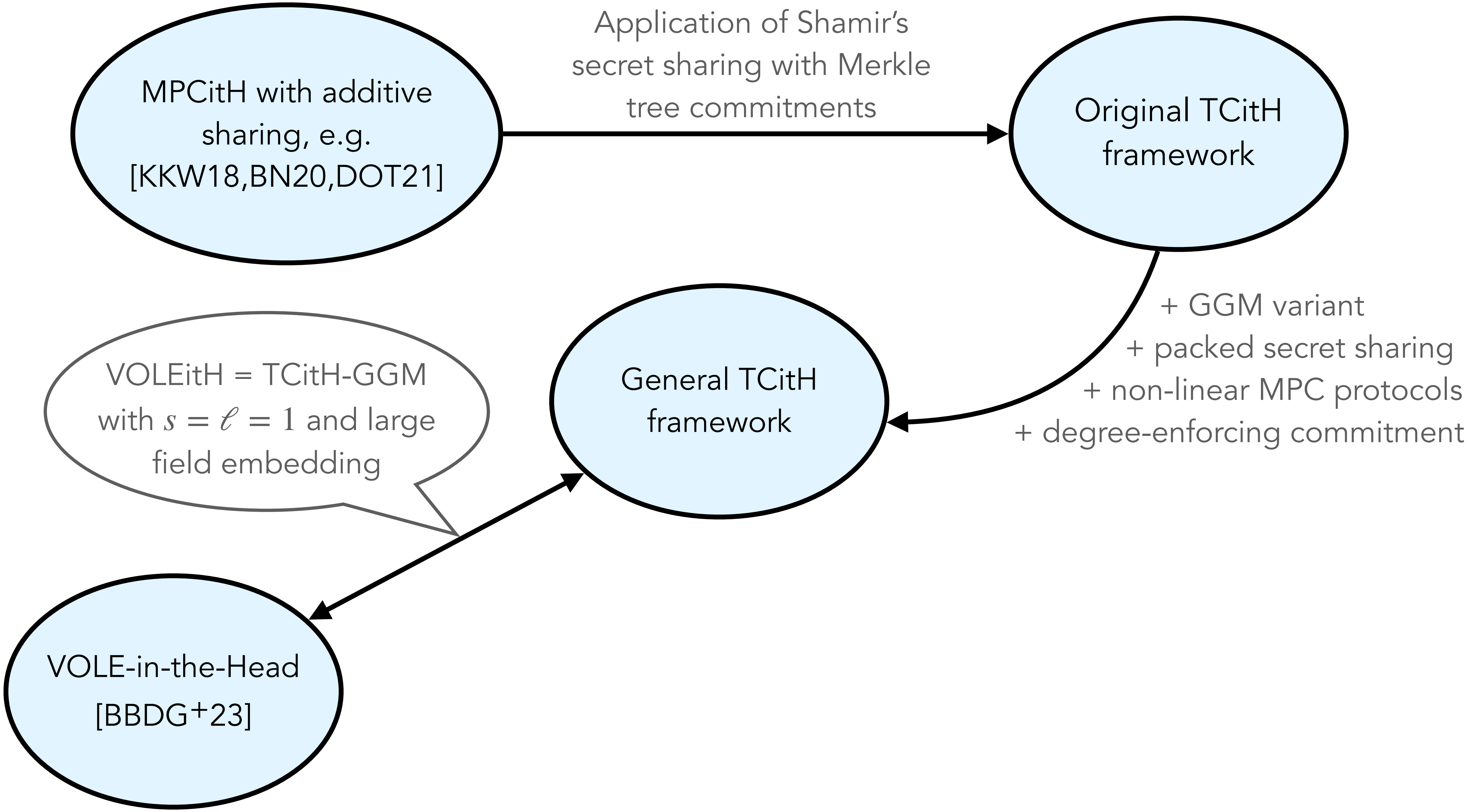
Application of Shamir's secret sharing with Merkle tree commitments

Original TCitH framework

+ GGM variant
+ packed secret sharing
+ non-linear MPC protocols
+ degree-enforcing commitment

General TCitH framework

Connections to other proof systems



MPCitH with additive sharing, e.g. [KKW18,BN20,DOT21]

Application of Shamir's secret sharing with Merkle tree commitments

Original TCitH framework

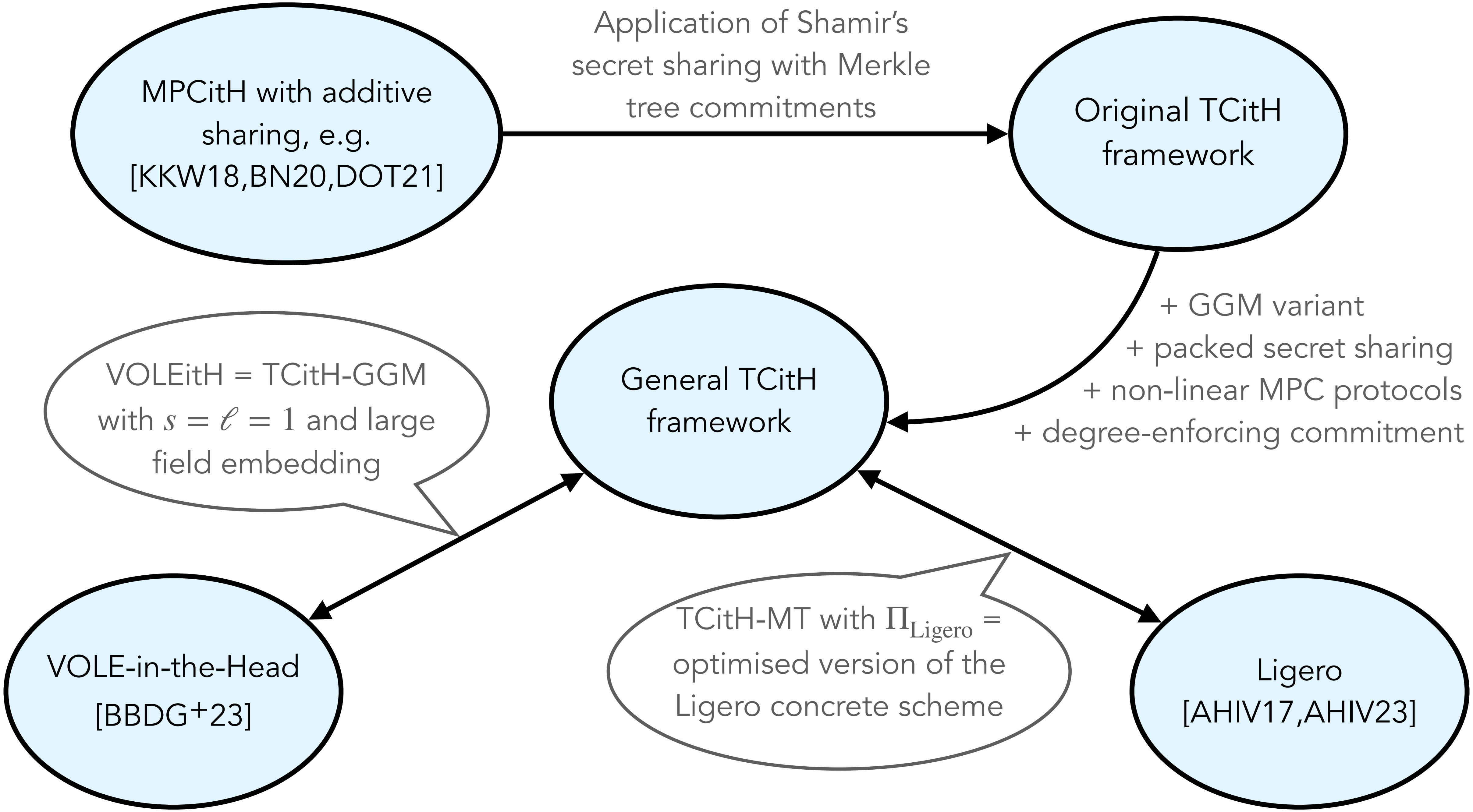
+ GGM variant
+ packed secret sharing
+ non-linear MPC protocols
+ degree-enforcing commitment

General TCitH framework

VOLEitH = TCitH-GGM with $s = \ell = 1$ and large field embedding

VOLE-in-the-Head [BBDG+23]

Connections to other proof systems



MPCitH with additive sharing, e.g. [KKW18,BN20,DOT21]

Application of Shamir's secret sharing with Merkle tree commitments

Original TCitH framework

+ GGM variant
+ packed secret sharing
+ non-linear MPC protocols
+ degree-enforcing commitment

General TCitH framework

VOLEitH = TCitH-GGM with $s = \ell = 1$ and large field embedding

VOLE-in-the-Head [BBDG$^+$23]

TCitH-MT with Π_{Ligero} = optimised version of the Ligero concrete scheme

Ligero [AHIV17,AHIV23]

Thank you!

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)