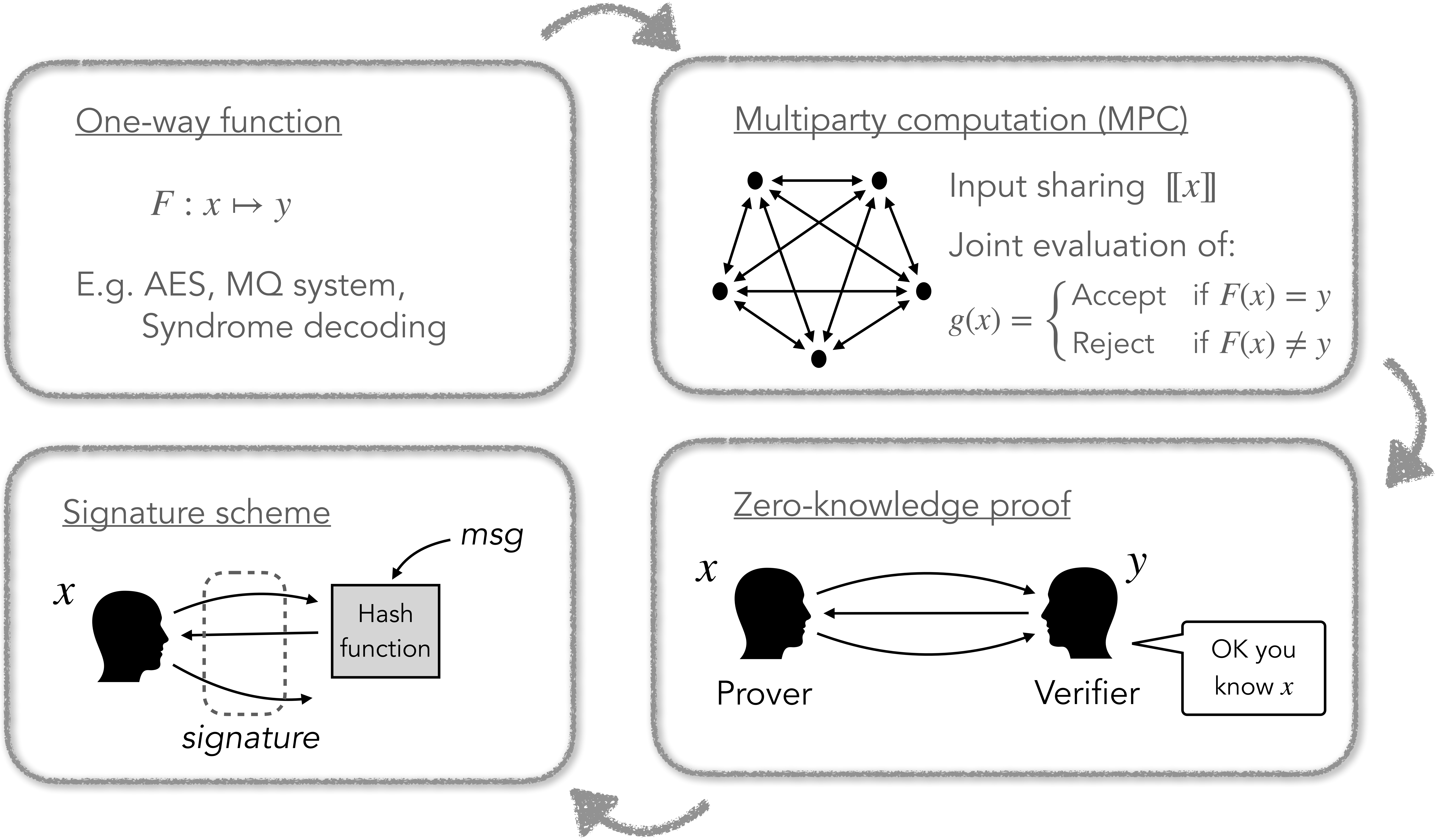
Post-Quantum Signatures from MPC in the Head

Matthieu Rivain

PQ-TLS Summer School

Jun 19, 2024, Anglet

CryptoExperts
WE INNOVATE TO SECURE YOUR BUSINESS

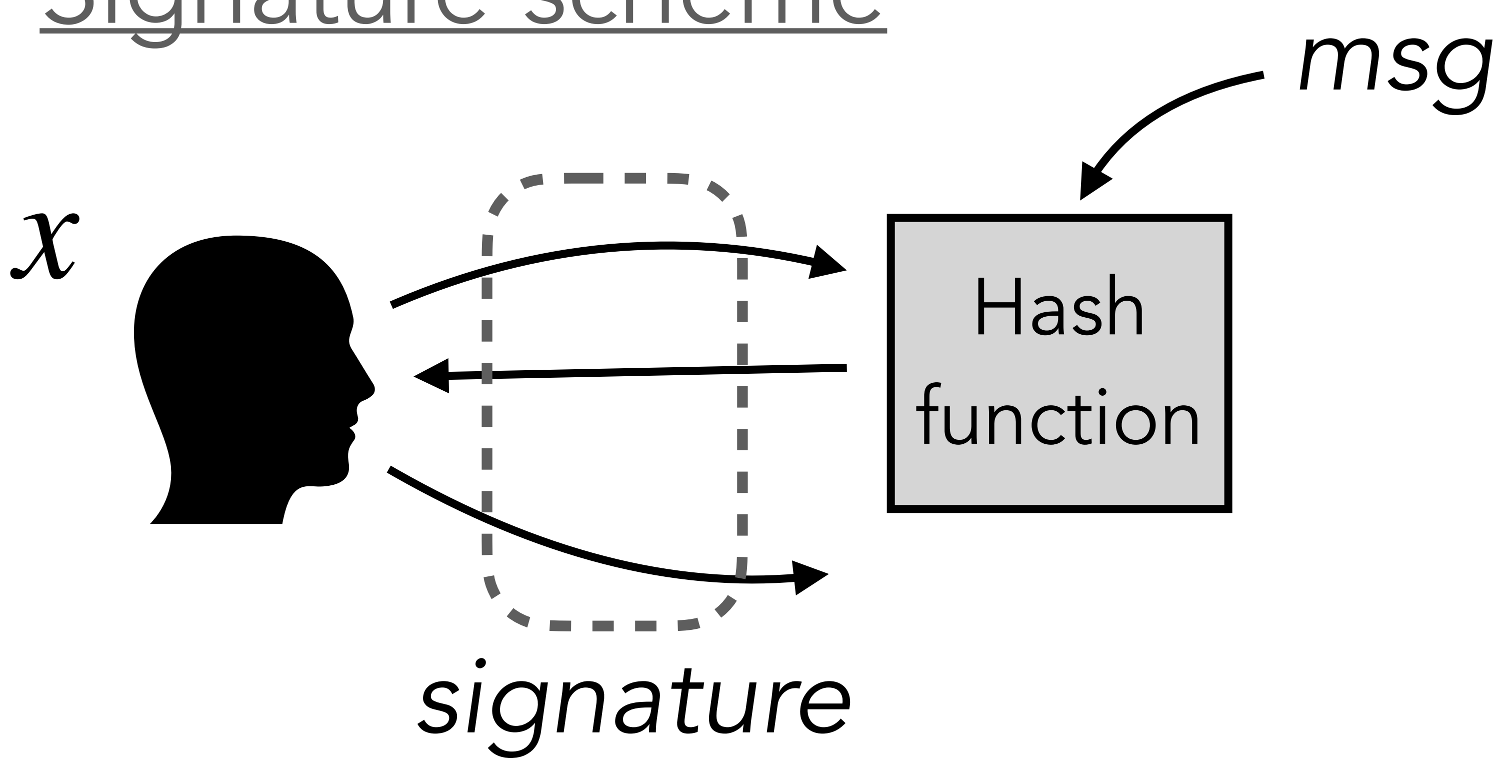

One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Zero-knowledge proof
x
y
Prover
Verifier
OK you know x
Signature scheme
x
msg
Hash function
signature

One-way function

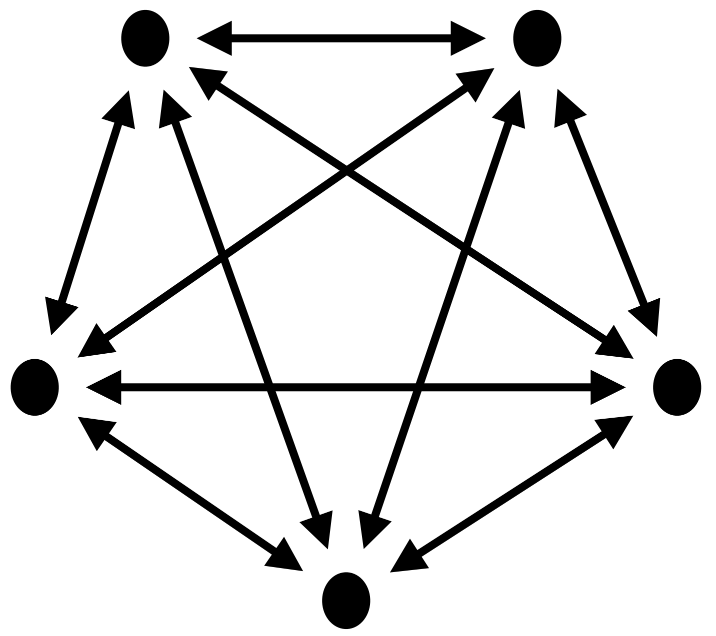
$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Signature scheme



Multiparty computation (MPC)



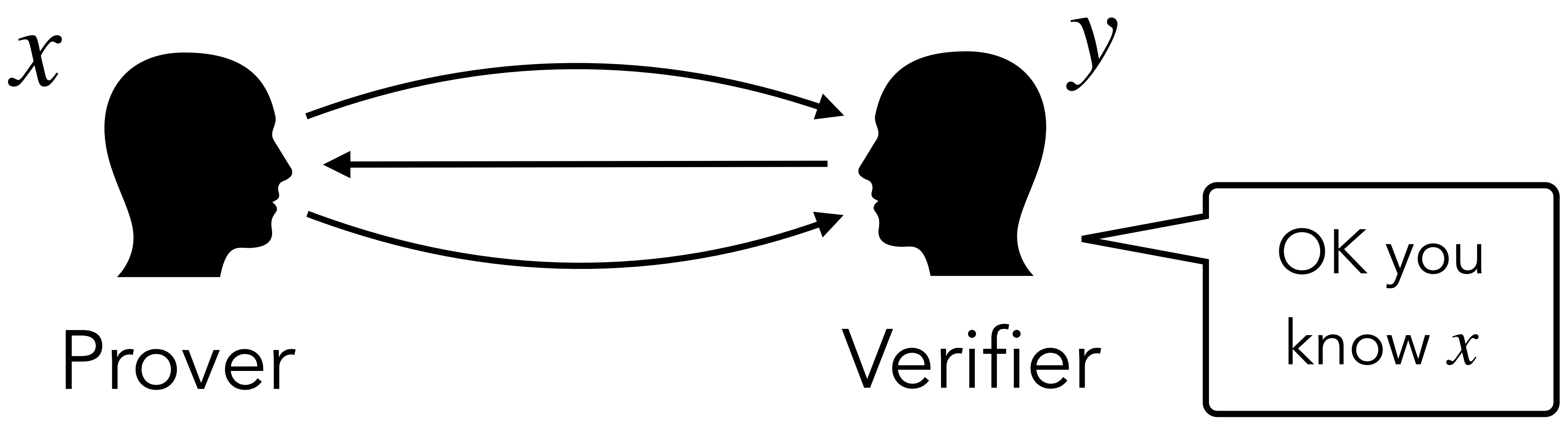
Input sharing $[\![x]\!]$

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

Zero-knowledge proof

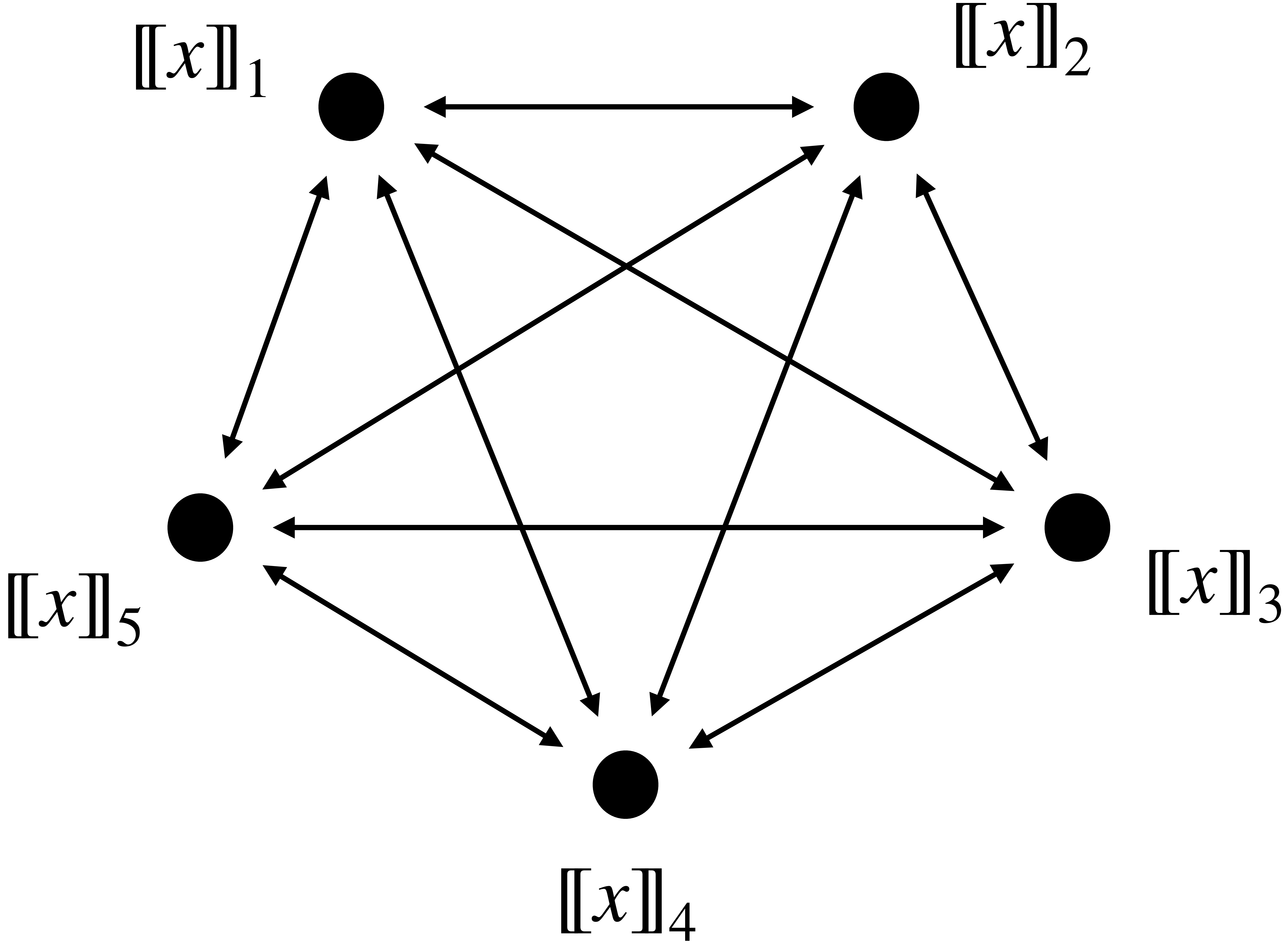


Roadmap

- MPC-in-the-Head with Additive Secret Sharing
- Optimisations
- SDitH Signature Scheme: MPCitH with Syndrome Decoding
- MPC-in-the-Head with Threshold Secret Sharing

MPC-in-the-Head with Additive Secret Sharing

MPC model

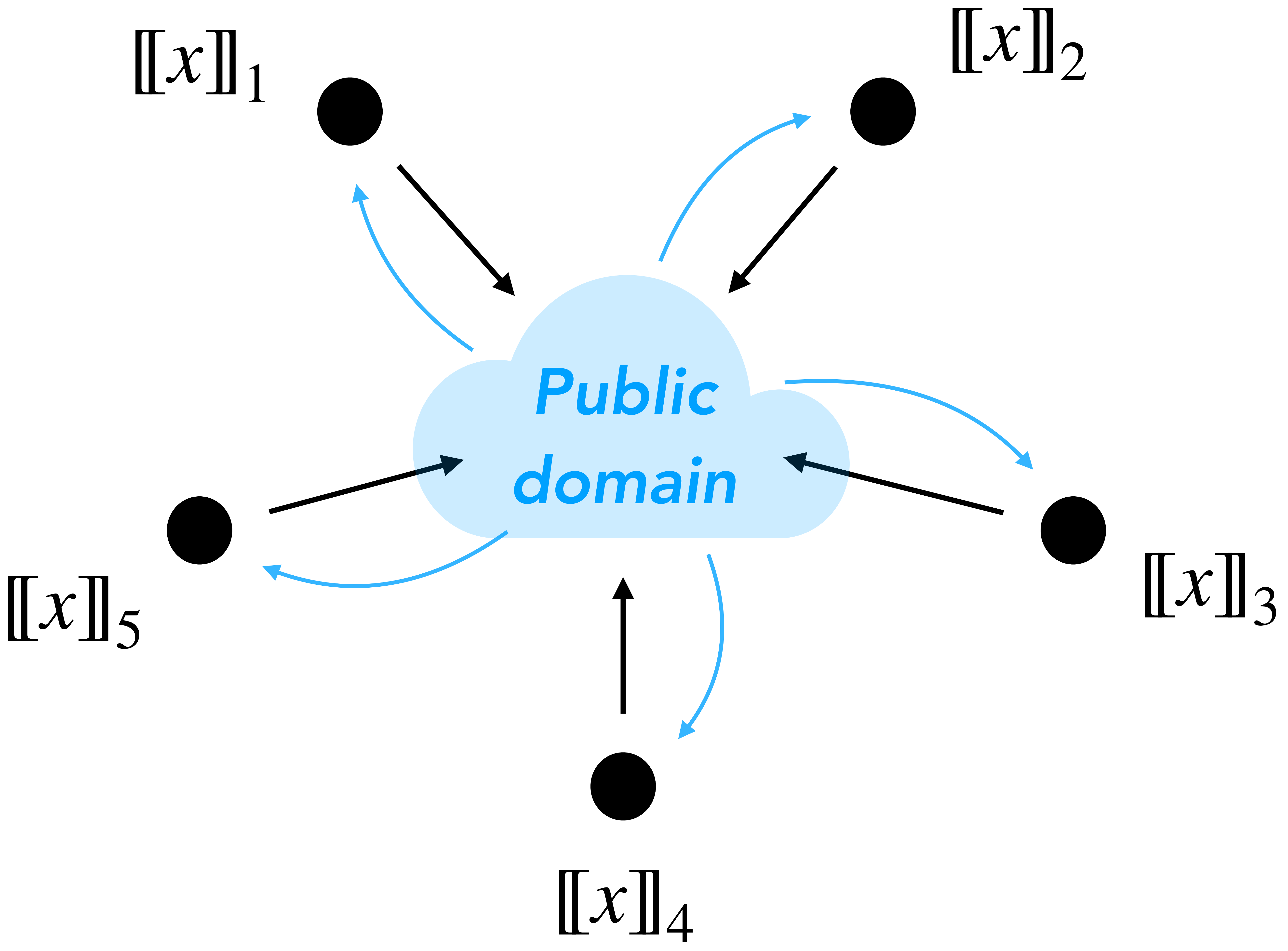


- **Jointly compute**

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

- $(N-1)$ **private:** the views of any $N-1$ parties provide no information on x
- **Semi-honest model:** assuming that the parties follow the steps of the protocol

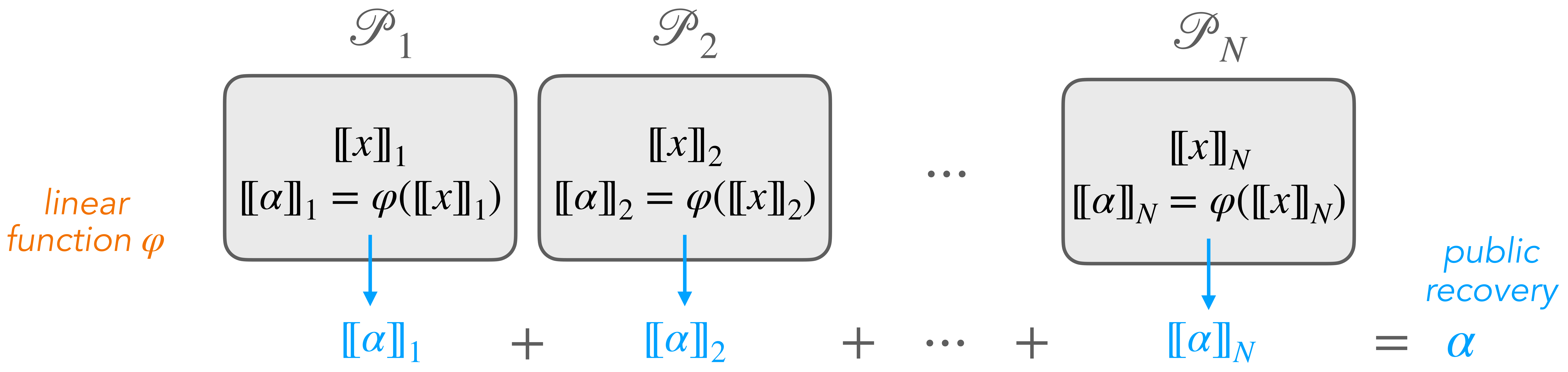
MPC model

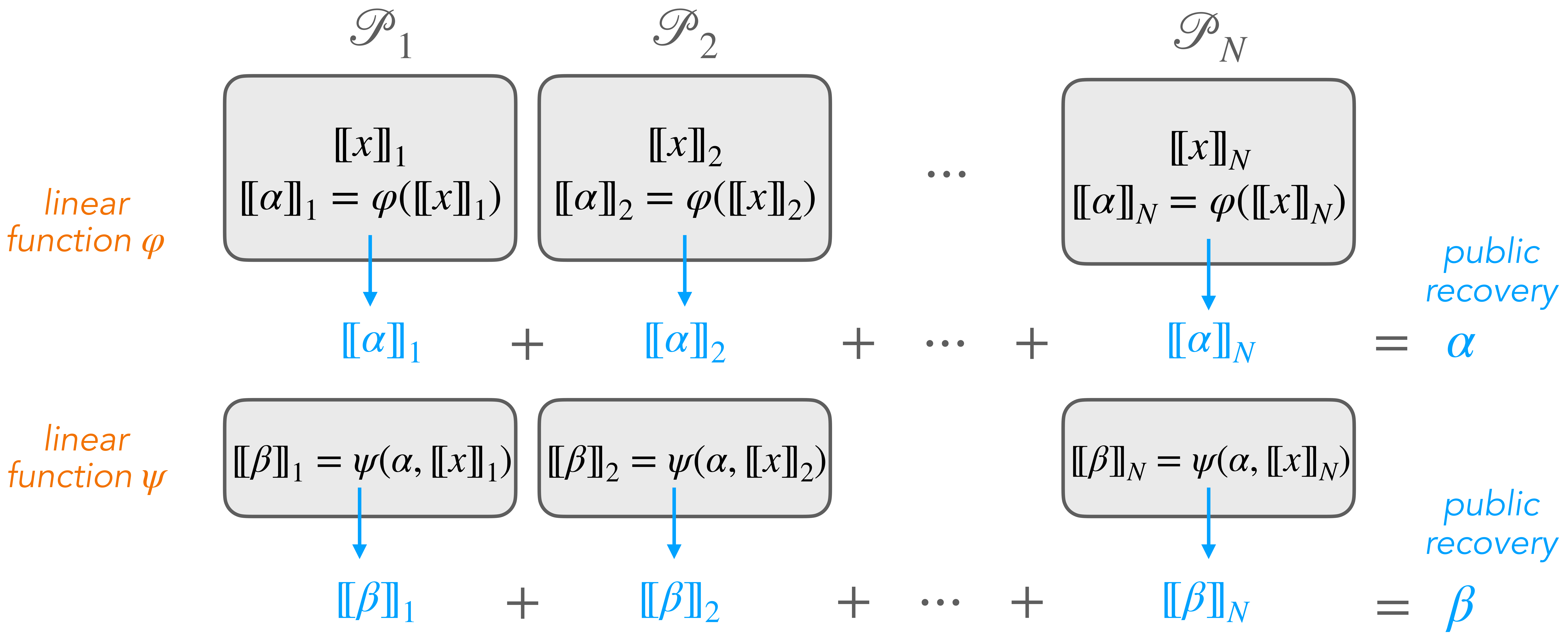


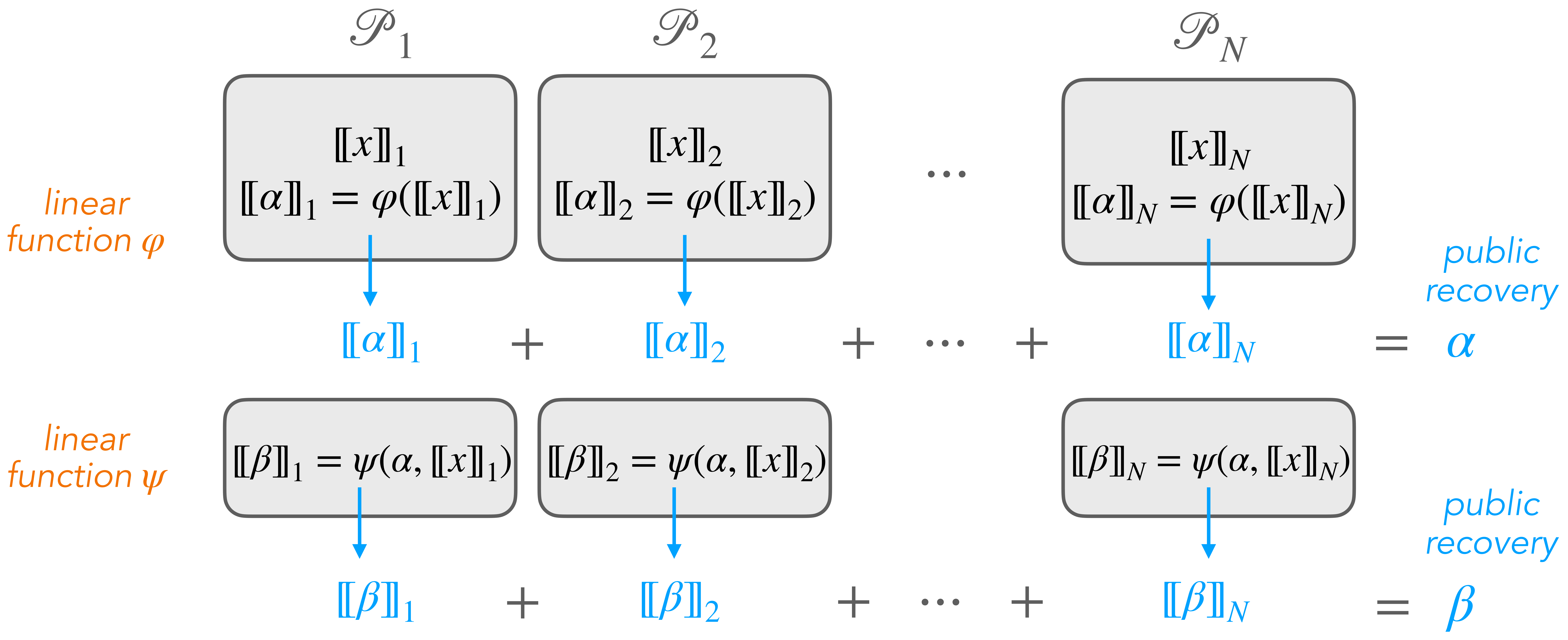
- **Jointly compute**

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

- $(N-1)$ **private:** the views of any $N-1$ parties provide no information on x
- **Semi-honest model:** assuming that the parties follow the steps of the protocol
- ***Broadcast model***
  - *Parties locally compute on their shares* $[\![x]\!] \mapsto [\![\alpha]\!]$
  - *Parties broadcast* $[\![\alpha]\!]$ *and recompute* α
  - *Parties start again (now knowing* α*)*


$\mathscr{P}_1$
$\mathscr{P}_2$
$\mathscr{P}_N$
$[\![x]\!]_1$
$[\![\alpha]\!]_1 = \varphi([\![x]\!]_1)$
$[\![x]\!]_2$
$[\![\alpha]\!]_2 = \varphi([\![x]\!]_2)$
$\cdots$
$[\![x]\!]_N$
$[\![\alpha]\!]_N = \varphi([\![x]\!]_N)$
linear function φ
public recovery
$[\![\alpha]\!]_1 + [\![\alpha]\!]_2 + \cdots + [\![\alpha]\!]_N = \alpha$


$\mathscr{P}_1$
$\mathscr{P}_2$
$\mathscr{P}_N$
linear function φ
$[\![x]\!]_1$
$[\![\alpha]\!]_1 = \varphi([\![x]\!]_1)$
$[\![x]\!]_2$
$[\![\alpha]\!]_2 = \varphi([\![x]\!]_2)$
$\cdots$
$[\![x]\!]_N$
$[\![\alpha]\!]_N = \varphi([\![x]\!]_N)$
public recovery
$[\![\alpha]\!]_1 + [\![\alpha]\!]_2 + \cdots + [\![\alpha]\!]_N = \alpha$
linear function ψ
$[\![\beta]\!]_1 = \psi(\alpha, [\![x]\!]_1)$
$[\![\beta]\!]_2 = \psi(\alpha, [\![x]\!]_2)$
$[\![\beta]\!]_N = \psi(\alpha, [\![x]\!]_N)$
public recovery
$[\![\beta]\!]_1 + [\![\beta]\!]_2 + \cdots + [\![\beta]\!]_N = \beta$



and so on…

$$g : (y, \alpha, \beta, \ldots) \mapsto \begin{cases} \text{Accept} \\ \text{Reject} \end{cases}$$

Example: matrix multiplication $y = Hx$

$\mathscr{P}_1$ | $\mathscr{P}_2$ | $\cdots$ | $\mathscr{P}_N$

$[\![x]\!]_1$, $[\![\alpha]\!]_1 = H \cdot [\![x]\!]_1$

$[\![x]\!]_2$, $[\![\alpha]\!]_2 = H \cdot [\![x]\!]_2$

$[\![x]\!]_N$, $[\![\alpha]\!]_N = H \cdot [\![x]\!]_N$

mult. by H is linear

public recovery

$$[\![\alpha]\!]_1 + [\![\alpha]\!]_2 + \cdots + [\![\alpha]\!]_N = \alpha$$

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

$$g(y, \alpha) = \text{Accept} \iff Hx = y$$

MPCitH transform

Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

$\cdots$

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

$\longrightarrow$

Prover

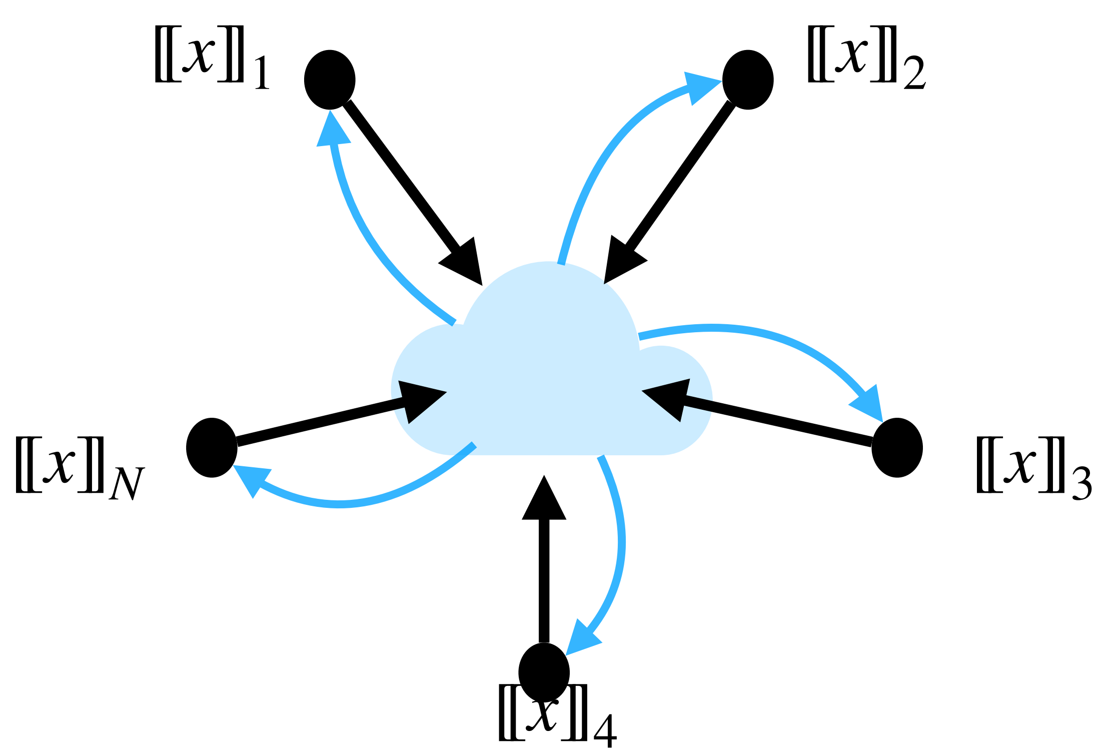
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\cdots$

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

send broadcast

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

Prover

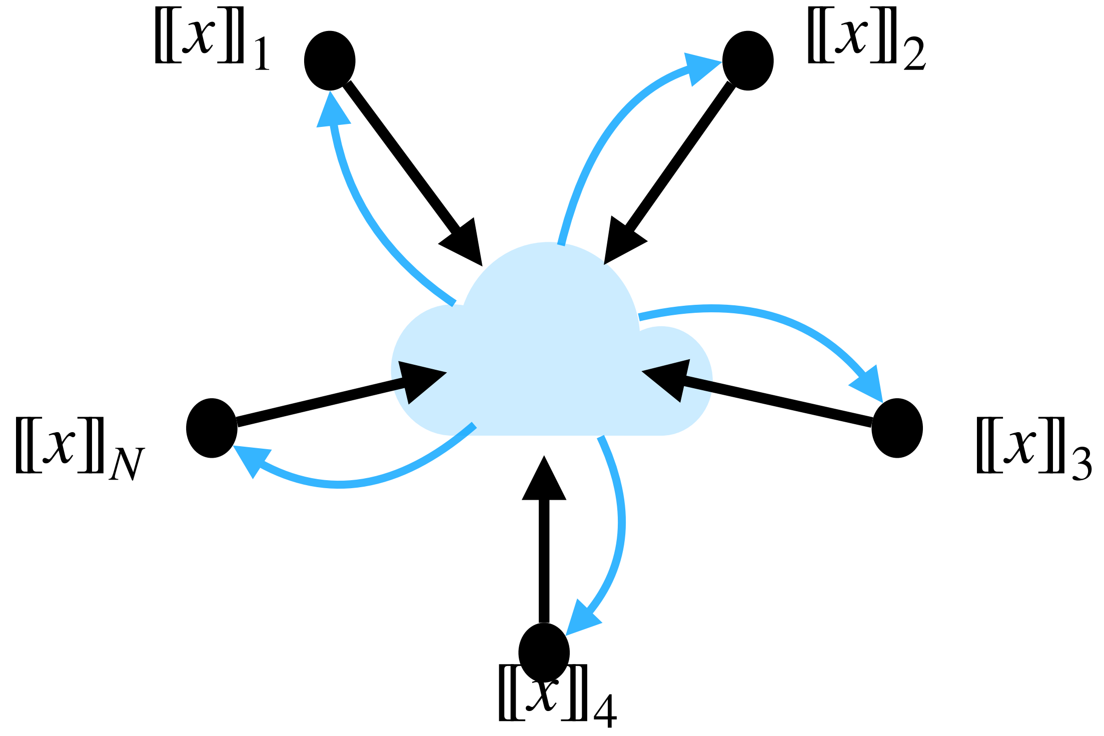
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\cdots$

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

③ Chose a random party

$i^* \xleftarrow{\$} \{1,\ldots,N\}$

i^*

$\longleftarrow$

Prover

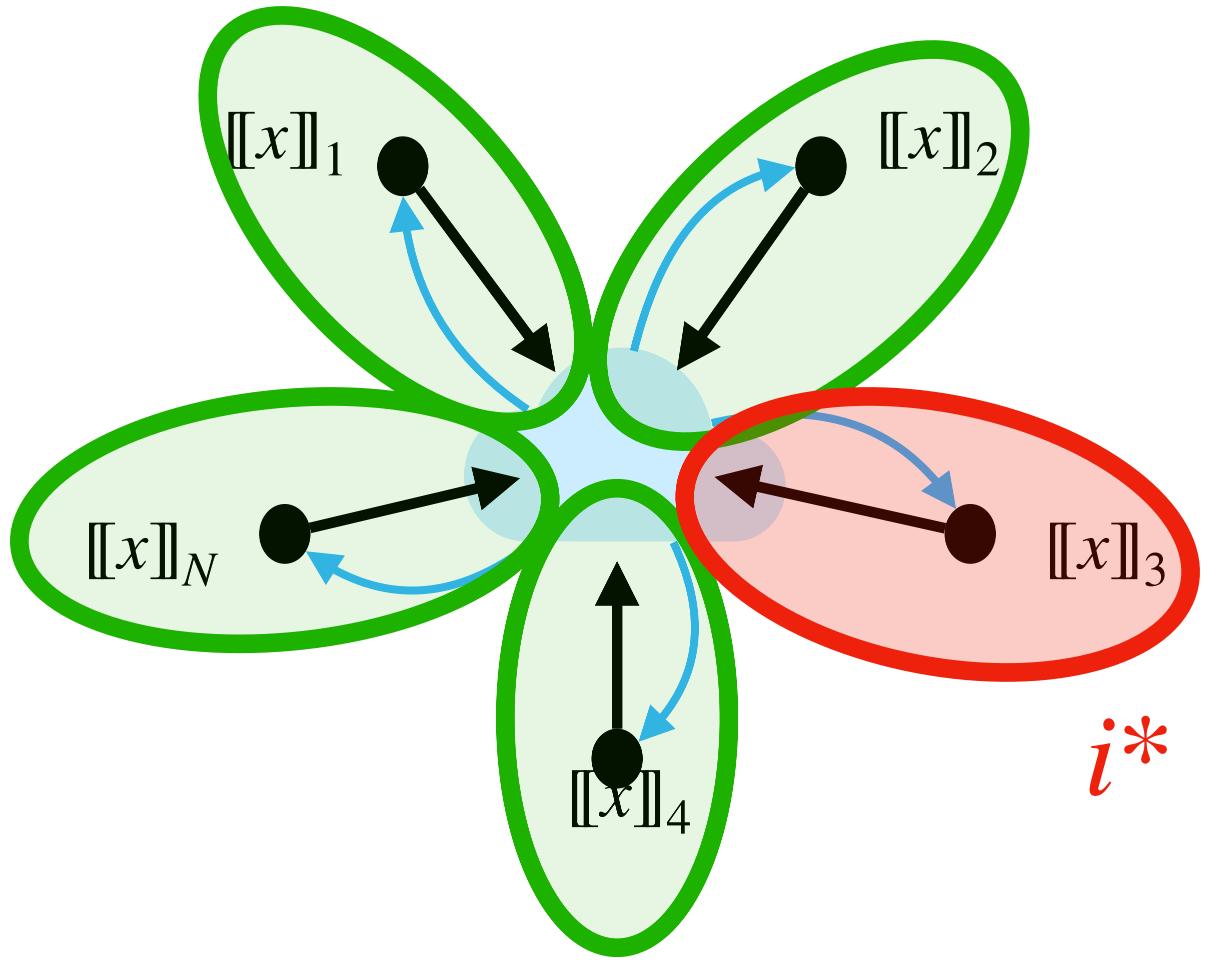
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



④ Open parties $\{1,\ldots,N\}\backslash\{i^*\}$

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

$\cdots$

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

send broadcast

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

← i^*

$([\![x]\!]_i, \rho_i)_{i \neq i^*}$ →

③ Chose a random party

$i^* \xleftarrow{\$} \{1,\ldots,N\}$

Prover

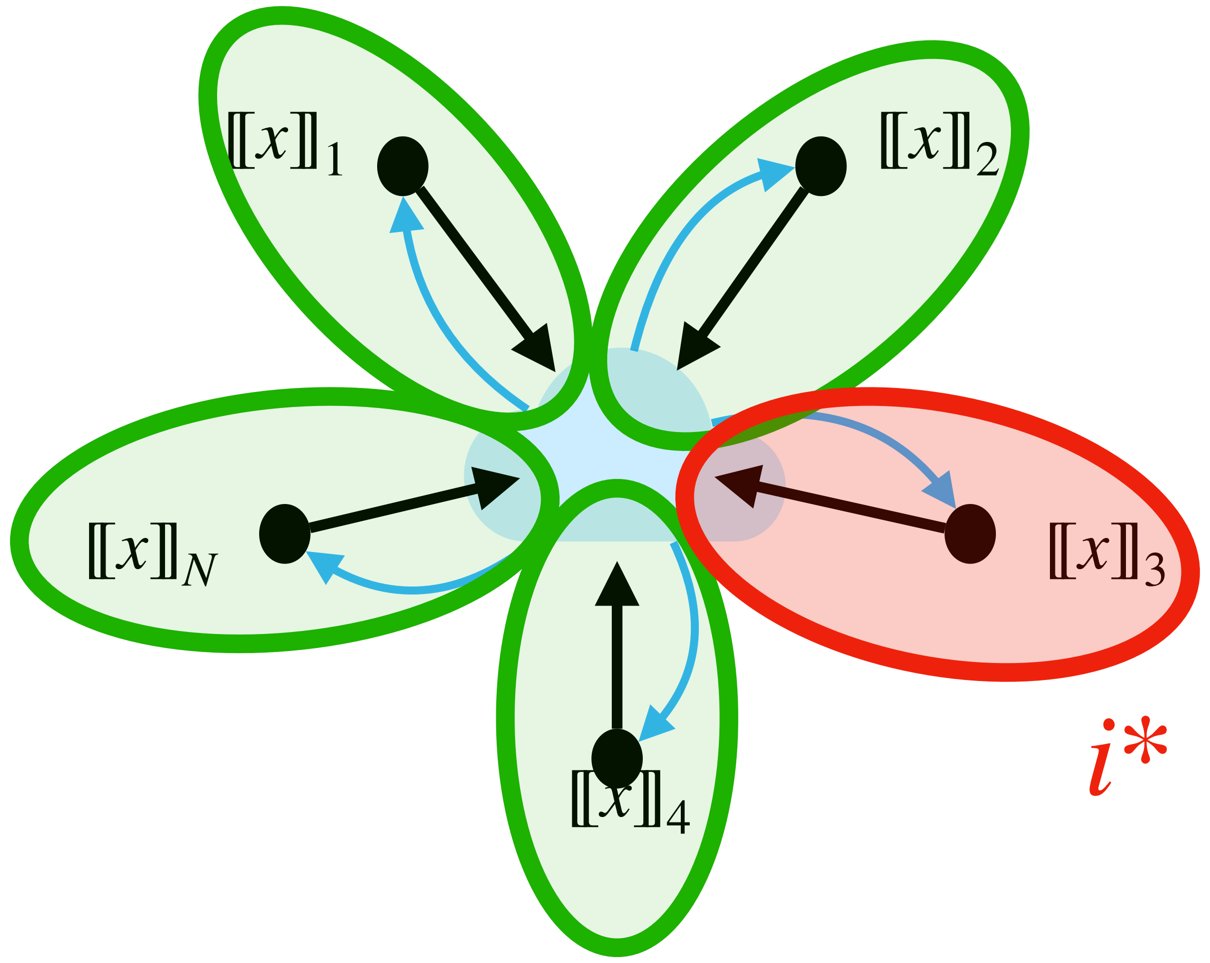
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



④ Open parties $\{1,\ldots,N\}\backslash\{i^*\}$

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

$\cdots$

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

send broadcast

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

i^*

$([\![x]\!]_i, \rho_i)_{i \neq i^*}$

③ Chose a random party

$i^* \xleftarrow{\$} \{1,\ldots,N\}$

⑤ Check $\forall i \neq i^*$

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

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

Prover

Verifier

MPCitH transform

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

MPCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private
- **Soundness**
  - if $g(y, \alpha) \neq \text{Accept}$ → Verifier rejects

MPCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private
- **Soundness**
  - if $g(y, \alpha) \neq$ Accept → Verifier rejects
  - if $g(y, \alpha) =$ Accept, then
    - either $[\![x]\!]$ = sharing of correct witness $F(x) = y$

      → Prover honest

MPCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private
- **Soundness**
  - if $g(y, \alpha) \neq$ Accept → Verifier rejects
  - if $g(y, \alpha) =$ Accept, then
    - either $[\![x]\!]$ = sharing of correct witness $F(x) = y$

      → Prover honest
    - or Prover has cheated for at least one party

      → Cheat undetected with proba $\dfrac{1}{N}$

MPCitH transform

- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private
- **Soundness**
  - if $g(y, \alpha) \neq$ Accept → Verifier rejects
  - if $g(y, \alpha) =$ Accept, then
    - either $[\![x]\!]$ = sharing of correct witness $F(x) = y$

      → Prover honest
    - or Prover has cheated for at least one party

      → Cheat undetected with proba $\frac{1}{N}$

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

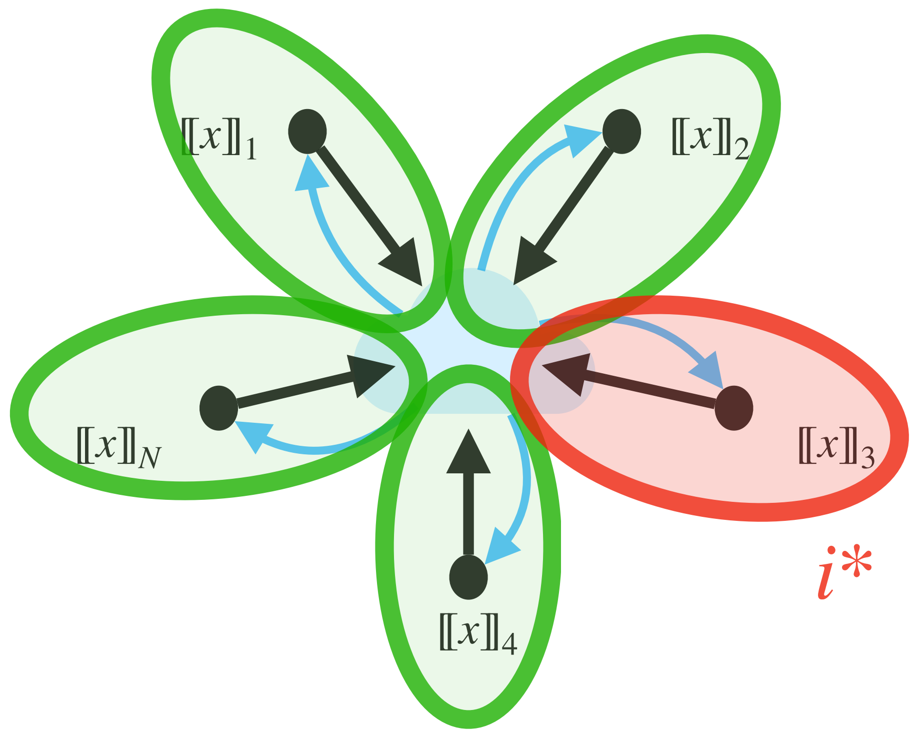
Soundness error

Example: matrix multiplication $y = Hx$

$\mathcal{P}_1$	$\mathcal{P}_2$		$\mathcal{P}_N$
$[\![x]\!]_1$ $[\![\alpha]\!]_1 = H \cdot [\![x]\!]_1$	$[\![x]\!]_2$ $[\![\alpha]\!]_2 = H \cdot [\![x]\!]_2$	$\cdots$	$[\![x]\!]_N$ $[\![\alpha]\!]_N = H \cdot [\![x]\!]_N$

$$[\![\alpha]\!]_1 \; + \; [\![\alpha]\!]_2 \; + \cdots + \; [\![\alpha]\!]_N \; = \; \alpha$$

Prover



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

$$\xrightarrow{\{[\![\alpha]\!]_i\}}$$

$$\xleftarrow{i^*}$$

$$\xrightarrow{\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}}$$

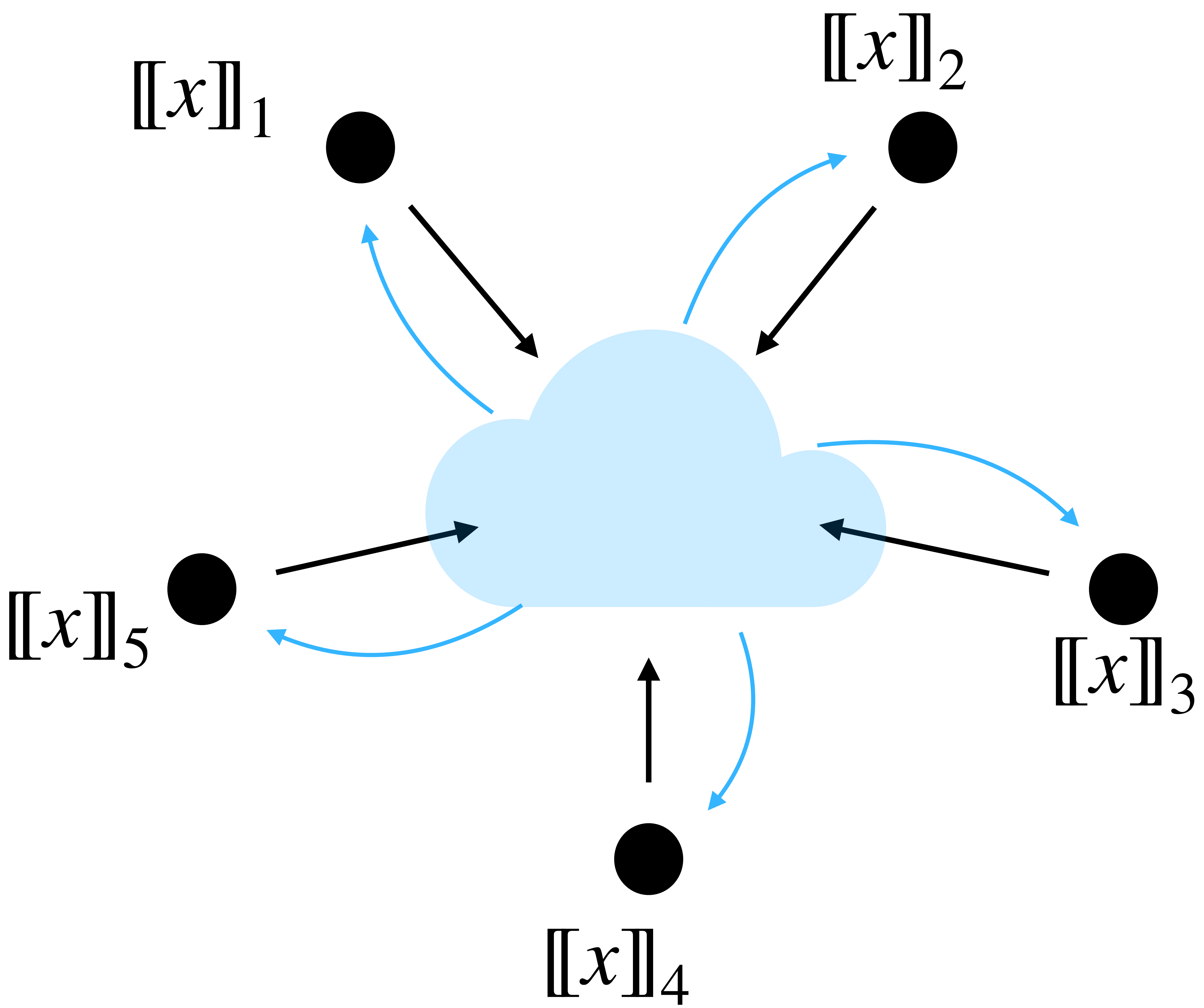
Verifier

Check $\forall i \neq i^*$

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

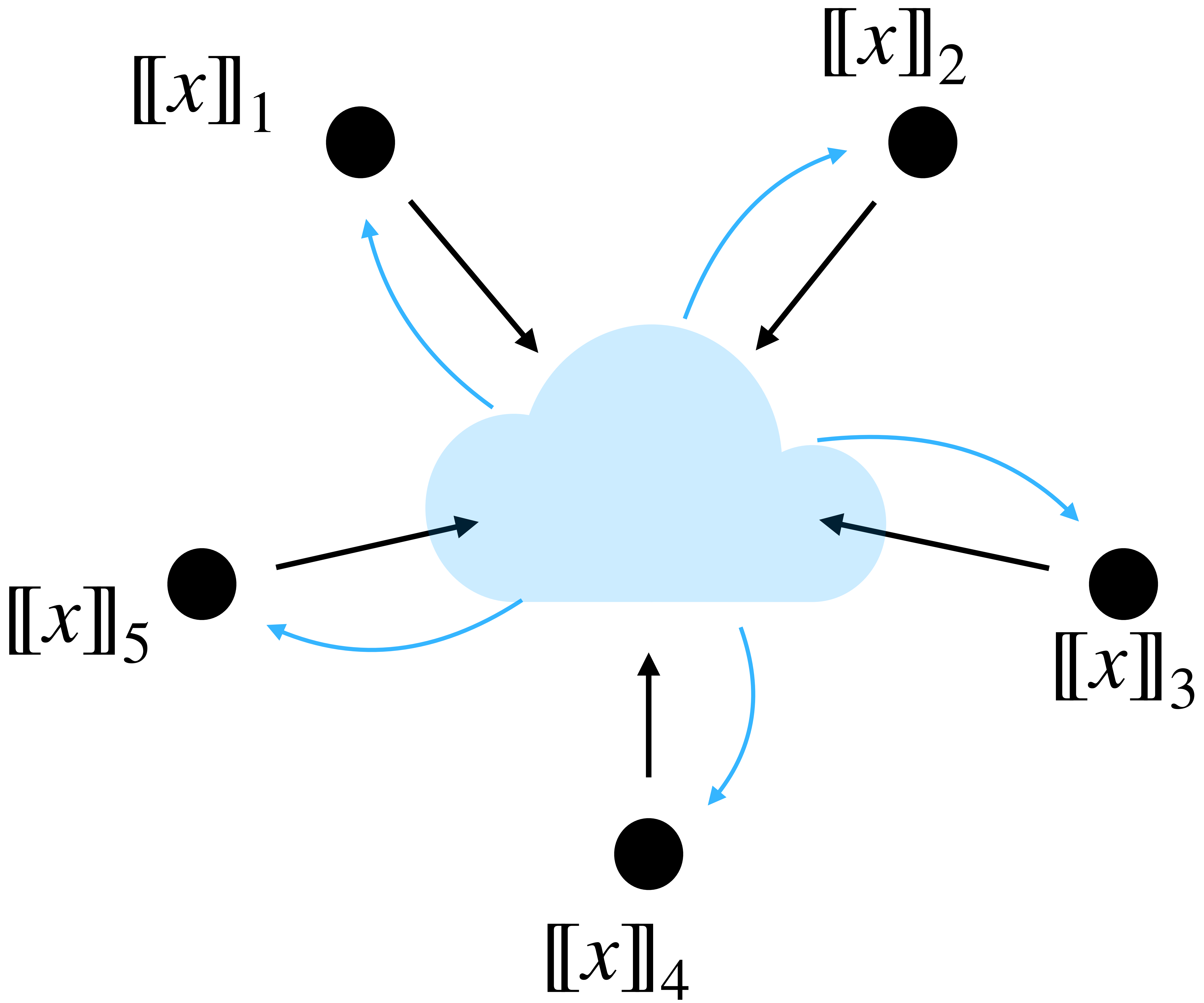
Check $\alpha := \Sigma_i [\![\alpha]\!]_i \; = \; y$

Complete MPC model

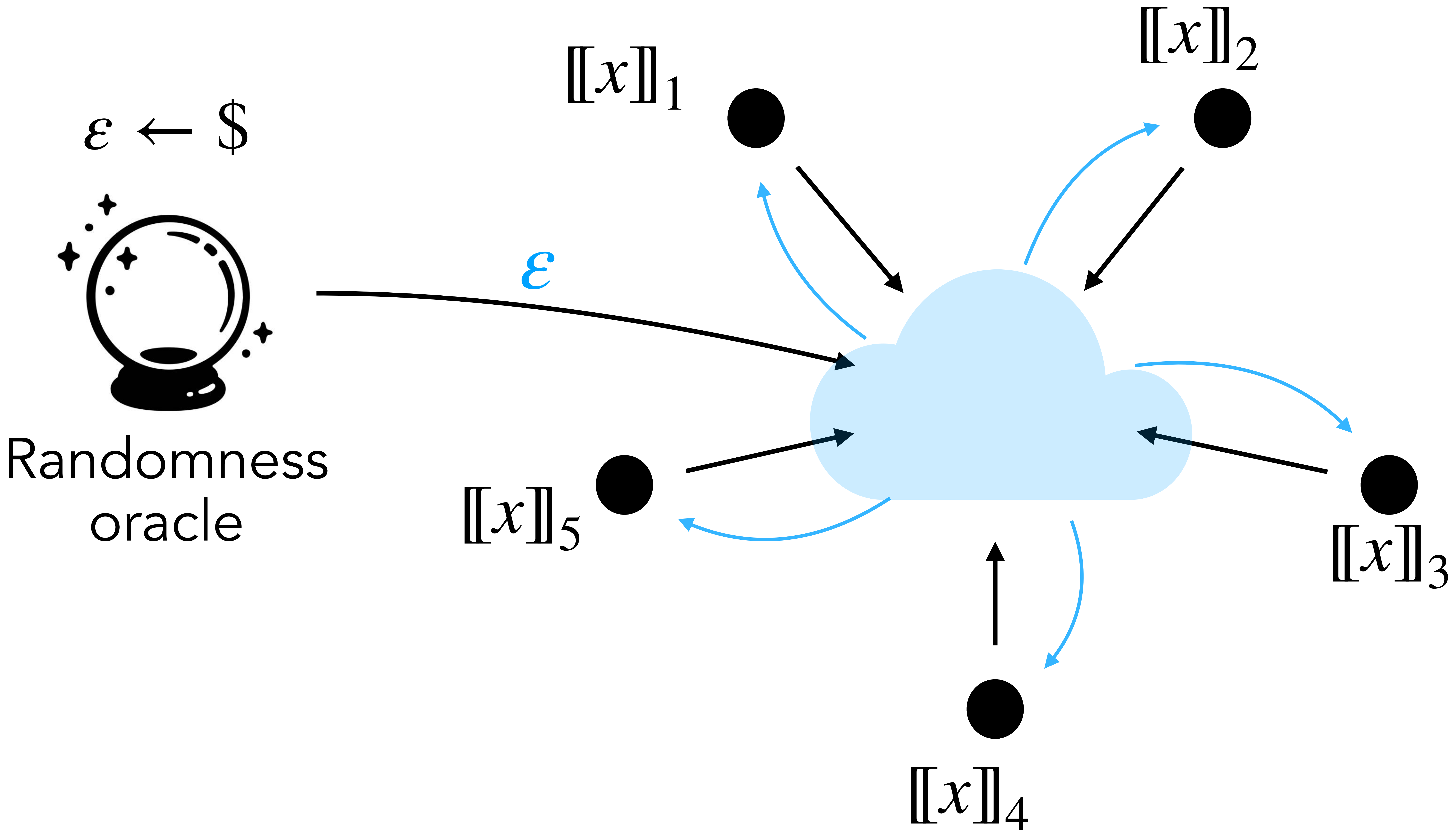


Complete MPC model

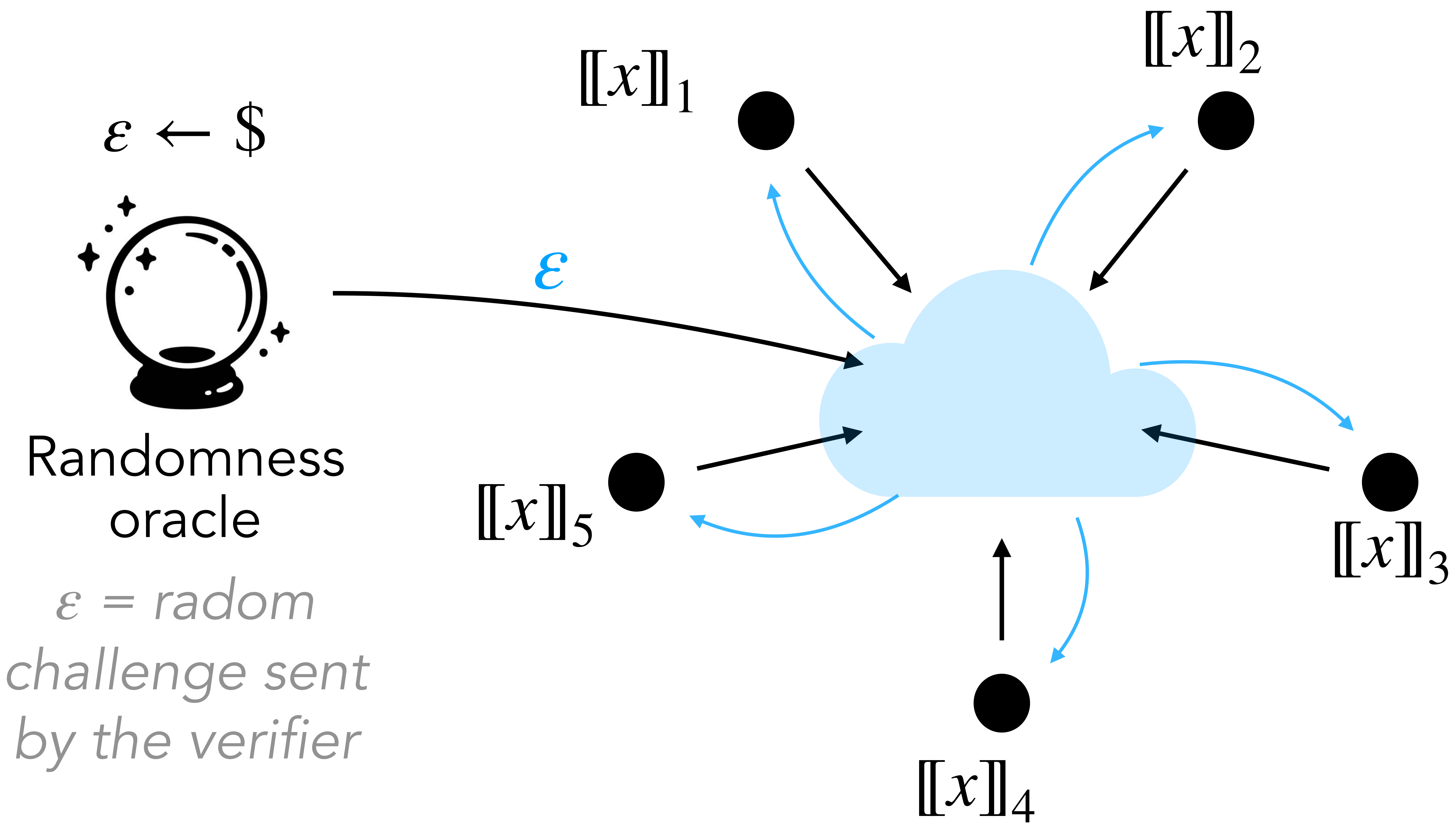
Randomness
oracle



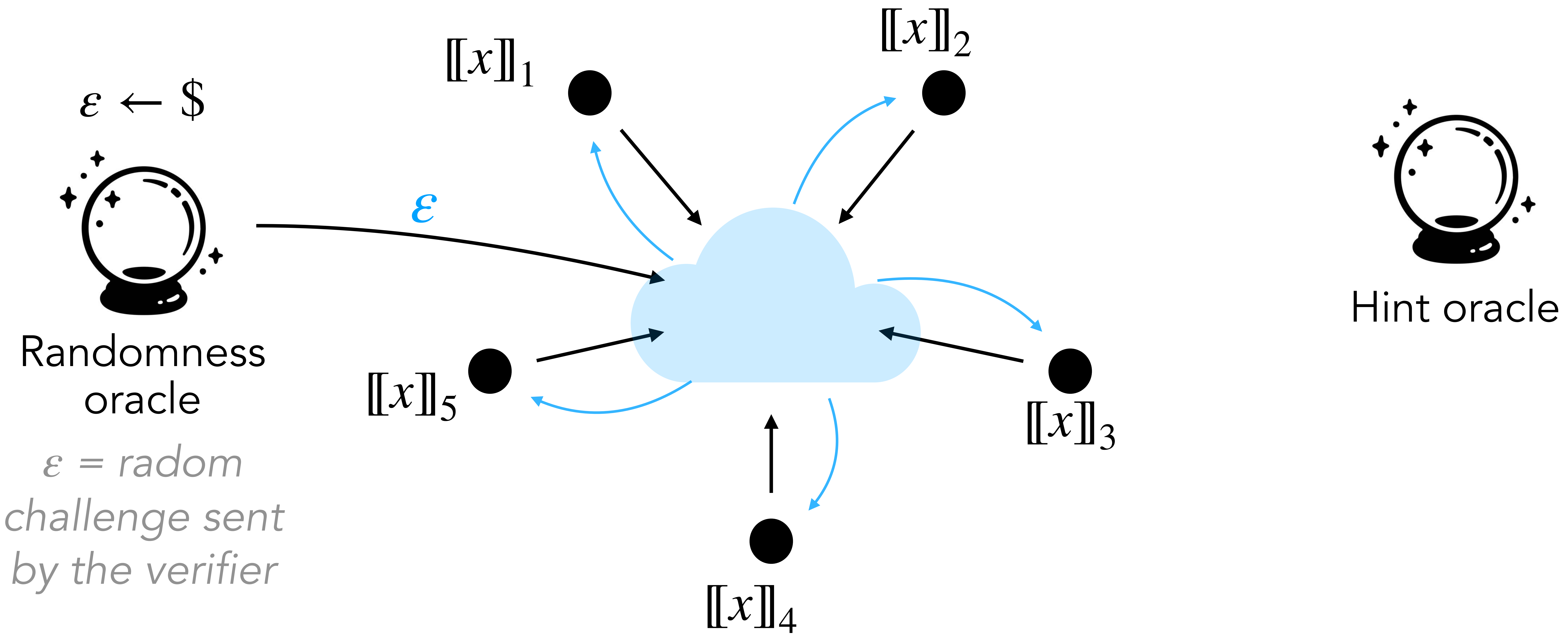
Complete MPC model



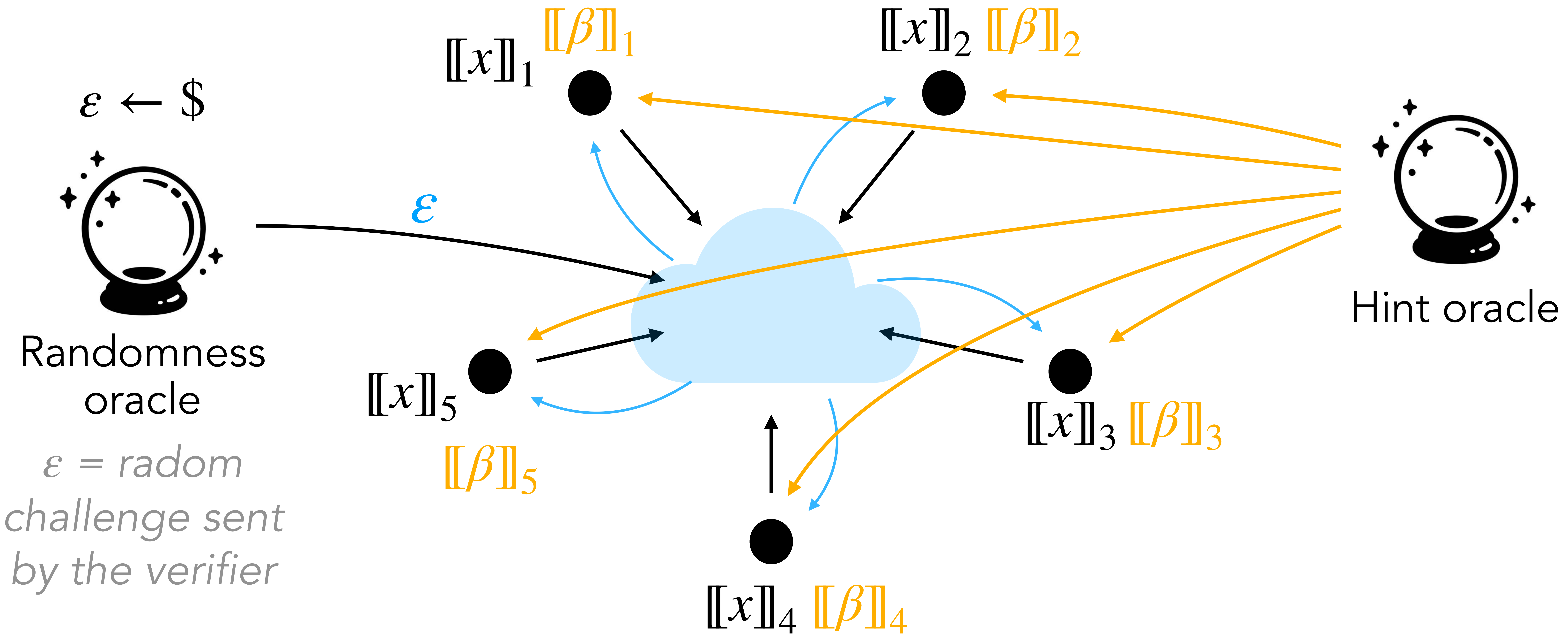
Complete MPC model



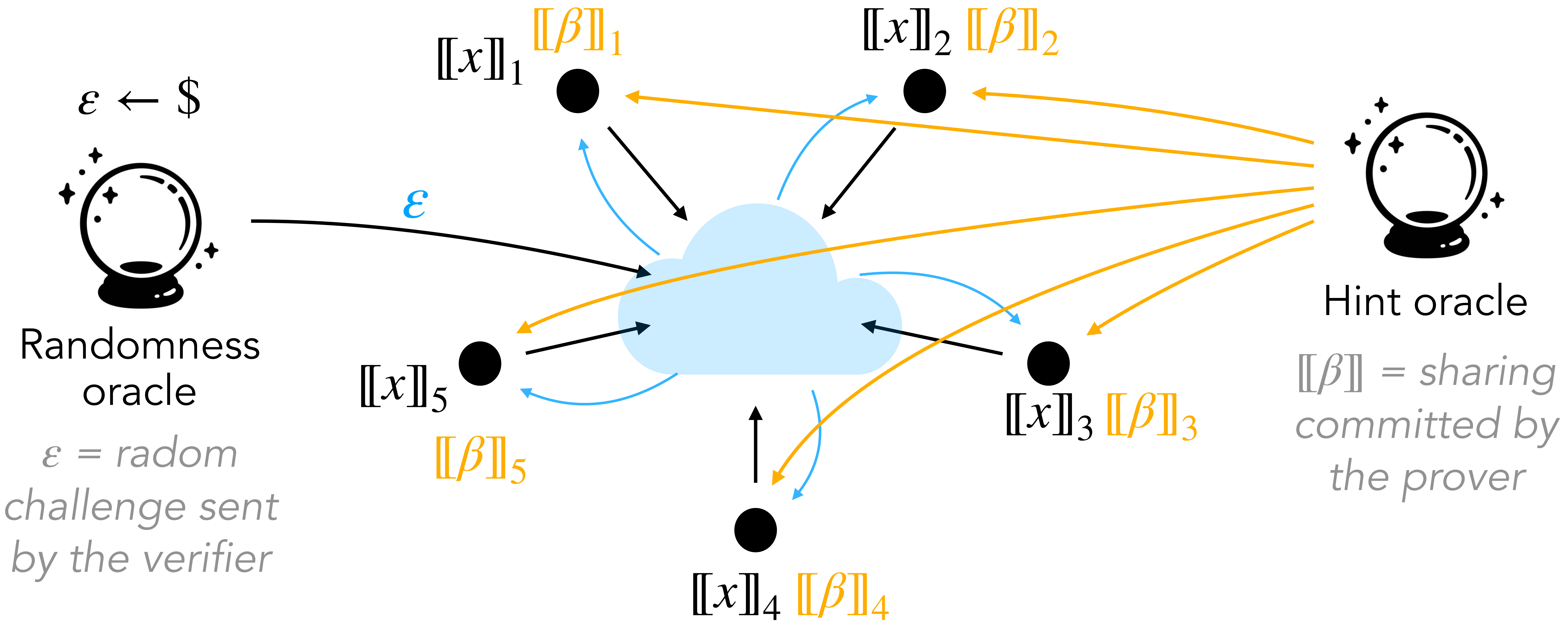
Complete MPC model



Complete MPC model



Complete MPC model



False positive probability

- False positive = MPC protocol outputs "Accept" while $[\![x]\!]$ s.t. $F(x) \neq y$

False positive probability

- False positive = MPC protocol outputs "Accept" while $[\![x]\!]$ s.t. $F(x) \neq y$
- False positive probability:

$$p = \max_{[\![\beta]\!]} P\left[\text{MPC} : ([\![x]\!], [\![\beta]\!], \varepsilon) \mapsto \text{"Accept"} \mid F(x) \neq y\right]$$

(over the randomness of ε)

False positive probability

- False positive = MPC protocol outputs “Accept” while $[\![x]\!]$ s.t. $F(x) \neq y$
- False positive probability:

$$p \;=\; \max_{[\![\beta]\!]} P\left[\text{MPC} : ([\![x]\!], [\![\beta]\!], \varepsilon) \mapsto \text{"Accept"} \mid F(x) \neq y\right]$$

(over the randomness of ε)

- Soundness error:

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

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$\cdots$

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

$\cdots$

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

$\leftarrow$ **hint** $ab = c$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$\cdots$

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

ε

$\leftarrow$ ***hint*** $ab = c$

$\leftarrow$ ***random*** ε

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$

$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$

$\downarrow$

$[\![\alpha]\!]_1 \ [\![\beta]\!]_1$

…

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

ε

$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$

$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$

$\downarrow$

$[\![\alpha]\!]_N \ [\![\beta]\!]_N$

← **hint** $ab = c$

← **random** ε

$\alpha = \epsilon x + a$

$\beta = y + b$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$

$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$

$[\![\alpha]\!]_1 \ [\![\beta]\!]_1$

$[\![v]\!]_1 = \epsilon[\![z]\!]_1 - [\![c]\!]_1 + \alpha[\![b]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$

$[\![v]\!]_1$

$\cdots$

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

ε

$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$

$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$

$[\![\alpha]\!]_N \ [\![\beta]\!]_N$

$[\![v]\!]_N = \epsilon[\![z]\!]_N - [\![c]\!]_N + \alpha[\![b]\!]_N + \beta[\![a]\!]_N - \alpha\beta$

$[\![v]\!]_N$

← **hint** $ab = c$

← **random** ε

$\alpha = \epsilon x + a$

$\beta = y + b$

v

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$

$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$

$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$

$[\![\alpha]\!]_1 \ [\![\beta]\!]_1$

$[\![v]\!]_1 = \epsilon[\![z]\!]_1 - [\![c]\!]_1 + \alpha[\![b]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$

$[\![v]\!]_1$

…

$\mathscr{P}_N$

$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$

$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$

ε

$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$

$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$

$[\![\alpha]\!]_N \ [\![\beta]\!]_N$

$[\![v]\!]_N = \epsilon[\![z]\!]_N - [\![c]\!]_N + \alpha[\![b]\!]_N + \beta[\![a]\!]_N - \alpha\beta$

$[\![v]\!]_N$

← hint $ab = c$

← random ε

$\alpha = \epsilon x + a$

$\beta = y + b$

v

$$g(v) = \begin{cases} \text{Accept} & \text{if } v = 0 \\ \text{Reject} & \text{if } v \neq 0 \end{cases}$$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$$
$$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$$
$$\varepsilon$$
$$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$$
$$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$$

$$\downarrow$$

$$[\![\alpha]\!]_1 \; [\![\beta]\!]_1$$

$$[\![v]\!]_1 = \epsilon[\![z]\!]_1 - [\![c]\!]_1 + \alpha[\![b]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$$

$$\downarrow$$

$$[\![v]\!]_1$$

$\cdots$

$\mathscr{P}_N$

$$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$$
$$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$$
$$\varepsilon$$
$$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$$
$$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$$

$$\downarrow$$

$$[\![\alpha]\!]_N \; [\![\beta]\!]_N$$

$$[\![v]\!]_N = \epsilon[\![z]\!]_N - [\![c]\!]_N + \alpha[\![b]\!]_N + \beta[\![a]\!]_N - \alpha\beta$$

$$\downarrow$$

$$[\![v]\!]_N$$

← **hint** $ab = c$

← **random** ε

$$\alpha = \epsilon x + a$$
$$\beta = y + b$$

$$v$$

$$g(v) = \begin{cases} \text{Accept} & \text{if } v = 0 \\ \text{Reject} & \text{if } v \neq 0 \end{cases}$$

If $xy = z$ and $ab = c$, then $v = 0$

If $xy \neq z$ or $ab \neq c$, then $\Pr[v = 0] = 1/|\mathbb{F}|$

Example: [BN20] check product $xy = z$

$\mathscr{P}_1$

$$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$$
$$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$$
$$\varepsilon$$
$$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$$
$$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$$

$\downarrow$ $[\![\alpha]\!]_1 \; [\![\beta]\!]_1$

$$[\![v]\!]_1 = \epsilon[\![z]\!]_1 - [\![c]\!]_1 + \alpha[\![b]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$$

$\downarrow$ $[\![v]\!]_1$

…

$\mathscr{P}_N$

$$[\![x]\!]_N, [\![y]\!]_N, [\![z]\!]_N$$
$$[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$$
$$\varepsilon$$
$$[\![\alpha]\!]_N = \varepsilon[\![x]\!]_N + [\![a]\!]_N$$
$$[\![\beta]\!]_N = [\![y]\!]_N + [\![b]\!]_N$$

$\downarrow$ $[\![\alpha]\!]_N \; [\![\beta]\!]_N$

$$[\![v]\!]_N = \epsilon[\![z]\!]_N - [\![c]\!]_N + \alpha[\![b]\!]_N + \beta[\![a]\!]_N - \alpha\beta$$

$\downarrow$ $[\![v]\!]_N$

← **hint** $ab = c$

← **random** ε

$\alpha = \epsilon x + a$
$\beta = y + b$

false positive probability

$$g(v) = \begin{cases} \text{Accept} & \text{if } v = 0 \\ \text{Reject} & \text{if } v \neq 0 \end{cases}$$

If $xy = z$ and $ab = c$, then $v = 0$

If $xy \neq z$ or $ab \neq c$, then $\Pr[v = 0] = 1/|\mathbb{F}|$

Verifying arbitrary circuits

- Product-check protocol ⇒ protocol for checking any arithmetic circuit $C(x) = y$
- Principle:
  - Let $\{c_i = a_i \cdot b_i\}$ all the multiplications in C
  - Extended witness: $w = x \,\|\, (c_1, \ldots, c_m)$
  - Compute $[\![y]\!]$ = linear function of $[\![w]\!]$ → check $[\![y]\!]$ = sharing of y
  - $[\![a_i]\!], [\![b_i]\!], [\![c_i]\!]$ = linear functions of $[\![w]\!]$ → product check on $[\![a_i]\!], [\![b_i]\!], [\![c_i]\!]$

Optimisations

Optimising communication (sig. size)

- **Signature = transcript P → V** (×τ iterations)
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → N commitments
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → N MPC broadcasts
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N - 1$ input shares + random tapes

Optimising communication (sig. size)

- **Signature = transcript P → V** ($\times\tau$ iterations)
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → N commitments
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → N MPC broadcasts
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i*}$ → $N-1$ input shares + random tapes
- First optimisation: **hashing**
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N \rightarrow h = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N), \ \alpha = \Sigma_i [\![\alpha]\!]_i$
  - Verification
    - $[\![\alpha]\!]_i = \varphi([\![x]\!]_i) \quad \forall i \neq i*$
    - $[\![\alpha]\!]_{i*} = \alpha - \Sigma_{i \neq i*} [\![\alpha]\!]_i$
    - Check $\mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N) = h$

Optimising communication (sig. size)

- **Signature = transcript P → V** (×τ iterations)
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → N commitments
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N-1$ input shares + random tapes
- First optimisation: **hashing**
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → $h = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$, $\alpha = \Sigma_i [\![\alpha]\!]_i$
  - Verification
    - $[\![\alpha]\!]_i = \varphi([\![x]\!]_i) \quad \forall i \neq i^*$
    - $[\![\alpha]\!]_{i^*} = \alpha - \Sigma_{i \neq i^*} [\![\alpha]\!]_i$
    - Check $\mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N) = h$

Optimising communication (sig. size)

- **Signature = transcript P → V** (×τ iterations)
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → ~~N commitments~~ → hash +1 commitment
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N-1$ input shares + random tapes
- First optimisation: **hashing**
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → $h = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$, $\alpha = \Sigma_i [\![\alpha]\!]_i$
  - Verification
    - $[\![\alpha]\!]_i = \varphi([\![x]\!]_i) \quad \forall i \neq i^*$
    - $[\![\alpha]\!]_{i^*} = \alpha - \Sigma_{i \neq i^*} [\![\alpha]\!]_i$
    - Check $\mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N) = h$

Optimising communication (sig. size)

- **Signature = transcript P → V** (×τ iterations)
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → ~~N commitments~~ → hash +1 commitment
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → $N-1$ input shares + random tapes **main cost**
- First optimisation: **hashing**
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → $h = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$, $\alpha = \Sigma_i [\![\alpha]\!]_i$
  - Verification
    - $[\![\alpha]\!]_i = \varphi([\![x]\!]_i) \quad \forall i \neq i^*$
    - $[\![\alpha]\!]_{i^*} = \alpha - \Sigma_{i \neq i^*} [\![\alpha]\!]_i$
    - Check $\mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N) = h$

Second optimisation: seed trees

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

Second optimisation: seed trees

- **[KKW18]** Katz, Kolesnikov, Wang: “Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures” (CCS 2018)
- Pseudorandom generation from seed
  - $([\![x]\!]_i, \rho_i) \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
  - $[\![x]\!]_N = x - \Sigma_{i=1}^{N} [\![x]\!]_i$

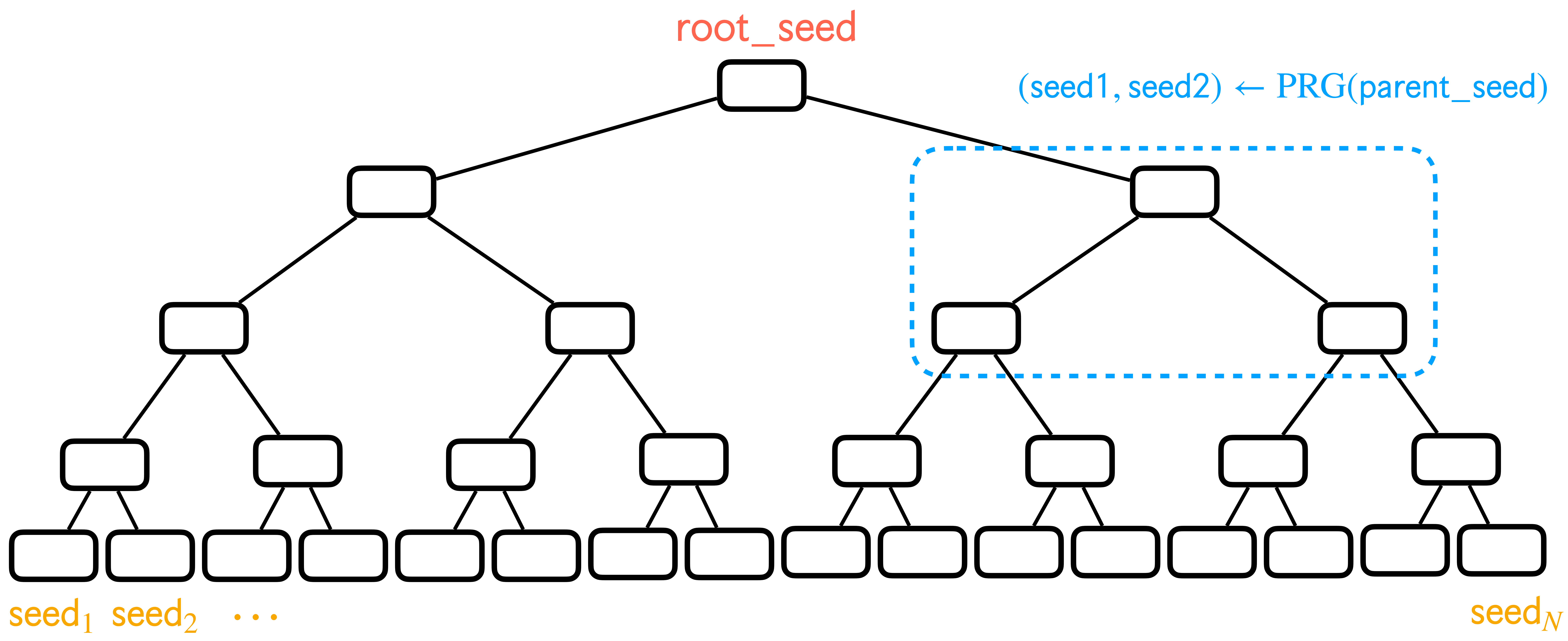
Second optimisation: seed trees

- **[KKW18]** Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)
- Pseudorandom generation from seed
  - $([\![x]\!]_i, \rho_i) \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
  - $[\![x]\!]_N = x - \Sigma_{i=1}^{N} [\![x]\!]_i$
- Seeds $\{\mathsf{seed}_i\}$ generated from a common "root seed"

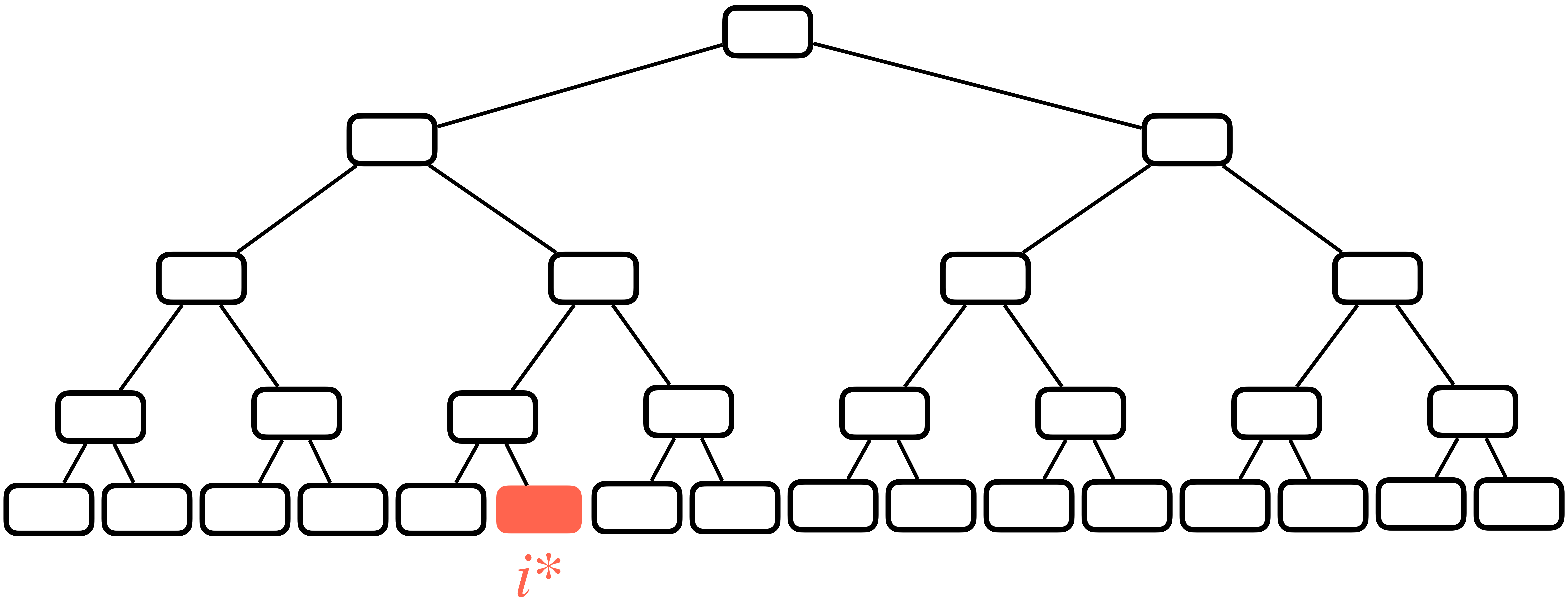
Second optimisation: seed trees

- **[KKW18]** Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)
- Pseudorandom generation from seed
  - $([\![x]\!]_i, \rho_i) \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
  - $[\![x]\!]_N = x - \Sigma_{i=1}^{N} [\![x]\!]_i$
- Seeds $\{\mathsf{seed}_i\}$ generated from a common "root seed"
- Goal: revealing $\{\mathsf{seed}_i\}_{i \neq i*}$ with less than $(N-1) \cdot \lambda$ bits

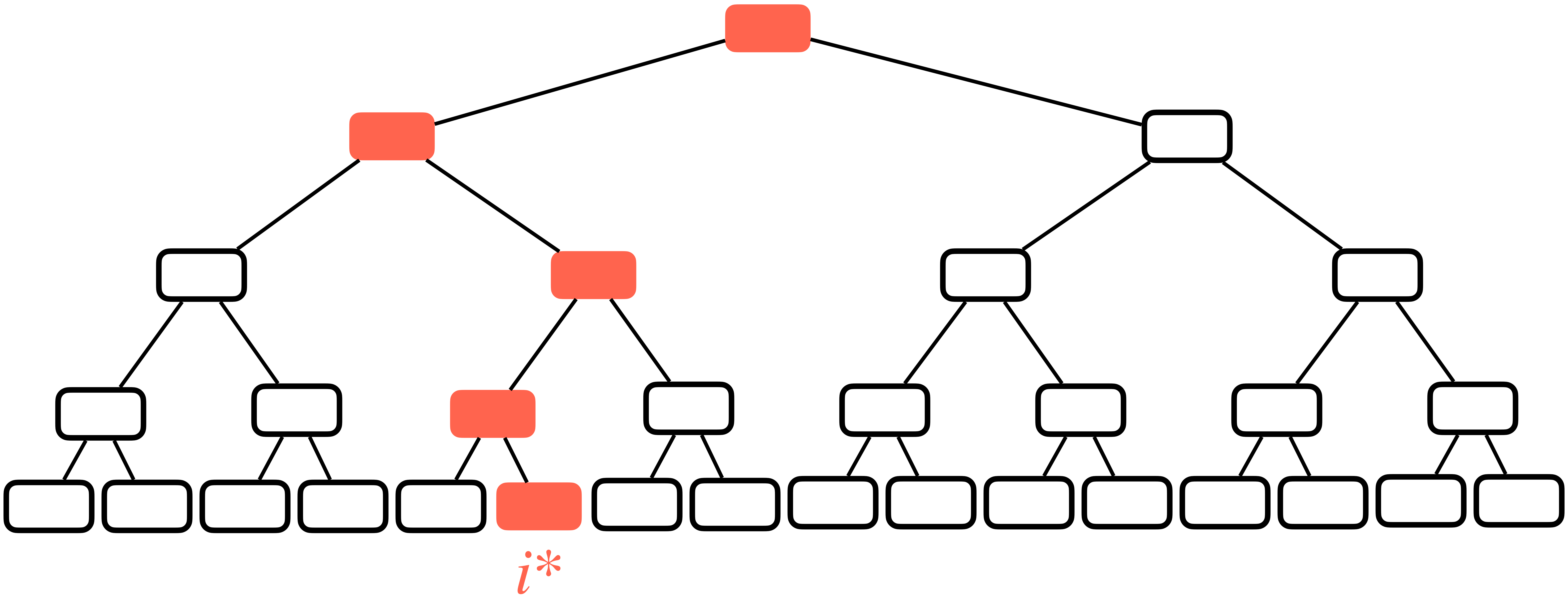
Second optimisation: seed trees



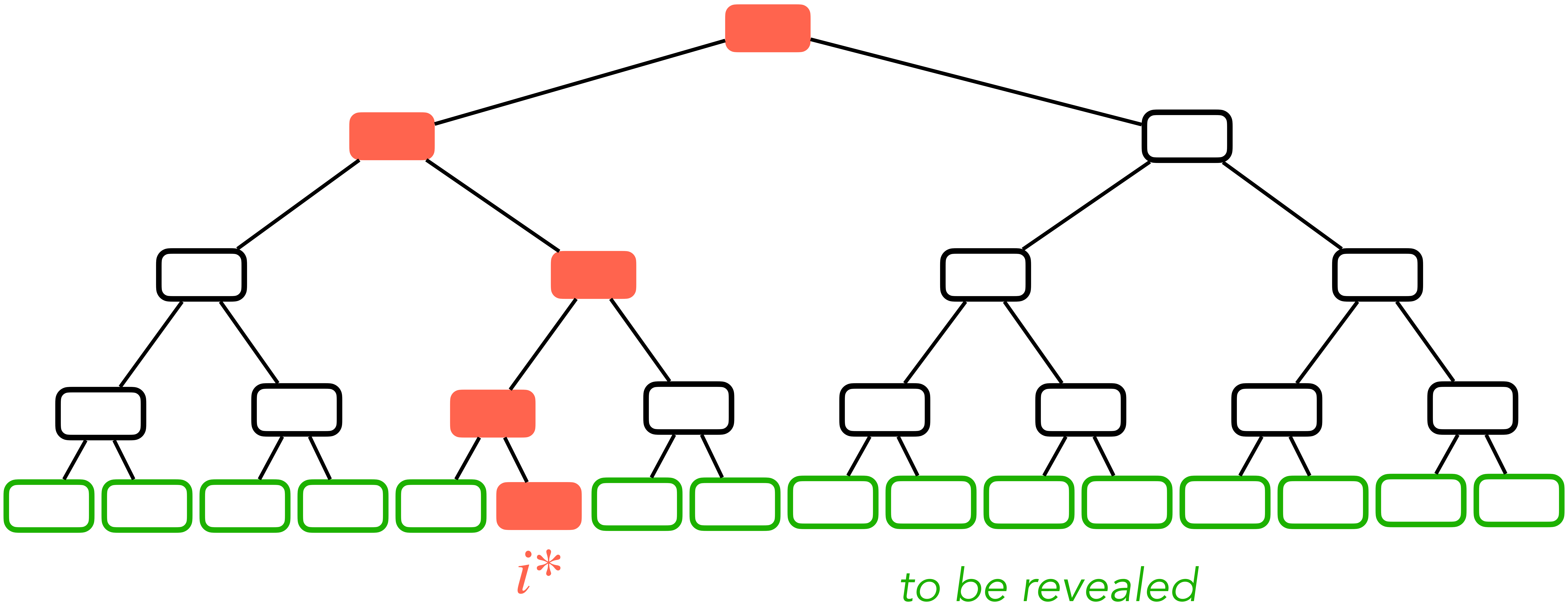
Second optimisation: seed trees



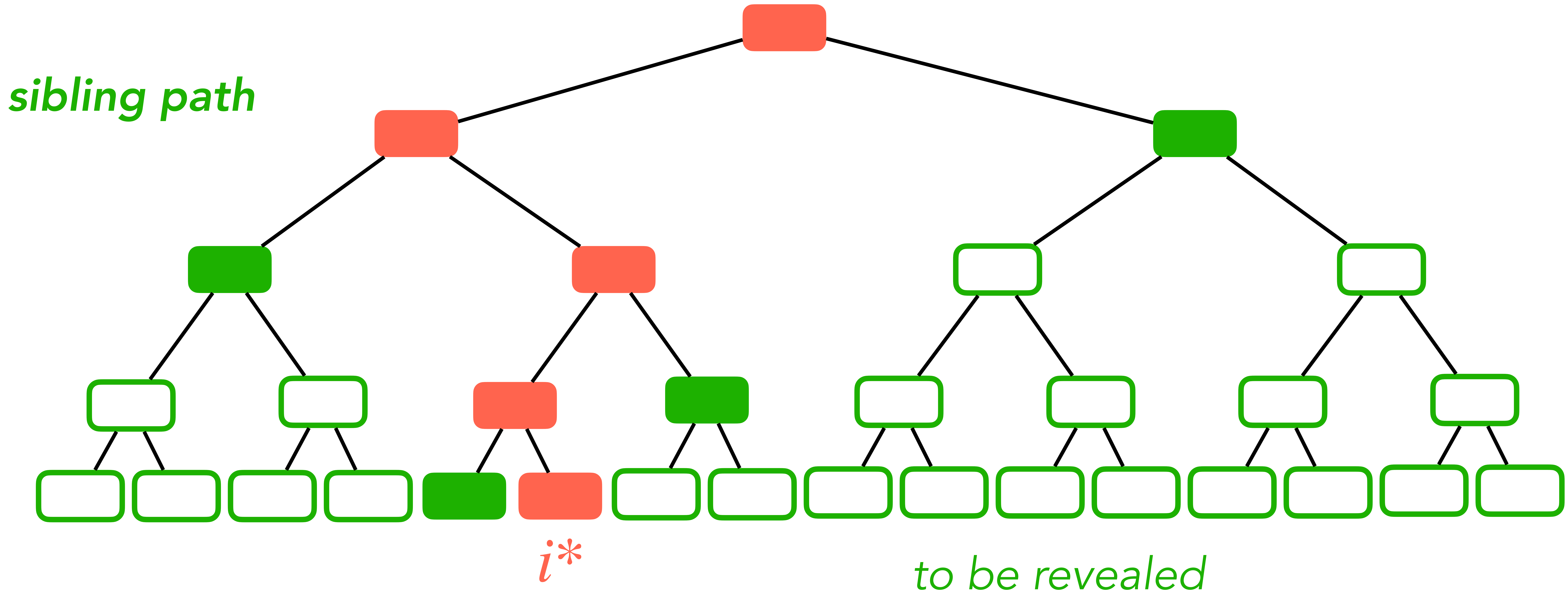
Second optimisation: seed trees



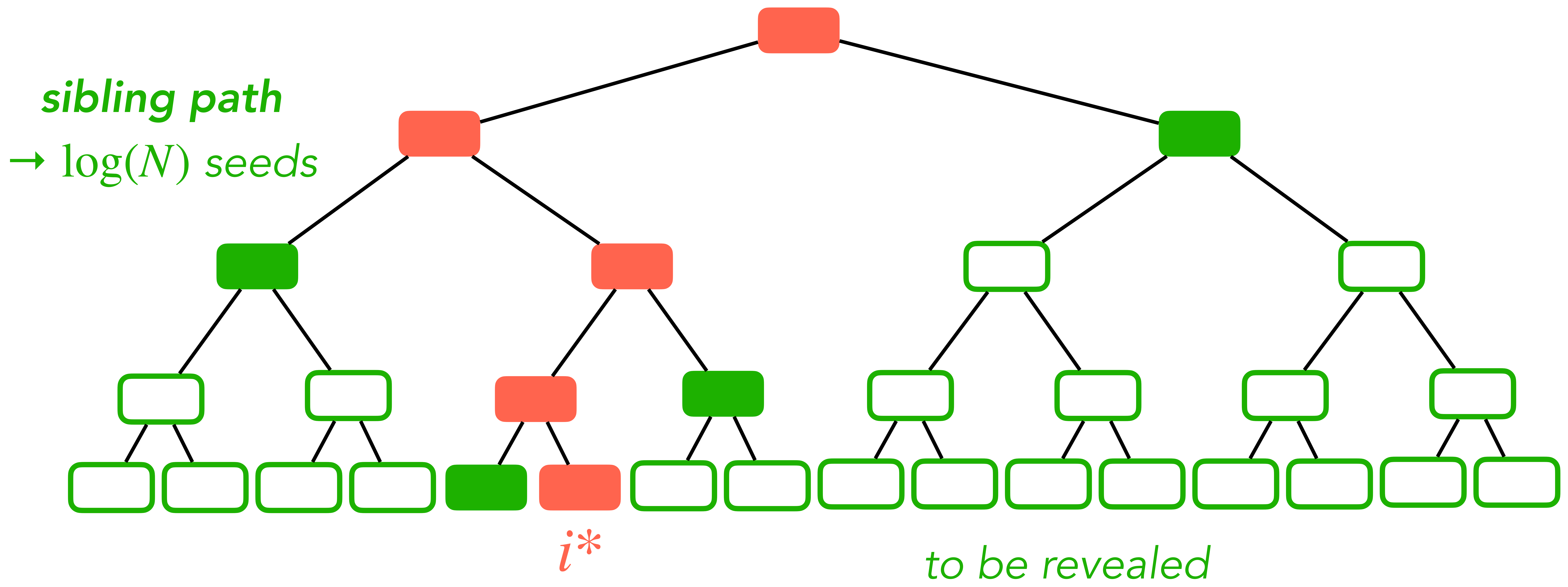
Second optimisation: seed trees



Second optimisation: seed trees



Second optimisation: seed trees



Second optimisation: seed trees

- Signature = transcript P → V
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → ~~N commitments~~ → hash +1 commitment
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → ~~$N-1$ input shares + random tapes~~ → $\log(N)$ seeds

    \+ $[\![x]\!]_N$ if $i^* \neq N$

Second optimisation: seed trees

- **Signature = transcript P → V**
  - $\{\mathrm{Com}^{\rho_i}([\![x]\!]_i)\}$ → ~~N commitments~~ → hash +1 commitment
  - $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ → ~~N MPC broadcasts~~ → hash (+1 MPC broadcast)
  - $\{[\![x]\!]_i, \rho_i\}_{i \neq i^*}$ → ~~$N-1$ input shares + random tapes~~ → $\log(N)$ seeds + $[\![x]\!]_N$ if $i^* \neq N$
- Verification
  - Sibling path → $\{\text{seed}_i\}_{i \neq i^*}$
  - $\text{seed}_i \to ([\![x]\!]_i, \rho_i) \quad \forall\, i \neq i^*$
  - …

Optimising computation: hypercube technique

- **[AGHHJY23]** Aguilar Melchor, Gama, Howe, Hülsing, Joseph, Yue. "The Return of the SDitH" (EUROCRYPT 2023)

Optimising computation: hypercube technique

- **[AGHHJY23]** Aguilar Melchor, Gama, Howe, Hülsing, Joseph, Yue. "The Return of the SDitH" (EUROCRYPT 2023)

- High-level principle

  - Apply MPC computation to sums of shares

$$\Sigma_{i \in I} [\![x_i]\!] \xrightarrow{\varphi} \Sigma_{i \in I} [\![\alpha_i]\!]$$

  - Only $\log N + 1$ such party computations necessary for the prover

  - Only $\log N$ for the verifier

Optimising computation: hypercube technique

- **[AGHHJY23]** Aguilar Melchor, Gama, Howe, Hülsing, Joseph, Yue. "The Return of the SDitH" (EUROCRYPT 2023)
- High-level principle
  - Apply MPC computation to sums of shares

$$\Sigma_{i\in I}\,[\![x_i]\!] \xrightarrow{\varphi} \Sigma_{i\in I}\,[\![\alpha_i]\!]$$

  - Only $\log N + 1$ such party computations necessary for the prover
  - Only $\log N$ for the verifier
- See Nicolas Gama's talk at EC: https://youtu.be/z6nE4fOWvZA (49:33)

SDitH Signature Scheme: MPCitH with SD

Syndrome decoding problem

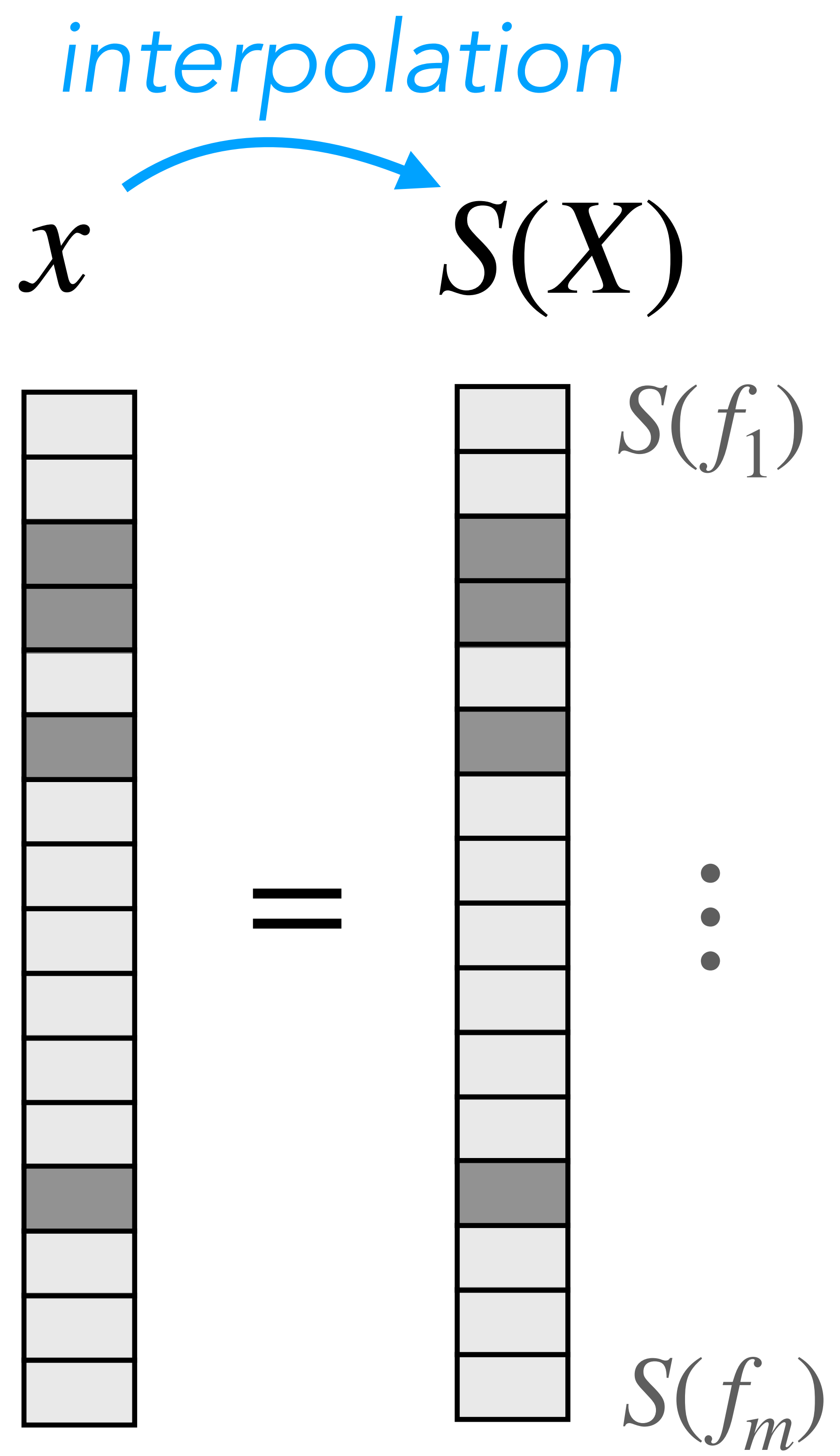
- Parameters
  - A field $\mathbb{F}_q$, $m \in \mathbb{N}$ *(code length)* , $k < m$ *(code dimension)* , $w < m$ *(weight)*
- Let
  - $H \leftarrow \mathbb{F}_q^{(m-k)\times m}$ *(random parity-check matrix)*
  - $x \leftarrow \mathbb{F}_q^m$ s.t. $\mathrm{wt}(x) \leq w$ *(SD solution)*
  - $y = Hx$ *(syndrome)*
- From (H, y) find x

Syndrome decoding problem

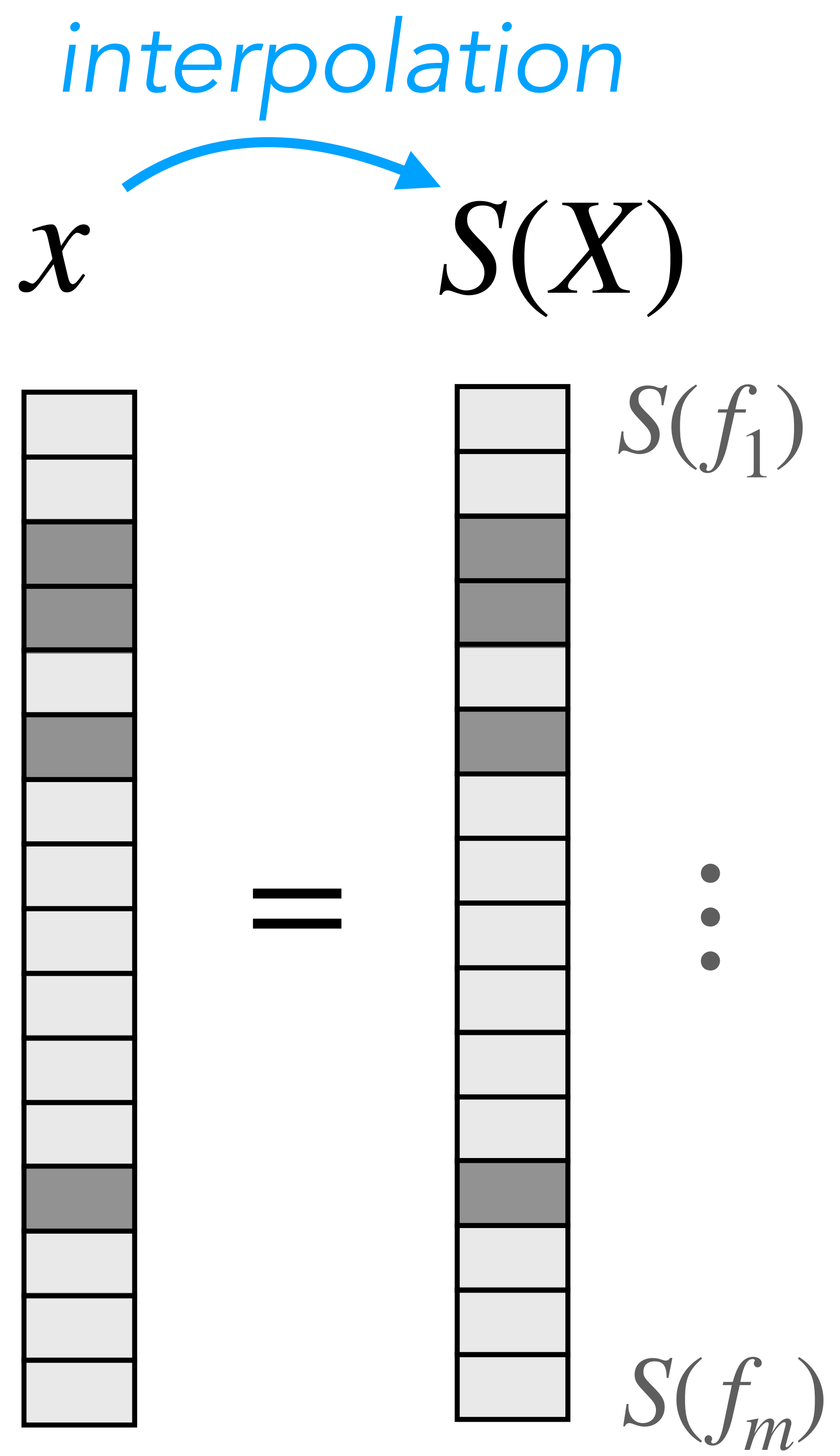
- Parameters
  - A field $\mathbb{F}_q$, $m \in \mathbb{N}$ *(code length)* , $k < m$ *(code dimension)* , $w < m$ *(weight)*
- Let
  - $H \leftarrow \mathbb{F}_q^{(m-k)\times m}$ *(random parity-check matrix)*
  - $x \leftarrow \mathbb{F}_q^m$ s.t. $\mathrm{wt}(x) \leq w$ *(SD solution)*
  - $y = Hx$ *(syndrome)*
- From (H, y) find x
- Standard form (wlog): $H = (H' \,|\, I_{m-k})$ $\Rightarrow$ $y = H'x_A + x_B$ where $x = (\underbrace{x_A}_{|x_A| = k} \,|\, \underbrace{x_B}_{|x_B| = m-k})$

$\Rightarrow$ $x_B = y - H'x_A$

Polynomial expression

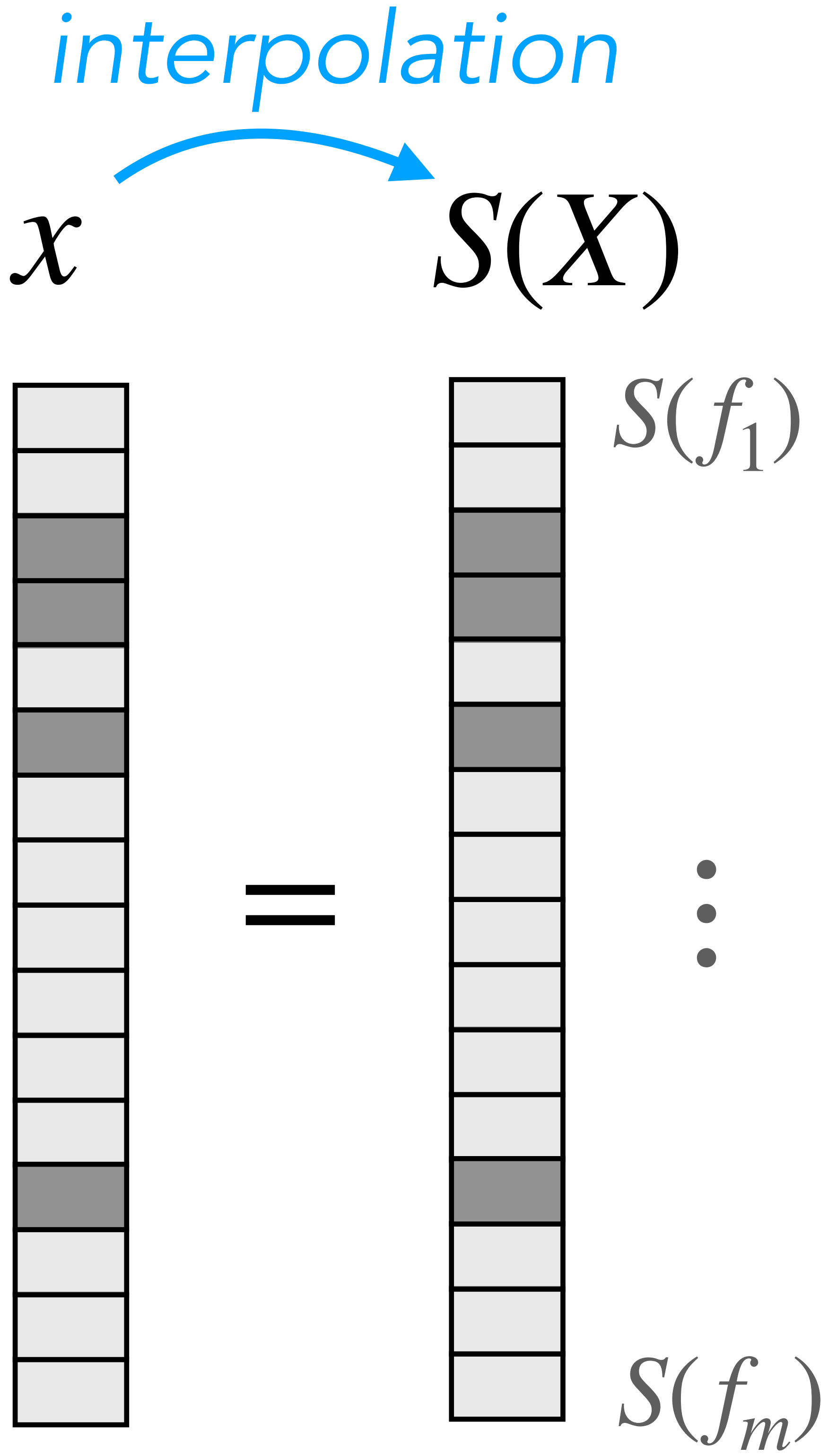


Polynomial expression



$$Q(X) = \prod_{i \in E} (X - f_i)$$

Polynomial expression

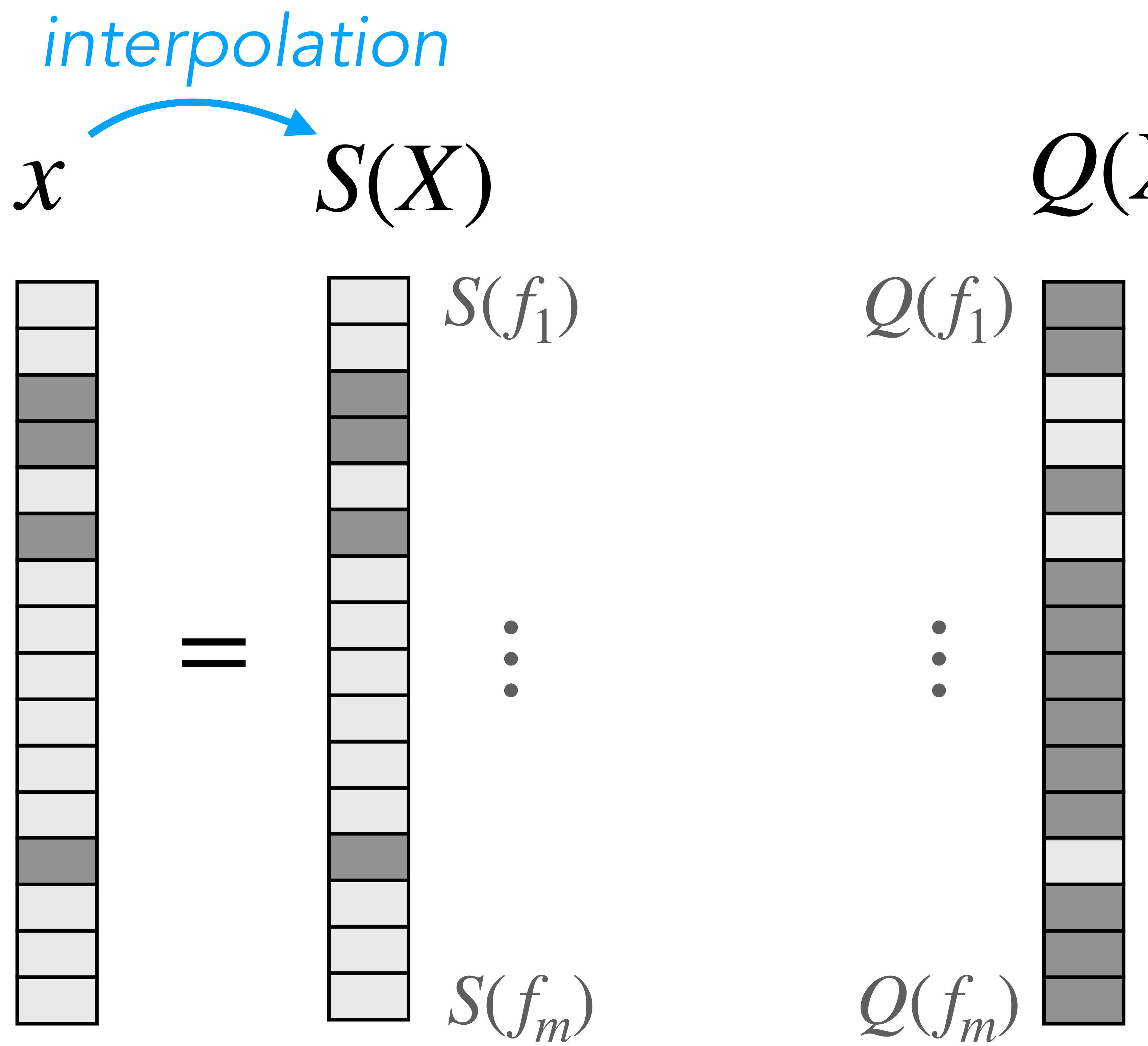


$$Q(X) = \prod_{i \in E} (X - f_i)$$

indices i s.t. $x_i \neq 0$

$|E| \leq w \;\Rightarrow\; \deg(Q) \leq w$

Polynomial expression



$$Q(X) = \prod_{i \in E} (X - f_i)$$

indices i s.t. $x_i \neq 0$

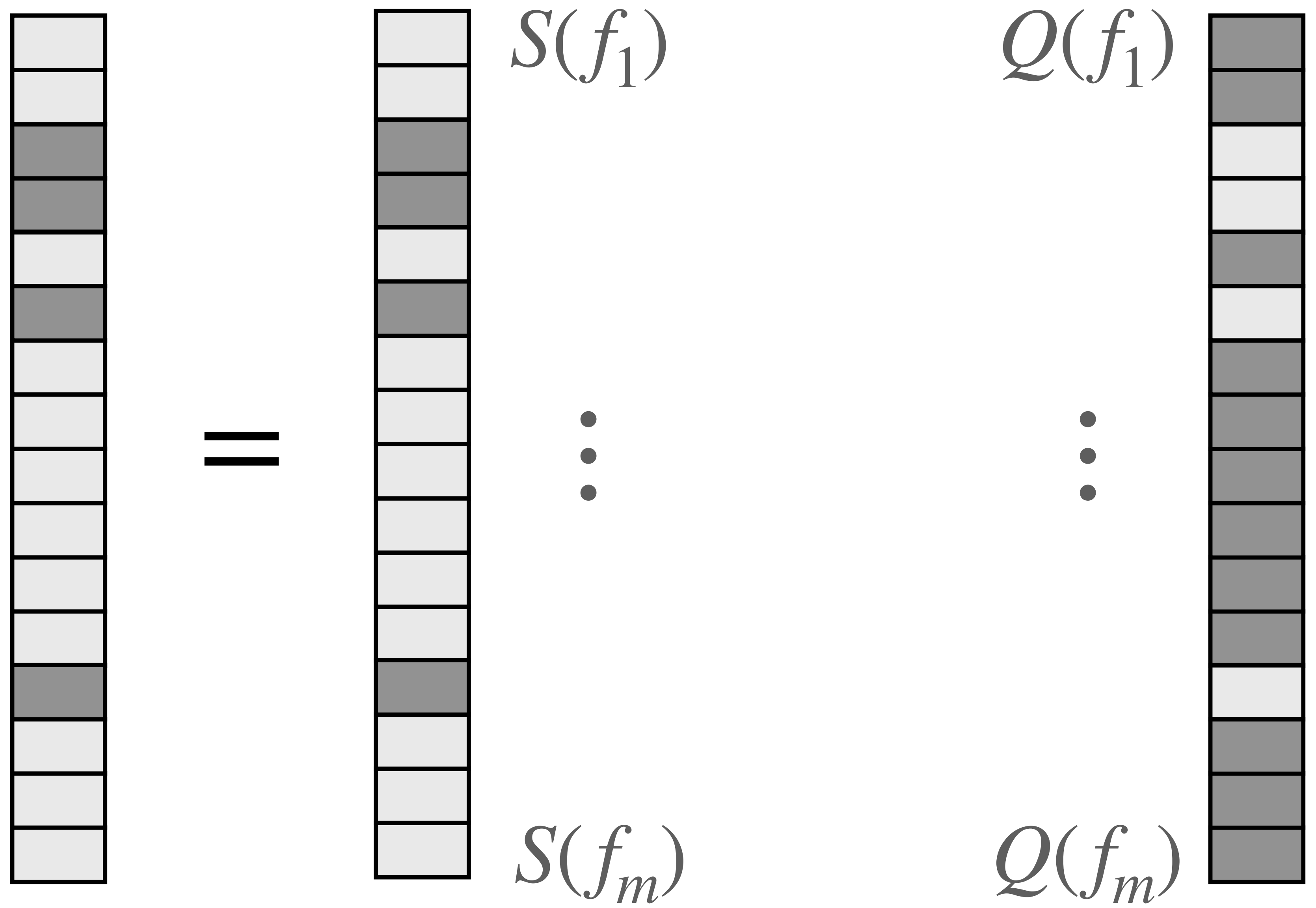
$|E| \leq w \;\Rightarrow\; \deg(Q) \leq w$

= zero coordinate

= non-zero coordinate

Polynomial expression

$$x \xrightarrow{\text{interpolation}} S(X) \qquad\qquad Q(X) = \prod_{i \in E} (X - f_i)$$



indices i s.t. $x_i \neq 0$

$|E| \leq w \;\Rightarrow\; \deg(Q) \leq w$

= zero coordinate

= non-zero coordinate

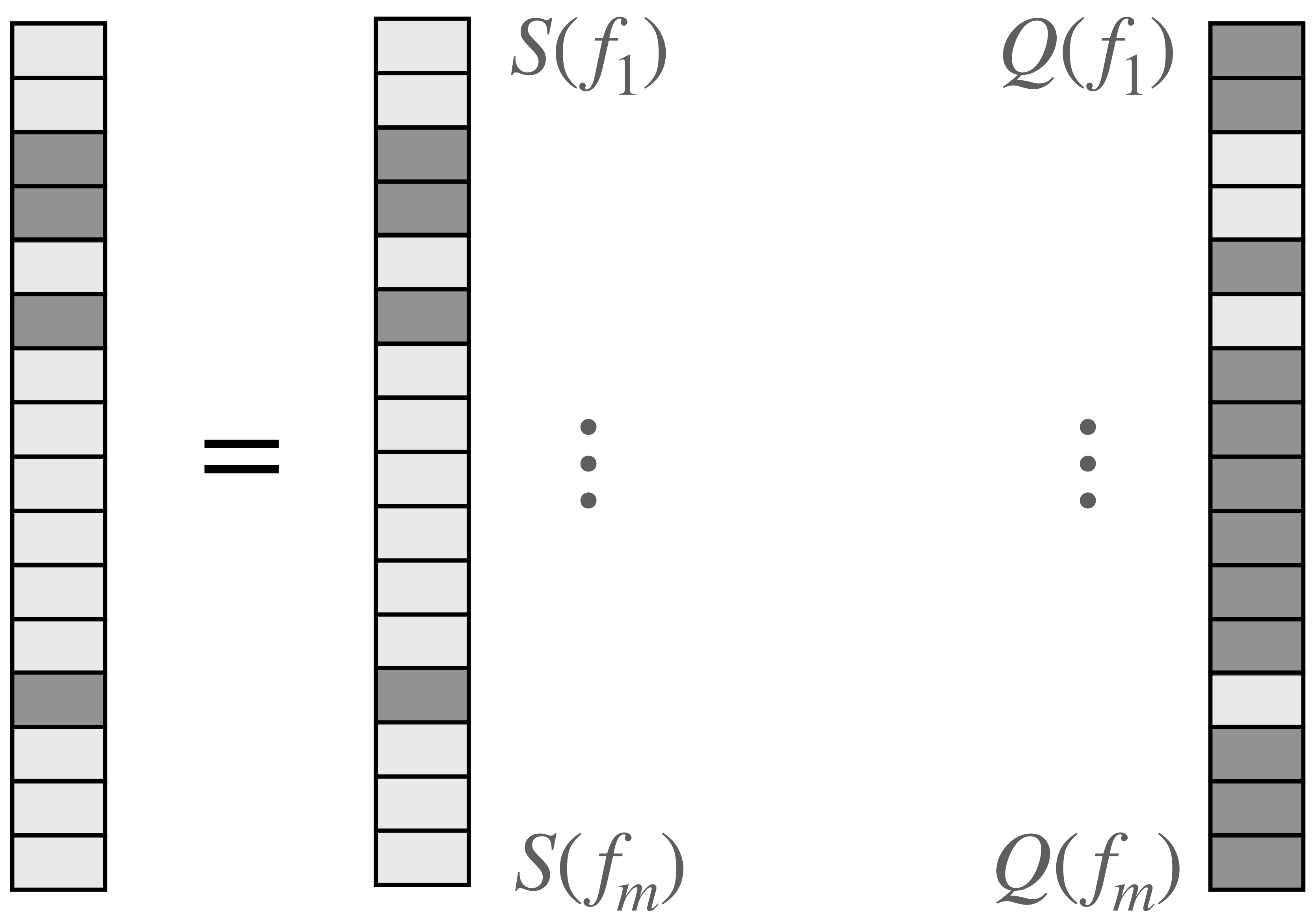
$\Rightarrow\; S(X) \cdot Q(X)$ evaluates to 0 in $f_1, \ldots, f_m$

some degree $\leqslant w - 1$ *polynomial*

$$\Rightarrow\; S(X) \cdot Q(X) = \underbrace{F(X)}_{\prod_{i \in [1:m]} (X - f_i)} \cdot P(X)$$

Polynomial expression

$$x \xrightarrow{\textit{interpolation}} S(X) \qquad Q(X) = \prod_{i \in E} (X - f_i)$$



$\Rightarrow$ $S(X) \cdot Q(X)$ evaluates to 0 in $f_1, \ldots, f_m$

If $\mathrm{wt}(x) \leq w$ then

$\exists\, Q$ of degree $\leq w$ s.t. $S(X) \cdot Q(X)$

evaluates to 0 in $f_1, \ldots, f_m$

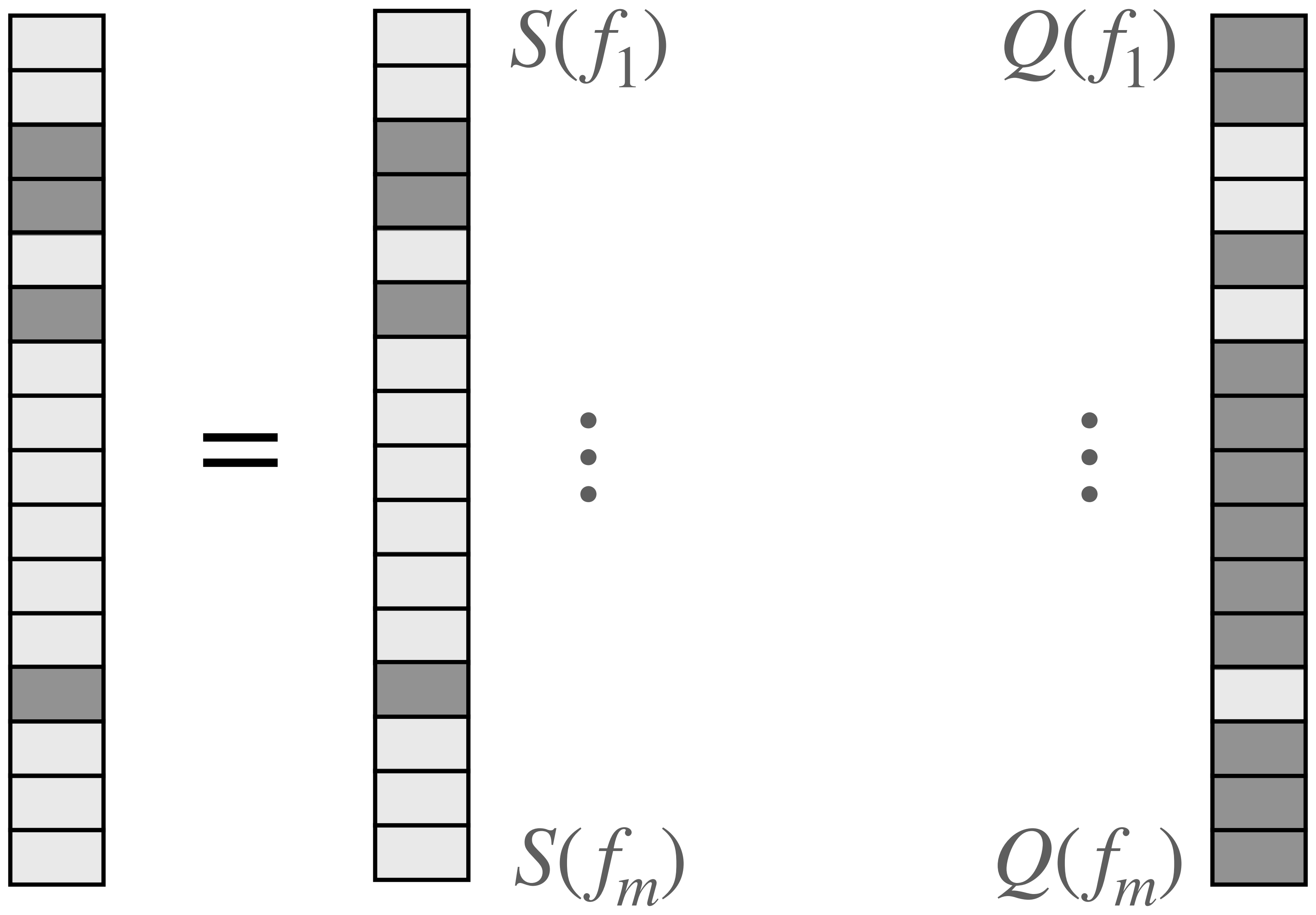
$\Leftrightarrow$

$\exists\, Q, P$ of degrees $\leq w$, $w - 1$ s.t

$S(X) \cdot Q(X) = F(X) \cdot P(X)$

Polynomial expression

$$x \xrightarrow{\text{interpolation}} S(X) \qquad\qquad Q(X) = \prod_{i \in E} (X - f_i)$$



$\Rightarrow$ $S(X) \cdot Q(X)$ evaluates to 0 in $f_1, \ldots, f_m$

If $\mathrm{wt}(x) \leq w$ then

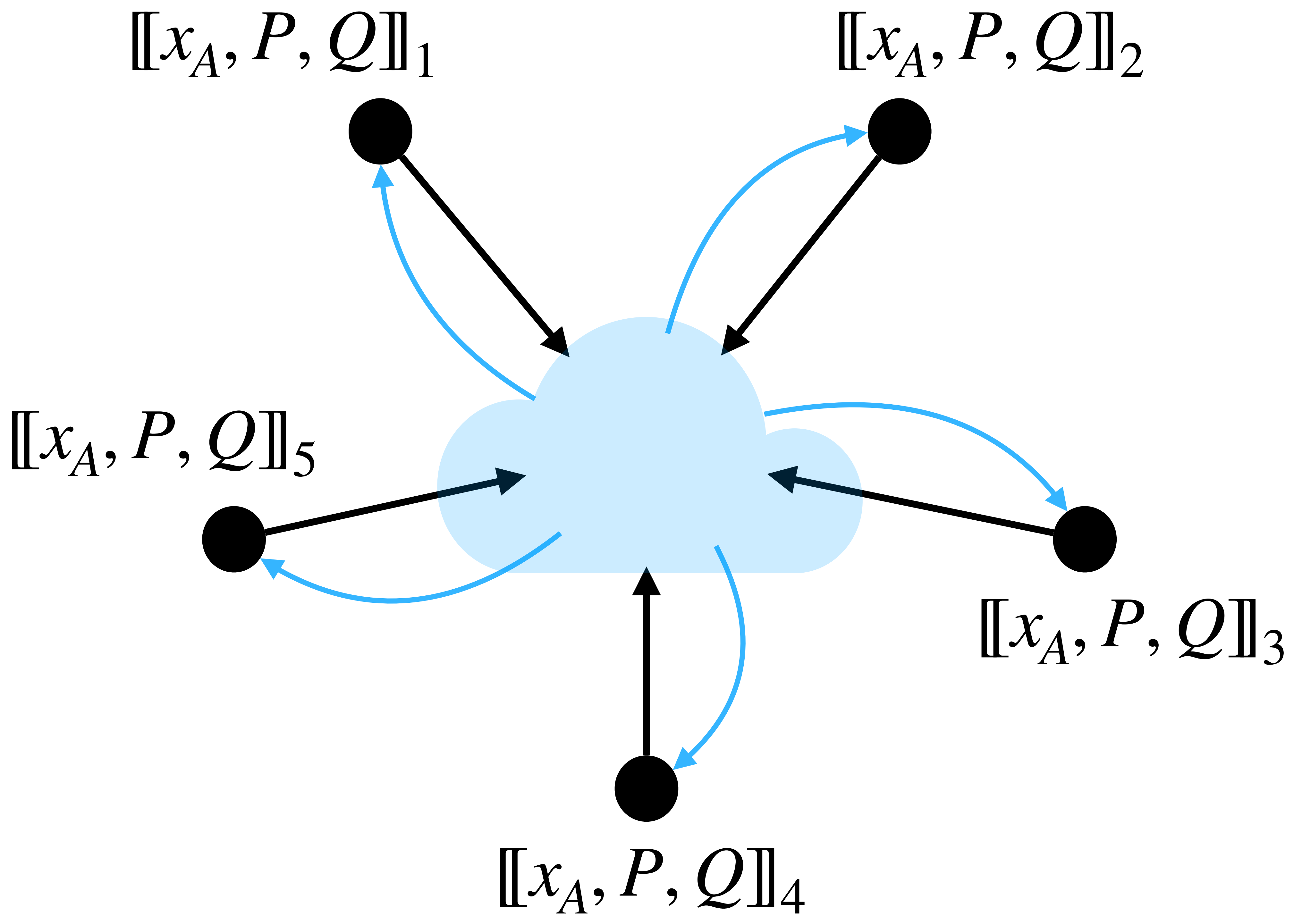
$\exists\, Q$ of degree $\leq w$ s.t. $S(X) \cdot Q(X)$ evaluates to 0 in $f_1, \ldots, f_m$

$\Leftrightarrow$

$\exists\, Q, P$ of degrees $\leq w$, $w - 1$ s.t
$S(X) \cdot Q(X) = F(X) \cdot P(X)$

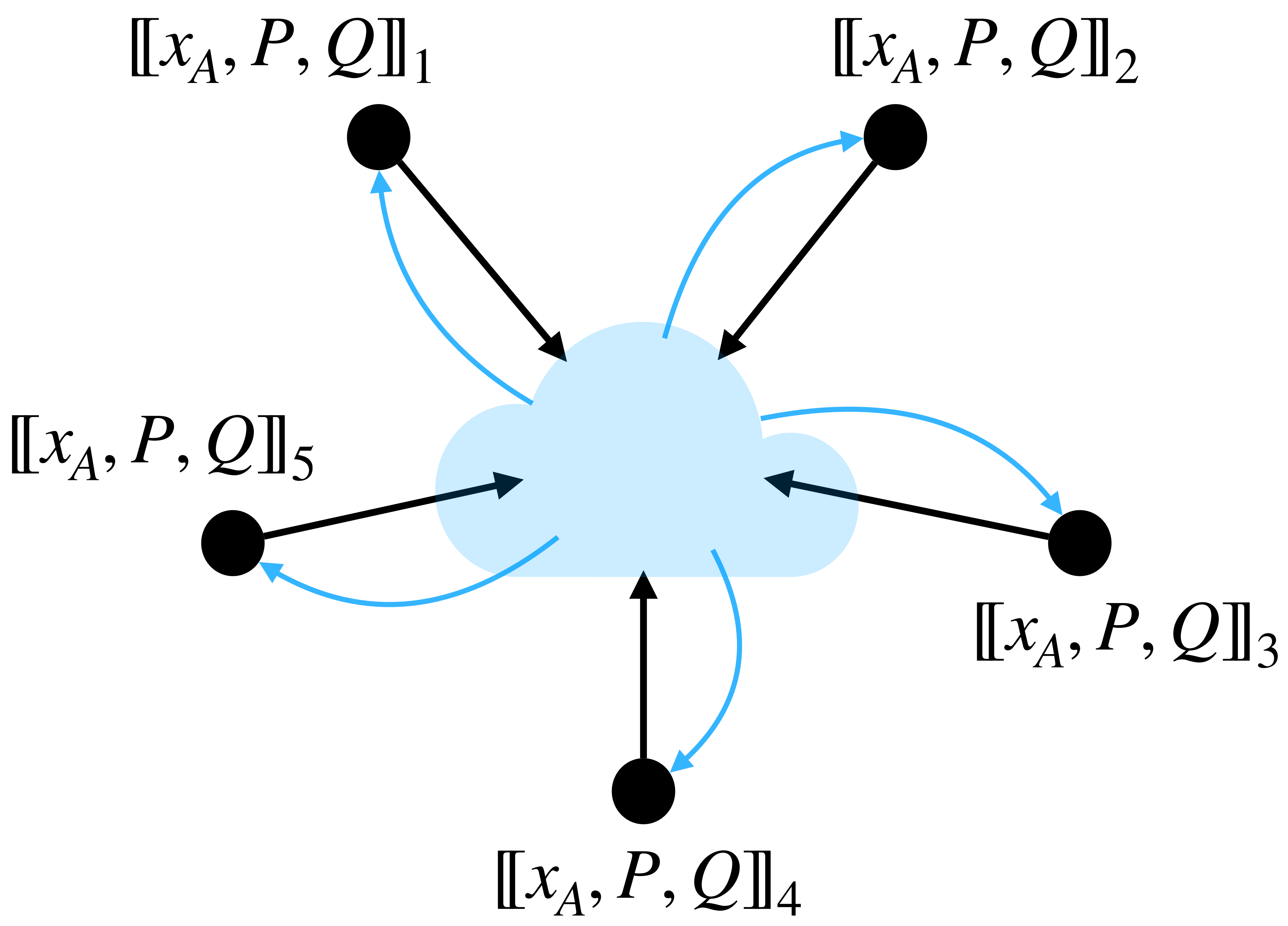
We'll show this

SDitH MPC protocol



- **Parties receive**
  - $[\![x_A]\!], [\![P]\!], [\![Q]\!]$ sharings of x_A, P, Q
  - (H', y) SD instance

SDitH MPC protocol



- **Parties receive**
  - $[\![x_A]\!], [\![P]\!], [\![Q]\!]$ sharings of x_A, P, Q
  - (H', y) SD instance
- **Parties jointly compute**

$$g(x_A, P, Q) = \begin{cases} \text{Accept} & \text{if } SQ = FP \\ \text{Reject} & \text{otherwise} \end{cases}$$

where $x_B = y - H'x_A$ and $S = \mathrm{Interp}(x_A \mid x_B)$

Schwartz–Zippel lemma

- Let P_1 and P_2 two degree-d polynomials of $\mathbb{F}[X]$
- Let r a random point of $\mathbb{F}$,

$$\Pr\left[P_1(r) = P_2(r) \mid P_1 \neq P_2\right] \leq \frac{d}{|\mathbb{F}|}$$

($P_1(r) = P_2(r) \quad \Leftrightarrow \quad r \in$ roots of $P_1 - P_2$)

Schwartz–Zippel lemma

- Let P_1 and P_2 two degree-d polynomials of $\mathbb{F}[X]$
- Let r a random point of $\mathbb{F}$,

$$\Pr\left[P_1(r) = P_2(r) \mid P_1 \neq P_2\right] \leq \frac{d}{|\mathbb{F}|}$$

$$(\, P_1(r) = P_2(r) \quad \Leftrightarrow \quad r \in \text{roots of } P_1 - P_2 \,)$$

- For a random $r \in \mathbb{F}_q^\eta$,

$$\Pr\left[S(r) \cdot Q(r) = F(r) \cdot P(r) \mid SQ \neq FP\right] \leq \frac{m + w - 1}{q^\eta}$$

SDitH MPC protocol

- Principle: check $SQ = FP$ on t random points (SZ lemma)

1. Locally compute $[\![x_B]\!] = y - H' [\![x_A]\!]$
2. Locally compute $[\![S]\!]$ by Lagrange interpolation of $[\![x]\!] = ([\![x_A]\!] \mid [\![x_B]\!])$
3. Randomness oracle $\rightarrow$ $r_1, \ldots, r_t \in \mathbb{F}_q^\eta$
4. Locally compute $[\![S(r_i)]\!], [\![Q(r_i)]\!], F(r_i) \cdot [\![P(r_i)]\!] \quad \forall i \in [1:t]$
5. Check the product $S(r_i) \cdot Q(r_i) = F(r_i) \cdot P(r_i)$ from the shares

SDitH MPC protocol

- Principle: check $SQ = FP$ on t random points (SZ lemma)

1. Locally compute $[\![x_B]\!] = y - H'[\![x_A]\!]$
2. Locally compute $[\![S]\!]$ by Lagrange interpolation of $[\![x]\!] = ([\![x_A]\!] \mid [\![x_B]\!])$
3. Randomness oracle $\rightarrow$ $r_1, \ldots, r_t \in \mathbb{F}_q^\eta$, $\varepsilon_1, \ldots, \varepsilon_t \in \mathbb{F}_q^\eta$
4. Locally compute $[\![S(r_i)]\!], [\![Q(r_i)]\!], F(r_i) \cdot [\![P(r_i)]\!] \quad \forall i \in [1:t]$
5. Check the product $S(r_i) \cdot Q(r_i) = F(r_i) \cdot P(r_i)$ from the shares
    - using [BN20] product-check protocol

SDitH MPC protocol

- Principle: check $SQ = FP$ on t random points (SZ lemma)

1. Locally compute $[\![x_B]\!] = y - H' [\![x_A]\!]$
2. Locally compute $[\![S]\!]$ by Lagrange interpolation of $[\![x]\!] = ([\![x_A]\!] \mid [\![x_B]\!])$
3. Randomness oracle $\rightarrow$ $r_1, \ldots, r_t \in \mathbb{F}_q^\eta$, $\varepsilon_1, \ldots, \varepsilon_t \in \mathbb{F}_q^\eta$
4. Locally compute $[\![S(r_i)]\!], [\![Q(r_i)]\!], F(r_i) \cdot [\![P(r_i)]\!] \quad \forall i \in [1:t]$
5. Check the product $S(r_i) \cdot Q(r_i) = F(r_i) \cdot P(r_i)$ from the shares
   - using [BN20] product-check protocol

- False positive probability: $p = \sum_{i=0}^{t} \binom{t}{i} \left(\frac{m+w-1}{q^\eta}\right)^i \left(1 - \frac{m+w-1}{q^\eta}\right)^{t-i} \left(\frac{1}{q^\eta}\right)^{t-i}$

SDitH signature scheme

Signature:

1. Generate random sharing $[\![x_A]\!]$, $[\![P]\!]$, $[\![Q]\!]$, $[\![a]\!]$, $[\![b]\!]$, $[\![c]\!]$
2. Commit the parties' shares:
$$[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i \xrightarrow{\text{Commit}} \mathsf{com}_i$$
3. Derive the first challenge (randomness of MPC protocol):
$$\mathsf{com}_1, \dots, \mathsf{com}_N \xrightarrow{\text{Hash}} h_1 \to r, \varepsilon$$
4. Simulate the MPC protocol:
$$[\![x_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!], r, \varepsilon \xrightarrow{\text{MPC}} [\![\alpha]\!], [\![\beta]\!], [\![v]\!]$$
5. Derive the second challenge (index of non-opened party):
$$h_1, [\![\alpha]\!], [\![\beta]\!], [\![v]\!] \xrightarrow{\text{Hash}} h_2 \to I$$
6. Build the signature from
$$h_1,\ h_2,\ \left\{[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i\right\}_{i \in I},\ \{\mathsf{com}_i, [\![\alpha]\!]_i, [\![\beta]\!]_i, [\![v]\!]_i\}_{i \notin I}$$

SDitH signature scheme

Signature:

1. Generate random sharing $[\![x_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!]$
2. Commit the parties' shares:

$$[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i \xrightarrow{\text{Commit}} \mathsf{com}_i$$

$\times\,\tau$

3. Derive the first challenge (randomness of MPC protocol):

$$\mathsf{com}_1, \ldots, \mathsf{com}_N \xrightarrow{\text{Hash}} h_1 \to r, \varepsilon$$

$\times\,\tau$

4. Simulate the MPC protocol:

$$[\![x_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!], r, \varepsilon \xrightarrow{\text{MPC}} [\![\alpha]\!], [\![\beta]\!], [\![v]\!]$$

$\times\,\tau$

5. Derive the second challenge (index of non-opened party):

$$h_1, [\![\alpha]\!], [\![\beta]\!], [\![v]\!] \xrightarrow{\text{Hash}} h_2 \to I$$

$\times\,\tau$

6. Build the signature from

$\times\,\tau$

$$h_1,\ h_2,\ \{[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i\}_{i \in I},\ \{\mathsf{com}_i, [\![\alpha]\!]_i, [\![\beta]\!]_i, [\![v]\!]_i\}_{i \notin I}$$

SDitH signature scheme

Signature:

1. Generate random sharing $[\![x_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!]$
2. Commit the parties' shares:
$$[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i \xrightarrow{\text{Commit}} \mathsf{com}_i$$

$\times\,\tau$

3. Derive the first challenge (randomness of MPC protocol):
$$\mathsf{com}_1, \ldots, \mathsf{com}_N \xrightarrow{\text{Hash}} h_1 \to r, \varepsilon$$

$\times\,\tau$

4. Simulate the MPC protocol:
$$[\![x_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!], r, \varepsilon \xrightarrow{\text{MPC}} [\![\alpha]\!], [\![\beta]\!], [\![v]\!]$$

$\times\,\tau$

hypercube $\rightarrow$ $\log N + 1$ party emulations

5. Derive the second challenge (index of non-opened party):
$$h_1, [\![\alpha]\!], [\![\beta]\!], [\![v]\!] \xrightarrow{\text{Hash}} h_2 \to I$$

$\times\,\tau$

6. Build the signature from

$\times\,\tau$

$$h_1,\ h_2,\ \{[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i\}_{i\in I},\ \{\mathsf{com}_i, [\![\alpha]\!]_i, [\![\beta]\!]_i, [\![v]\!]_i\}_{i\notin I}$$

SDitH signature scheme

Parameter Set	MPCitH Parameters						Sizes (in bytes)			
	N	ℓ	τ	η	t	p	pk	sk	Sig. Avg	Sig. Max
SDitH-L1-hyp	2^8	–	17	4	3	$2^{-70.6}$	132	432	8 476	8 496
SDitH-L3-hyp	2^8	–	26	4	3	$2^{-71.8}$	180	628	19 498	19 544
SDitH-L5-hyp	2^8	–	34	4	4	$2^{-94.2}$	244	838	33 843	33 924

Instance	KeyGen		Sign		Verify	
	ms	cycles	sign ms	cycles	verify ms	cycles
SDitH-gf256-L1-hyp	5.47	14.2M	4.18	10.8M	3.74	9.7M
SDitH-gf256-L3-hyp	6.41	16.6M	10.13	26.2M	8.83	22.9M
SDitH-gf256-L5-hyp	11.06	28.7M	19.25	49.9M	16.98	44.0M
SDitH-gf251-L1-hyp	3.05	7.9M	8.17	21.2M	7.83	20.3M
SDitH-gf251-L3-hyp	3.67	9.5M	17.98	46.6M	17.08	44.3M
SDitH-gf251-L5-hyp	6.36	16.5M	32.73	84.8M	31.26	81.0M

SDitH signature scheme

Parameter Set	MPCitH Parameters						Sizes (in bytes)			
	N	ℓ	τ	η	t	p	pk	sk	Sig. Avg	Sig. Max
SDitH-L1-hyp	2^8	–	17	4	3	$2^{-70.6}$	132	432	8 476	8 496
SDitH-L3-hyp	2^8	–	26	4	3	$2^{-71.8}$	180	628	19 498	19 544
SDitH-L5-hyp	2^8	–	34	4	4	$2^{-94.2}$	244	838	33 843	33 924

128-bit security

Instance	KeyGen		Sign		Verify	
	ms	cycles	sign ms	cycles	verify ms	cycles
SDitH-gf256-L1-hyp	5.47	14.2M	4.18	10.8M	3.74	9.7M
SDitH-gf256-L3-hyp	6.41	16.6M	10.13	26.2M	8.83	22.9M
SDitH-gf256-L5-hyp	11.06	28.7M	19.25	49.9M	16.98	44.0M
SDitH-gf251-L1-hyp	3.05	7.9M	8.17	21.2M	7.83	20.3M
SDitH-gf251-L3-hyp	3.67	9.5M	17.98	46.6M	17.08	44.3M
SDitH-gf251-L5-hyp	6.36	16.5M	32.73	84.8M	31.26	81.0M

SDitH signature scheme

☝️ ∃ variant based in MPCitH with threshold secret sharing

Parameter Set	MPCitH Parameters						Sizes (in bytes)			
	N	ℓ	τ	η	t	p	pk	sk	Sig. Avg	Sig. Max
SDitH-L1-hyp	2^8	–	17	4	3	$2^{-70.6}$	132	432	8 476	8 496
SDitH-L3-hyp	2^8	–	26	4	3	$2^{-71.8}$	180	628	19 498	19 544
SDitH-L5-hyp	2^8	–	34	4	4	$2^{-94.2}$	244	838	33 843	33 924

128-bit security

Instance	KeyGen		Sign		Verify	
	ms	cycles	sign ms	cycles	verify ms	cycles
SDitH-gf256-L1-hyp	5.47	14.2M	4.18	10.8M	3.74	9.7M
SDitH-gf256-L3-hyp	6.41	16.6M	10.13	26.2M	8.83	22.9M
SDitH-gf256-L5-hyp	11.06	28.7M	19.25	49.9M	16.98	44.0M
SDitH-gf251-L1-hyp	3.05	7.9M	8.17	21.2M	7.83	20.3M
SDitH-gf251-L3-hyp	3.67	9.5M	17.98	46.6M	17.08	44.3M
SDitH-gf251-L5-hyp	6.36	16.5M	32.73	84.8M	31.26	81.0M

MPC in the Head with Threshold Secret Sharing (a.k.a. TCitH)

Background: Shamir's secret sharing

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

- Generate
  - Let $(r_1, \ldots, r_\ell) \leftarrow \$$
  - Let P the polynomial of coefficients $(x, r_1, \ldots, r_\ell)$

$$\begin{cases} [\![x]\!]_1 = P(f_1) \\ \quad\vdots \\ [\![x]\!]_N = P(f_N) \end{cases} \qquad \text{with} \quad f_1, \ldots, f_N \in \mathbb{F} \text{ distinct field elements}$$

Background: Shamir's secret sharing

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

- Generate
  - Let $(r_1, \ldots, r_\ell) \leftarrow \$$
  - Let P the polynomial of coefficients $(x, r_1, \ldots, r_\ell)$

$$\begin{cases} [\![x]\!]_1 = P(f_1) \\ \quad \vdots \\ [\![x]\!]_N = P(f_N) \end{cases} \quad \text{with} \quad f_1, \ldots, f_N \in \mathbb{F} \text{ distinct field elements}$$

- Reconstruct
  - Interpolate P from $[\![x]\!]_1, \ldots, [\![x]\!]_N$
  - $x = P(0)$

Background: Shamir's secret sharing

- $(\ell + 1, N)$-**threshold** linear secret sharing scheme (LSSS)
  - Linearity: $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$

Background: Shamir's secret sharing

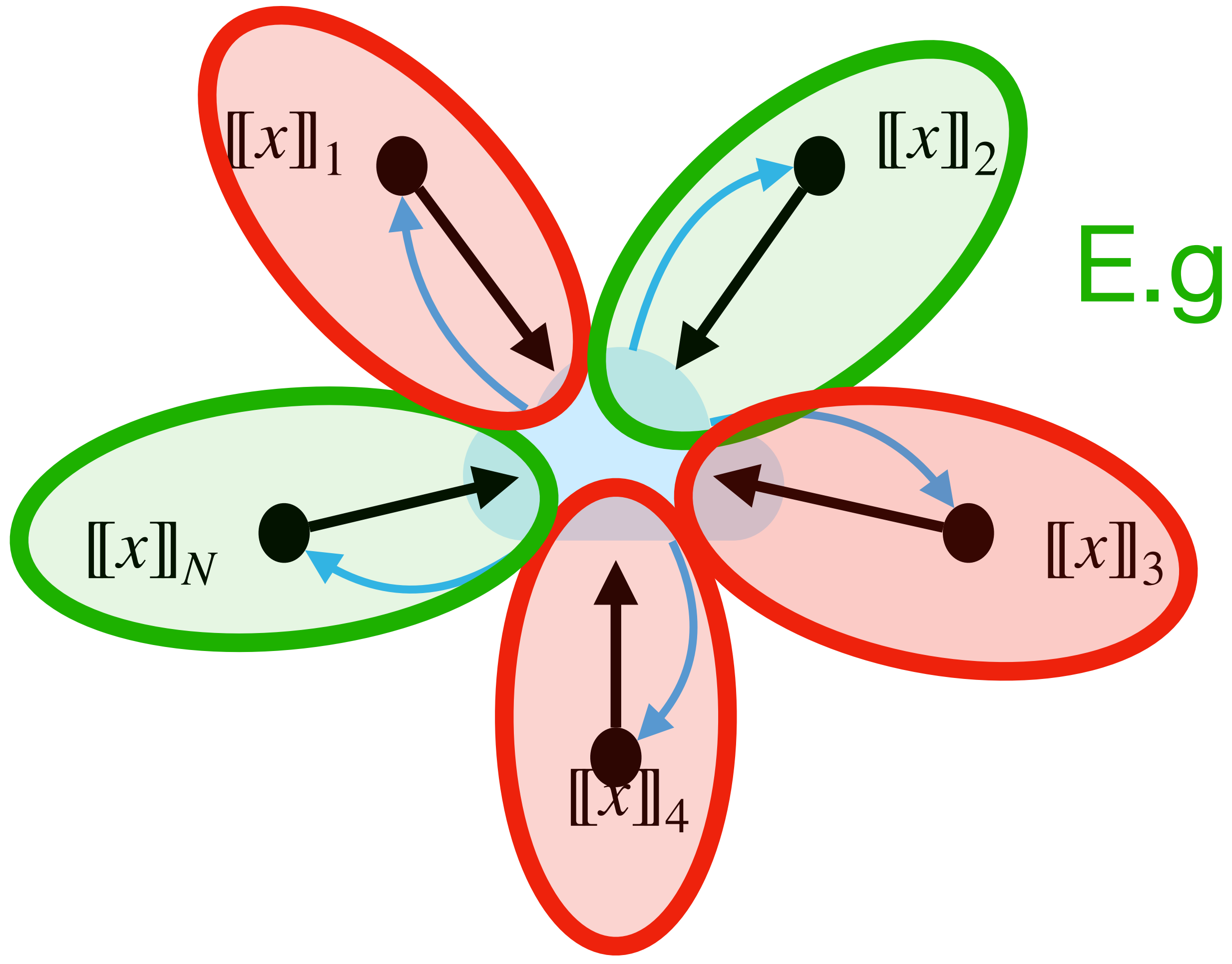
- $(\ell + 1, N)$-**threshold** linear secret sharing scheme (LSSS)
  - Linearity: $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$
  - Any set of ℓ shares is random and independent of x
  - Any set of $\ell + 1$ shares → coefficients $(x, r_1, \ldots, r_\ell)$ → all the shares

Background: Shamir's secret sharing

- $(\ell + 1, N)$-**threshold** linear secret sharing scheme (LSSS)
  - Linearity: $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$
  - Any set of ℓ shares is random and independent of x
  - Any set of $\ell + 1$ shares → coefficients $(x, r_1, \ldots, r_\ell)$ → all the shares
- $[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ is a **Reed-Solomon codeword** of $(x, r_1, \ldots, r_\ell)$

MPCitH with threshold LSSS (a.k.a TCitH)

- **[FR23]** Feneuil, Rivain. "Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head" (Asiacrypt 2023)
- ZK property $\Rightarrow$ only open ℓ parties
  - Verifier challenges a set $I \subseteq \{1,\ldots,N\}$ s.t. $|I| = \ell$
  - Prover opens $\{[\![x]\!]_i, \rho_i\}_{i \in I}$



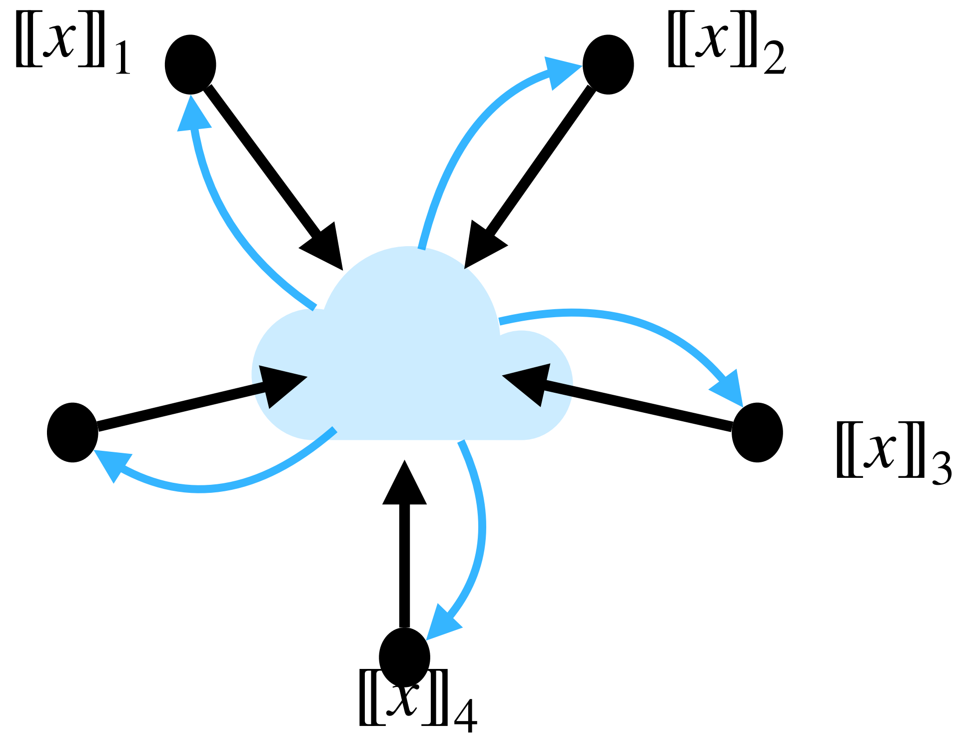
E.g. $\ell = 2$

MPCitH with threshold LSSS (a.k.a TCitH)

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 Chose 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 with threshold LSSS (a.k.a TCitH)

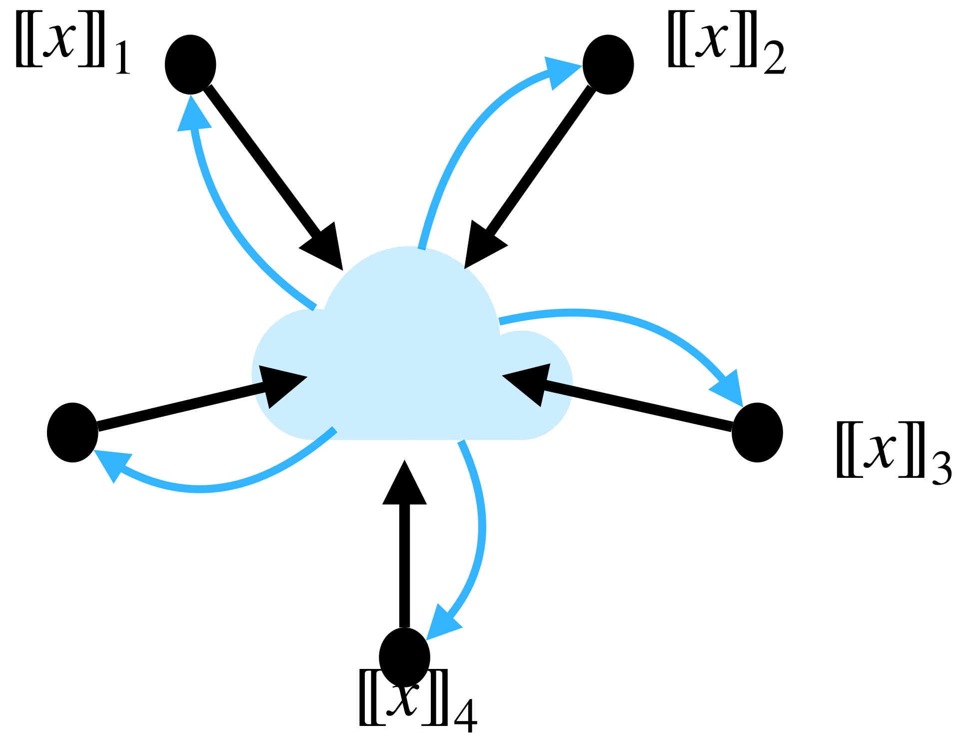
Threshold LSSS $\Rightarrow$ *cannot generate shares from seeds*

Prover

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

Verifier

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

← I

④ Open parties in I

$([\![x]\!]_i, \rho_i)_{i \in I}$ →

⑤ Check $\forall i \in I$
- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

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

Prover

Verifier

MPCitH with threshold LSSS (a.k.a TCitH)

① Generate and commit shares

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

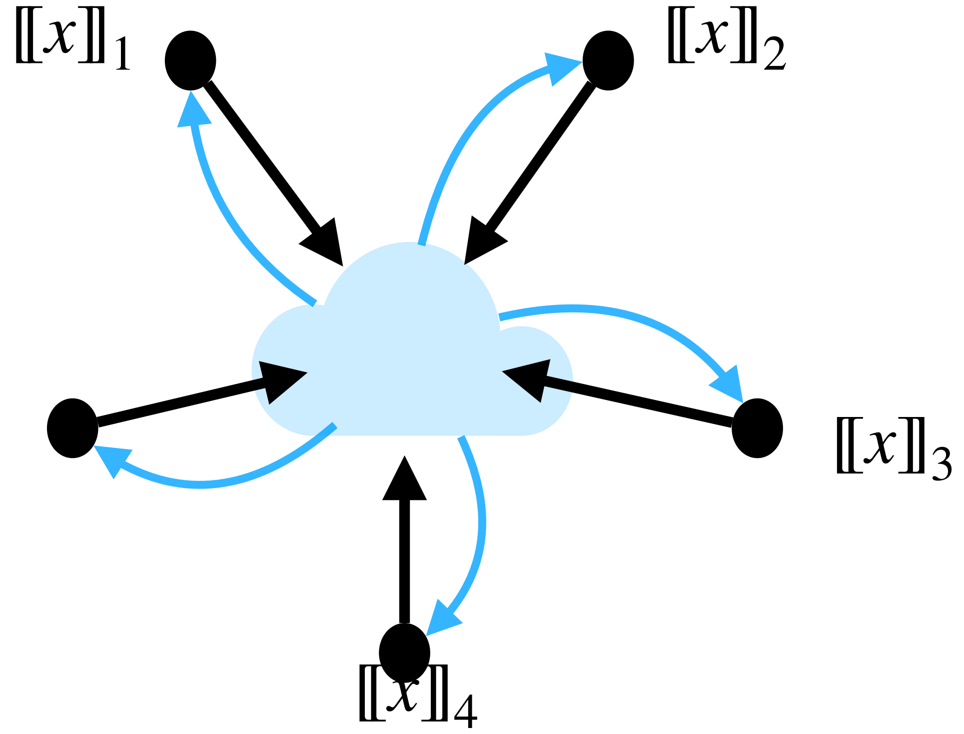
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

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

Threshold LSSS $\Rightarrow$ cannot generate shares from seeds

② Run MPC in their head



send broadcast

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

$[\![\alpha]\!]$ is an RS codeword $\Rightarrow \ell + 1$ shares fully determine the sharing

③ Chose random set of parties

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

I

④ Open parties in I

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

⑤ Check $\forall i \in I$

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

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

Prover

Verifier

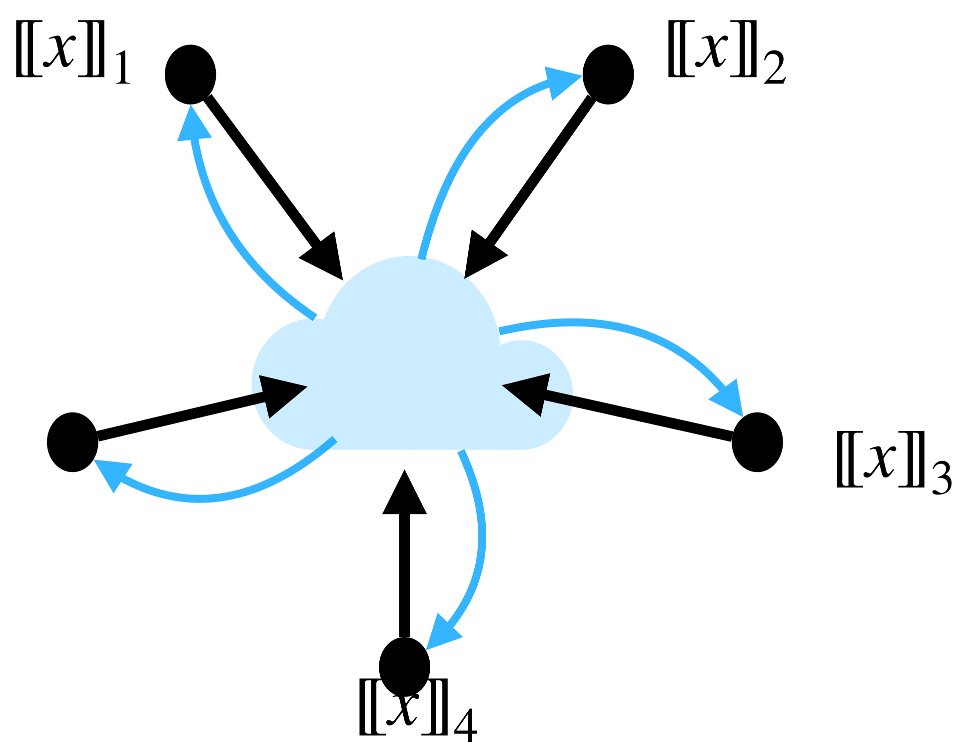
MPCitH with threshold LSSS (a.k.a TCitH)

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

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

Threshold LSSS $\Rightarrow$ cannot generate shares from seeds

② Run MPC in their head



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

$[\![\alpha]\!]$ is an RS codeword $\Rightarrow \ell + 1$ shares fully determine the sharing

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

I

④ Open parties in I

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

⑤ Check $\forall i \in I$
- Commitments $\mathrm{Com}^{\rho_i}([\![x]\!]_i)$
- MPC computation $[\![\alpha]\!]_i = \varphi([\![x]\!]_i)$

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

$\Rightarrow$ only $\ell + 1$ party computations required

Prover

Verifier

MPCitH with threshold LSSS (a.k.a TCitH)

① Generate and commit shares

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

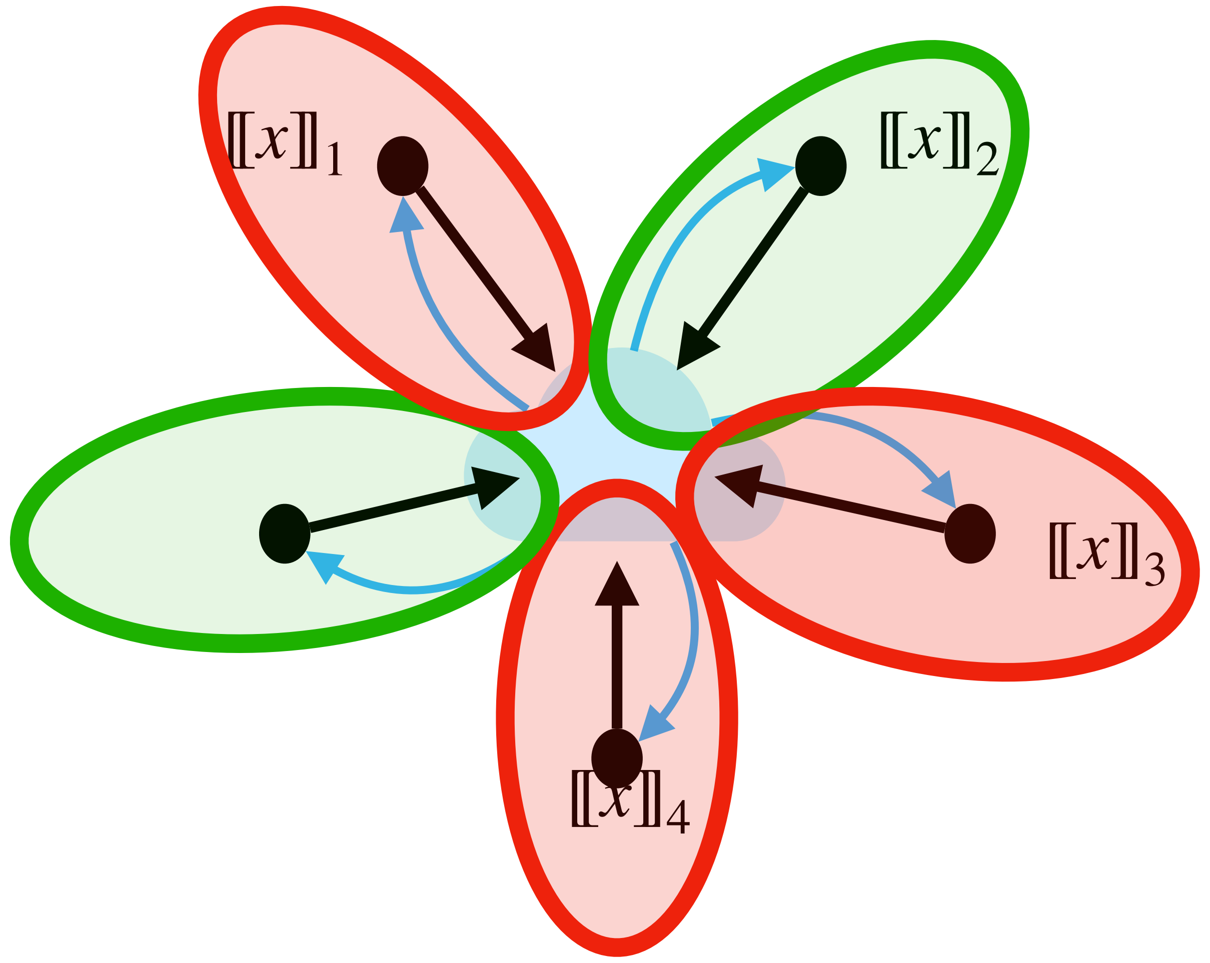
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

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

Threshold LSSS $\Rightarrow$ cannot generate shares from seeds

② Run MPC in their head



send broadcast

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

$[\![\alpha]\!]$ is an RS codeword $\Rightarrow \ell + 1$ shares fully determine the sharing

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

I

④ Open parties in I

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

⑤ Check $\forall i \in I$

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

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

Prover

ℓ parties opened instead of $N - 1$

$\Rightarrow$ only $\ell + 1$ party computations required

Verifier

MPCitH with threshold LSSS (a.k.a TCitH)

① Generate and commit shares

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

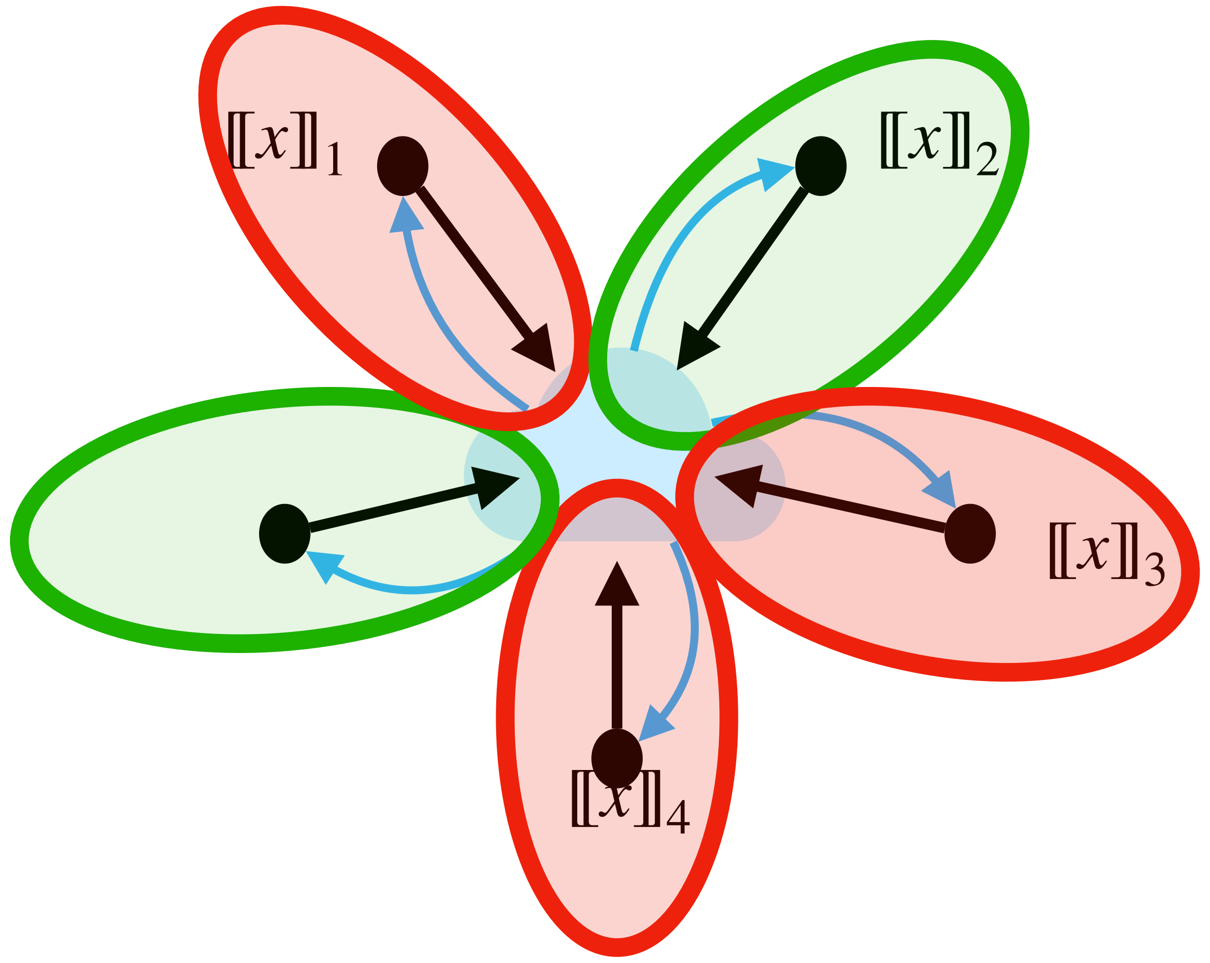
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

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

Threshold LSSS $\Rightarrow$ cannot generate shares from seeds

② Run MPC in their head



send broadcast

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

$[\![\alpha]\!]$ is an RS codeword $\Rightarrow \ell + 1$ shares fully determine the sharing

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

I

④ Open parties in I

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

⑤ Check $\forall i \in I$

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

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

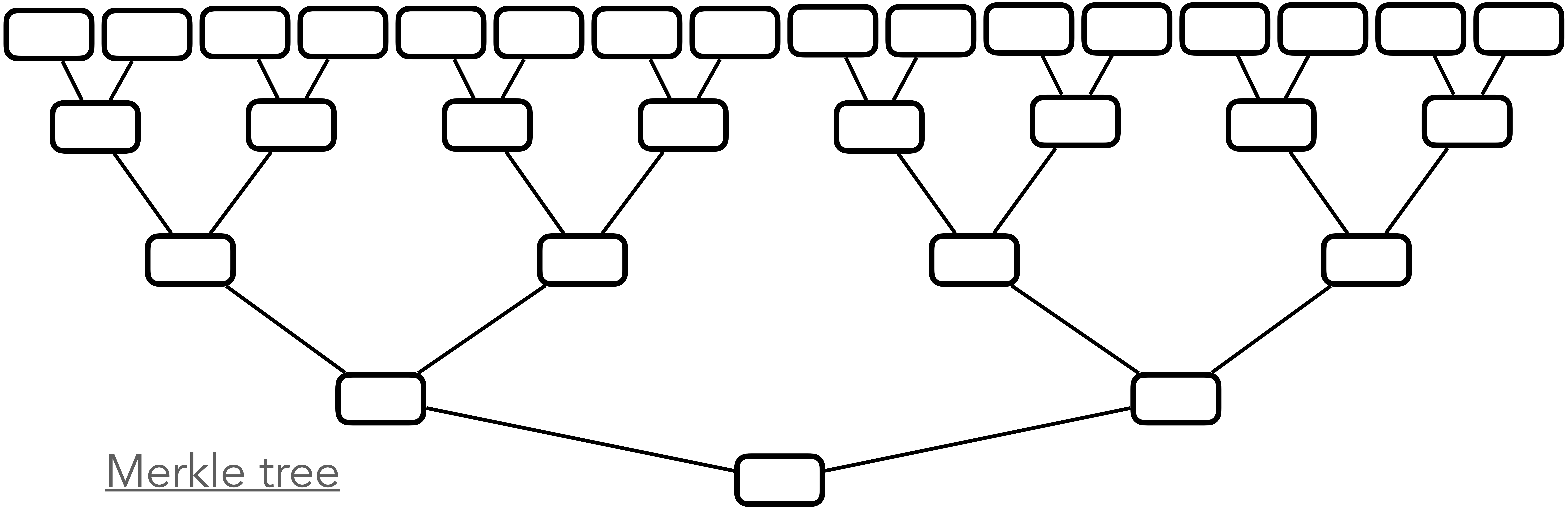
Prover

ℓ parties opened instead of $N - 1$

$\Rightarrow$ only $\ell + 1$ party computations required

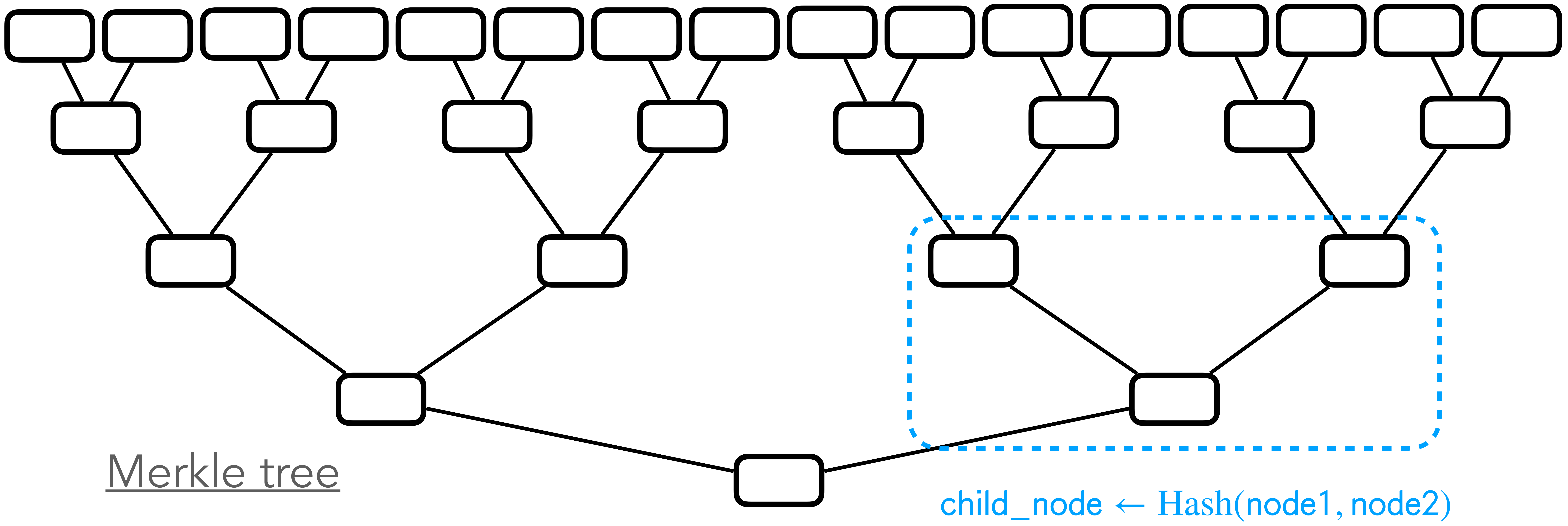
only ℓ party computations required

Sharing and commitments

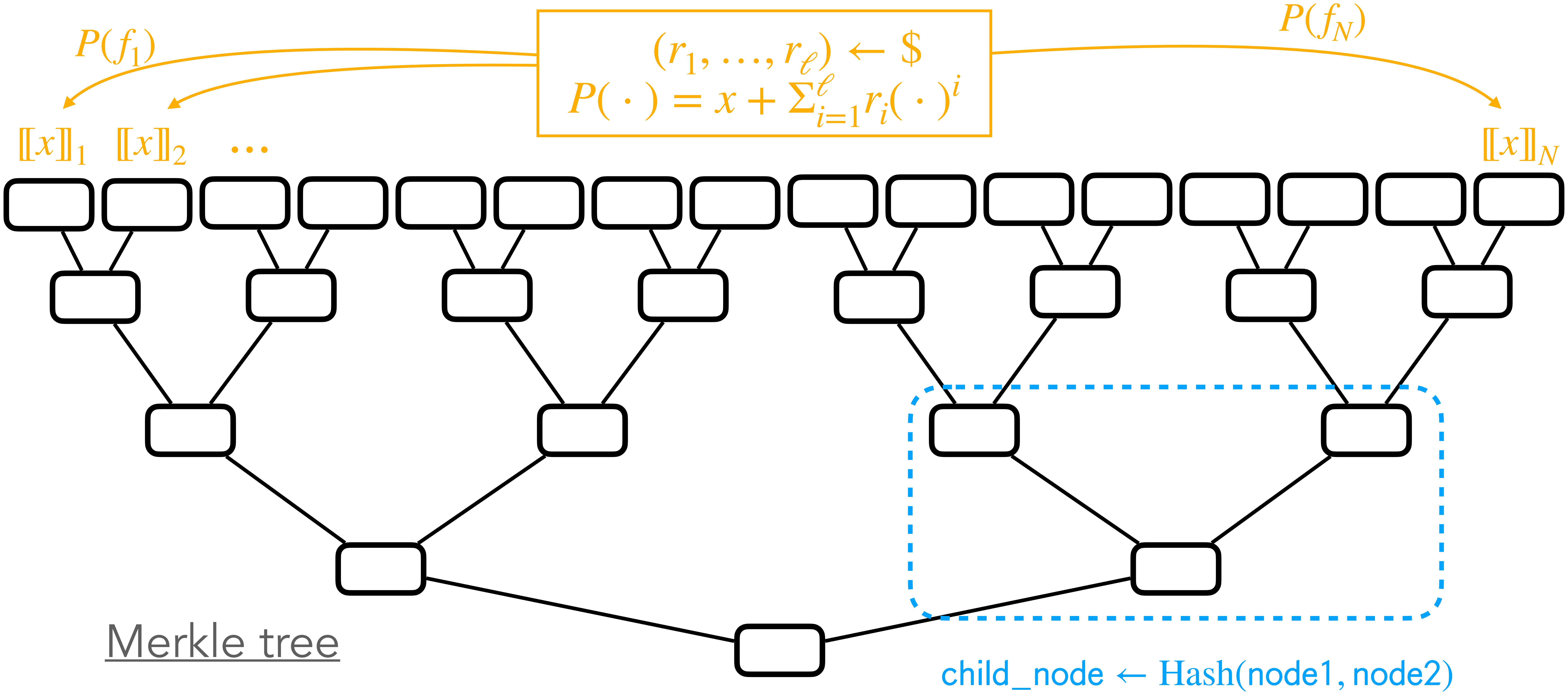


Merkle tree

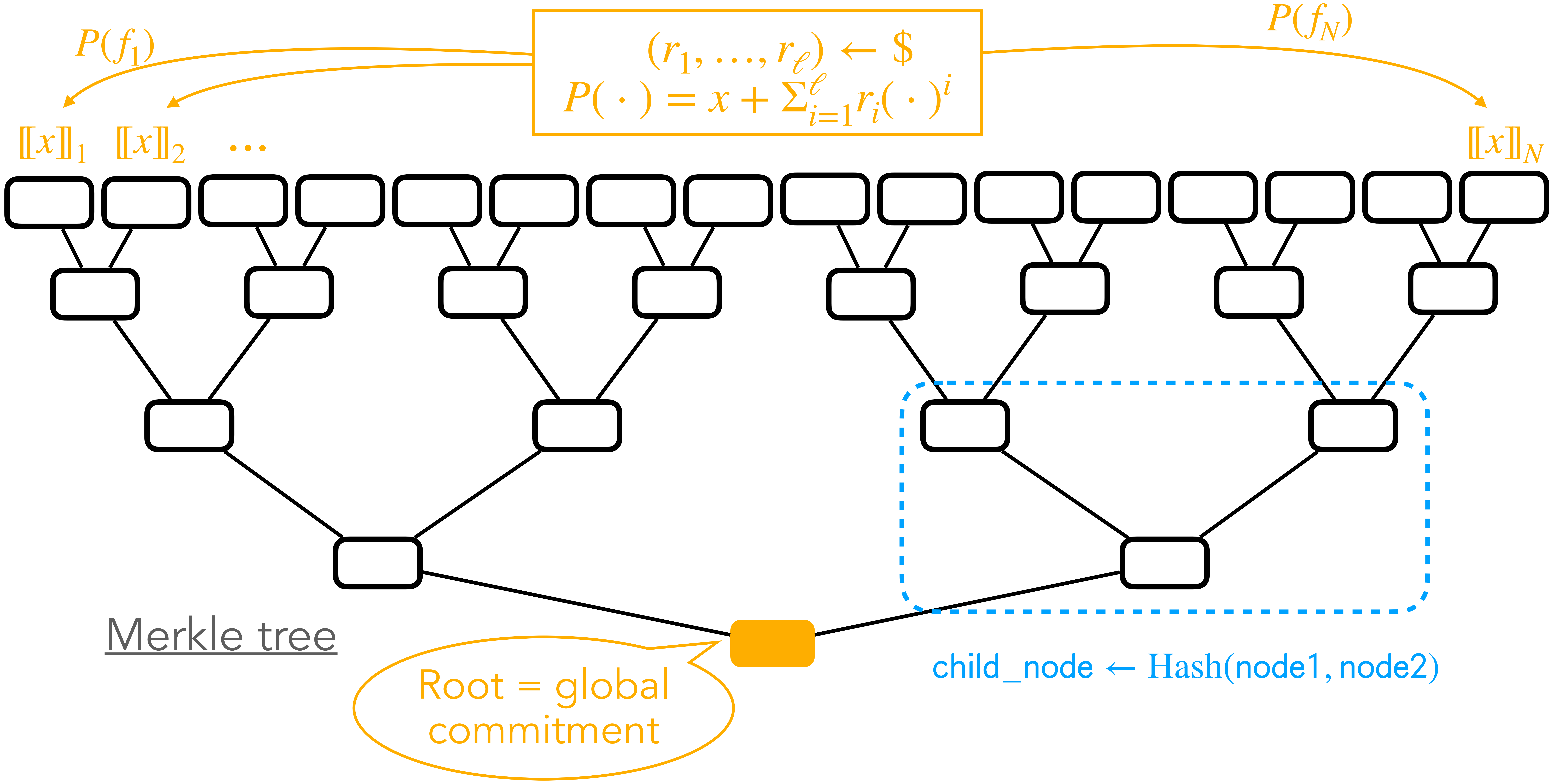
Sharing and commitments



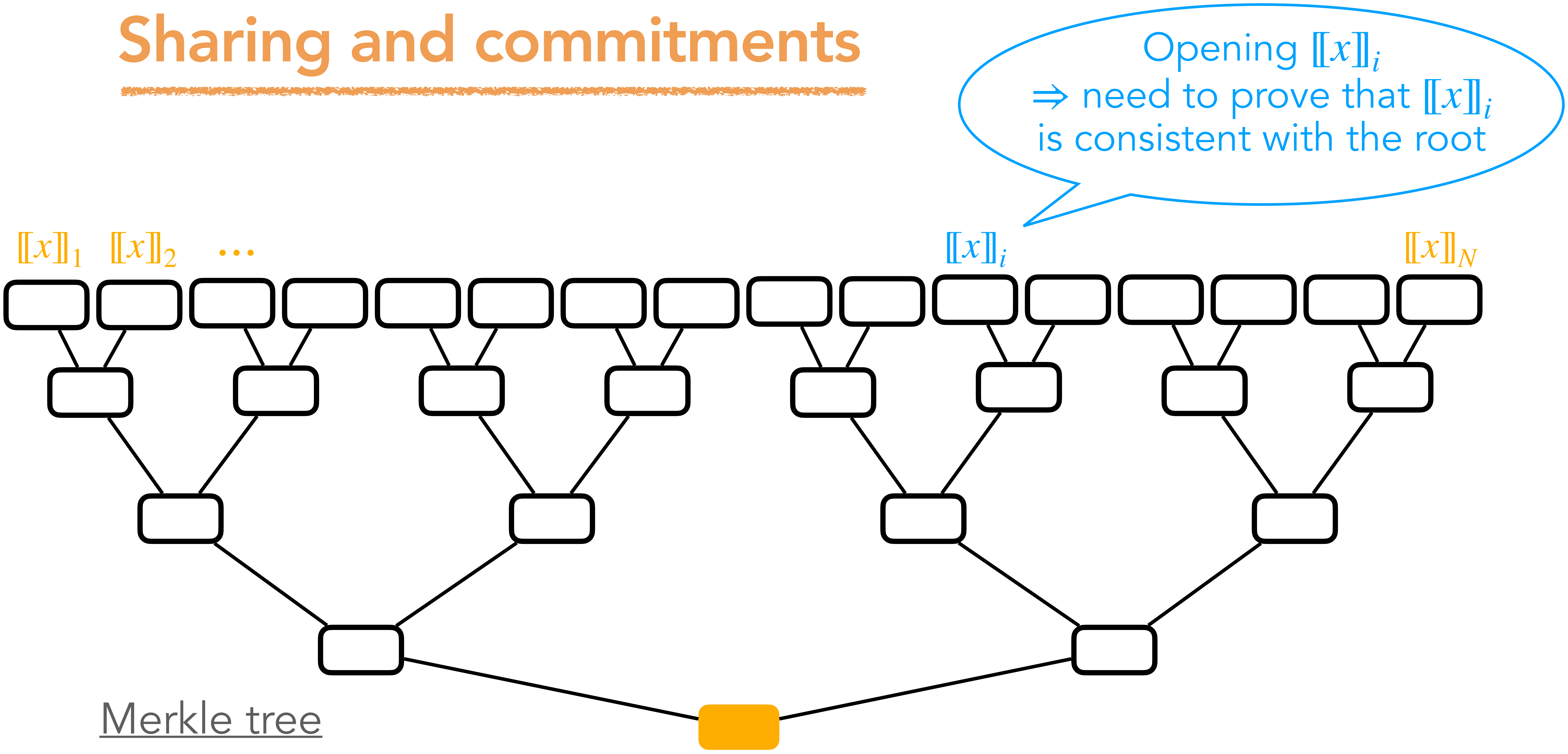
Sharing and commitments



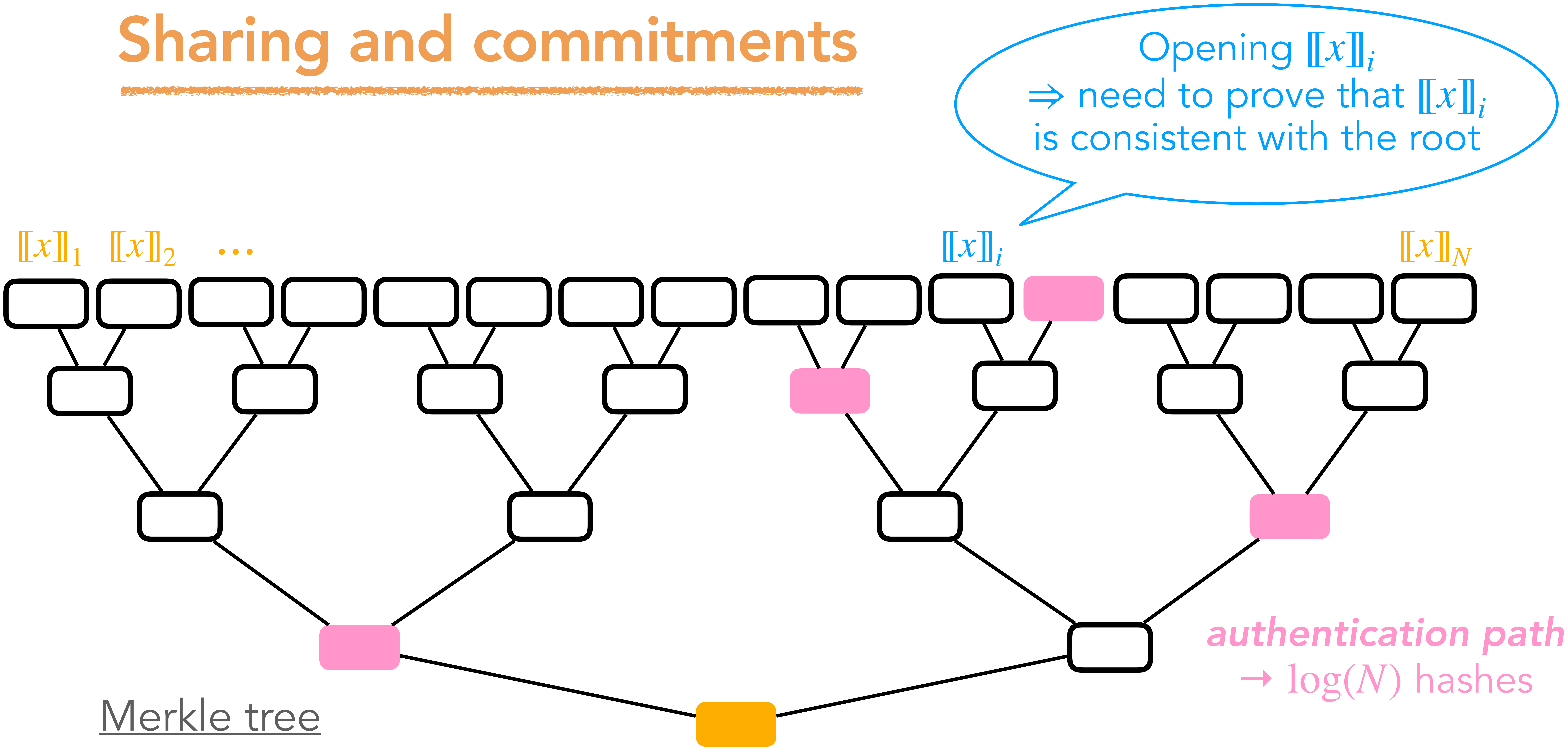
Sharing and commitments



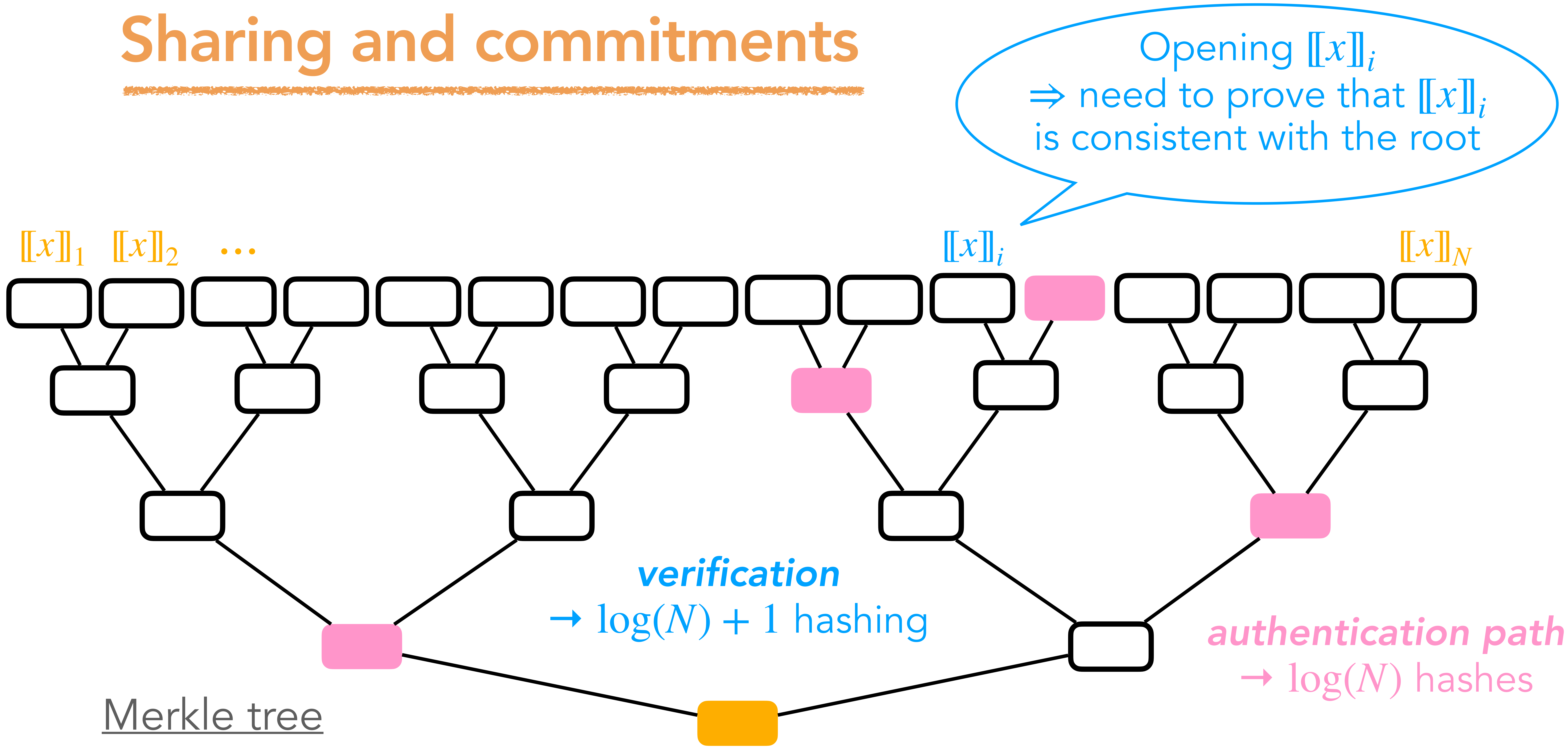
Sharing and commitments



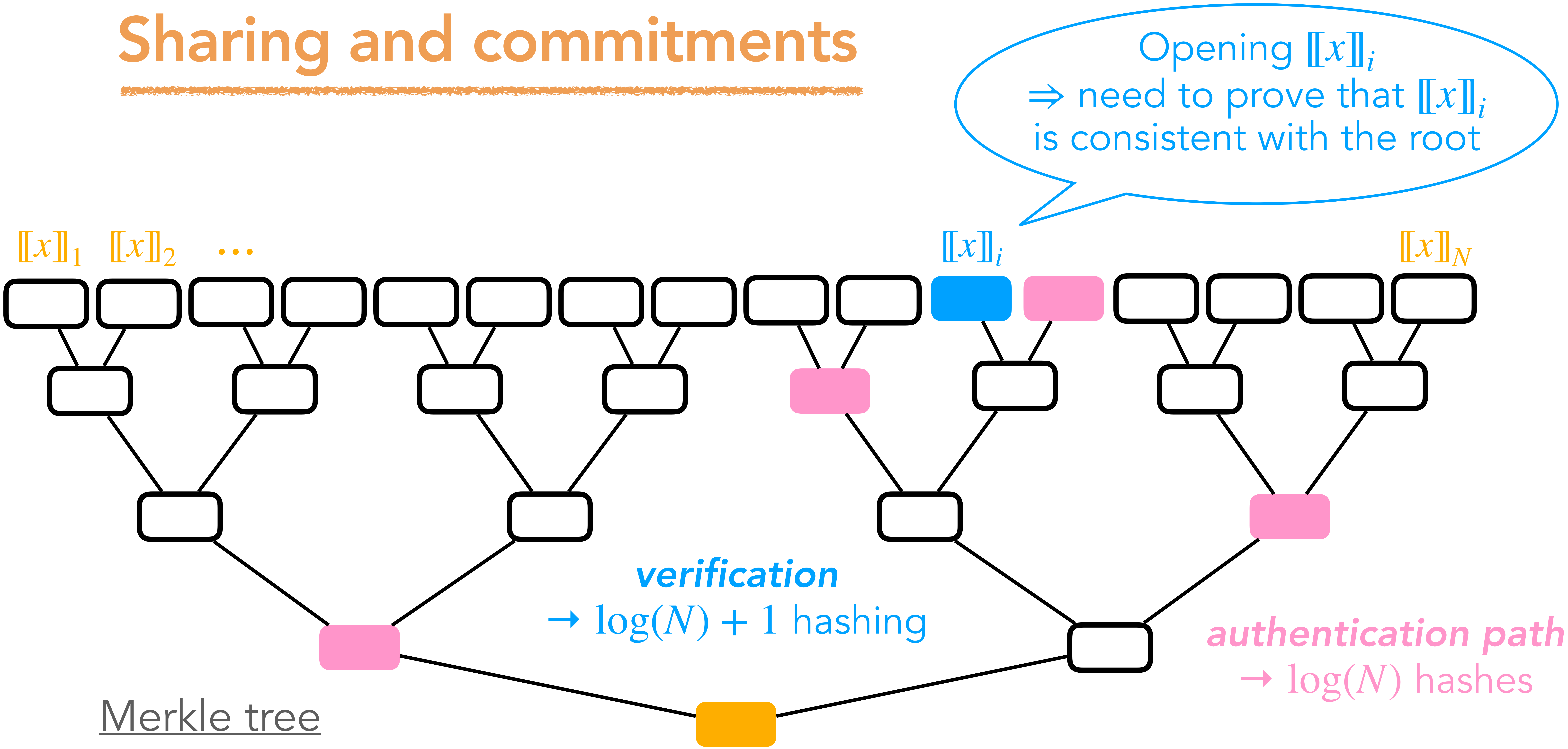
Sharing and commitments



Sharing and commitments

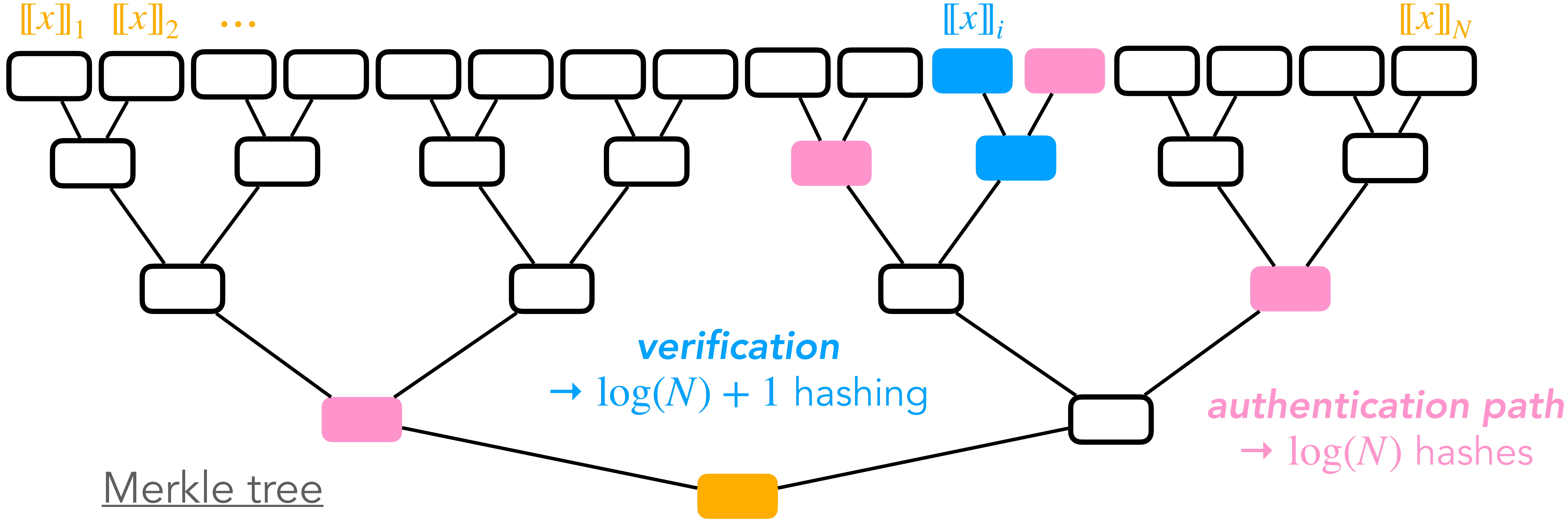


Sharing and commitments



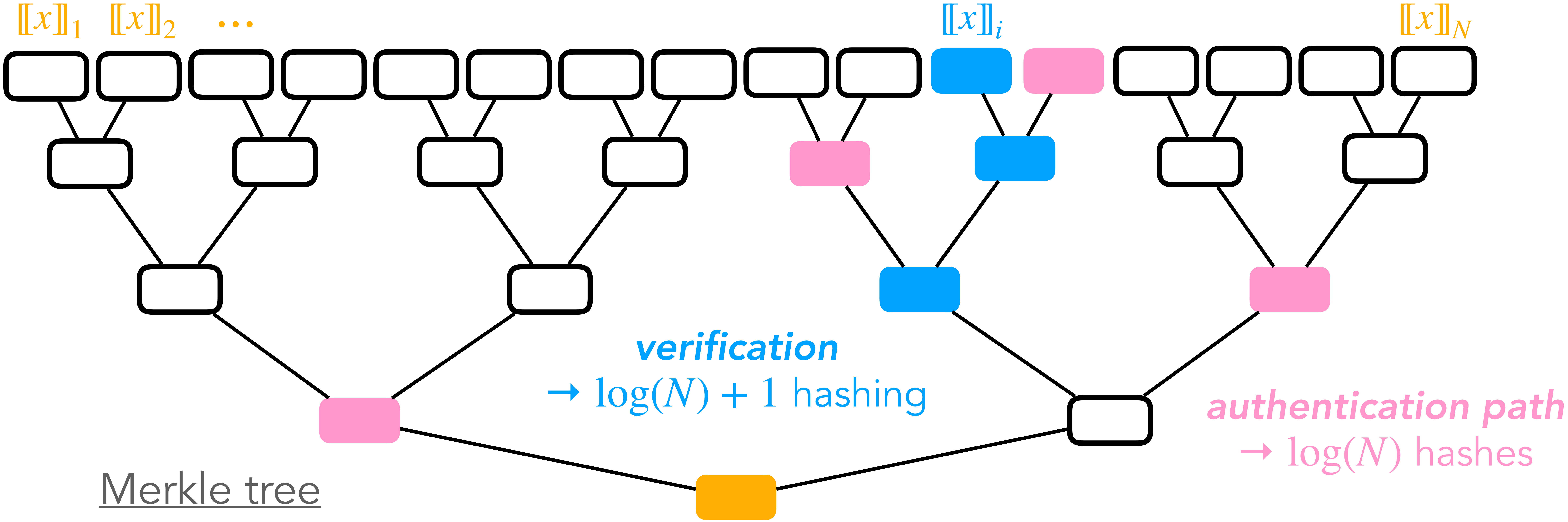
Sharing and commitments

Opening $[\![x]\!]_i$
$\Rightarrow$ need to prove that $[\![x]\!]_i$ is consistent with the root

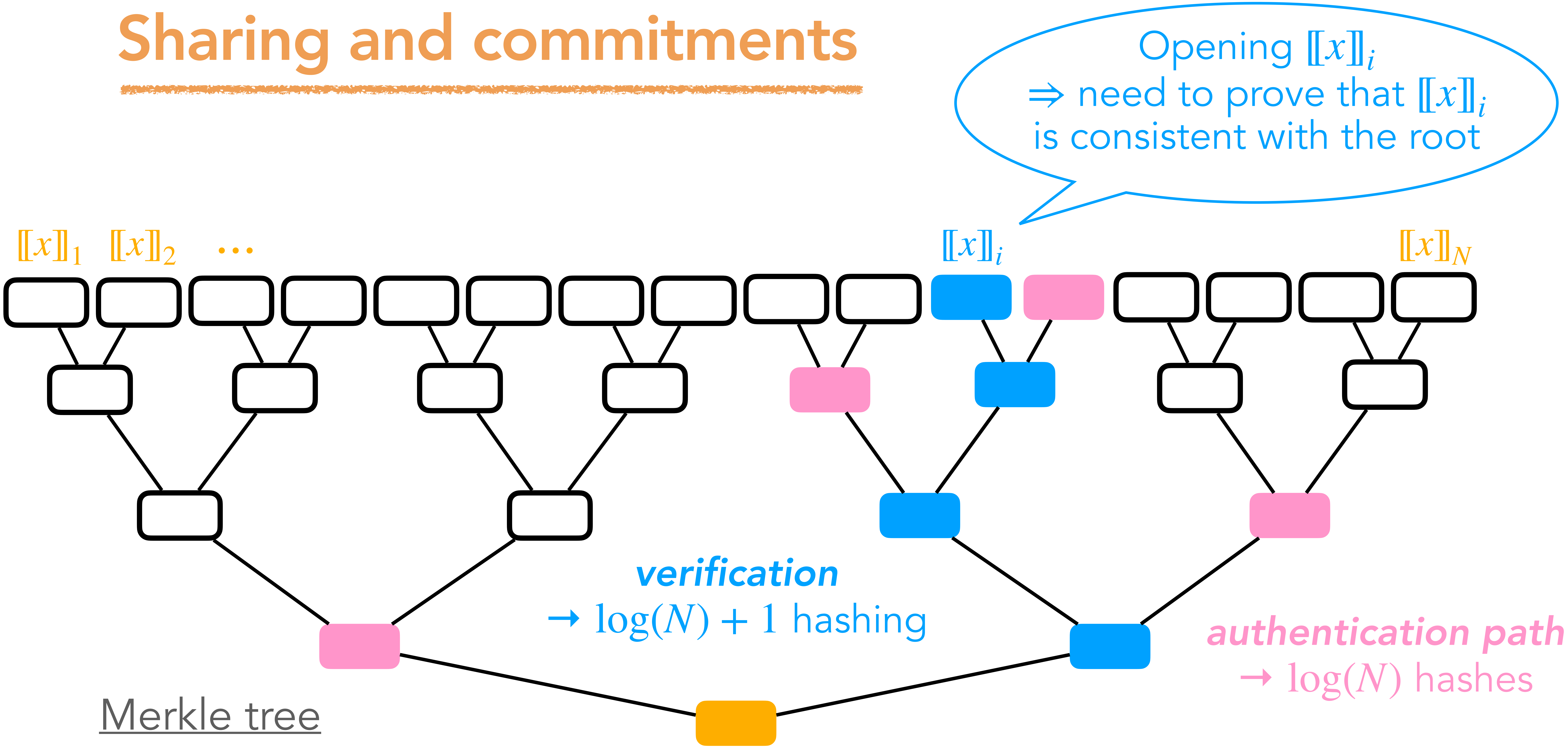


Sharing and commitments

Opening $[\![x]\!]_i$
$\Rightarrow$ need to prove that $[\![x]\!]_i$ is consistent with the root



Sharing and commitments

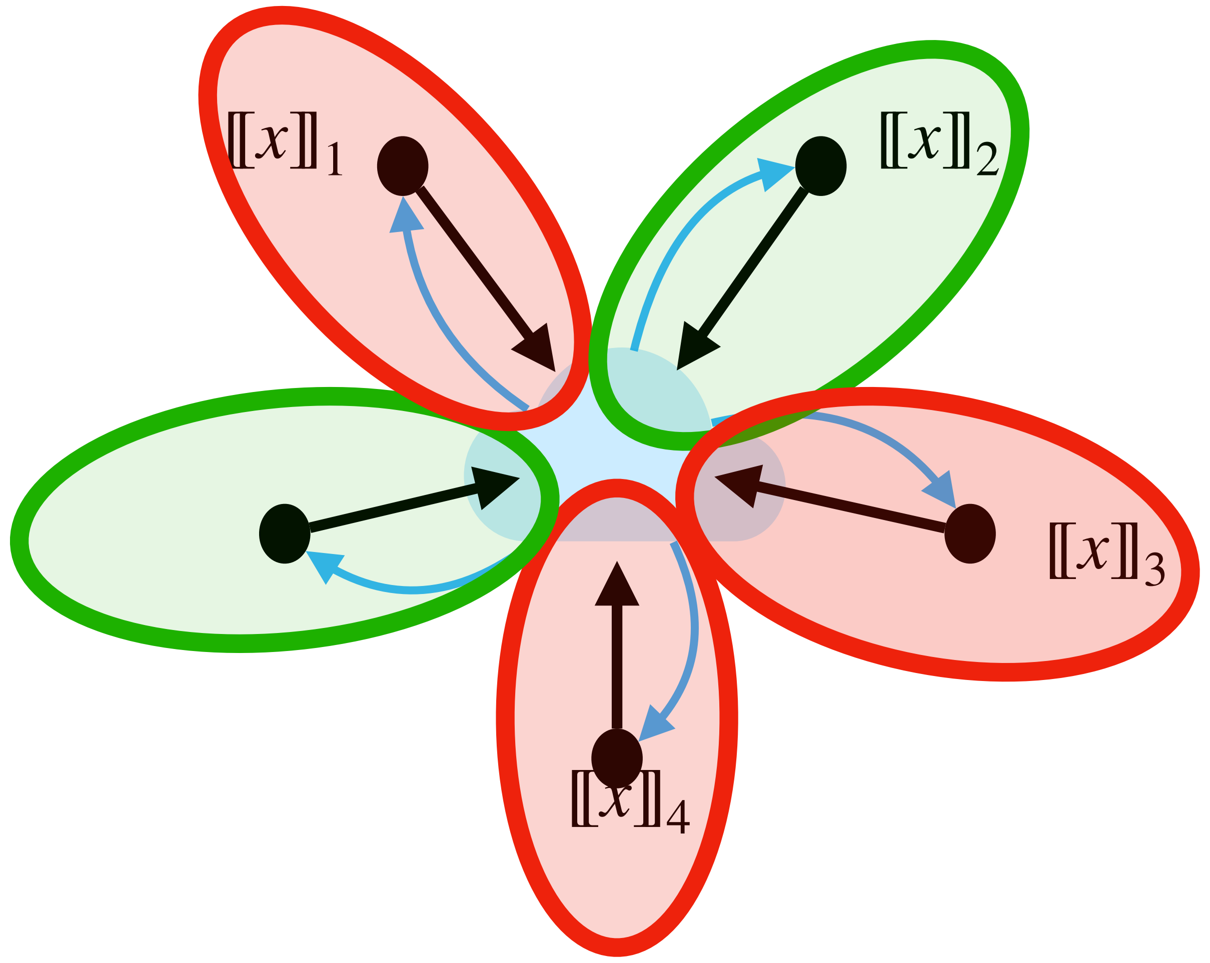


MPCitH with threshold LSSS (a.k.a TCitH)

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Merkle root →

send broadcast

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

← I

$\{[\![x]\!]_i \boxed{\text{auth}_i}\}_{i \in I}$ →

authentication path

③ Chose 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

Soundness

- One would expect:

$$P[\text{cheat detected}] = \frac{\ell}{N} \quad \Rightarrow \quad \boxed{\text{Soundness error} = 1 - \frac{\ell}{N}}$$

Soundness

- One would expect:

$$P[\text{cheat detected}] = \frac{\ell}{N} \quad \Rightarrow \quad \boxed{\text{Soundness error} = 1 - \frac{\ell}{N}}$$

🤔 *not really good*

Soundness

not really good

- One would expect:

$$P[\text{cheat detected}] = \frac{\ell}{N} \quad \Rightarrow \quad \boxed{\text{Soundness error} = 1 - \frac{\ell}{N}}$$

- But the verifier also check broadcast sharings $[\![\alpha]\!]$
  - must be valid Shamir's secret sharings
  - i.e. valid Reed-Solomon codewords

    $\Rightarrow$ limits the cheating possibilities

Soundness

- One would expect:

$$P[\text{cheat detected}] = \frac{\ell}{N} \quad \Rightarrow \quad \boxed{\cancel{\text{Soundness error} = 1 - \frac{\ell}{N}}}$$

- But the verifier also check broadcast sharings $[\![\alpha]\!]$
  - must be valid Shamir's secret sharings
  - i.e. valid Reed-Solomon codewords

  $\Rightarrow$ limits the cheating possibilities

- We actually have: $\boxed{\text{Soundness error} = \frac{1}{\binom{N}{\ell}}}$

Soundness

- One would expect:

$$P[\text{cheat detected}] = \frac{\ell}{N} \quad \Rightarrow \quad \boxed{\cancel{\text{Soundness error} = 1 - \frac{\ell}{N}}}$$

- But the verifier also check broadcast sharings $[\![\alpha]\!]$
  - must be valid Shamir's secret sharings
  - i.e. valid Reed-Solomon codewords

  $\Rightarrow$ limits the cheating possibilities

- We actually have: $\boxed{\text{Soundness error} = \frac{1}{\binom{N}{\ell}}}$

with false positives

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

Soundness

- One would expect:

$$P[\text{cheat detected}] = \frac{\ell}{N} \quad \Rightarrow \quad \boxed{\cancel{\text{Soundness error} = 1 - \frac{\ell}{N}}}$$

- But the verifier also check broadcast sharings $[\![\alpha]\!]$
  - must be valid Shamir's secret sharings
  - i.e. valid Reed-Solomon codewords

  $\Rightarrow$ limits the cheating possibilities

- We actually have: $\boxed{\text{Soundness error} = \frac{1}{\binom{N}{\ell}}}$

with false positives

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

not so great

Soundness

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

Why? 🤔

Soundness

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

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

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

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 $\rightarrow$ Prover : random $\{\gamma_j\}$
- Prover $\rightarrow$ Verifier : $[\![\xi]\!] = \Sigma_j\, \gamma_i \cdot [\![x_j]\!]$
- Before MPC computation

Soundness

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

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

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

🤩

Comparison

$\ell = 1 \;\Rightarrow\;$ 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: $O(N)$

😀

Comparison

$\ell = 1 \;\Rightarrow\;$ 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: $O(\log N)$	😀

Comparison

$\ell = 1 \;\Rightarrow\;$ 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	🙁

Comparison

$\ell = 1 \;\Rightarrow\;$ 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
Number of parties		$N \leq \lvert\mathbb{F}\rvert$

Comparison

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

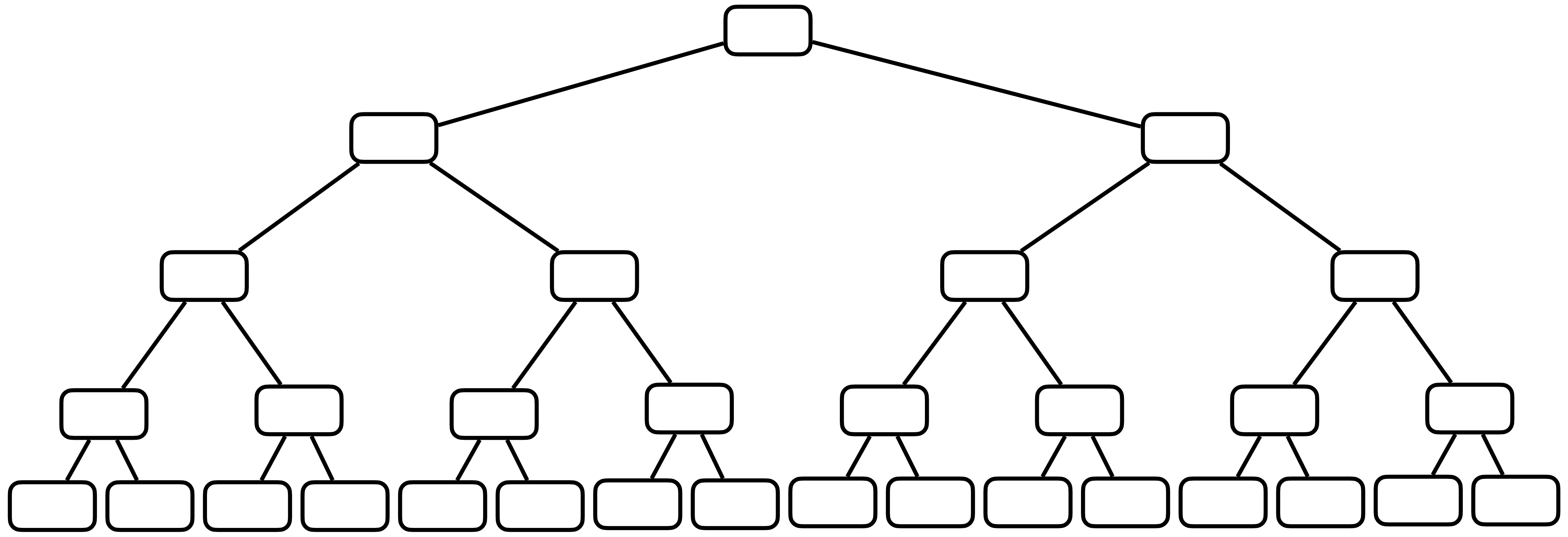
	MPCitH + seed trees + hypercube [AGHHJY23]	**TCitH** $\ell = 1$
Prover runtime	Party emulations: log *N* +1 Symmetric crypto: *O(N)*	Party emulations: 2 Symmetric crypto: *O(N)*
Verifier runtime	Party emulations: log *N* Symmetric crypto: *O(N)*	Party emulations: 1 Symmetric crypto: *O(log N)*
Size of tree	128-bit security: ~2KB 256-bit security: ~8KB	128-bit security: ~4KB 256-bit security: ~16KB
Number of parties		$N \leq \lvert\mathbb{F}\rvert$

Getting rid of these limitations

→ TCitH with GGM tree

TCitH with GGM trees

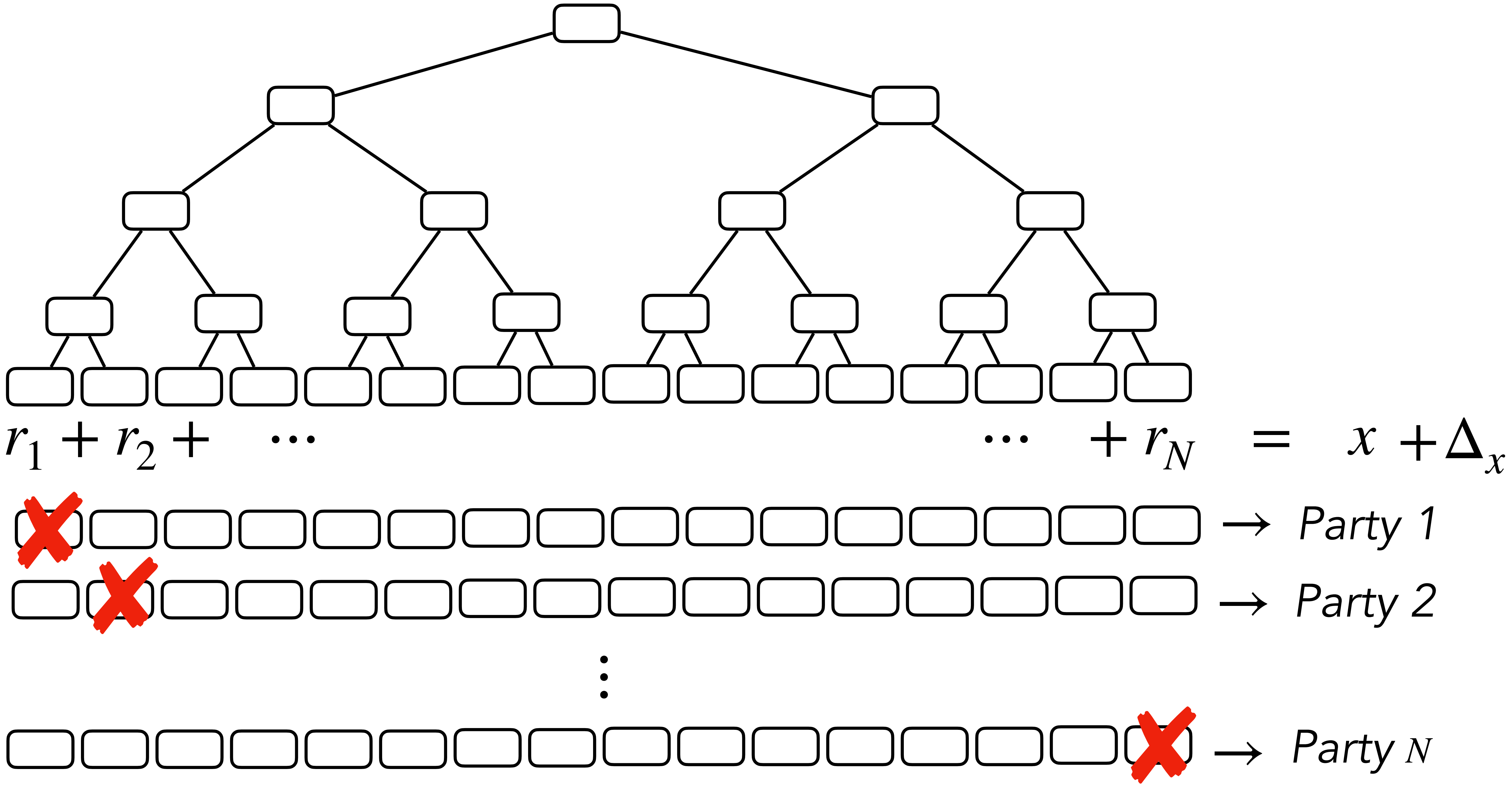
<u>Step 1</u>: Generate a replicated secret sharing of x [ISN89]



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

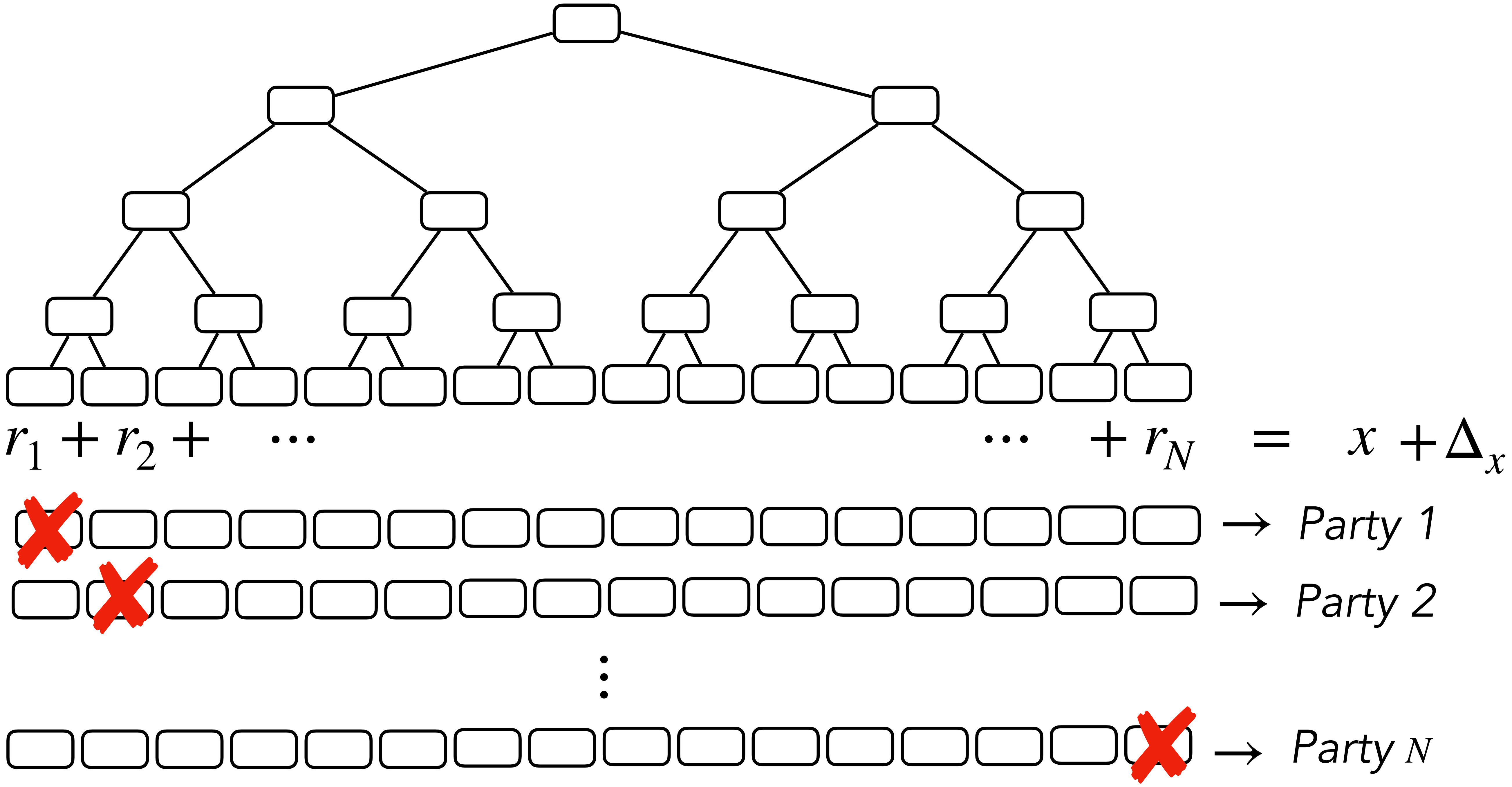
TCitH with GGM trees

<u>Step 1</u>: Generate a replicated secret sharing of x [ISN89]



TCitH with GGM trees

Step 1: Generate a replicated secret sharing of x [ISN89]



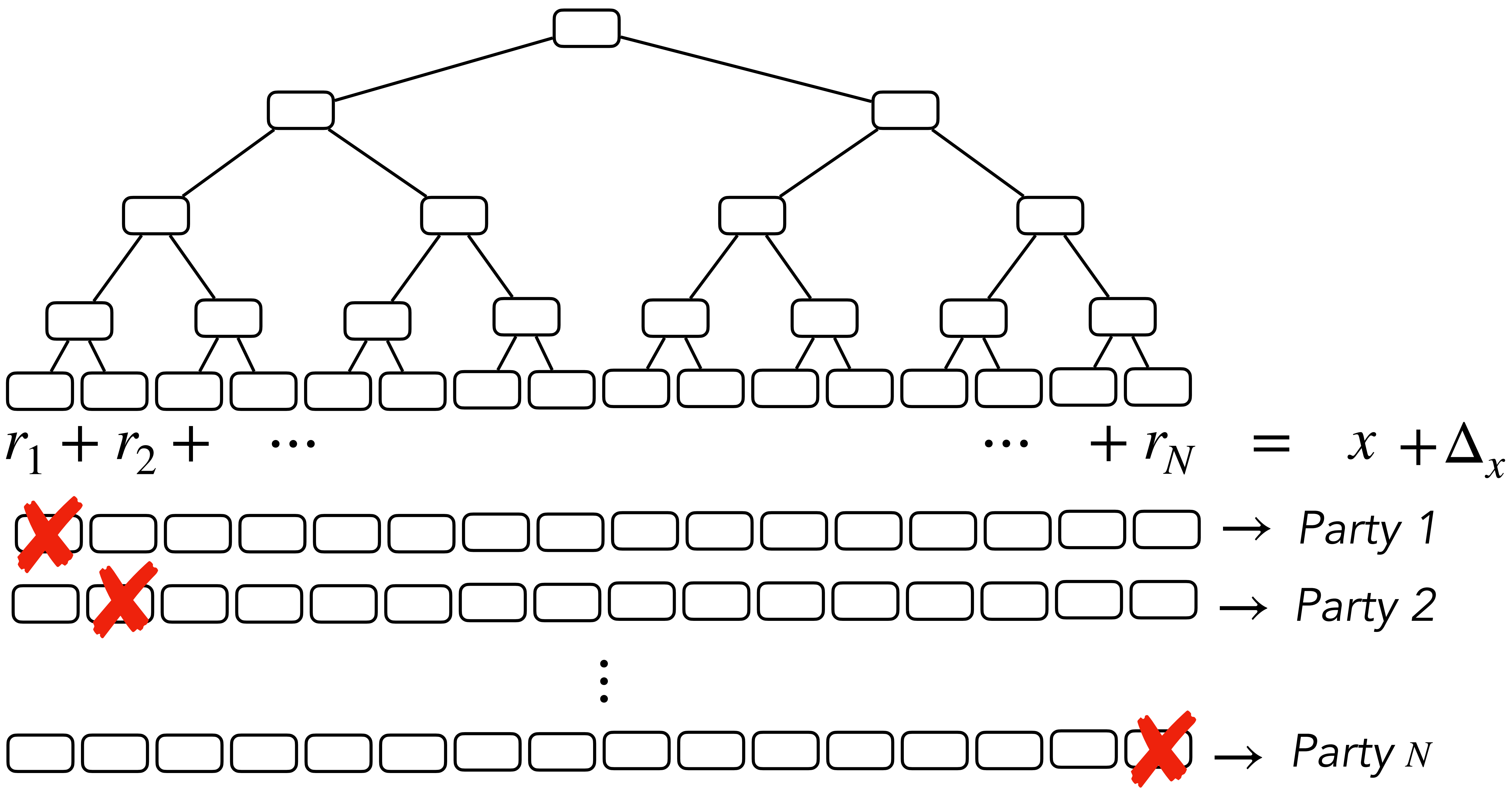
Step 2: Convert it into a Shamir's secret sharing [CDI05]

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 of x [ISN89]



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

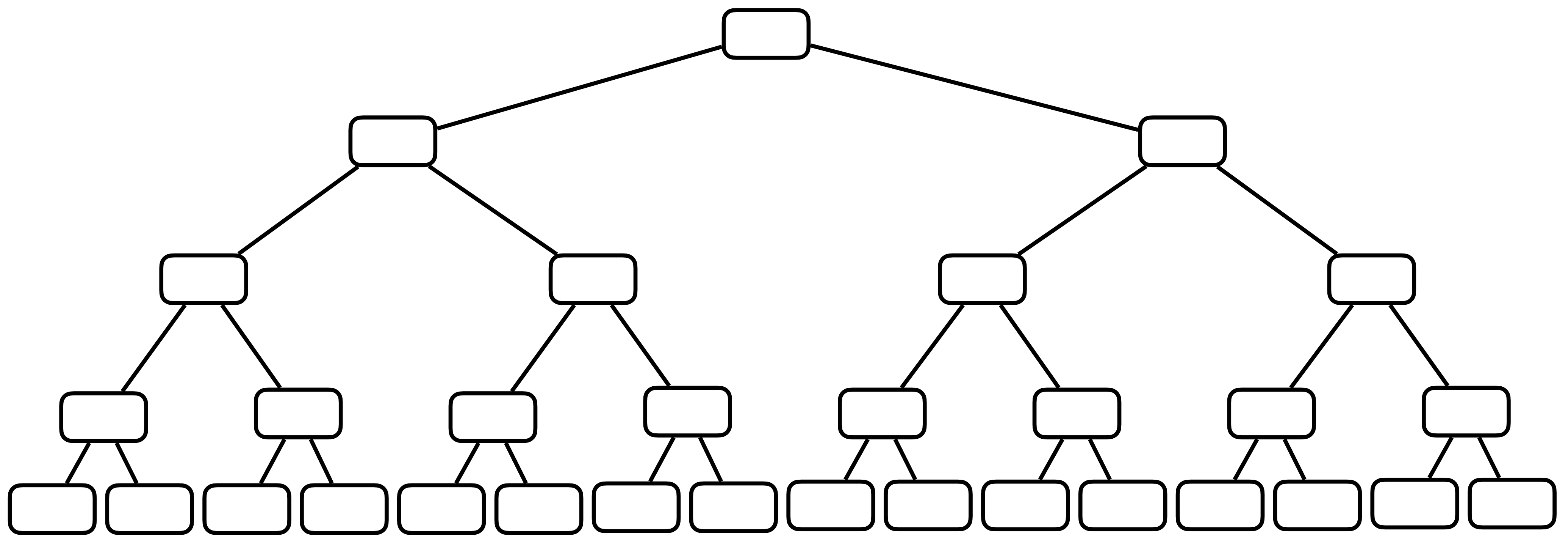
Let $P(X) = \Delta_x + \sum_j r_j P_j(X)$

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

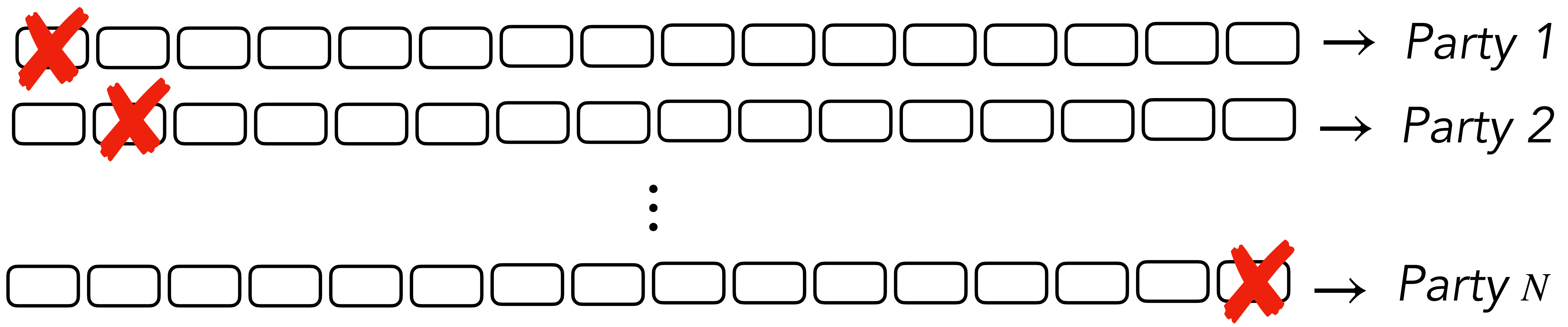
💡 $[\![x]\!] = (P(e_1), \ldots, P(e_N))$ is a valid Shamir's secret sharing of x

TCitH with GGM trees

Step 1: Generate a replicated secret sharing of x [ISN89]



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



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

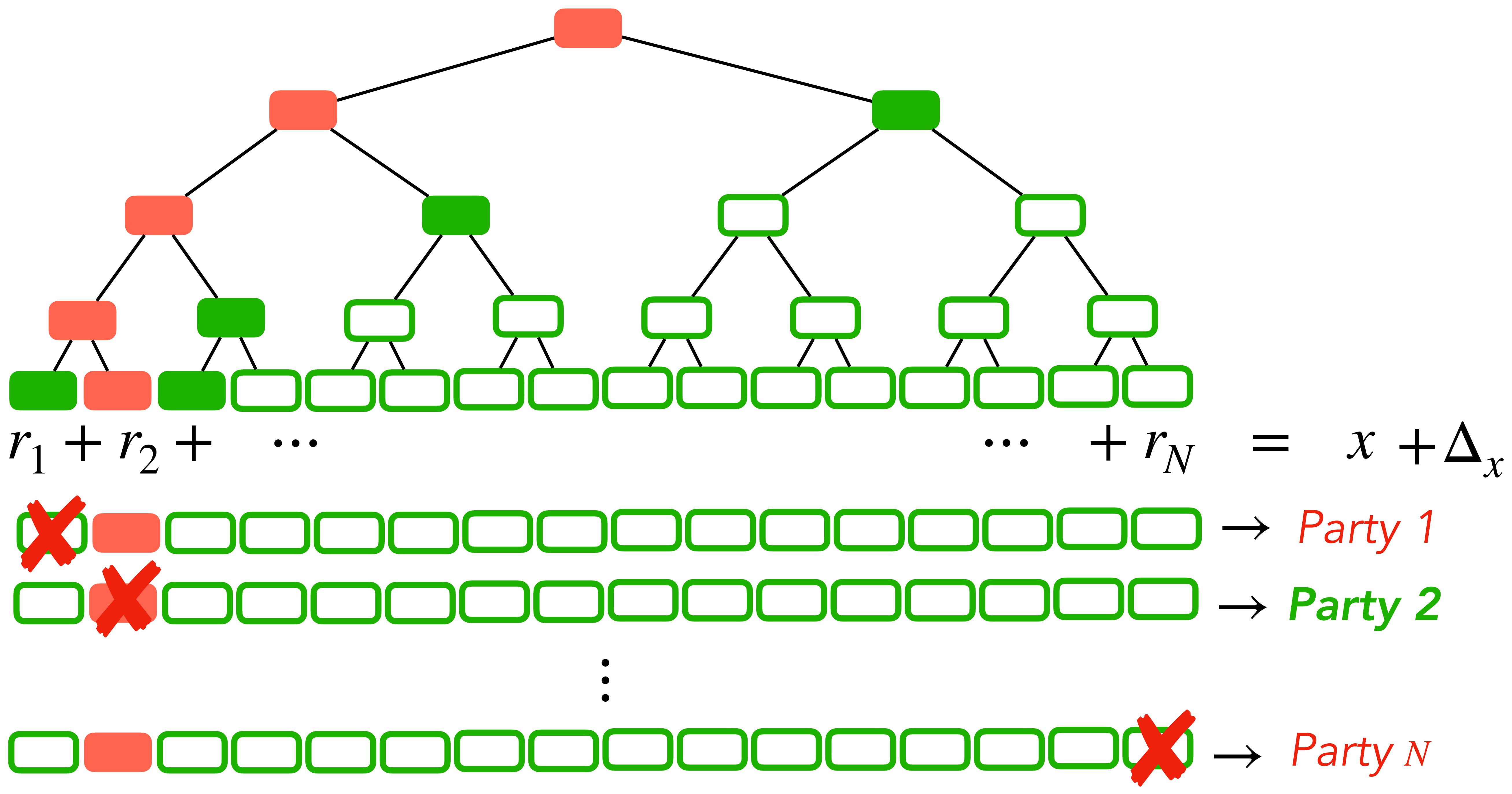
💡 Party i can compute

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

(since $P_i(e_i) = 0$)

TCitH with GGM trees

Step 1: Generate a replicated secret sharing of x [ISN89]



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

🛠️ Can be adapted to $\ell > 1$

TCitH with GGM trees

🛠️ Can be adapted to $\ell > 1$

🌲 Size of GGM tree

TCitH with GGM trees

🛠️ Can be adapted to $\ell > 1$

🌲 Size of GGM tree

😇 Good soundness (only valid sharings)

TCitH with GGM trees

🛠 Can be adapted to $\ell > 1$

🌲 Size of GGM tree

😇 Good soundness (only valid sharings)

🐌 Loose fast verification

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:

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

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

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

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

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

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

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

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

Here: $\ell \cdot \deg f_j$ $\left(\frac{1}{|\mathbb{F}|}\right)^{\#\alpha}$

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

Soundness error

Signature from MQ and TCitH

MQ Problem

- Parameters
  - A field $\mathbb{F}_q$, $n \in \mathbb{N}$ (# variables) , $m \in \mathbb{N}$ (# equations)
- Let
  - $x \leftarrow \mathbb{F}_q^n$ *(MQ solution)*
  - $A_i \leftarrow \mathbb{F}_q^{n \times n} \quad \forall i \in [1:m]$ *(m random matrices)*
  - $b_i \leftarrow \mathbb{F}_q^n \quad \forall i \in [1:m]$ *(m random vectors)*
  - $y = (y_1, \ldots, y_m) \in \mathbb{F}_q^m$ s.t. $\begin{cases} y_1 &= x^T A_1 x + b_1^T x \\ &\vdots \\ y_m &= x^T A_m x + b_m^T x \end{cases}$
- From $\big(\{A_i\}, \{b_i\}, y\big)$ find x

Checking a MQ instance = checking m quadratic constraints on the secret x

We can directly apply the previous protocol

$\Downarrow$

|sig| $\approx$ 3 kB

Shorter Signatures from TCitH-GGM

	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 from TCitH-GGM

	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 %

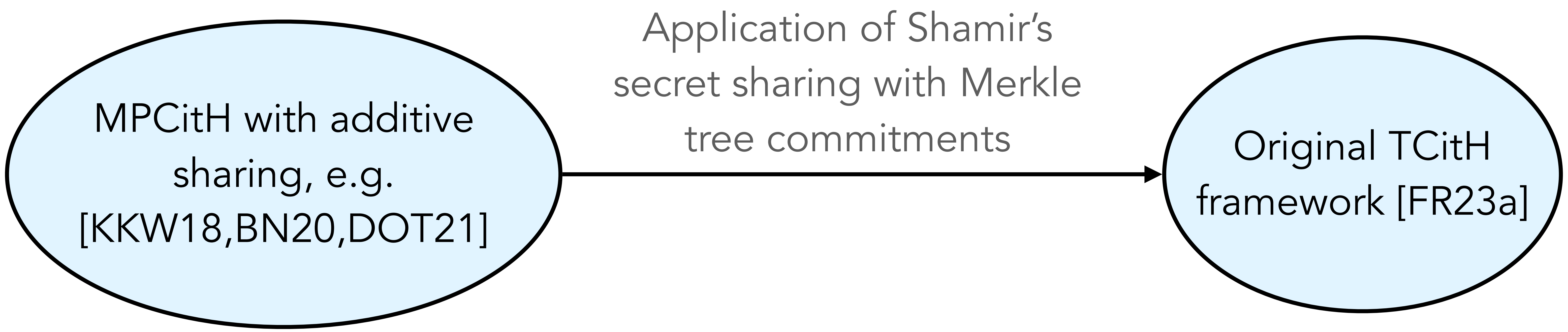
* $N = 256$ * $N = 2048$

Shorter Signatures from TCitH-GGM

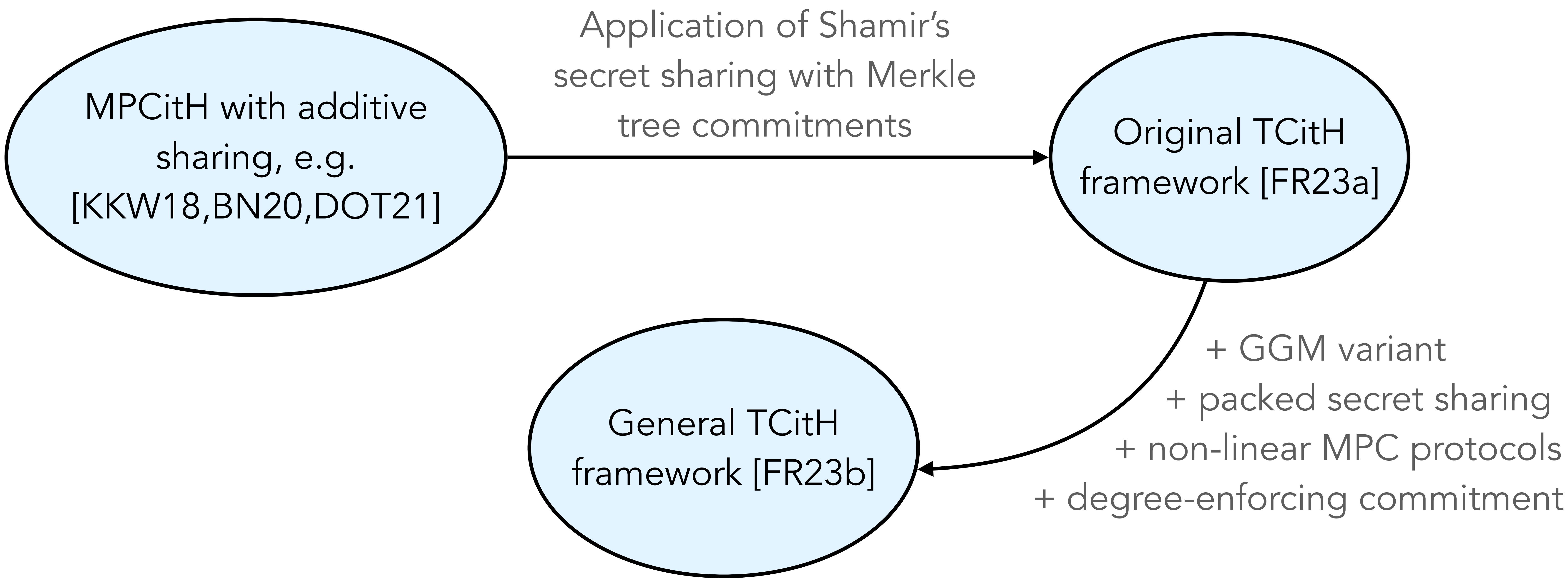
Two very recent works :

- [BBMO+24] 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: **tree merging** & **proof of work**
  - **Apply to TCitH-GGM** (gain of ~500 B at 128-bit security)
- [BFGNR24] 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**

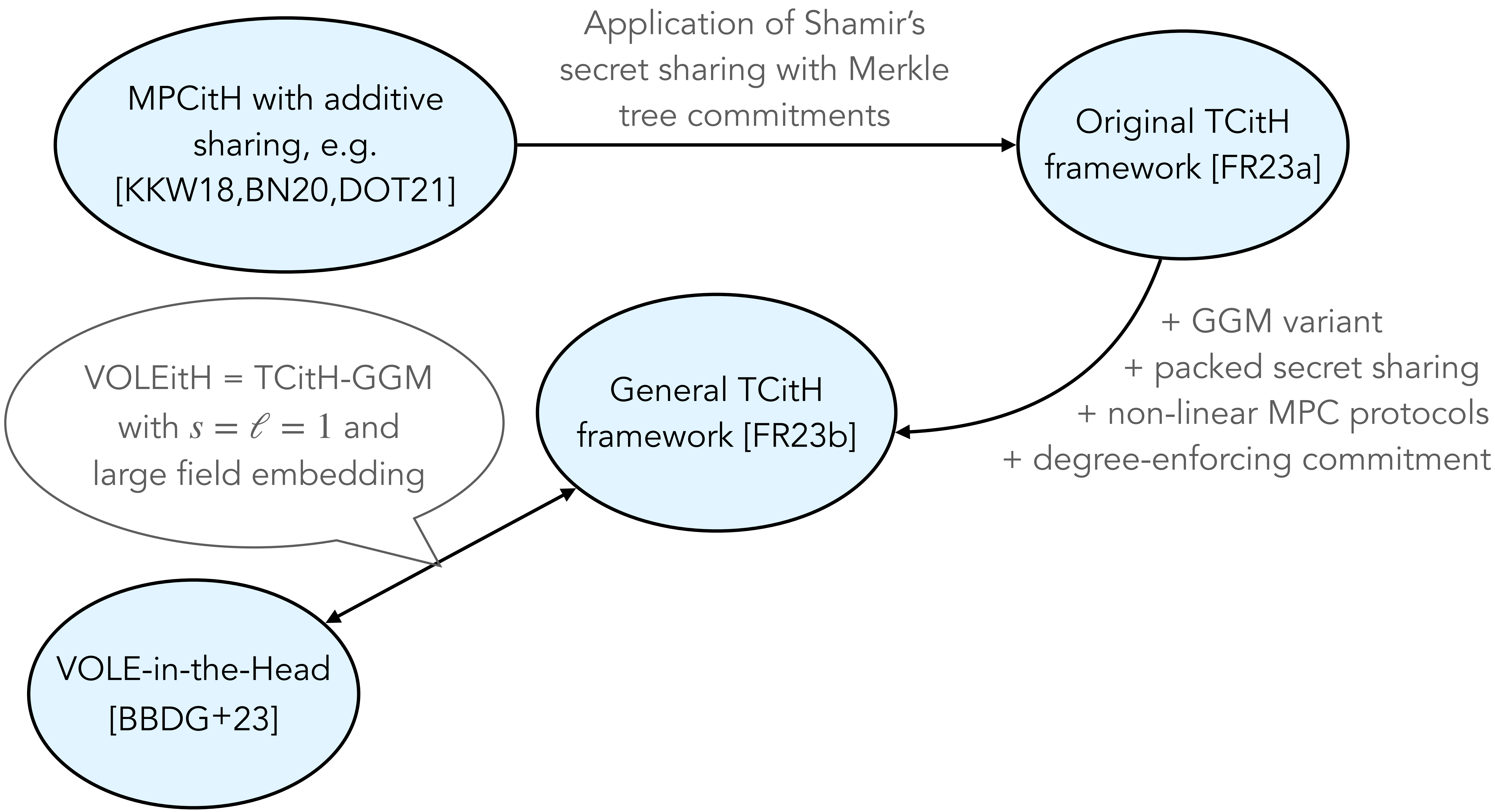
Connection to other proof systems



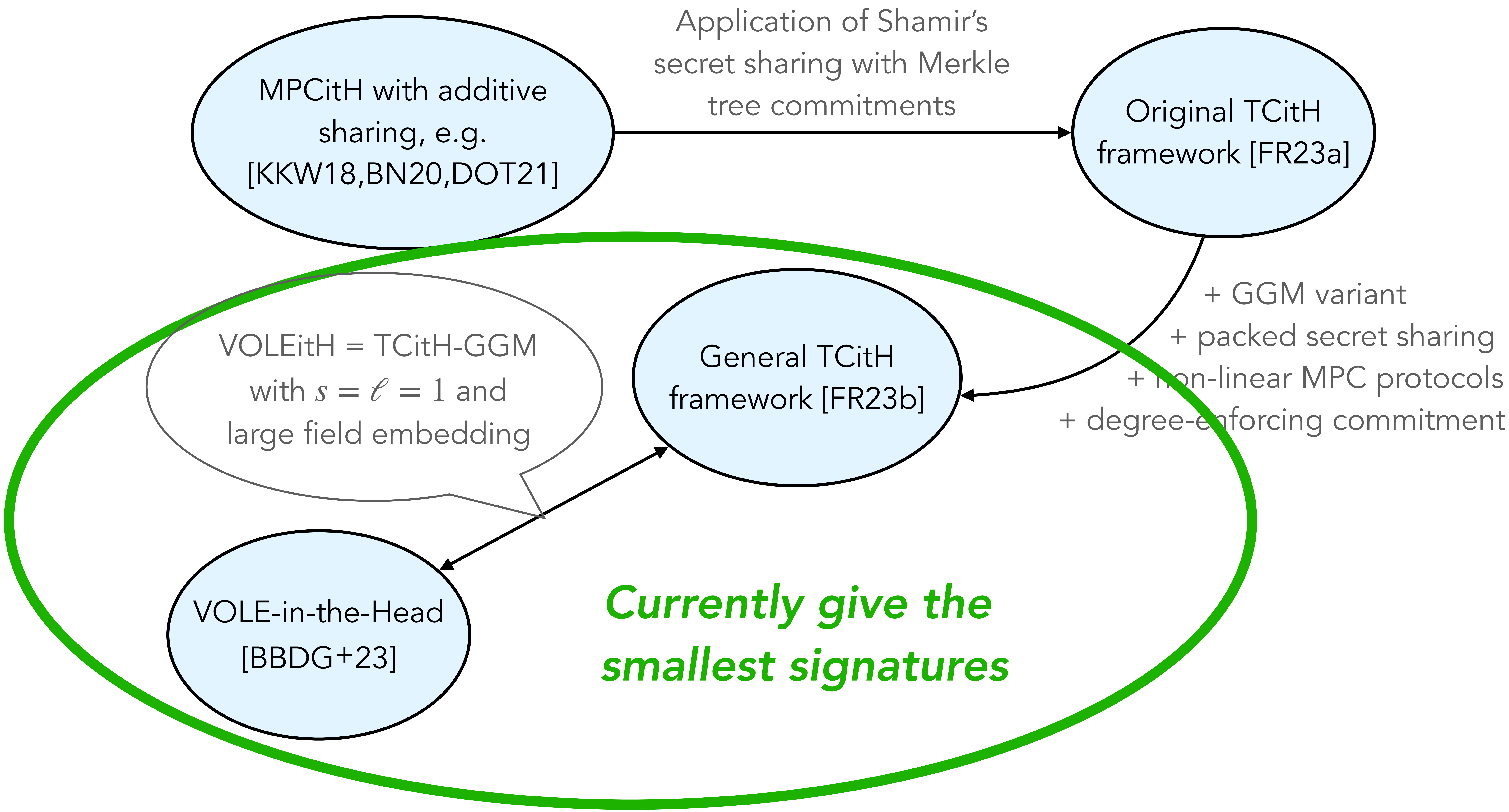
Connection to other proof systems



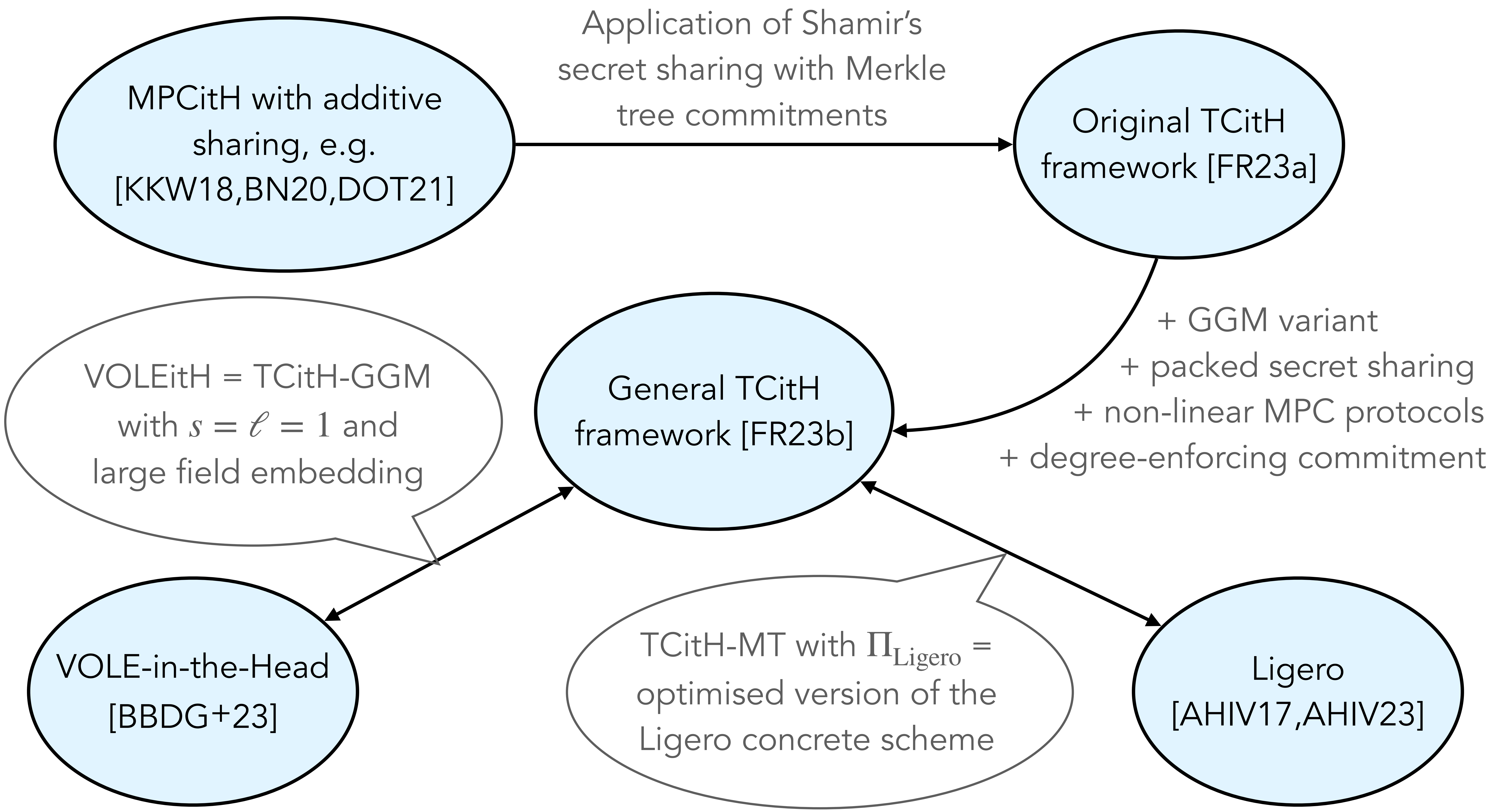
Connection to other proof systems



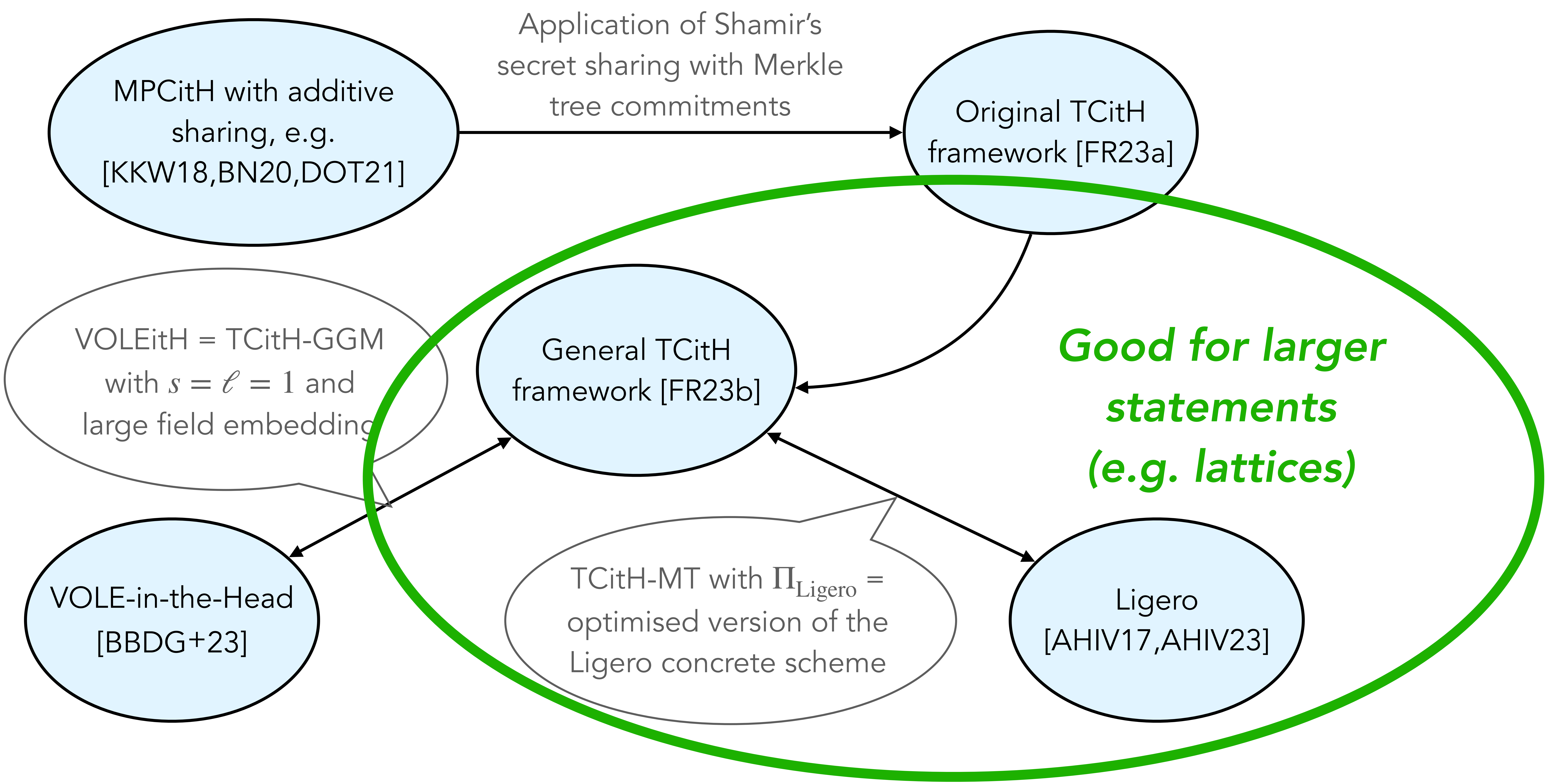
Connection to other proof systems



Connection to other proof systems



Connection to other proof systems



Conclusion

Conclusion

- MPC-in-the-Head
    - **Versatile** approach to build ZK proofs and (PQ) signatures
    - Drastic improvements since 2017
      (in particular thanks to **GGM trees** [KKW18])
    - Applicable to any one-way function
      → **conservative / unstructured** PQ assumptions
    - Instrumental to advanced signatures / ZK proofs
      → e.g. current shortest PQ ring signatures [FR23b]

Conclusion

- MPC-in-the-Head
  - **Versatile** approach to build ZK proofs and (PQ) signatures
  - Drastic improvements since 2017
    (in particular thanks to **GGM trees** [KKW18])
  - Applicable to any one-way function
    → **conservative / unstructured** PQ assumptions
  - Instrumental to advanced signatures / ZK proofs
    → e.g. current shortest PQ ring signatures [FR23b]
- State of the art still moving!
  - New frameworks: **VOLEitH** [BBDG+23], **TCitH** [FR23b]
  - Compression of GGM trees [BBMO+24]
  - Improvements for most MPCitH-based NIST submissions

Conclusion

- MPC-in-the-Head
  - **Versatile** approach to build ZK proofs and (PQ) signatures
  - Drastic improvements since 2017
    (in particular thanks to **GGM trees** [KKW18])
  - Applicable to any one-way function
    → **conservative / unstructured** PQ assumptions
  - Instrumental to advanced signatures / ZK proofs
    → e.g. current shortest PQ ring signatures [FR23b]
- State of the art still moving!
  - New frameworks: **VOLEitH** [BBDG+23], **TCitH** [FR23b]
  - Compression of GGM trees [BBMO+24]
  - Improvements for most MPCitH-based NIST submissions

What next?

You find out!