Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head

Thibauld Feneuil[1,2], Matthieu Rivain[1]

Asiacrypt 2023

December 4, 2023 — Guangzhou (China)

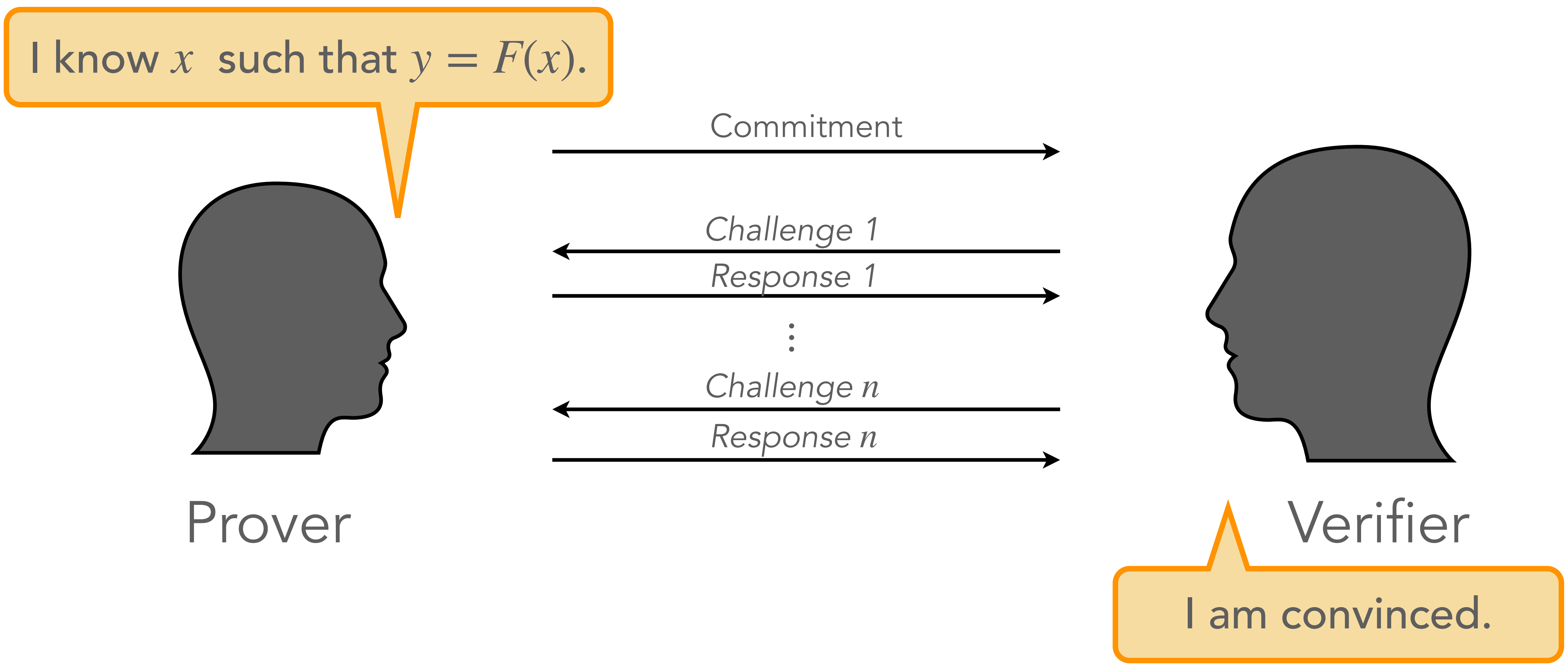
1. CryptoExperts, Paris, France

2. Sorbonne University, CNRS, INRIA, Institut de Mathématiques de Jussieu-Paris Rive Gauche, Ouragan, Paris, France

Table of Contents

MPCitH: general principle

Zero-Knowledge Proof of Knowledge

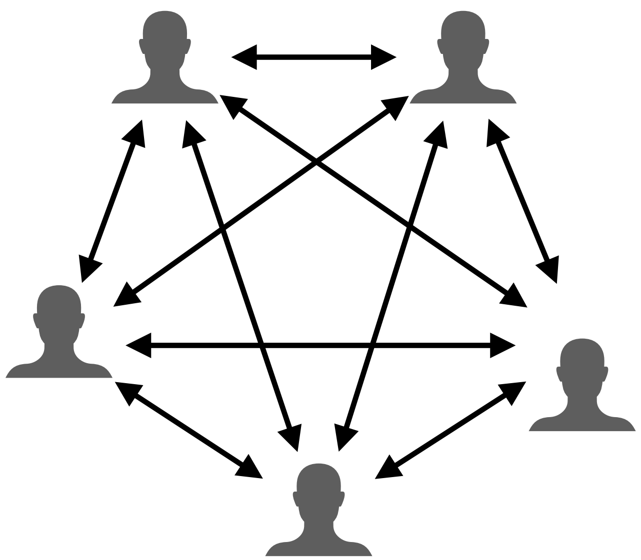


- **Completeness:** Pr[verif ✓ | honest prover] = 1
- **Soundness:** Pr[verif ✓ | malicious prover] $\leq \varepsilon$ (e.g. 2^{-128})
- **Zero-knowledge:** verifier learns nothing on the pre-image x.

MPC in the Head

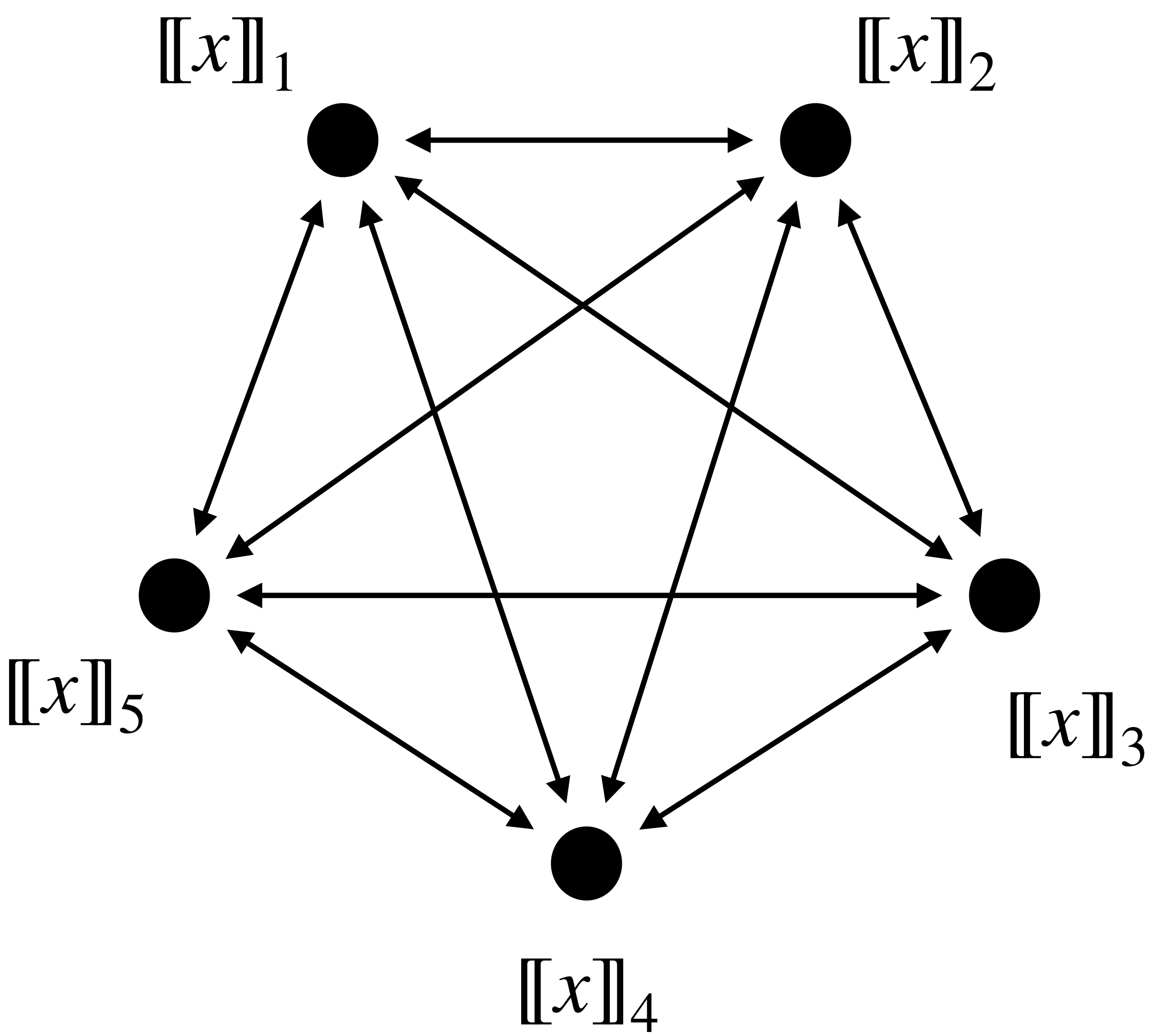
- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)

- Turn a *multiparty computation* (MPC) into a zero-knowledge proof



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

MPC model



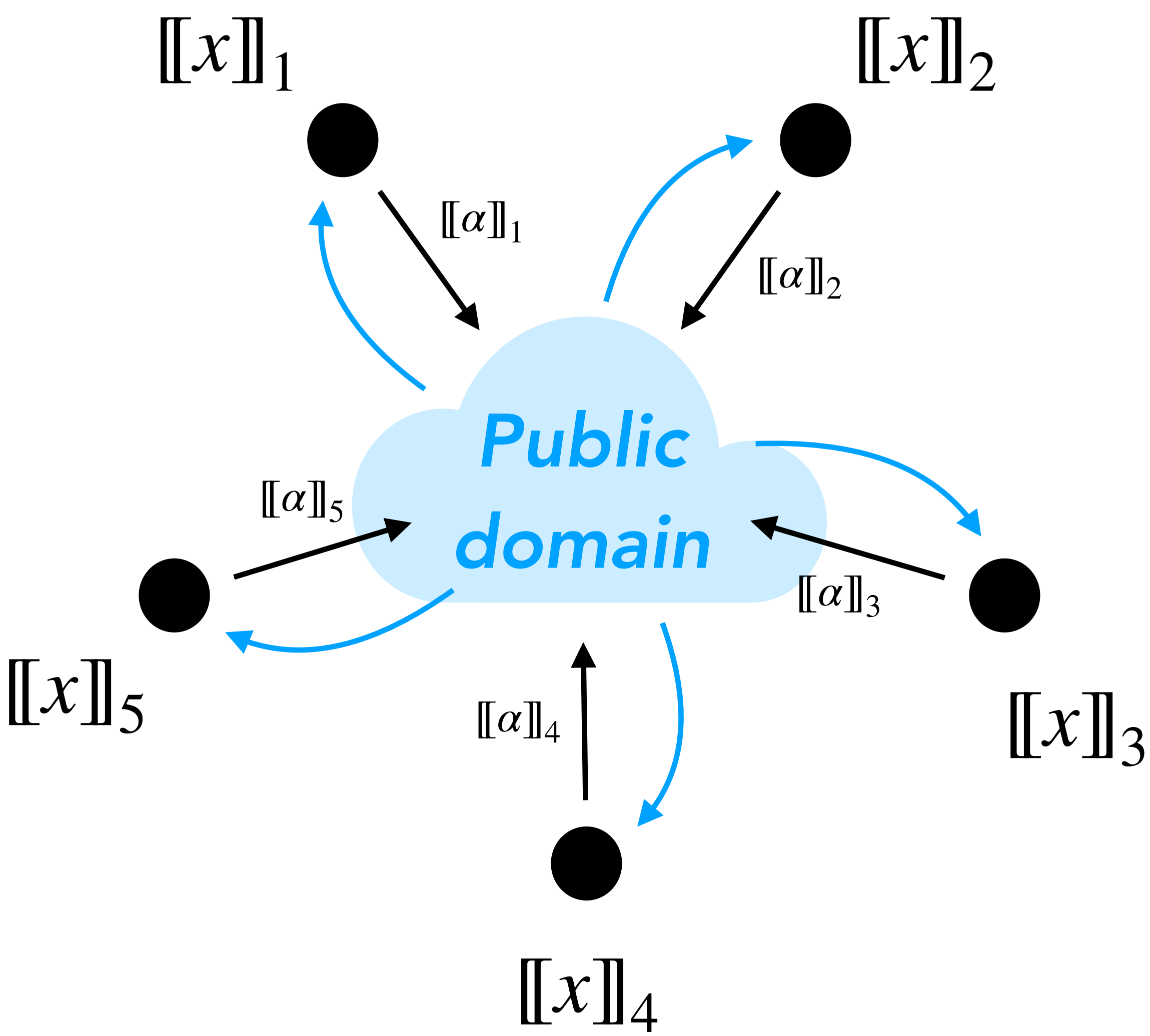
$$x = [\![x]\!]_1 + [\![x]\!]_2 + \ldots + [\![x]\!]_N$$

- **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



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

- **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*)*

MPCitH transform

Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

$\cdots$

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

Prover

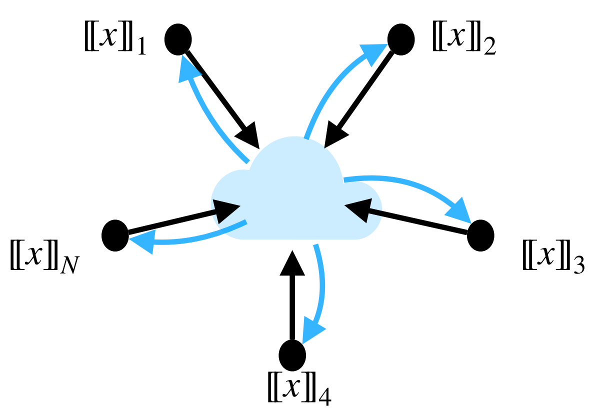
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\cdots$

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

send broadcast

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

Prover

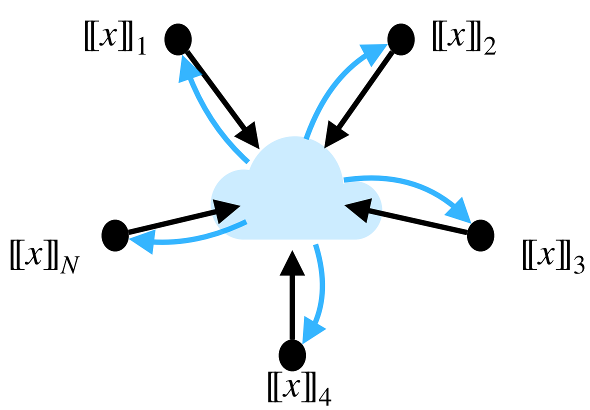
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

...

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

send broadcast

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

i^*

③ Choose a random party

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

Prover

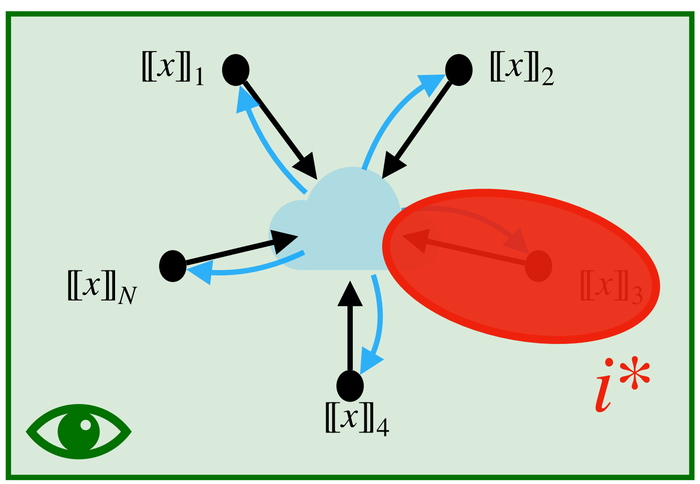
Verifier

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 Choose a random party

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

Prover

Verifier

MPCitH transform

① Generate and commit shares

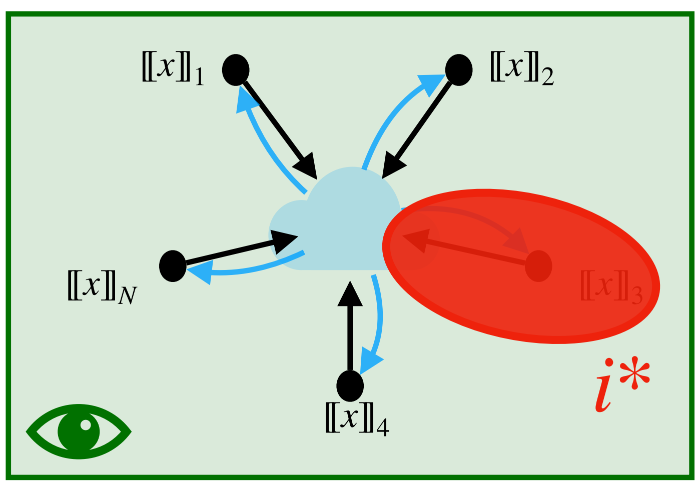
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

$\cdots$

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

② Run MPC in their head



send broadcast

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

③ Choose a random party

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

i^*

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

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

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

① Generate and commit shares

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

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

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

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

$\cdots$

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

Malicious Prover

Verifier

MPCitH transform

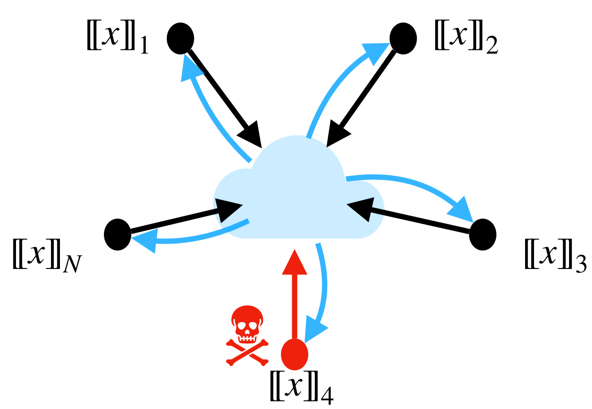
① Generate and commit shares

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

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

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

② Run MPC in their head



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

$\cdots$

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

send broadcast

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

Malicious Prover

Verifier

MPCitH transform

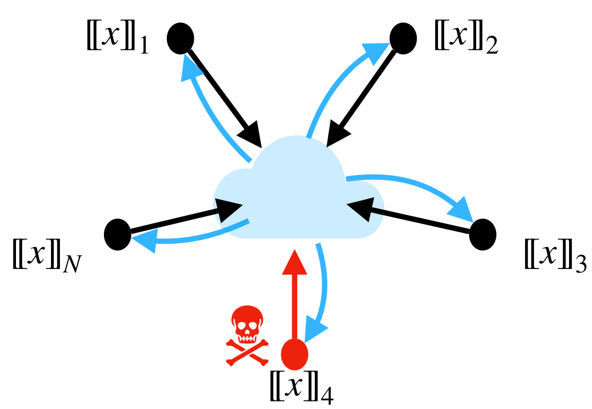
① Generate and commit shares

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

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

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

② Run MPC in their head



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

$\cdots$

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

③ Choose a random party

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

i^*

$\longleftarrow$

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

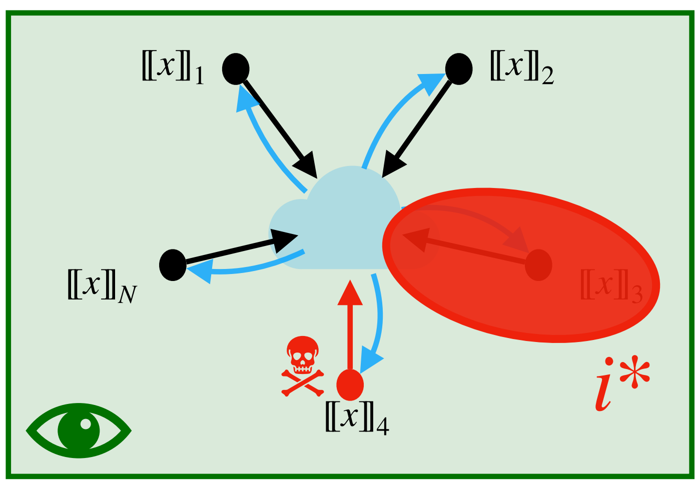
$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

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

...

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

② Run MPC in their head



send broadcast

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

③ Choose a random party

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

i^*

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

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

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

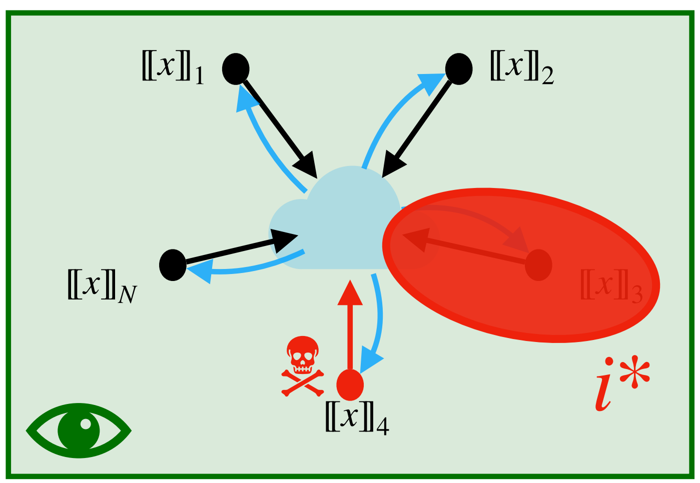
$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

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

$\ldots$

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

② Run MPC in their head



send broadcast

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

③ Choose a random party

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

i^*

⑤ Check $\forall i \neq i^*$

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

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

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

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

Malicious Prover

Verifier

Cheating detected!

MPCitH transform

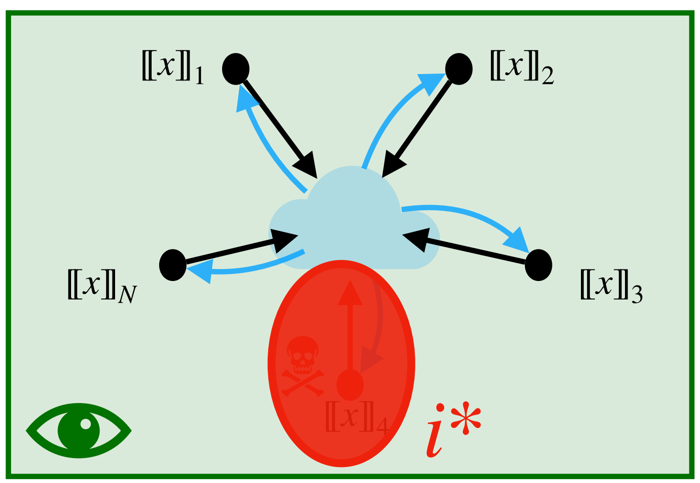
① Generate and commit shares

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

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

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

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 Choose 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 $g(y, \alpha) = \mathrm{Accept}$

Malicious Prover

Verifier

Seems OK.

MPCitH transform

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

MPCitH transform

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

- **Soundness:**

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

MPCitH transform

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

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

- **Parallel repetition**

Protocol repeated τ times in parallel, soundness error $\left(\frac{1}{N}\right)^{\tau}$

MPCitH transform

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

$$\mathbb{P}(\text{malicious prover convinces the verifier}) = \mathbb{P}(\text{corrupted party remains hidden}) = \frac{1}{N}$$

- **Parallel repetition**

Protocol repeated τ times in parallel, soundness error $\left(\frac{1}{N}\right)^{\tau}$

What about the computational cost ?

Computational Cost

① Generate and commit shares

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

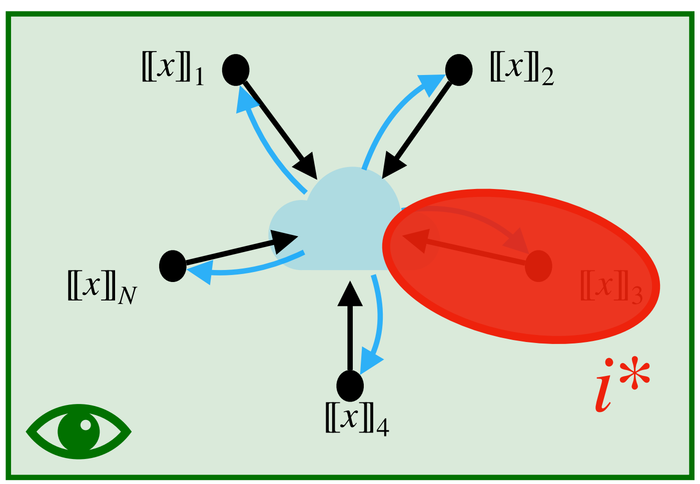
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\ldots$

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

$\longrightarrow$

② Run MPC in their head



send broadcast

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

$\longrightarrow$

③ Choose a random party

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

i^*

$\longleftarrow$

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

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

$\longrightarrow$

⑤ Check $\forall i \neq i^*$

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

Check $g(y, \alpha) = \mathrm{Accept}$

Prover

Verifier

Computational Cost

- **Typical parameters:**

$$N = 256,\ \tau = 17$$

Number of party emulations: $\tau \cdot N = 4352$

Computational Cost

- **Typical parameters**:

$$N = 256,\ \tau = 17$$

Number of party emulations: $\tau \cdot N = 4352$

- **Hypercube Technique**:

Number of party emulations: $\tau \cdot (1 + \log_2 N) = 153$

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

Computational Cost

- **Typical parameters**:

$$N = 256,\ \tau = 17$$

Number of party emulations: $\tau \cdot N = 4352$

- **Hypercube Technique**:

Number of party emulations: $\tau \cdot (1 + \log_2 N) = 153$

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

- **Our Approach**:

Number of party emulations: $\tau \cdot (1 + \ell) = 34$

an additional parameter

The Threshold Approach

In the *threshold* approach, we use a **low-threshold** linear sharing scheme. For example, the Shamir's $(\ell + 1, N)$-secret sharing scheme.

To share a value x,

- sample $r_1, r_2, \ldots, r_\ell$ uniformly at random,
- build the polynomial $P(X) = x + \sum_{k=0}^{\ell} r_k \cdot X^k$,
- Set the share $[\![x]\!]_i \leftarrow P(e_i)$, where e_i is publicly known.

The Threshold Approach

In the *threshold* approach, we use a **low-threshold** linear sharing scheme. For example, the Shamir's $(\ell + 1, N)$-secret sharing scheme.

Properties:

- Linearity: $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$
- Any set of ℓ shares is random and independent of x
- Any set of $\ell + 1$ shares $\rightarrow$ all the shares (and the secret)

The Threshold Approach

In the *threshold* approach, we use a **low-threshold** linear sharing scheme. For example, the Shamir's $(\ell + 1, N)$-secret sharing scheme.

Properties:

- Linearity: $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$
- Any set of ℓ shares is random and independent of x
- Any set of $\ell + 1$ shares $\rightarrow$ all the shares (and the secret)

Zero-Knowledge:

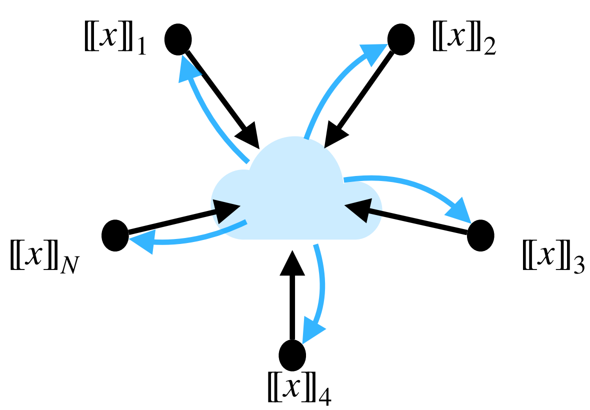
The prover opens only ℓ parties (instead of $N - 1$).

In practice, $\ell \in \{1,2,3\}$

MPCitH Transform with Threshold LSSS

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

② Run MPC in their head



④ Open parties in I

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

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

I
$\longleftarrow$

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

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

⑤ Check $\forall i \in I$

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

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

Prover

Verifier

MPCitH Transform with Threshold LSSS

① Generate and commit shares

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

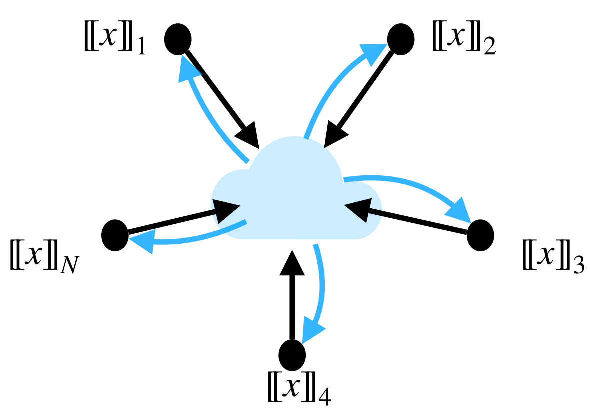
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\ldots$

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

Threshold LSSS ⇒ cannot generate shares from seeds

② Run MPC in their head



send broadcast

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

③ Choose a random set of parties

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

I

④ Open parties in I

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

⑤ Check $\forall i \in I$

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

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

Prover

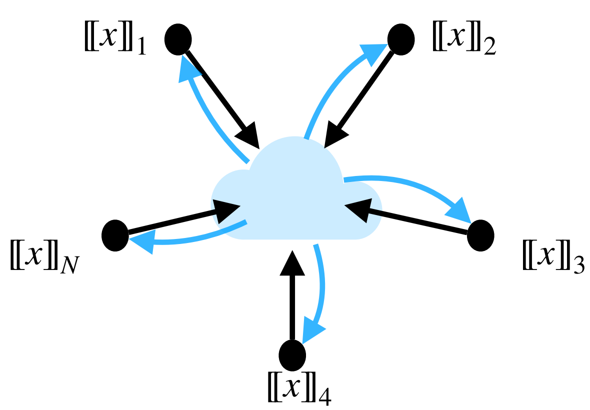
Verifier

MPCitH Transform with Threshold LSSS

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 Choose a random set of parties

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

⑤ Check $\forall i \in I$

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

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

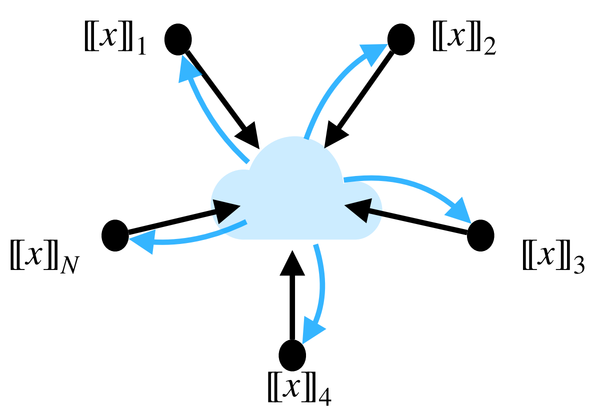
Prover

Verifier

MPCitH Transform with Threshold LSSS

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

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

authentication path

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

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

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

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

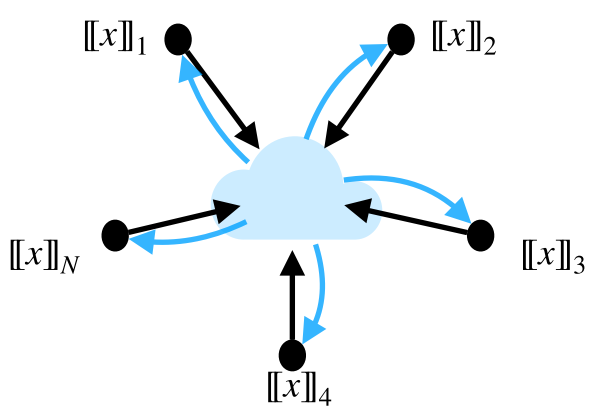
Prover

Verifier

MPCitH Transform with Threshold LSSS

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

$[\![\alpha]\!]$ is redundant
$\Rightarrow \ell + 1$ shares fully determine the sharing
$\Rightarrow$ **only $\ell + 1$ party computations required**

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

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

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

Prover

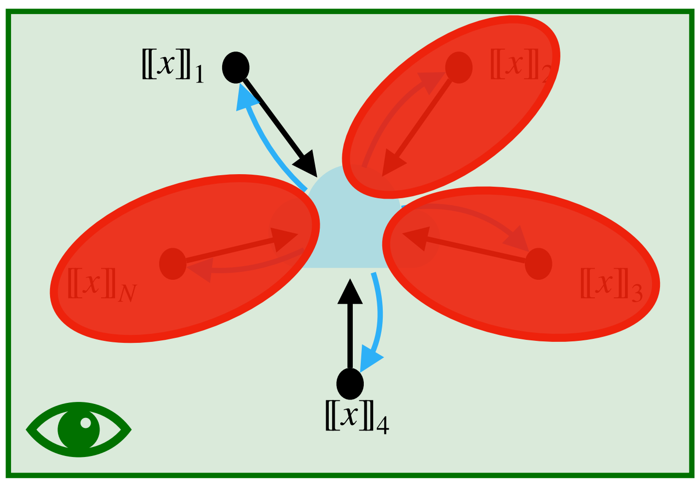
Verifier

MPCitH Transform with Threshold LSSS

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

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

authentication path

$[\![\alpha]\!]$ is redundant

$\Rightarrow \ell + 1$ shares fully determine the sharing

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

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

ℓ parties opened instead of $N - 1$

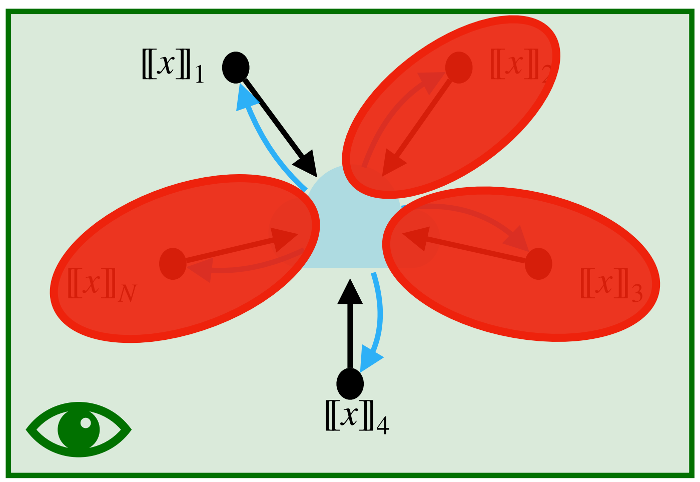
Prover

Verifier

MPCitH Transform with Threshold LSSS

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

② Run MPC in their head



④ Open parties in I

Prover

Merkle root

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

I

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

authentication path

$[\![\alpha]\!]$ is redundant
$\Rightarrow \ell + 1$ shares fully determine the sharing
$\Rightarrow$ **only $\ell + 1$ party computations required**

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

ℓ parties opened instead of $N - 1$

only ℓ party computations required

The Threshold Approach - Soundness

- ***Soundness error (for any ℓ):***

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

⚠️ The term $\binom{N}{\ell}$ should be polynomial in the security level.

- ***Soundness error (for $\ell = 1$):***

$$\frac{1}{N} + p \cdot \frac{(N-1)}{2}$$

instead of $\frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right)$.

The Threshold Approach

	Additive sharing + hypercube technique	**Threshold LSSS** with $\ell = 1$
Soundness error	$\frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right)$	$\frac{1}{N} + p \cdot \frac{(N-1)}{2}$
Prover # party computations	$1 + \log_2 N$	2
Verifier # party computations	$\log_2 N$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

The Threshold Approach

	Additive sharing + hypercube technique	**Threshold LSSS** with $\ell = 1$
Soundness error	$\frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right)$	$\frac{1}{N} + p \cdot \frac{(N-1)}{2}$
Prover # party computations	$1 + \log_2 N$	2
Verifier # party computations	$\log_2 N$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

Much cheaper emulation

The Threshold Approach

	Additive sharing + hypercube technique	**Threshold LSSS** with $\ell = 1$
Soundness error	$\frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right)$	$\frac{1}{N} + p \cdot \frac{(N-1)}{2}$
Prover # party computations	$1 + \log_2 N$	2
Verifier # party computations	$\log_2 N$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	$\boxed{\text{Merkle tree } 2\lambda \cdot \log N}$

Fast verification algorithm

The Threshold Approach

	Additive sharing + hypercube technique	**Threshold LSSS** with $\ell = 1$
Soundness error	$\frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right)$	$\frac{1}{N} + p \cdot \frac{(N-1)}{2}$
Prover # party computations	$1 + \log_2 N$	2
Verifier # party computations	$\log_2 N$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

Larger proof transcripts

The Threshold Approach

Require $N \leq |\mathbb{F}|$

	Additive sharing + hypercube technique	**Threshold LSSS** with $\ell = 1$
Soundness error	$\frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right)$	$\frac{1}{N} + p \cdot \frac{(N-1)}{2}$
Prover # party computations	$1 + \log_2 N$	2
Verifier # party computations	$\log_2 N$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

Applications

Applications

- New trade-offs for MPCitH-based zero-knowledge proof systems
  - Larger proof sizes, faster algorithms, fast verification

Applications

- <u>New trade-offs</u> for post-quantum MPCitH-based signature schemes
  - Larger signature sizes, faster algorithms, fast verification

Additive sharing
(with hypercube optimisation)

	Size	Signing time	Verification time
`SDitH-gf256-L1`	8 260 B	5.18 ms	4.81 ms
`SDitH-gf251-L1`		8.51 ms	8.16 ms
`SDitH-gf256-L1`	10 424 B	1.97 ms	0.62 ms
`SDitH-gf251-L1`		1.71 ms	0.23 ms

Threshold LSSS

Benchmark of the SDitH submission package of the NIST call

Applications

- A new batching strategy for MPCitH-based proof system
  - By packing several witness in the Shamir's secret sharing
  - Compatible with several former MPCitH-based proof arguments (as Limbo)

	#gates = 2^8	**#gates = 2^{16}**
Non batched	6 KB	390 KB
Batch 100 proofs	0.6 KB / proof	28 KB / proof
Batch 10000 proofs	0.6 KB / proof	27 KB / proof

Batched proofs for circuits over GF(256) using Limbo

Conclusion

Conclusion

- Replacing additive sharings with threshold sharings provides new trade-offs that lowers the cost of emulating the multiparty computation.
- The threshold approach enables us to have fast verification algorithms.
- That also offers an efficient batching strategy for some MPCitH-based proof systems.

Conclusion

- Replacing additive sharings with threshold sharings provides new trade-offs that lowers the cost of emulating the multiparty computation.

- The threshold approach enables us to have fast verification algorithms.

- That also offers an efficient batching strategy for some MPCitH-based proof systems.

- The threshold approach has been recently improved in a new work:

  [FR23] Feneuil, Rivain. *Threshold Computation in the Head: Improved Framework for Post-Quantum Signatures and Zero-Knowledge Arguments.* ePrint 2023/1573.

Conclusion

- Replacing additive sharings with threshold sharings provides new trade-offs that lowers the cost of emulating the multiparty computation.

- The threshold approach enables us to have fast verification algorithms.

- That also offers an efficient batching strategy for some MPCitH-based proof systems.

- The threshold approach has been recently improved in a new work:

  [FR23] Feneuil, Rivain. *Threshold Computation in the Head: Improved Framework for Post-Quantum Signatures and Zero-Knowledge Arguments.* ePrint 2023/1573.

Thank you for your attention !

thibauld.feneuil@cryptoexperts.com
matthieu.rivain@cryptoexperts.com