Rank Metric in the Head

RYDE

*N. Aragon, M. Bardet, L. Bidoux,
J.-J. Chi-Domínguez, V. Dyseryn,
T. Feneuil, P. Gaborit, A. Joux,
M. Rivain, J.-P. Tillich, A. Vincotte*

MIRA

*N. Aragon, M. Bardet, L. Bidoux,
J.-J. Chi-Domínguez, V. Dyseryn,
T. Feneuil, P. Gaborit, R. Neveu,
M. Rivain, J.-P. Tillich*

2nd Oxford Post-Quantum Cryptography Summit

September 4, 2023, University of Oxford (UK)

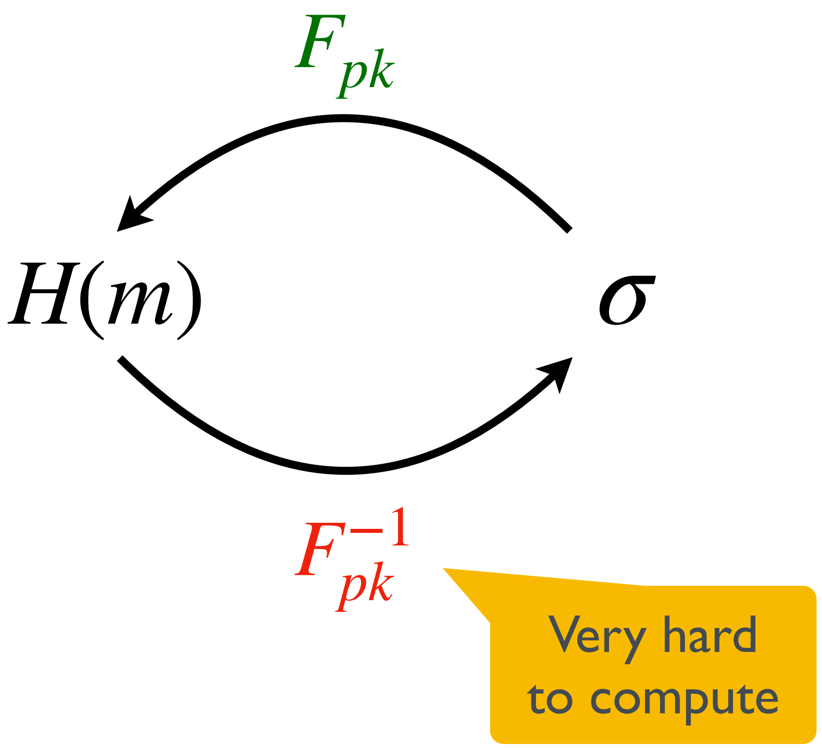
Table of Contents

- Introduction
- MPC-in-the-Head: general principle
- Optimisations + Hypercube technique
- From MPC-in-the-Head to signatures
- Performances

Introduction

How to build signature schemes?

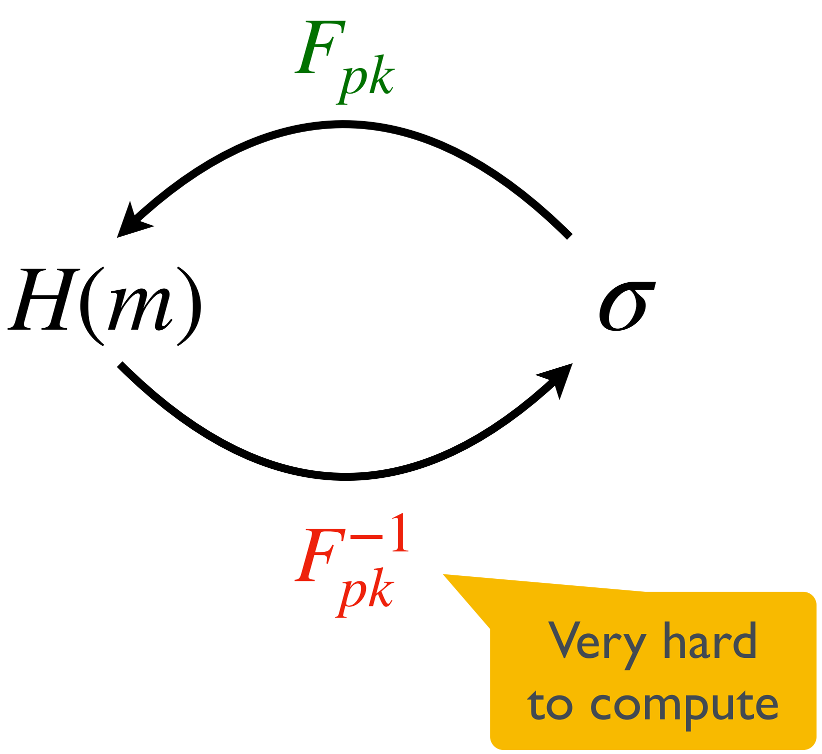
Hash & Sign



- Short signatures
- "Trapdoor" in the public key

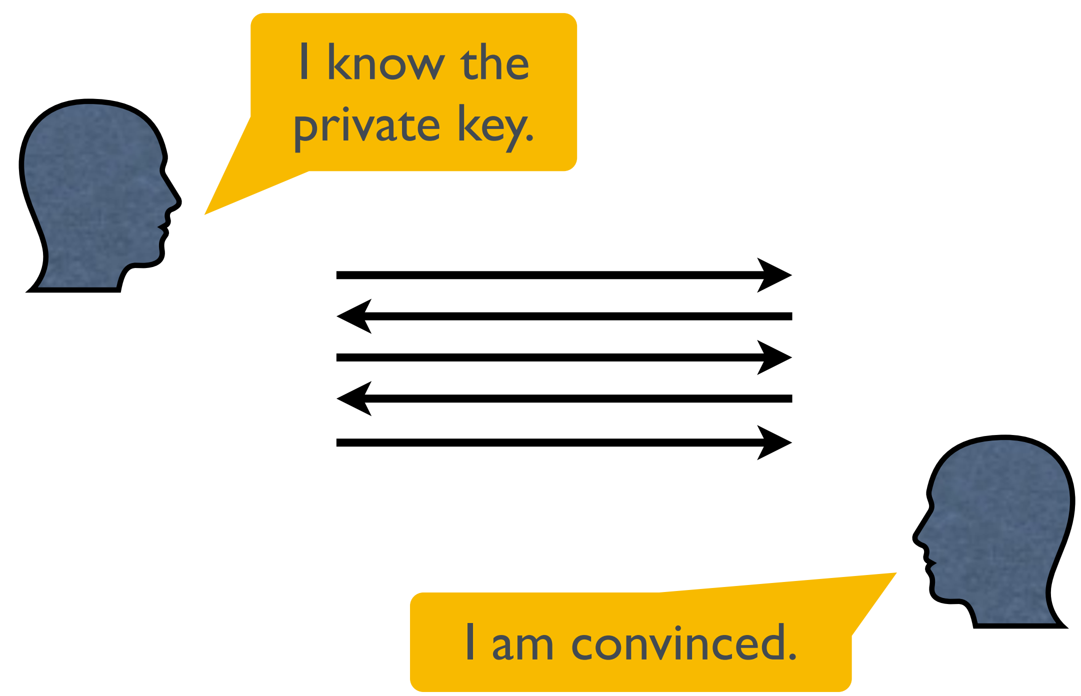
How to build signature schemes?

Hash & Sign



- Short signatures
- “Trapdoor” in the public key

From a zero-knowledge proof



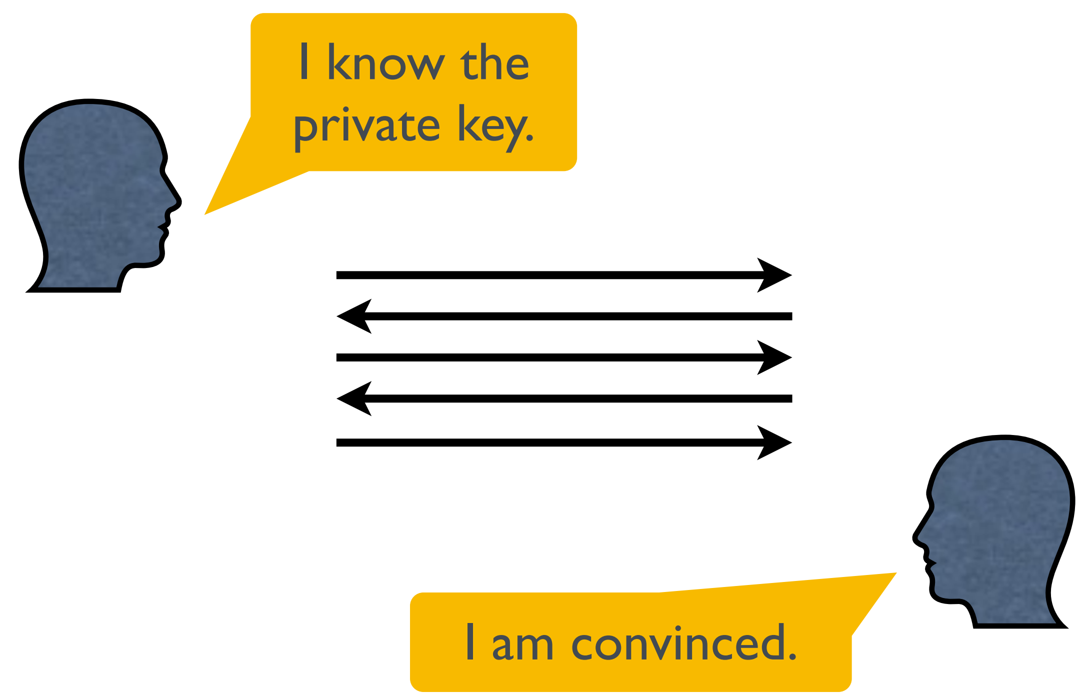
- Large(r) signatures
- Short public key

How to build signature schemes?

Hash & Sign

- Short signatures
- "Trapdoor" in the public key

From a zero-knowledge proof



- Large(r) signatures
- Short public key

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 x

MPC in the Head

- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)
- Turn an MPC protocol into a zero knowledge proof of knowledge
- **Generic**: can be applied to any cryptographic problem

MPC in the Head

- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)
- Turn an MPC protocol into a zero knowledge proof of knowledge
- **Generic**: can be applied to any cryptographic problem

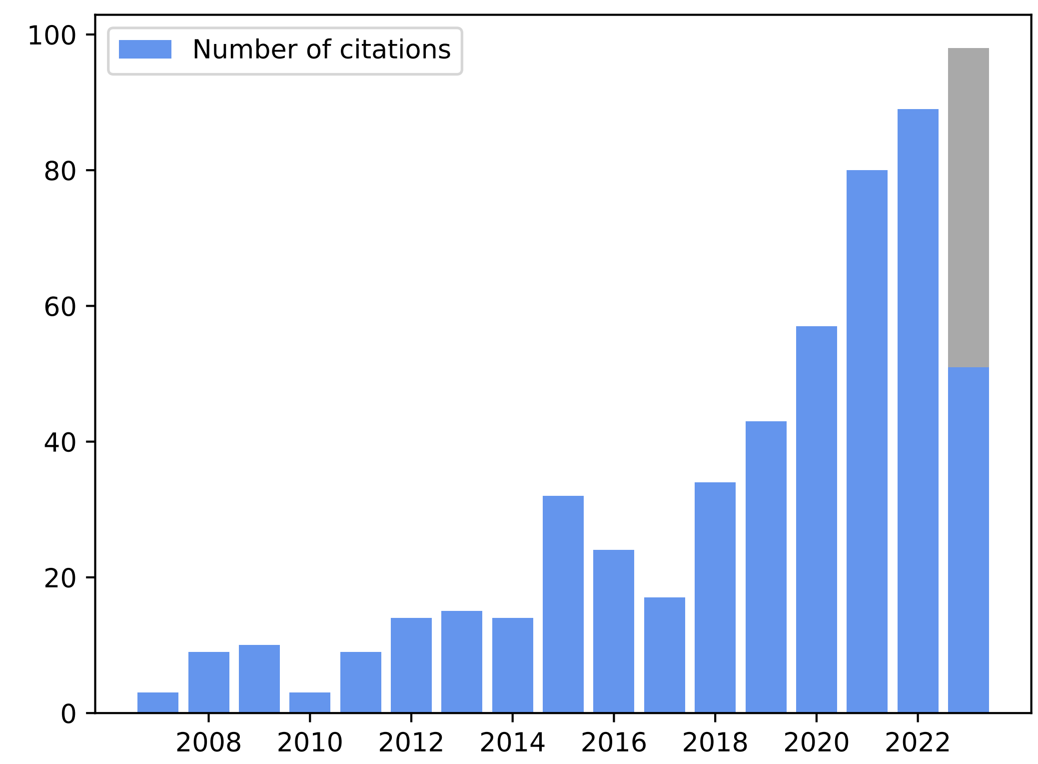


Figure: Number of citations to [IKOS07] by year

Source: Google Scholar

MPC in the Head

- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)
- Turn an MPC protocol into a zero knowledge proof of knowledge
- **Generic**: can be applied to any cryptographic problem

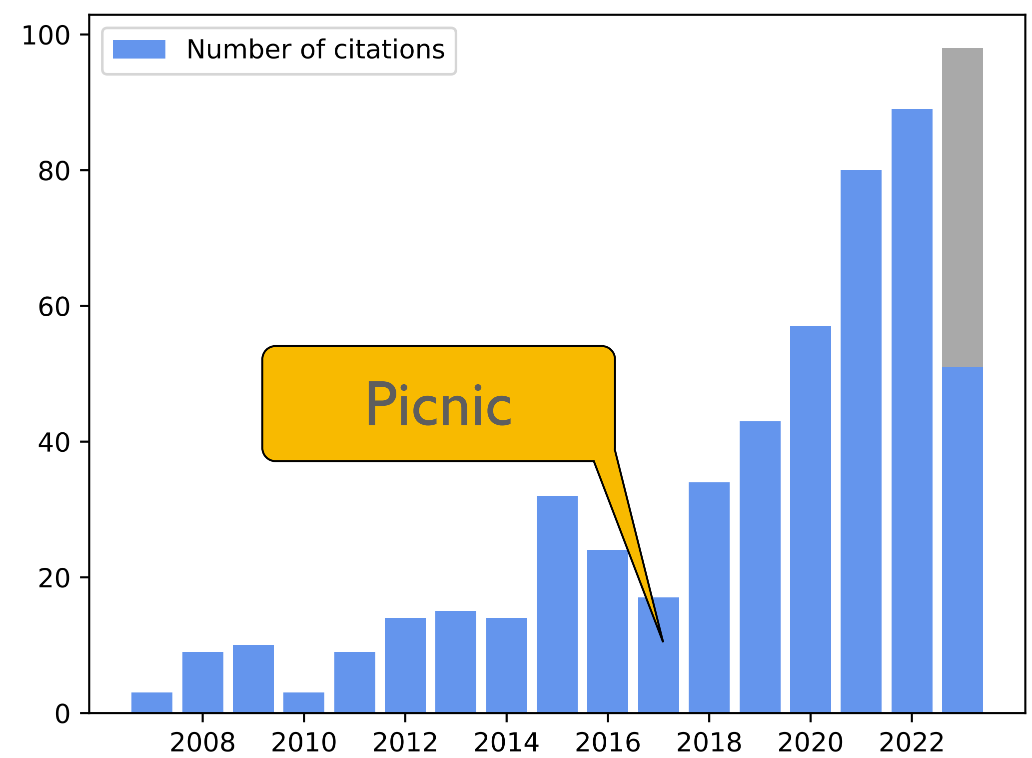


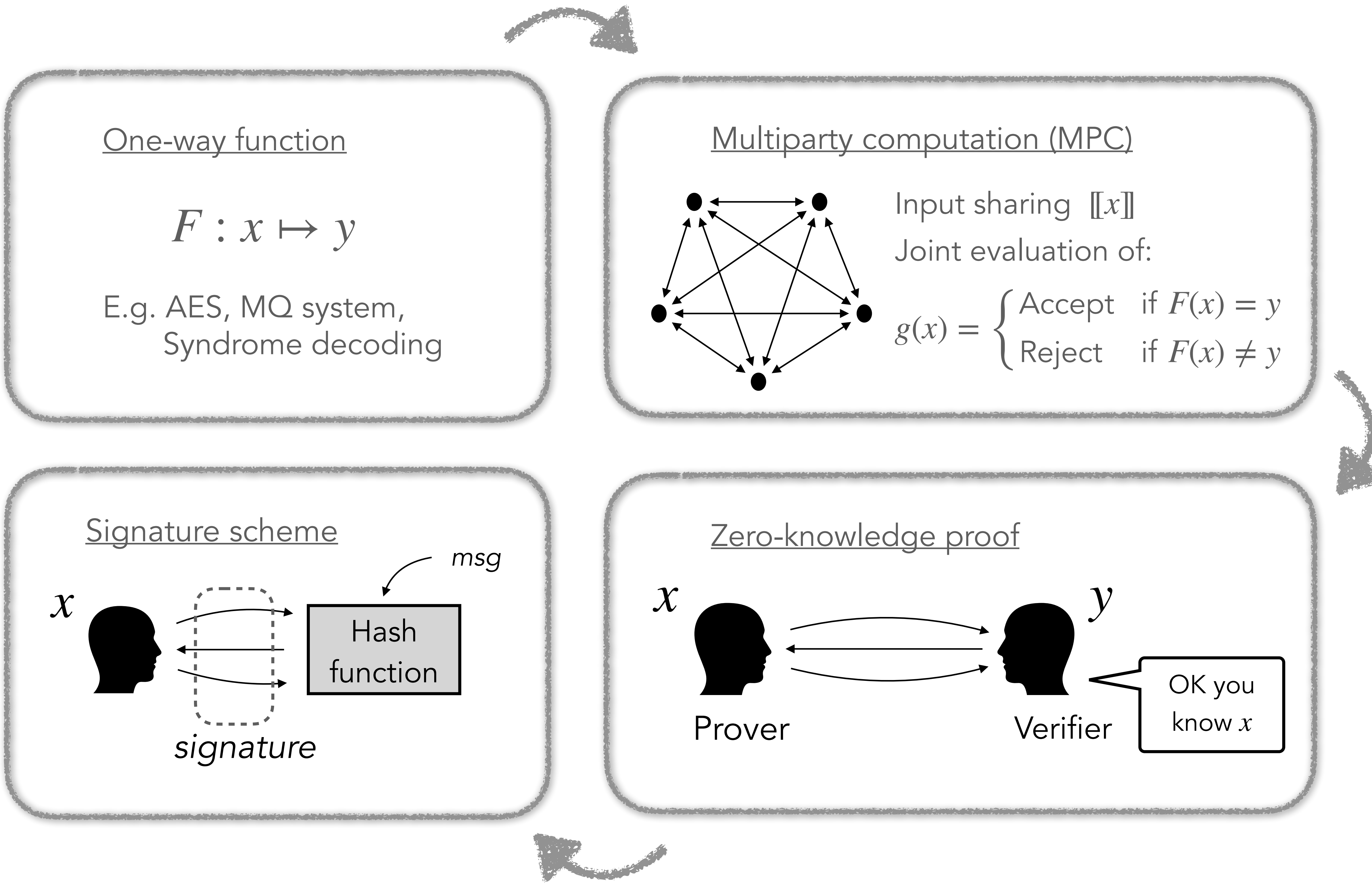
Figure: Number of citations to [IKOS07] by year

Source: Google Scholar

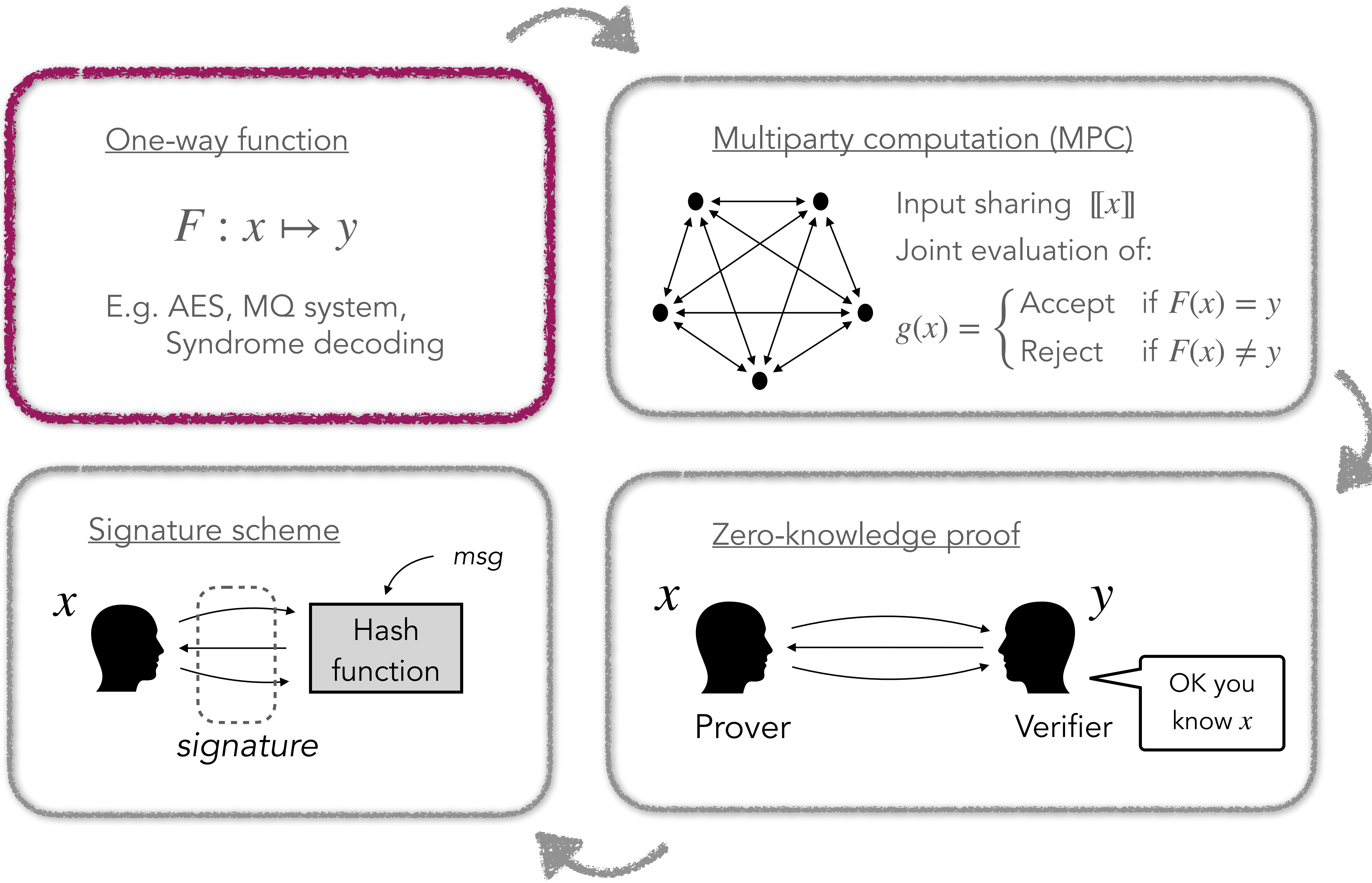
MPC in the Head

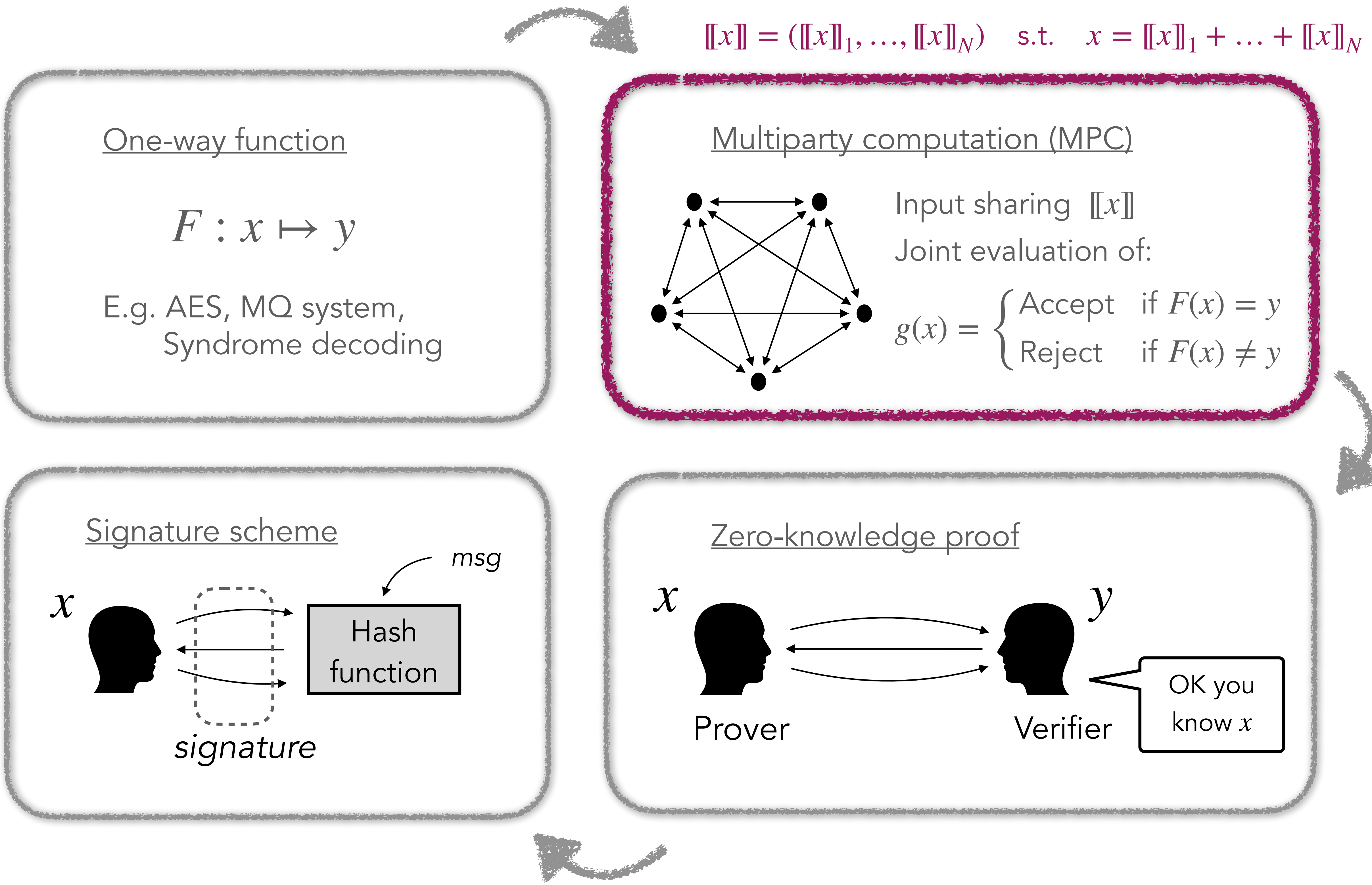
- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)
- Turn an MPC protocol into a zero knowledge proof of knowledge
- **Generic**: can be applied to any cryptographic problem
- Convenient to build (candidate) **post-quantum signature** schemes
- **Picnic**: submission to NIST (2017)
- First round of recent NIST call: 8 MPCitH schemes / 40 submissions

AIMer	*MQOM*
Biscuit	*PERK*
MIRA	*RYDE*
MiRitH	*SDitH*

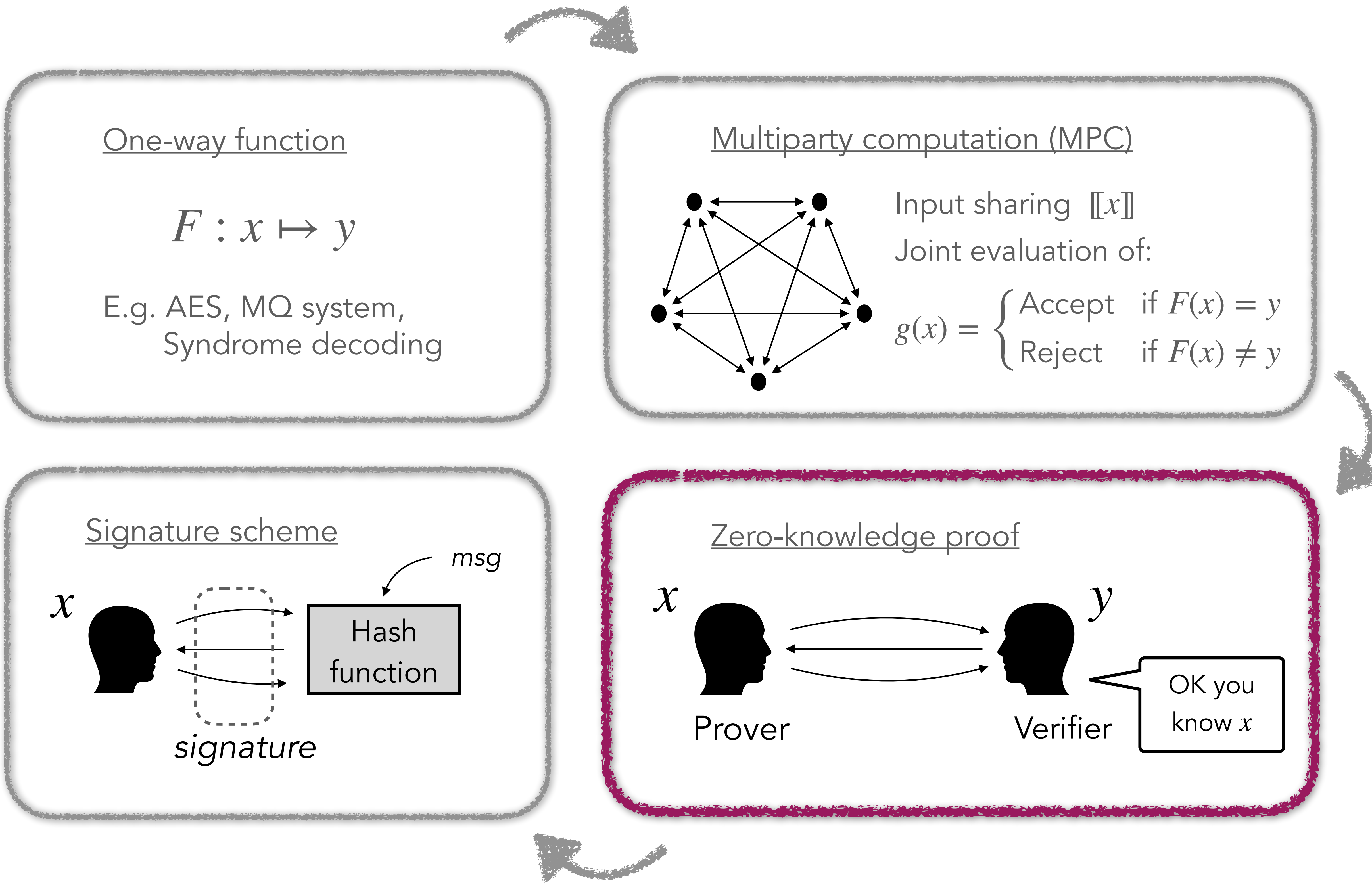


One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Signature scheme
msg
x
Hash function
signature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

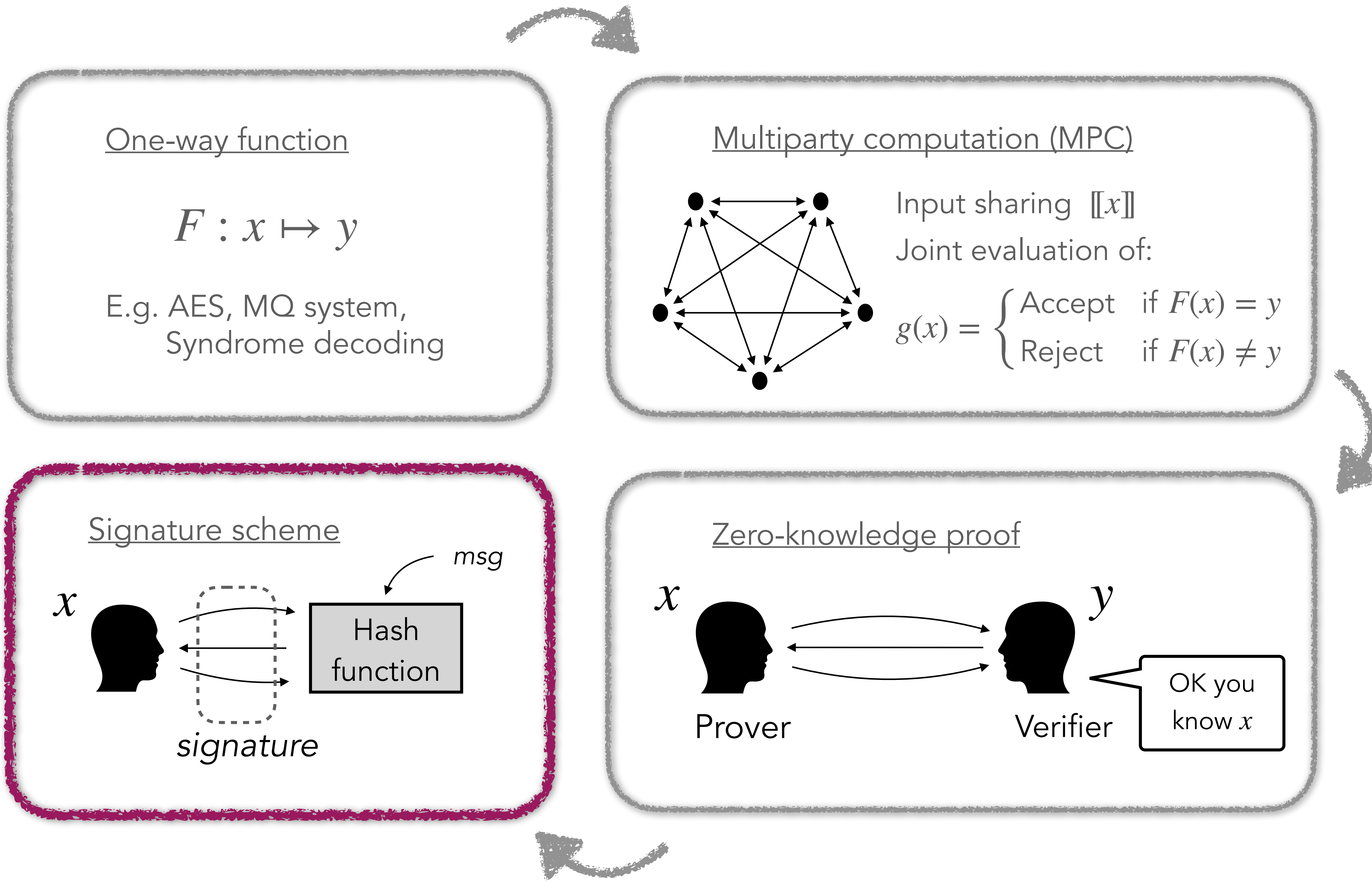

One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Signature scheme
x
msg
Hash function
signature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



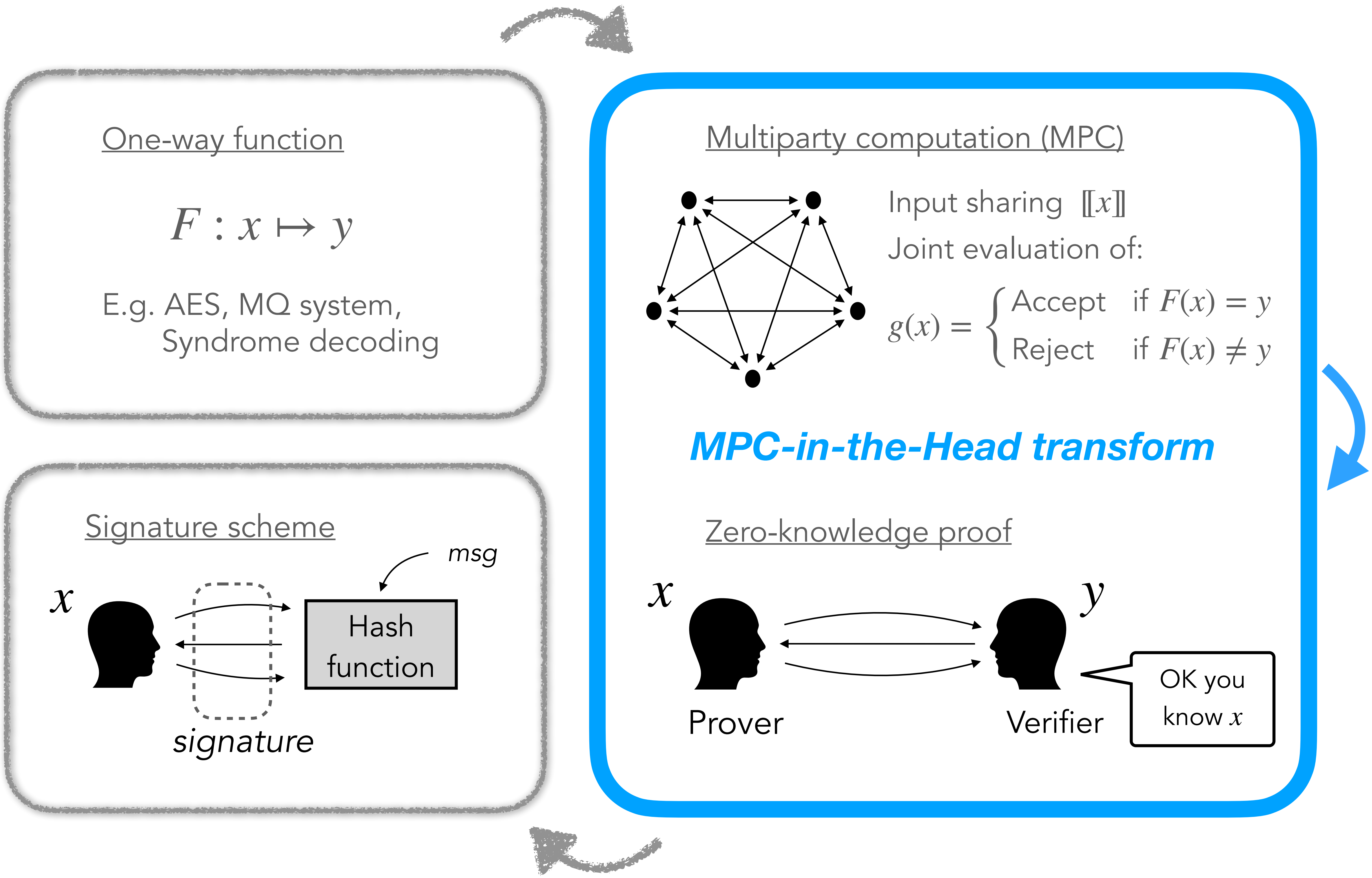
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ s.t. $x = [\![x]\!]_1 + \ldots + [\![x]\!]_N$
One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Signature scheme
x
msg
Hash function
signature
Zero-knowledge proof
x
y
Prover
Verifier
OK you know x



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

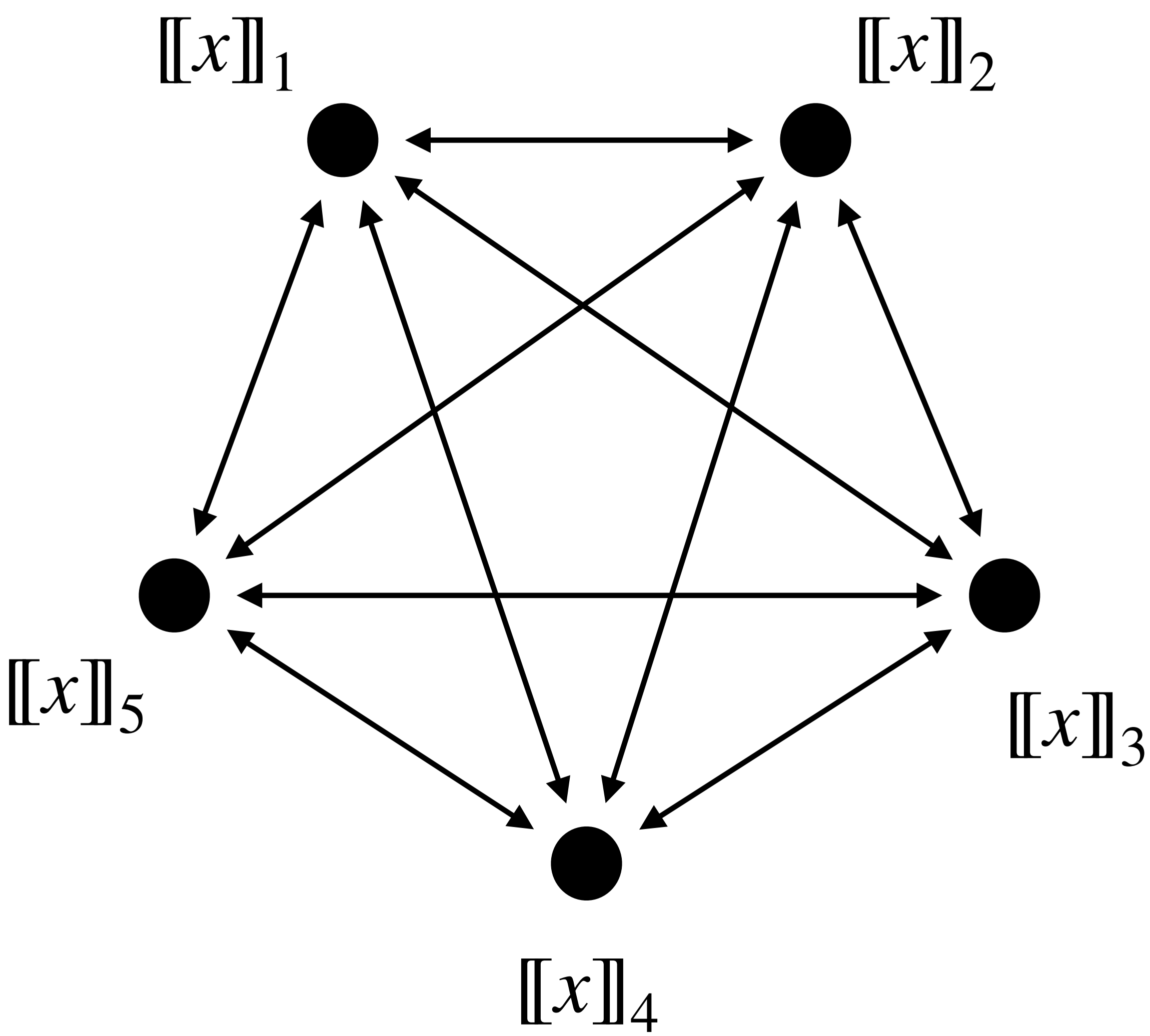


One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Zero-knowledge proof
x
y
Prover
Verifier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

MPCitH: general principle

MPC model



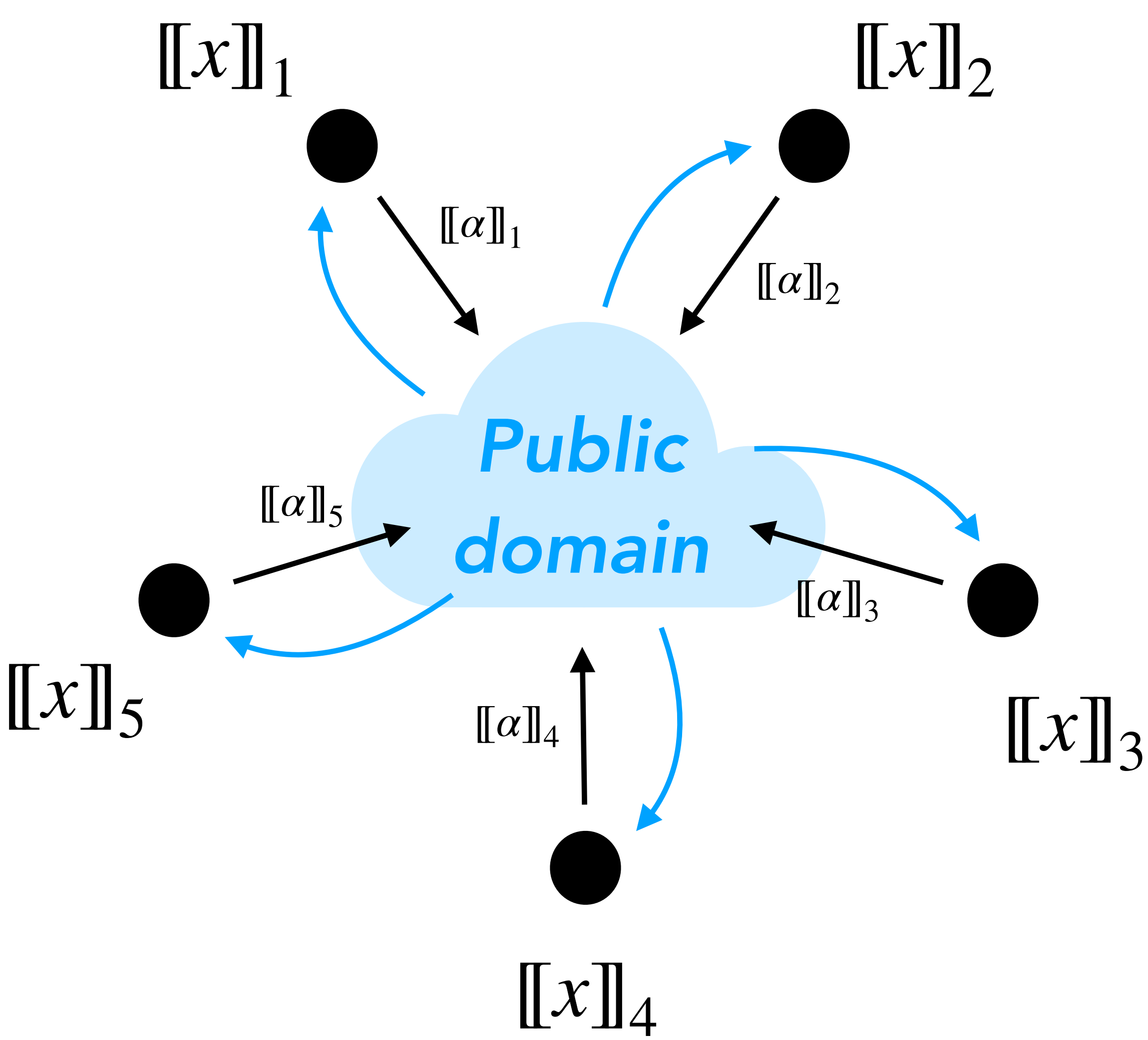
$$x = [\![x]\!]_1 + [\![x]\!]_2 + \ldots + [\![x]\!]_N$$

- **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

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

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

$\cdots$

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

Prover

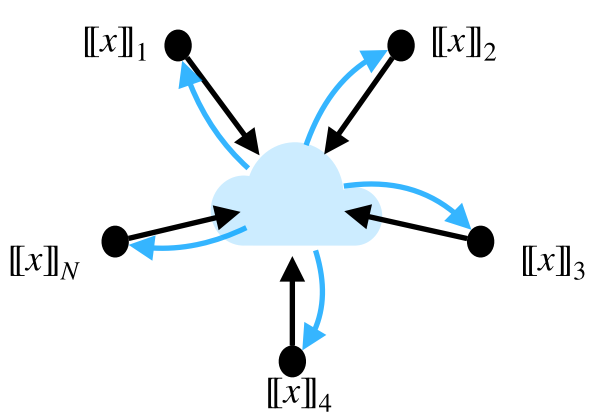
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

...

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

send broadcast

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

Prover

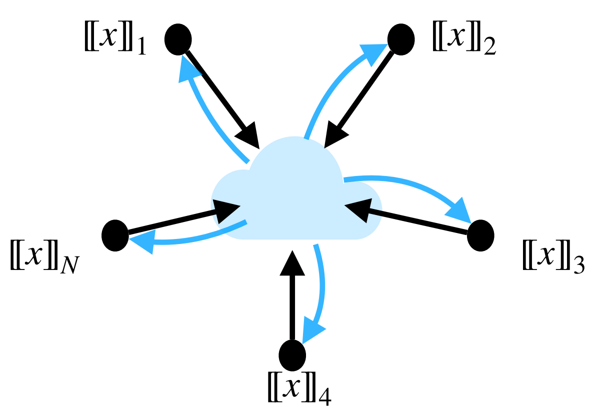
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

...

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

send broadcast

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

i^*

③ Choose a random party

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

Prover

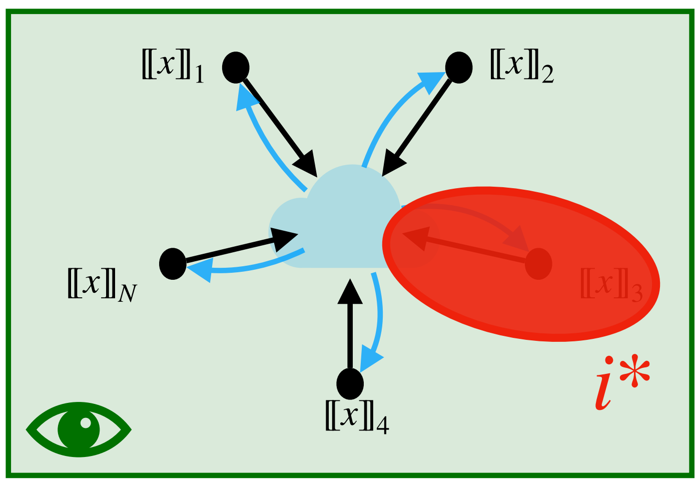
Verifier

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

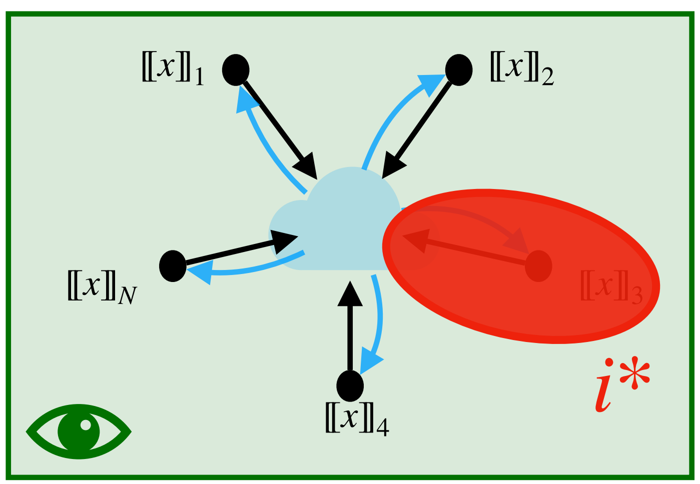
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

…

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

send broadcast

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

← i^*

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

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

Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

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

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

$\ldots$

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

→

Malicious Prover

Verifier

MPCitH transform

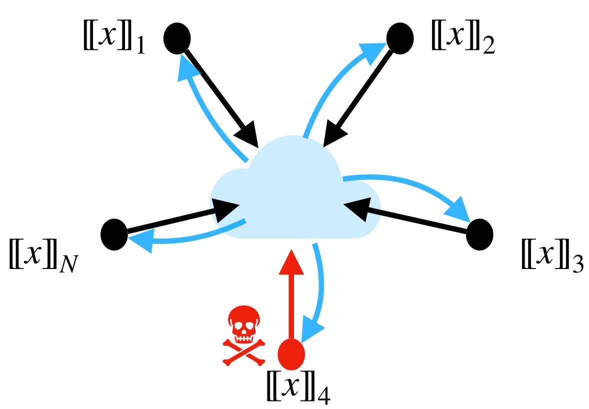
① Generate and commit shares

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

We have $F(x) \neq y$ 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

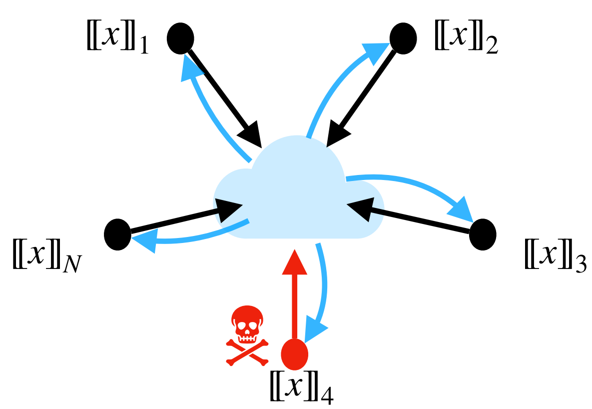
① Generate and commit shares

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

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

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

② Run MPC in their head



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

$\ldots$

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

send broadcast

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

③ Choose a random party

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

i^*

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

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

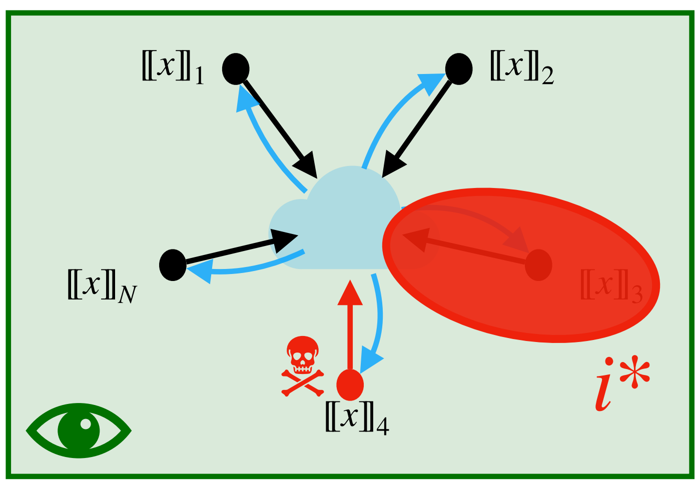
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

...

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

→

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

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

→

Malicious Prover

Verifier

MPCitH transform

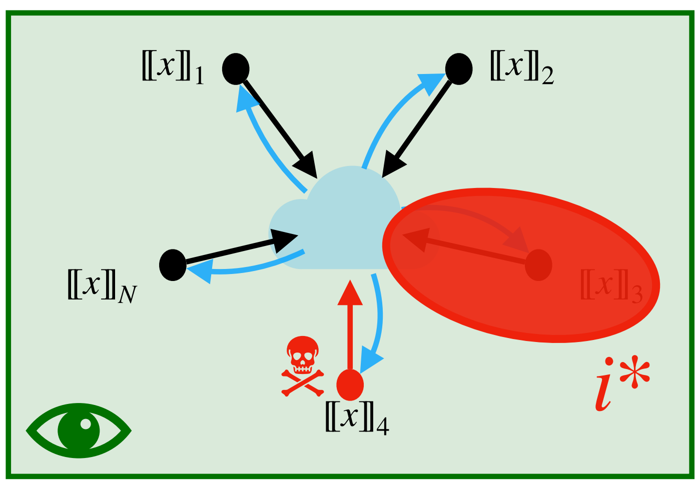
① Generate and commit shares

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

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

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

② Run MPC in their head



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

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

$\ldots$

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

send broadcast

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

← i^*

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

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

Malicious Prover

Verifier

Cheating detected!

MPCitH transform

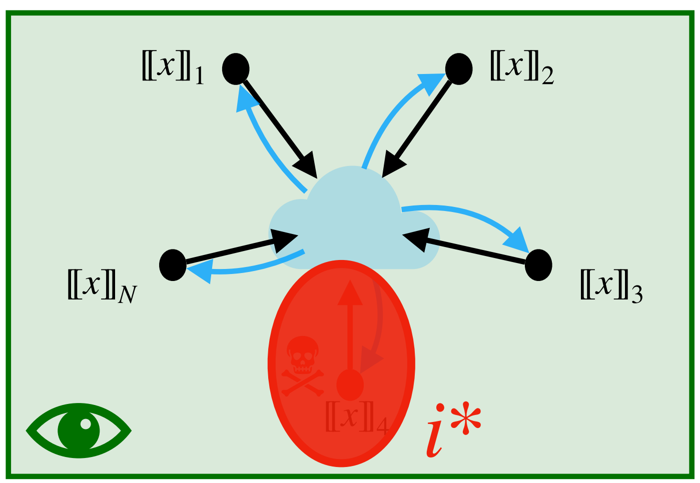
① Generate and commit shares

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

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 $\tilde{g}(y, \alpha) =$ 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:**

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

- **Parallel repetition**

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

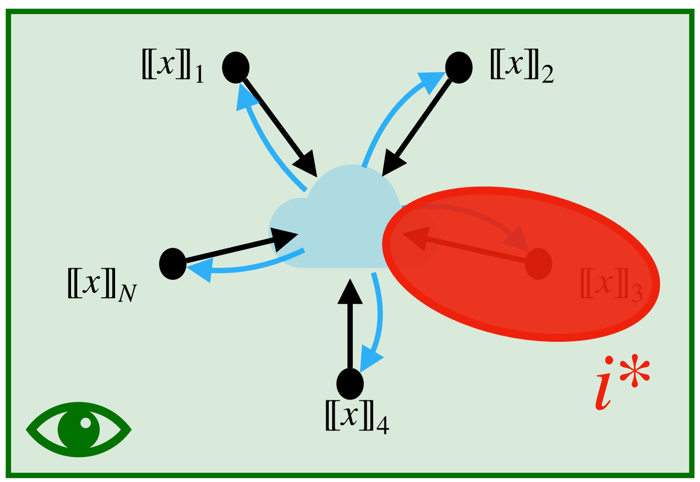
Optimisations

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

...

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

→

send broadcast

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

→

③ Choose a random party

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

i^*

←

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

→

⑤ Check $\forall i \neq i^*$

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

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

<u>Prover</u> <u>Verifier</u>

Naive MPCitH transformation

Size of the broadcast (per party)

Size of a commitment digest

Size of the MPC input (per party)

$$\text{Size} \approx \tau \cdot \left(N \cdot 2\lambda + N \cdot |\alpha| + (N-1) \cdot |x| \right)$$

Number of repetitions to achieve the desired security level

$$\tau \approx \frac{\lambda}{\log_2 N}$$

Naive MPCitH transformation

Size of the broadcast (per party)

Size of a commitment digest

Size of the MPC input (per party)

$$\text{Size} \approx \tau \cdot \left(N \cdot 2\lambda + N \cdot |\alpha| + (N-1) \cdot |x|\right)$$

Number of repetitions to achieve the desired security level

$$\tau \approx \frac{\lambda}{\log_2 N}$$

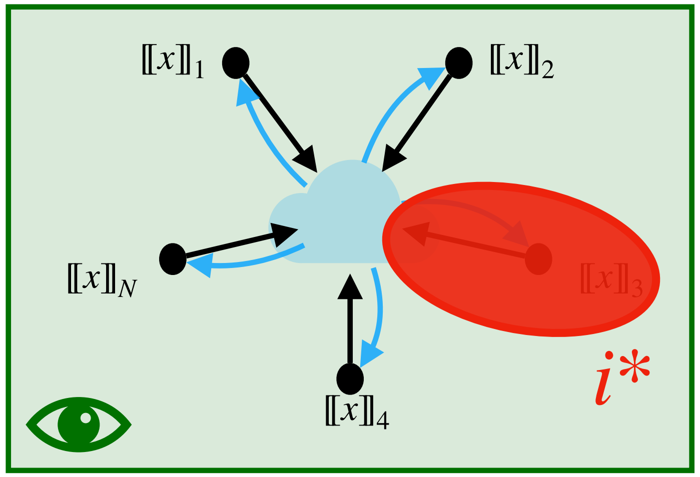
The signature sizes would be of at least 30 KB.

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

...

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

send broadcast

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

i^*

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

③ Choose a random party

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

⑤ Check $\forall i \neq i^*$

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

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

Prover

Verifier

MPCitH transform

① Generate and commit shares

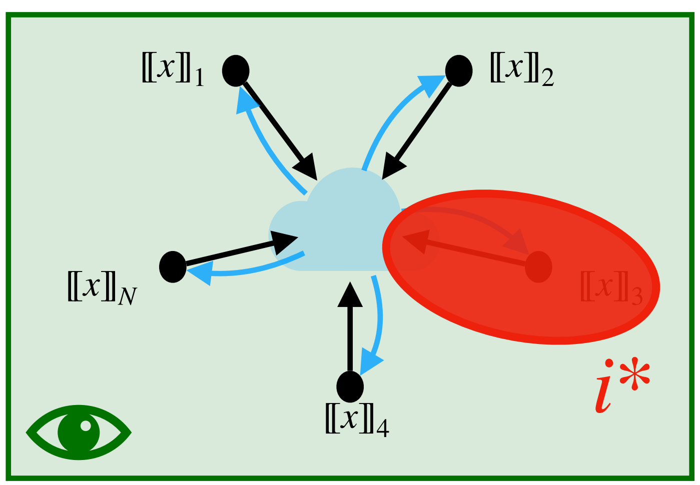
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

Compute

$\forall i, \mathsf{com}_i = \mathrm{Com}^{\rho_i}([\![x]\!]_i)$

$h_1 = \mathrm{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$ →

② Run MPC in their head



$h_2 = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$ →

③ Choose a random party

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

← i^*

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

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

⑤ Compute $\forall i \neq i^*$

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

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

Check $h_1 = \mathrm{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$

Check $h_2 = \mathrm{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$

Prover

Verifier

MPCitH transform

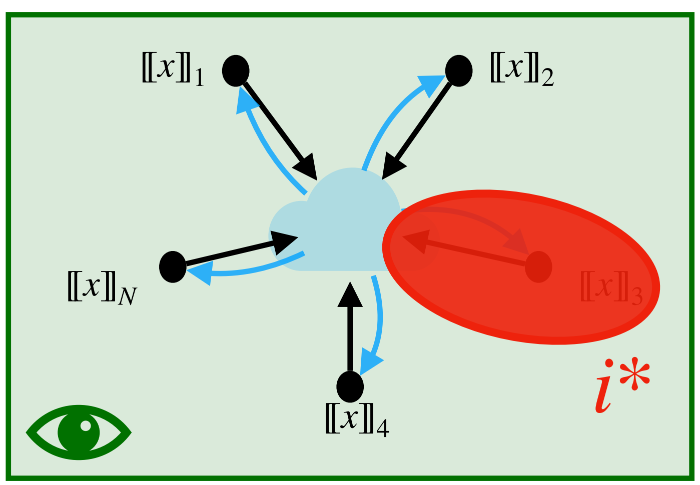
① Generate and commit shares

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

Compute

$\forall i, \text{com}_i = \text{Com}^{\rho_i}([\![x]\!]_i)$

② Run MPC in their head



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

$h_1 = \text{Hash}(\text{com}_1, \ldots, \text{com}_N)$ →

$h_2 = \text{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$ →

← i^*

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

③ Choose a random party

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

⑤ Compute $\forall i \neq i^*$

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

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

Check $h_1 = \text{Hash}(\text{com}_1, \ldots, \text{com}_N)$

Check $h_2 = \text{Hash}([\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N)$

<u>Prover</u>

<u>Verifier</u>

Using a Seed Tree

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

$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + \ldots + [\![x]\!]_{N-1} + [\![x]\!]_N$$

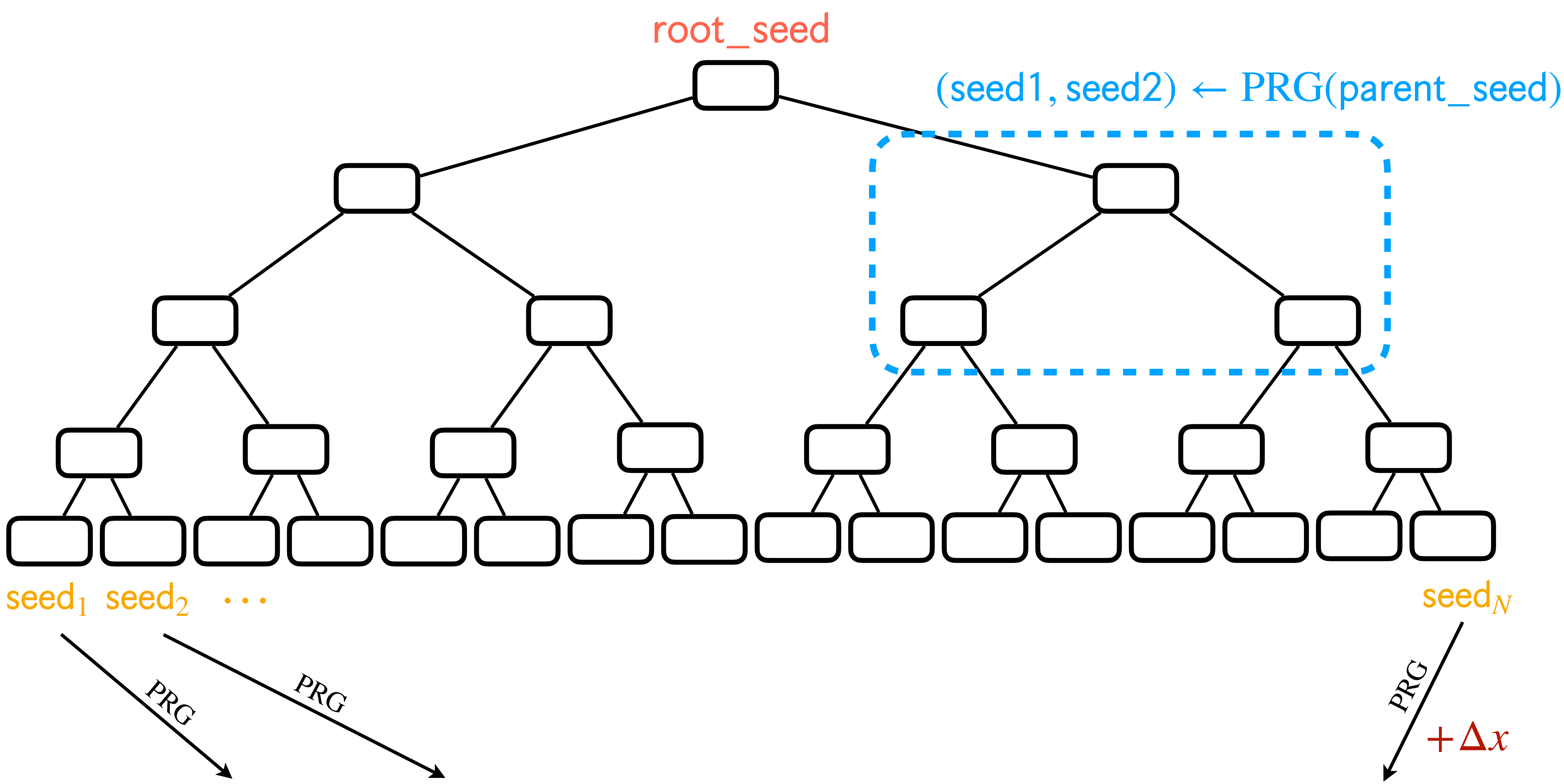
Using a Seed Tree

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

$$\begin{array}{ccccccccccccc} & \text{seed}_1 & & \text{seed}_2 & & \text{seed}_3 & & & & \text{seed}_{N-1} & & \text{seed}_N \\ & \big\downarrow{\scriptstyle \text{PRG}} & & \big\downarrow{\scriptstyle \text{PRG}} & & \big\downarrow{\scriptstyle \text{PRG}} & & & & \big\downarrow{\scriptstyle \text{PRG}} & & \big\downarrow{\scriptstyle \text{PRG}} \; {\color{red}+\Delta x} \\ x = & [\![x]\!]_1 & + & [\![x]\!]_2 & + & [\![x]\!]_3 & + & \ldots & + & [\![x]\!]_{N-1} & + & [\![x]\!]_N \end{array}$$

Using a Seed Tree

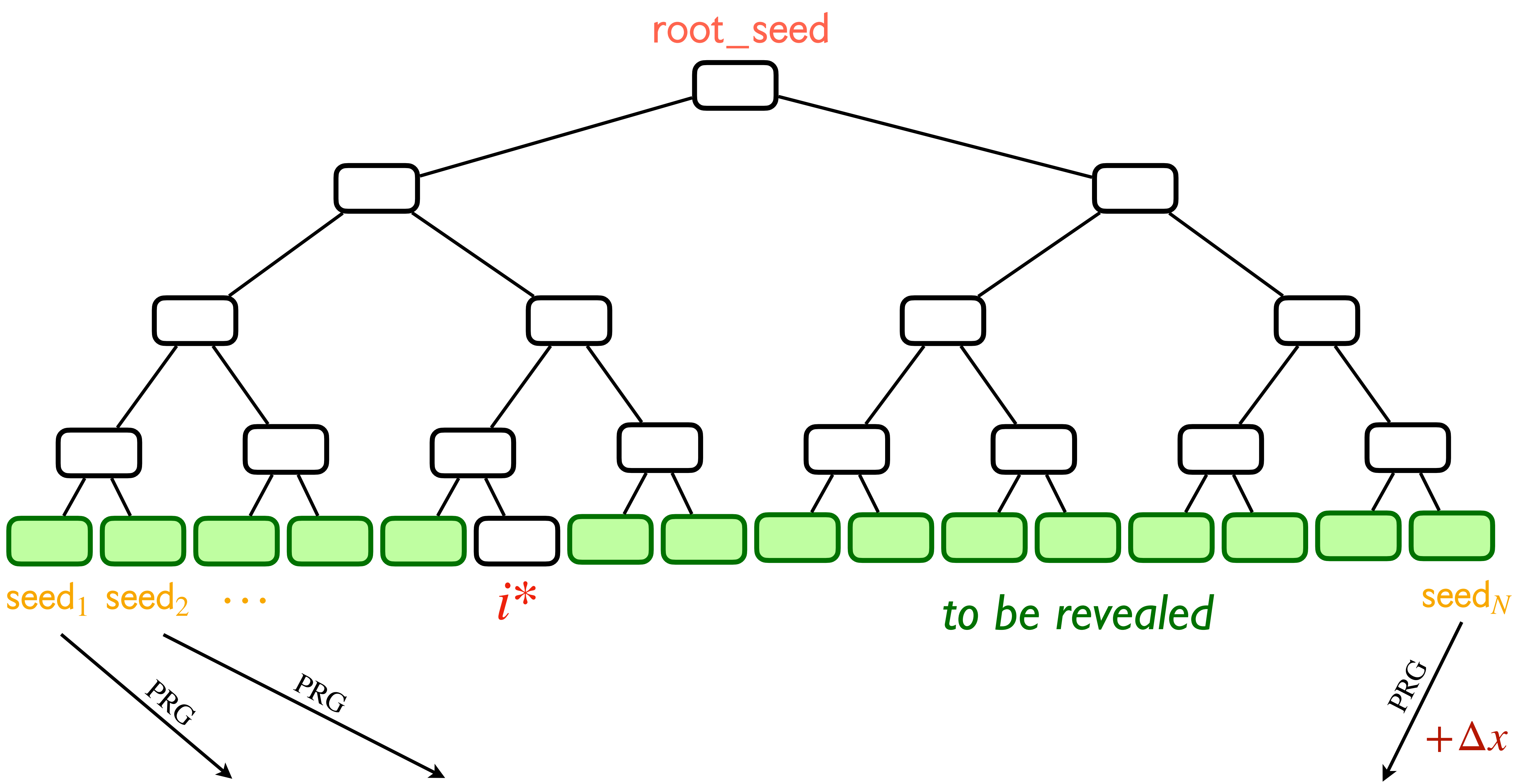
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + \ldots + [\![x]\!]_{N-1} + [\![x]\!]_N$$

Using a Seed Tree

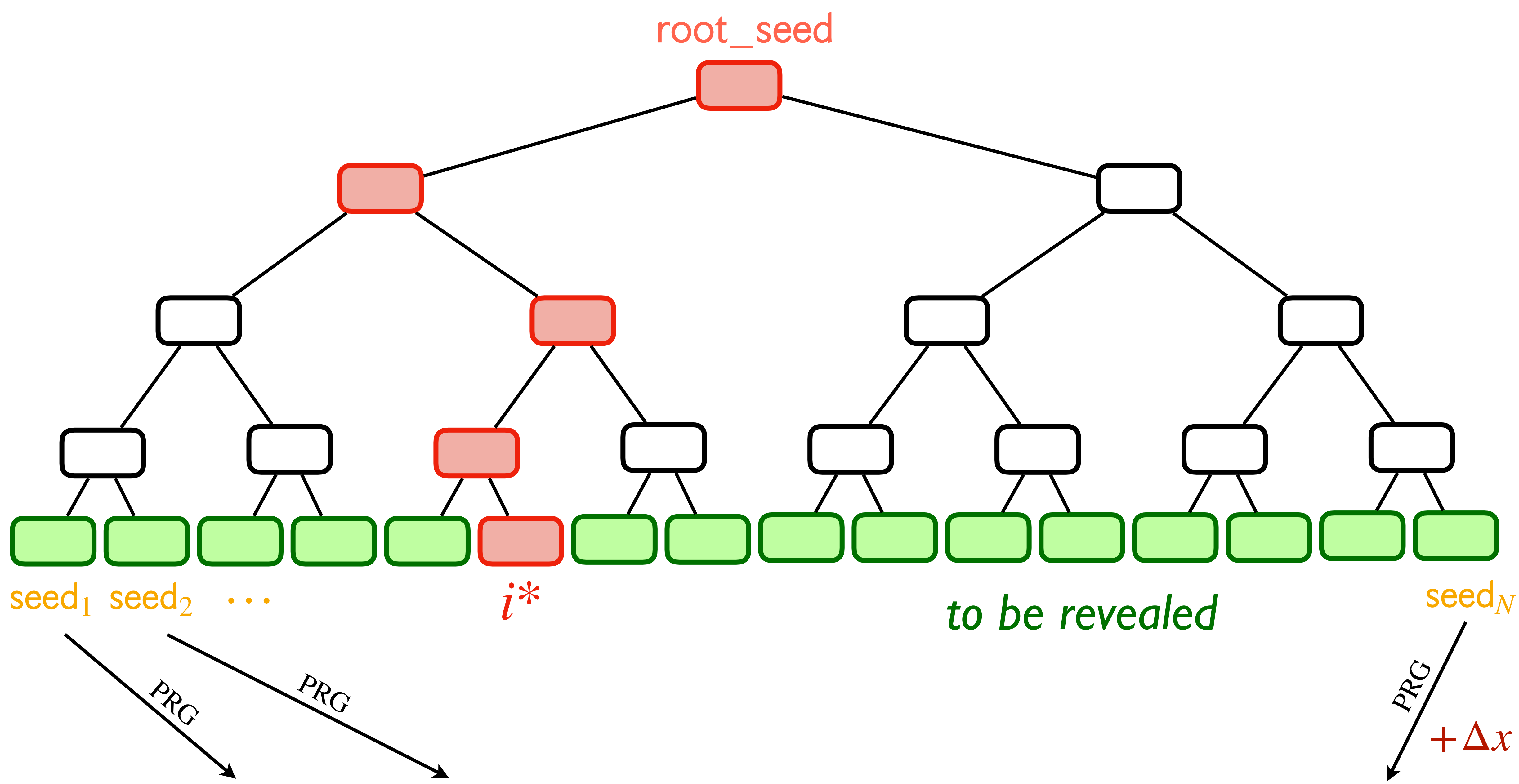
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + \ldots + [\![x]\!]_{N-1} + [\![x]\!]_N$$

Using a Seed Tree

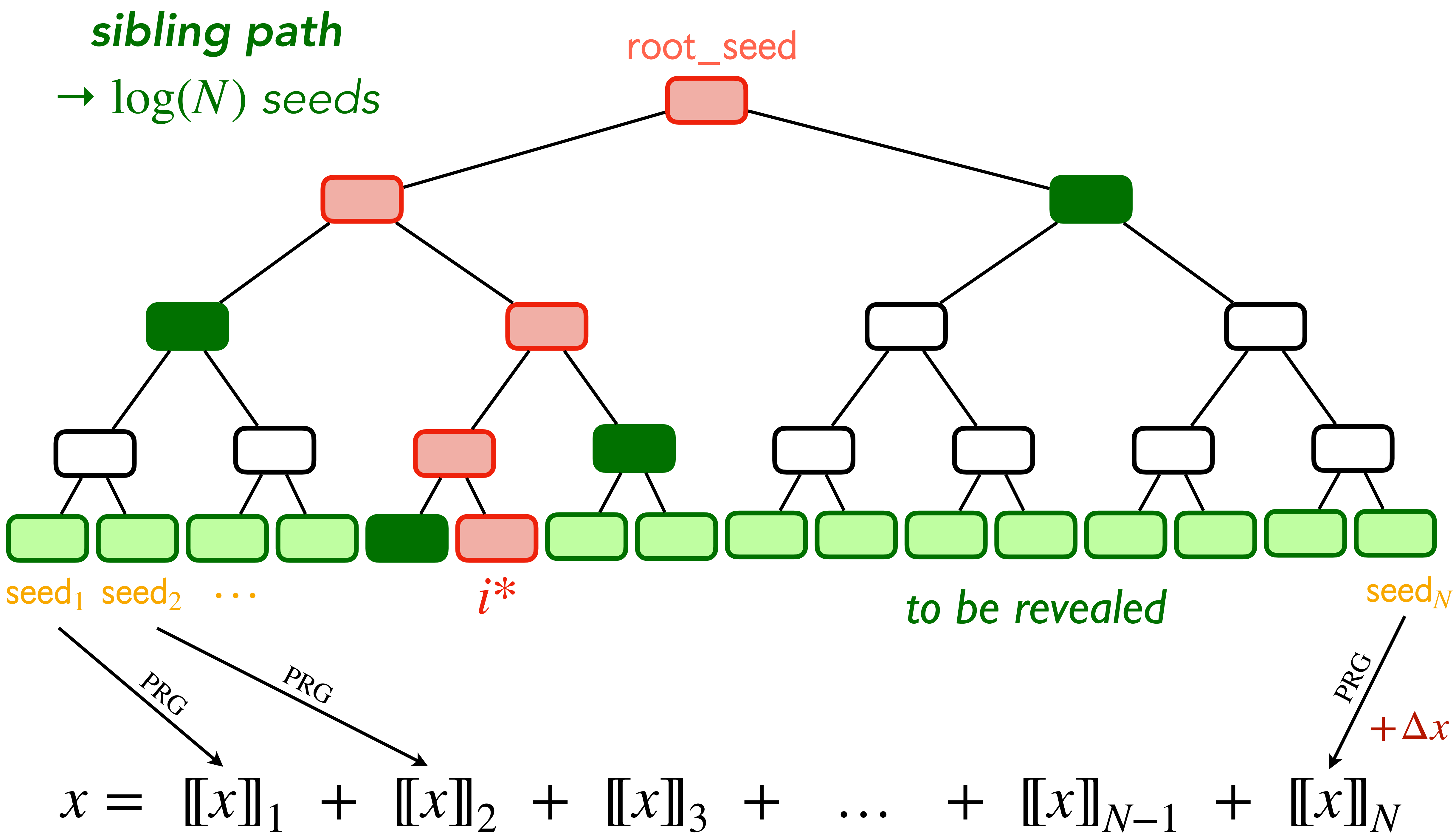
[KKW18] Katz, Kolesnikov, Wang: “Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures” (CCS 2018)



$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + \ldots + [\![x]\!]_{N-1} + [\![x]\!]_N$$

Using a Seed Tree

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



Traditional MPCitH transformation

Size of the broadcast

Size of the auxiliary value

Path in the seed (GGM) tree

$$\text{Size} \approx \tau \cdot \left(|\Delta x| + |\alpha| + \lambda \cdot \log_2 N + 2\lambda \right)$$

Commitment
of the hidden party

Number of repetitions to achieve the desired security level

$$\tau \approx \frac{\lambda}{\log_2 N}$$

Traditional MPCitH transformation

Size of the broadcast

Size of the auxiliary value

Path in the seed (GGM) tree

$$\text{Size} \approx \tau \cdot \left(|\Delta x| + |\alpha| + \lambda \cdot \log_2 N + 2\lambda \right)$$

Commitment
of the hidden party

Number of repetitions to achieve the desired security level

$$\tau \approx \frac{\lambda}{\log_2 N}$$

Emulating τ MPC protocols with N parties is very expensive.

The Hypercube Technique

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

Traditional: one sharing of x

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

The Hypercube Technique

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

Traditional: one sharing of x

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

Hypercube: D sharings of x, with the same auxiliary value Δx

$$x = \left\{ \begin{array}{c} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{array} \right\} + \Delta x$$

such that $N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$

The Hypercube Technique

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

$$x = \left\{ \begin{array}{c} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{array} \right\} + \Delta x$$

$$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$$

How to build these D sharings?

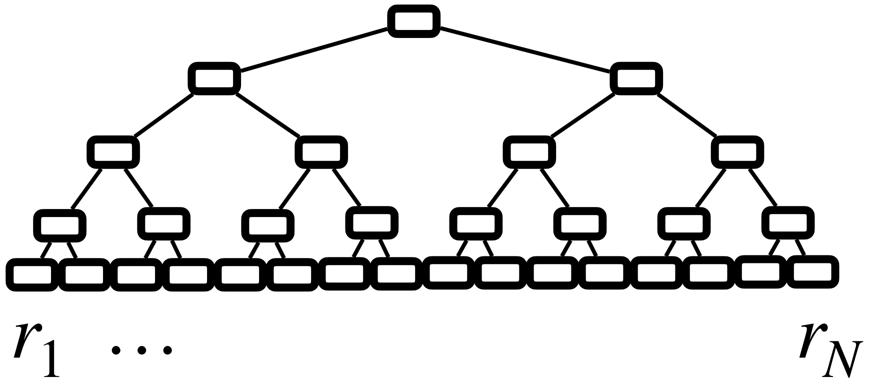
The Hypercube Technique

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

$$x = \begin{Bmatrix} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{Bmatrix} + \Delta x$$

$$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$$

How to build these D sharings?



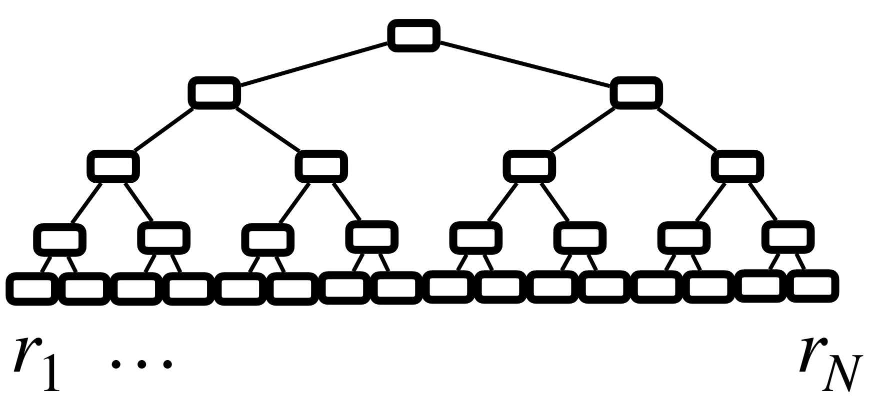
The Hypercube Technique

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

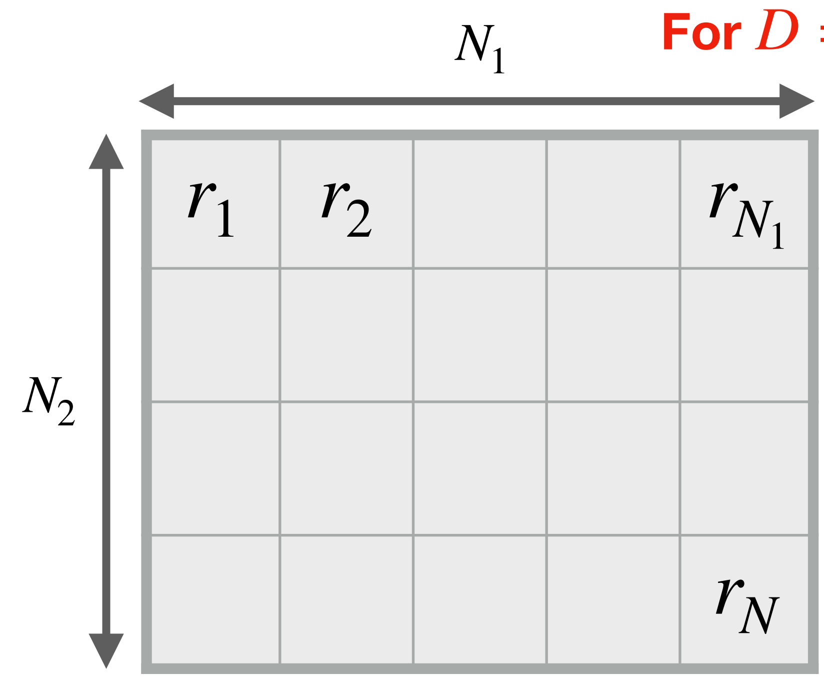
$$x = \begin{Bmatrix} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{Bmatrix} + \Delta x$$

$$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$$

How to build these D sharings?



For $D = 2$



The Hypercube Technique

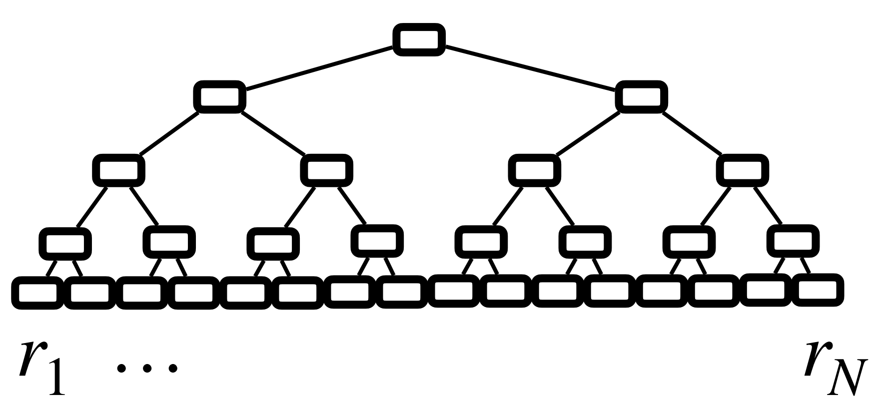
[AGHHJY23] Aguilar-Melchor, Gama, Howe, Hülsing, Joseph, Yue: "The Return of the SDitH" (Eurocrypt 2023)

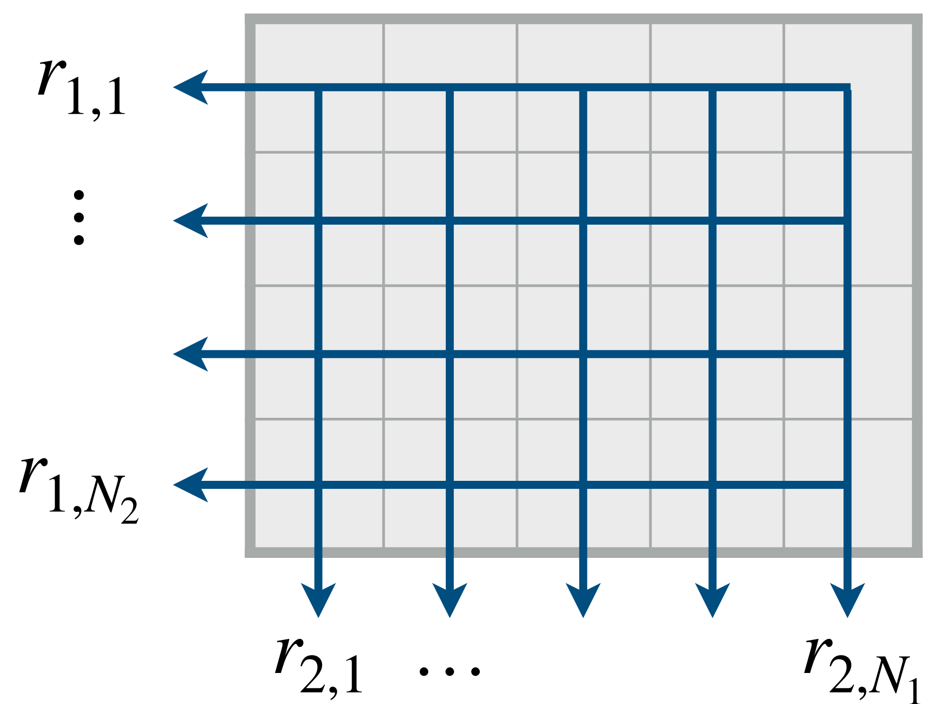
$$x = \left\{ \begin{matrix} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{matrix} \right\} + \Delta x$$

$$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$$

How to build these D sharings?

For $D = 2$





The Hypercube Technique

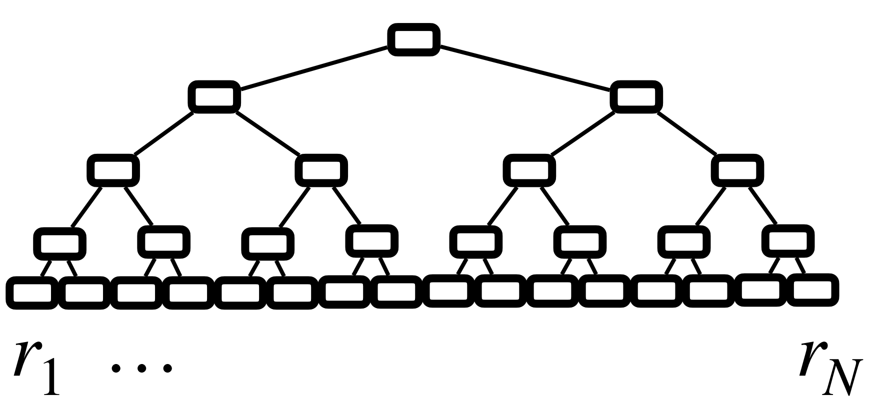
[AGHHJY23] Aguilar-Melchor, Gama, Howe, Hülsing, Joseph, Yue: "The Return of the SDitH" (Eurocrypt 2023)

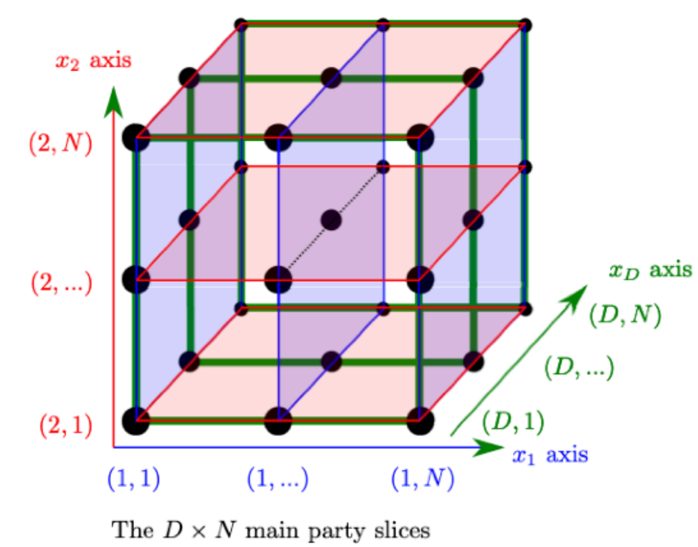
$$x = \left\{ \begin{array}{c} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{array} \right\} + \Delta x$$

$$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$$

How to build these D sharings?

For $D \geq 2$





Source: Figure from [AGHHJY23]

The Hypercube Technique

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

$$x = \left\{ \begin{matrix} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{matrix} \right\} + \Delta x$$

$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$

What about the soundness?

We emulate D sub-protocols…

$$\frac{1}{N_1} \cdot \frac{1}{N_2} \cdot \ldots \cdot \frac{1}{N_D} = \frac{1}{N}$$

Same soundness error as before!

The Hypercube Technique

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

$$x = \begin{Bmatrix} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{Bmatrix} + \Delta x$$

$$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$$

What about the signature size?

$$\text{Size} \approx \tau \cdot \left(|\Delta x| + |\alpha| + \lambda \cdot \log_2 N + 2\lambda \right)$$

Same signature size as before!

The Hypercube Technique

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

$$x = \left\{ \begin{matrix} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{matrix} \right\} + \Delta x$$

$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$

What about the emulation cost?

We emulate D sub-protocols…

$$N_1 + N_2 + \ldots + N_D$$

The Hypercube Technique

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

$$x = \left\{ \begin{matrix} r_{1,1} + r_{1,2} + \dots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \dots + r_{2,N_2} \\ \dots \\ r_{D,1} + r_{D,2} + \dots + r_{D,N_D} \end{matrix} \right\} + \Delta x$$

$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$

What about the emulation cost?

We emulate D sub-protocols…

$$N_1 + N_2 + \ldots + N_D$$

The Hypercube Technique

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

$$x = \left\{ \begin{array}{c} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{array} \right\} + \Delta x$$

$$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$$

What about the emulation cost?

We emulate D sub-protocols…

$$\sout{N_1 + N_2 + \ldots + N_D}$$

$$1 + (N_1 - 1) + (N_2 - 1) + \ldots + (N_D - 1)$$

The Hypercube Technique

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

$$x = \left\{ \begin{array}{c} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{array} \right\} + \Delta x$$

$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$

What about the emulation cost?

We emulate D sub-protocols…

$$\sout{N_1 + N_2 + \ldots + N_D}$$

$$1 + (N_1 - 1) + (N_2 - 1) + \ldots + (N_D - 1)$$

instead of $N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$

The Hypercube Technique

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

$$x = \begin{Bmatrix} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{Bmatrix} + \Delta x$$

$N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$

What about the emulation cost?

We emulate D sub-protocols…

~~$N_1 + N_2 + \ldots + N_D$~~

$$1 + (N_1 - 1) + (N_2 - 1) + \ldots + (N_D - 1)$$

$D = \log_2 N$
$N_1 = \ldots = N_D = 2$

$$1 + \log_2 N$$

instead of $N = N_1 \cdot N_2 \cdot \ldots \cdot N_D$

The Hypercube Technique

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

Traditional: N party emulations per repetition

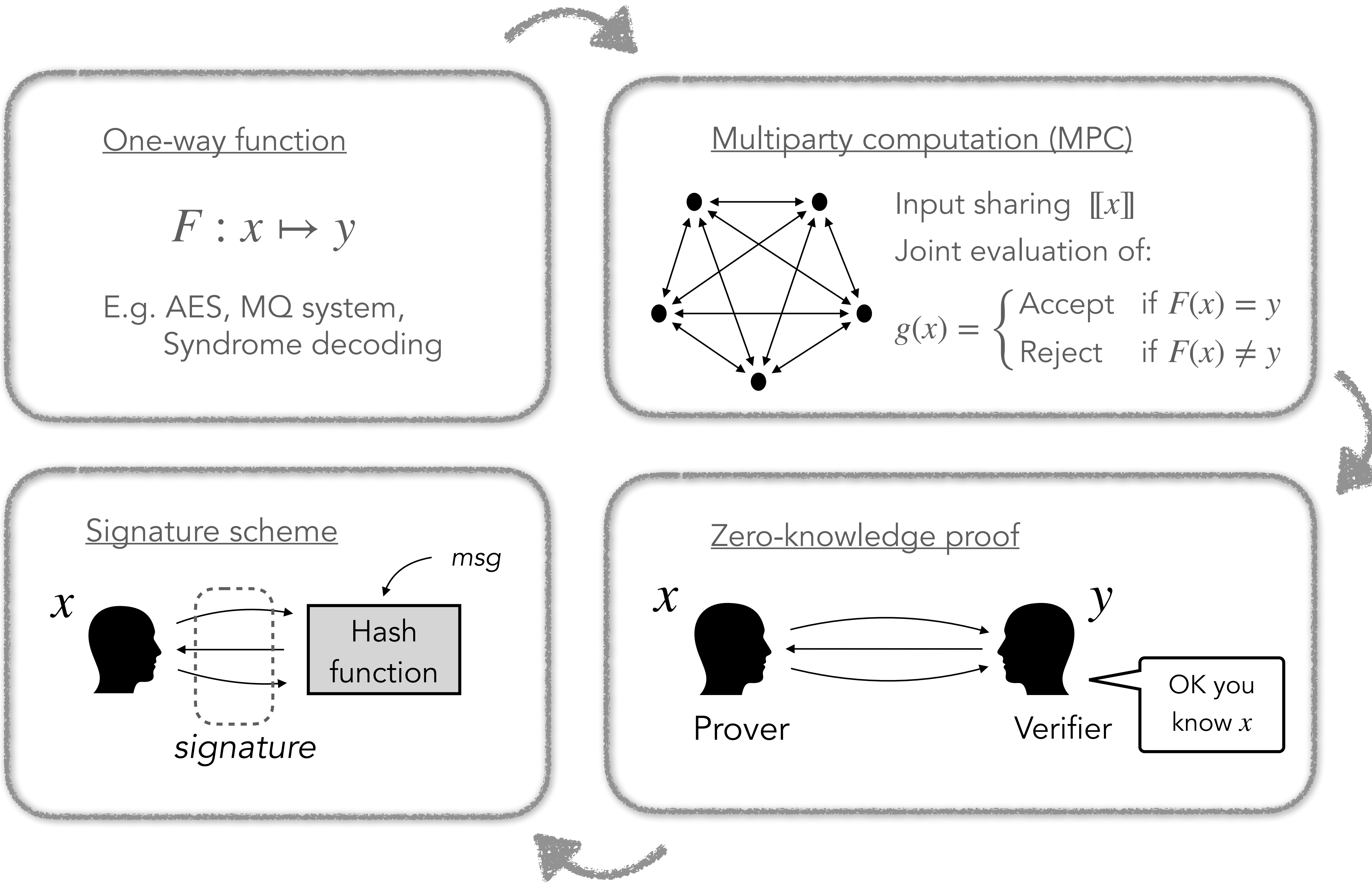
$$D = \log_2 N$$
$$N_1 = \ldots = N_D = 2$$

$$N = 256$$

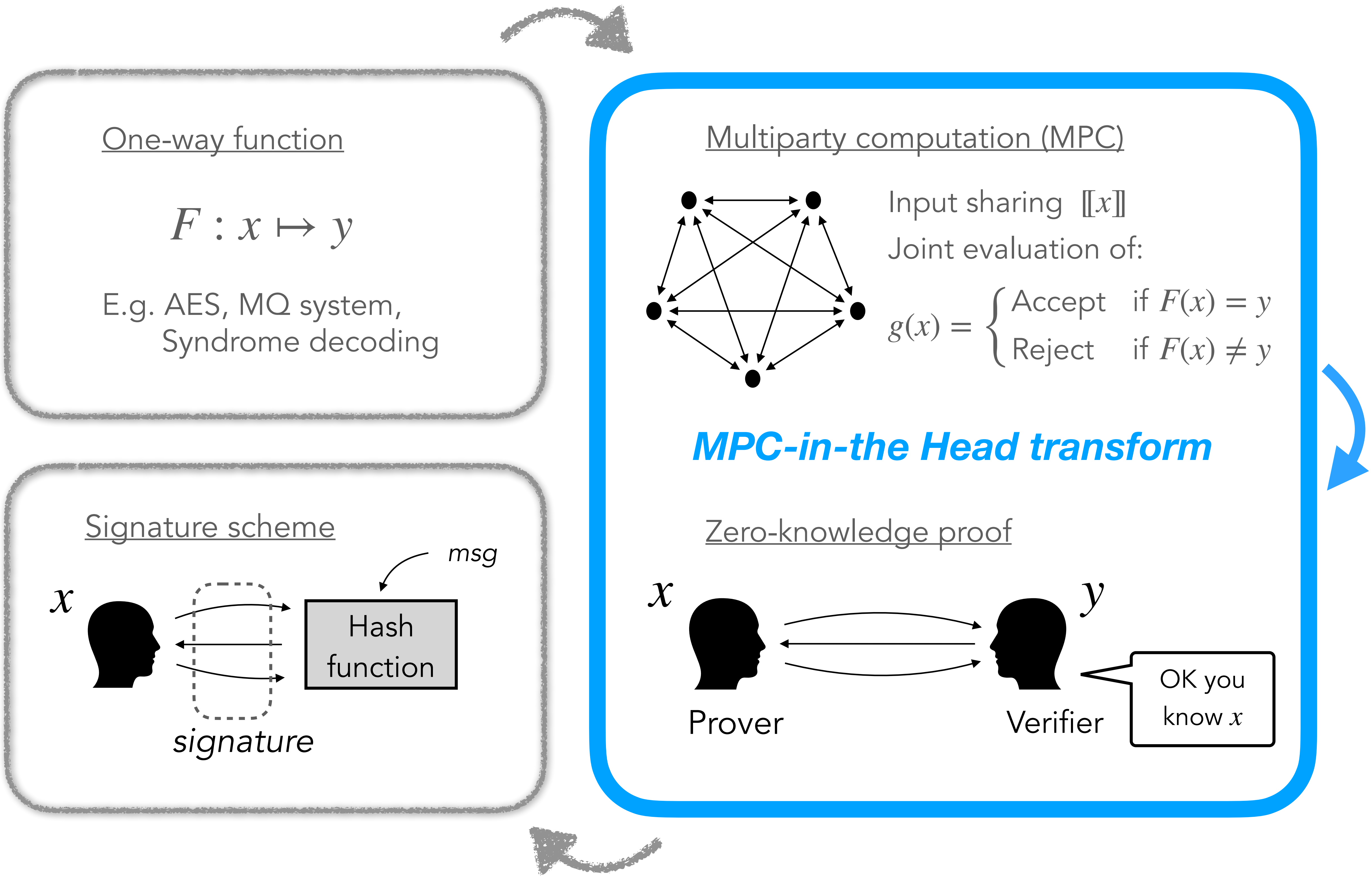
Hypercube: $1 + \log_2 N$ party emulations per repetition

$$1 + \log_2 N = 9$$

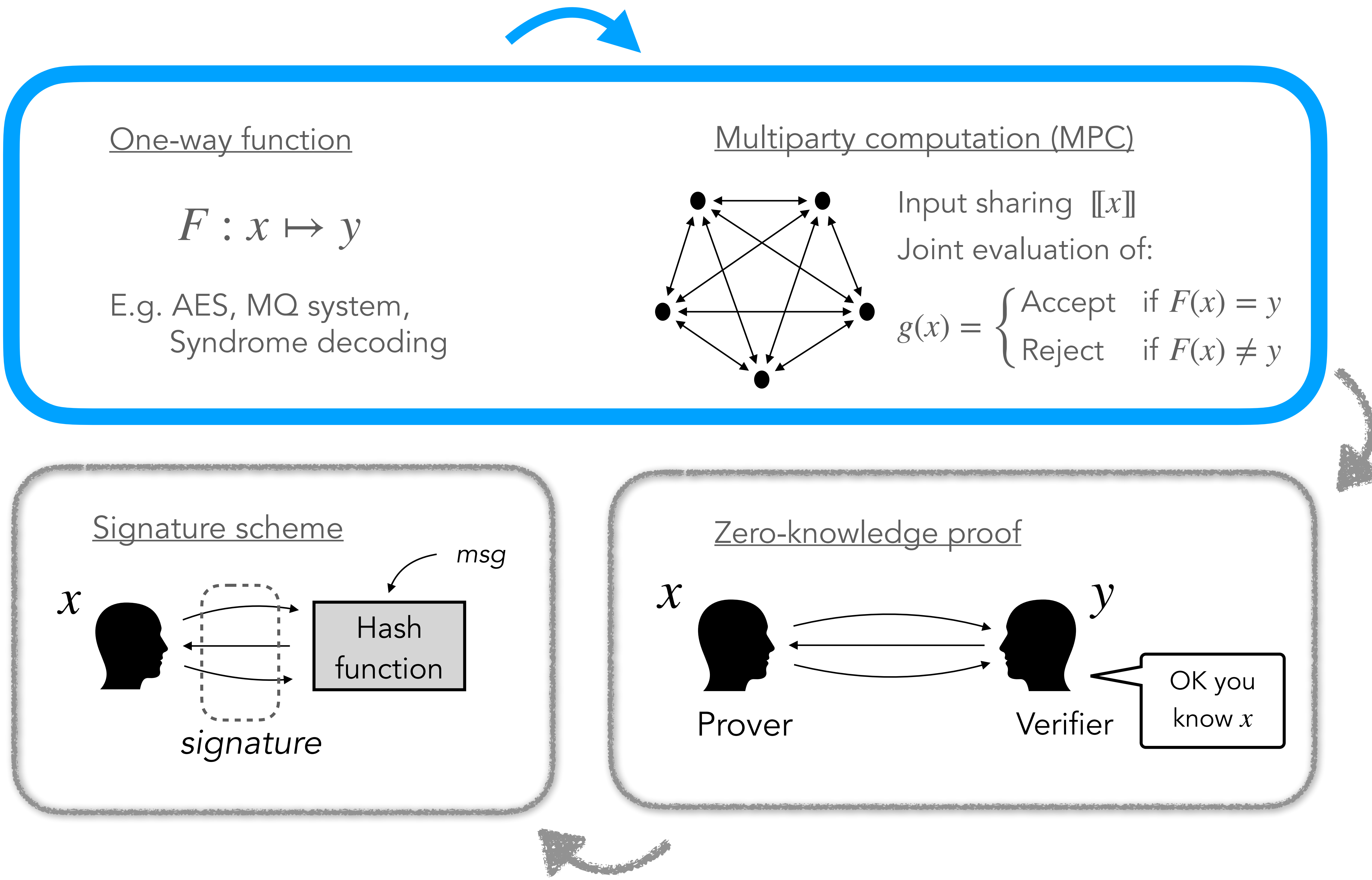
From MPC-in-the-Head to signatures



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



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

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

- RYDE: Syndrome decoding problem in the rank metric

  From a matrix $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and a vector $y \in \mathbb{F}_{q^m}^{n-k}$, find a vector $x \in \mathbb{F}_{q^m}^{n}$ s.t.

$$y = Hx \quad \text{and} \quad \dim \operatorname{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

- RYDE: Syndrome decoding problem in the rank metric

  From a matrix $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and a vector $y \in \mathbb{F}_{q^m}^{n-k}$, find a vector $x \in \mathbb{F}_{q^m}^{n}$ s.t.

$$y = Hx \quad \text{and} \quad \dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

- MIRA: MinRank problem

  From $k+1$ matrices $M_0, \ldots, M_k \in \mathbb{F}_q^{m\times n}$, find a vector $x \in \mathbb{F}_q^k$ such that

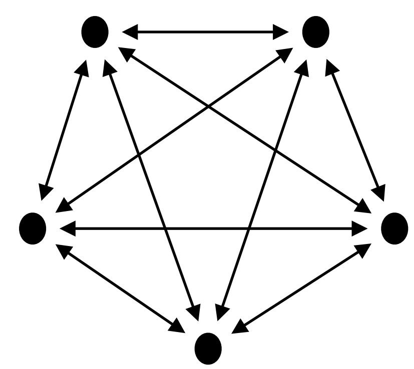
$$\mathrm{rank}\left(M_0 + \sum_{i=1}^{k} x_i M_i\right) \leq r.$$

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

- RYDE: Syndrome decoding problem in the rank metric

  From a matrix $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and a vector $y \in \mathbb{F}_{q^m}^{n-k}$, find a vector $x \in \mathbb{F}_{q^m}^{n}$ s.t.

$$y = Hx \quad \text{and} \quad \dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

- MIRA: MinRank problem

  From $k+1$ matrices $M_0, \ldots, M_k \in \mathbb{F}_q^{m\times n}$, find a vector $x \in \mathbb{F}_q^k$ such that

$$\mathrm{rank}\left(M_0 + \sum_{i=1}^{k} x_i M_i\right) \leq r.$$

Rank Syndrome Decoding Problem

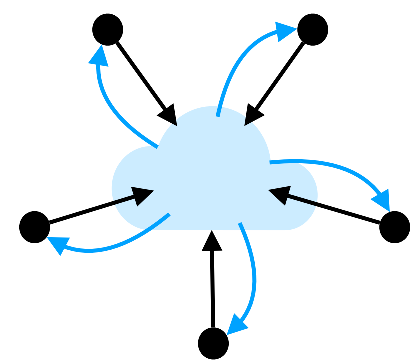
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$,
find x such that

$$y = Hx$$

and

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

Multiparty computation (MPC)



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

- Check the linear constraint
- Check the rank constraint

- *Input of the MPC protocol*: a sharing $[\![x]\!]$ satisfying

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

The case of ***RYDE***

Rank Syndrome Decoding Problem

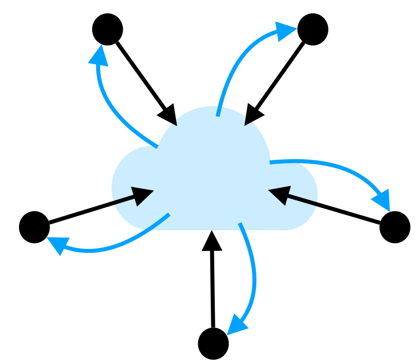
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$,
find x such that

$$y = Hx$$

and

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \le r.$$

Multiparty computation (MPC)



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

- Check the linear constraint
- Check the rank constraint

- *Input of the MPC protocol*: a sharing $[\![x]\!]$ satisfying

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

- Each party will compute

$$[\![Hx]\!]_i \leftarrow H[\![x]\!]_i$$

  and broadcast $[\![Hx]\!]_i$.

- Everyone can check that $[\![Hx]\!]_1 + \ldots + [\![Hx]\!]_N$ is equal to y.

*The case of **RYDE***

Rank Syndrome Decoding Problem

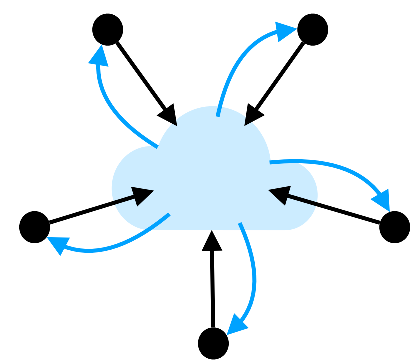
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$,
find x such that

$$y = Hx$$

and

$$\dim \operatorname{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

Multiparty computation (MPC)



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

- Check the linear constraint ✔
- Check the rank constraint

- *Input of the MPC protocol*: a sharing $[\![x]\!]$ satisfying

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

- Each party will compute

$$[\![Hx]\!]_i \leftarrow H[\![x]\!]_i$$

and broadcast $[\![Hx]\!]_i$.

$$y = \sum_i [\![Hx]\!]_i = H \sum_i [\![x]\!]_i = Hx$$

- Everyone can check that $[\![Hx]\!]_1 + \ldots + [\![Hx]\!]_N$ is equal to y.

The case of ***RYDE***

Rank Syndrome Decoding Problem

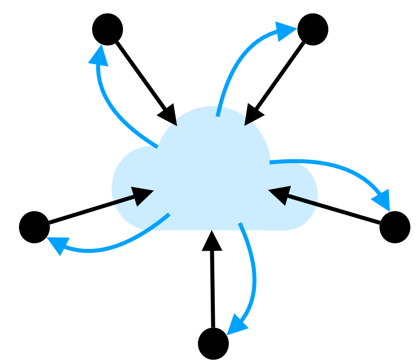
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$,
find x such that

$$y = Hx$$

and

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

Multiparty computation (MPC)



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

- Check the linear constraint ✔
- Check the rank constraint

- *Input of the MPC protocol*: a sharing $[\![x]\!]$ satisfying

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

- How to check efficiently that x has a rank weight of at most r?

The case of RYDE

Rank Syndrome Decoding Problem

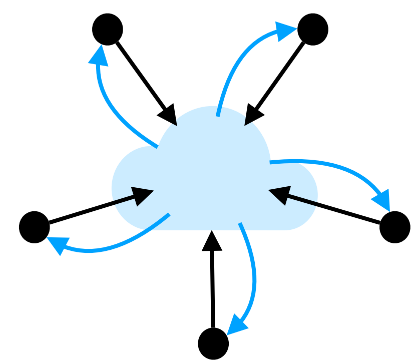
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$, find x such that

$$y = Hx$$

and

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

Multiparty computation (MPC)



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

- Check the linear constraint ✔
- Check the rank constraint

- *Input of the MPC protocol*: a sharing $[\![x]\!]$ satisfying

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

- How to check efficiently that x has a rank weight of at most r?

[Fen22] Feneuil: "Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP" (ePrint 2022/1512)

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r$$

$$\Longleftrightarrow$$

$$\exists \text{ degree-}q^r\ q\text{-polynomial } P := \sum_{i=0}^{r} a_i X^{q^i} \text{ s.t. } P(x_1) = \ldots = P(x_n) = 0$$

The case of ***RYDE***

Rank Syndrome Decoding Problem

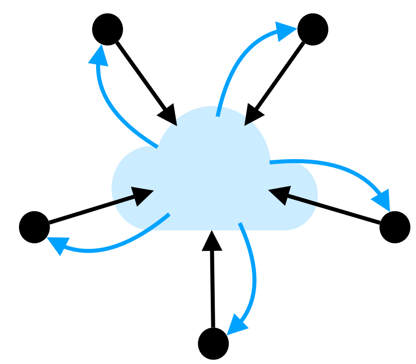
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$,
find x such that

$$y = Hx$$

and

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \le r.$$

Multiparty computation (MPC)



Input sharings $[\![x]\!]$ and $[\![P]\!]$

- Check the linear constraint ✔
- Check the rank constraint

- *Input of the MPC protocol*: two sharings $[\![x]\!]$ and $[\![P]\!] := \sum_{i=0}^{r} [\![a_i]\!] X^{q^i}$.

*The case of **RYDE***

<u>Rank Syndrome Decoding Problem</u>

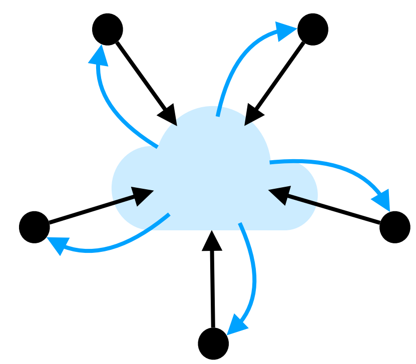
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$,
find x such that

$$y = Hx$$

and

$$\dim \operatorname{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \le r.$$

<u>Multiparty computation (MPC)</u>



Input sharings $[\![x]\!]$ and $[\![P]\!]$

- Check the linear constraint ✔
- Check the rank constraint

- *<u>Input of the MPC protocol</u>*: two sharings $[\![x]\!]$ and $[\![P]\!] := \sum_{i=0}^{r} [\![a_i]\!] X^{q^i}$.
- The parties will compute $[\![x_j^{q^i}]\!]$ from $[\![x_j]\!]$ for all (i, j).

The case of RYDE

Rank Syndrome Decoding Problem

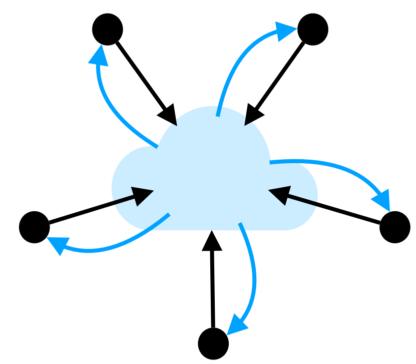
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$,
find x such that

$$y = Hx$$

and

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

Multiparty computation (MPC)



Input sharings $[\![x]\!]$ and $[\![P]\!]$

- Check the linear constraint ✔
- Check the rank constraint ✔

- *Input of the MPC protocol*: two sharings $[\![x]\!]$ and $[\![P]\!] := \sum_{i=0}^{r} [\![a_i]\!] X^{q^i}$.
- The parties will compute $[\![x_j^{q^i}]\!]$ from $[\![x_j]\!]$ for all (i, j).
- For all j, the parties will check that $P(x_j) = 0$ by checking that

$$\left\langle \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_r \end{pmatrix}, \begin{pmatrix} x_j \\ x_j^q \\ \vdots \\ x_j^{q^r} \end{pmatrix} \right\rangle = 0,$$

using a [BN20]-like MPC protocol.

[BN20] Baum, Nof: "Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography" (PKC 2020)

The case of RYDE

Rank Syndrome Decoding Problem

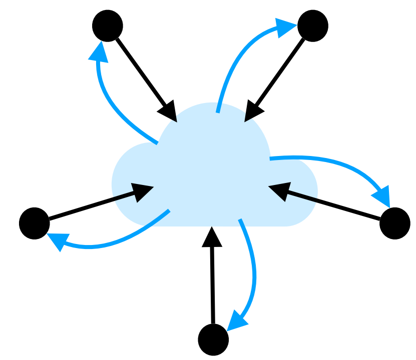
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$,
find x such that

$$y = Hx$$

and

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

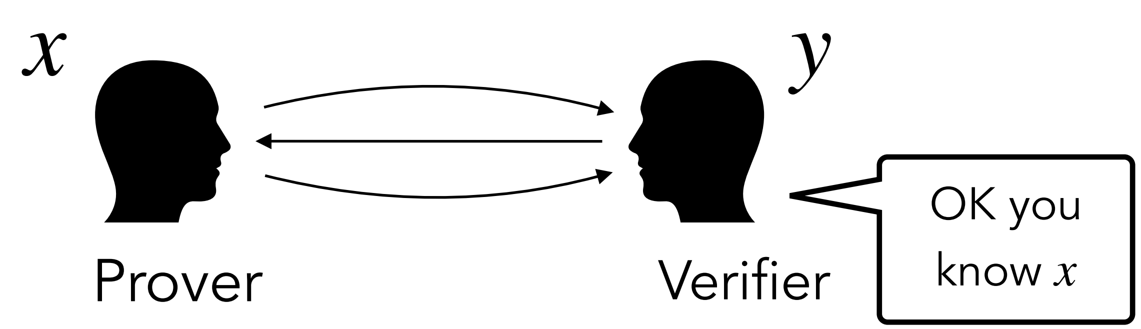
Multiparty computation (MPC)



Input sharings $[\![x]\!]$ and $[\![P]\!]$

- Check the linear constraint
- Check the rank constraint

Zero-knowledge proof



The case of RYDE

Rank Syndrome Decoding Problem

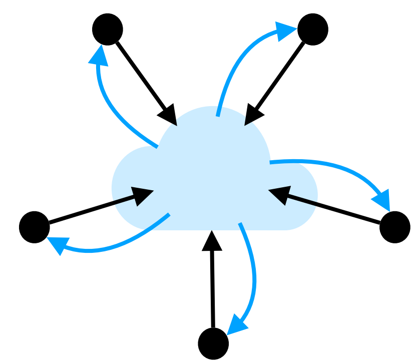
From $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$, find x such that

$$y = Hx$$

and

$$\dim \mathrm{Span}_{\mathbb{F}_q}(x_1, \ldots, x_n) \leq r.$$

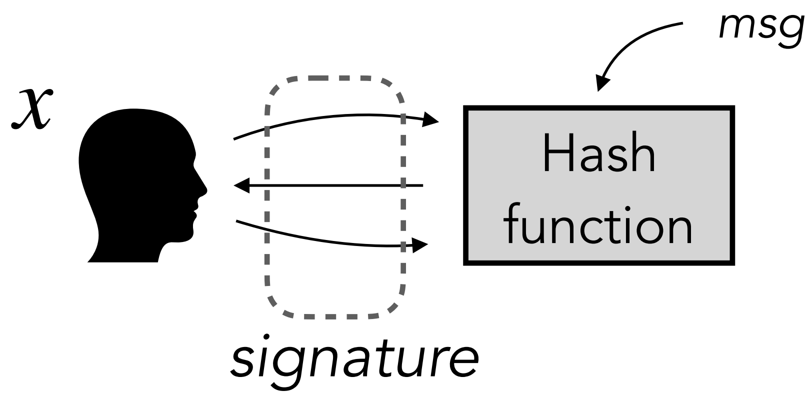
Multiparty computation (MPC)



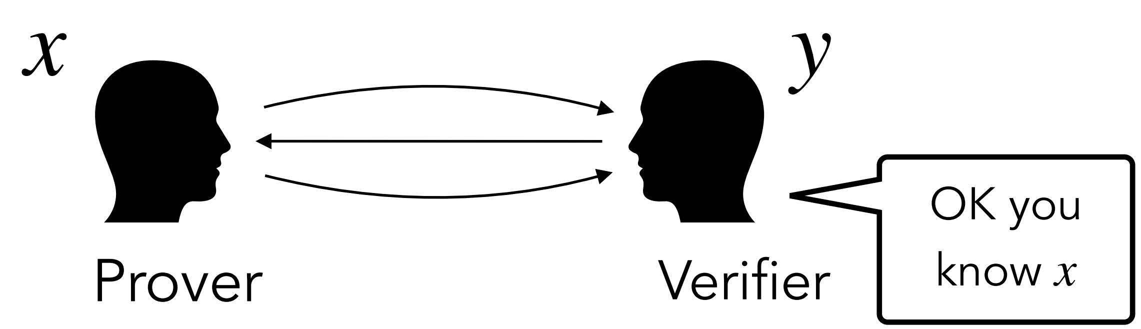
Input sharings $[\![x]\!]$ and $[\![P]\!]$

- Check the linear constraint
- Check the rank constraint

RYDE



Zero-knowledge proof



Fiat-Shamir transform

Should take [KZ20] attack into account (since there are 5 rounds)!

[KZ20] Kales, Zaverucha. "An attack on some signature schemes constructed from five-pass identification schemes" (CANS20)

MinRank Problem

From $M_0, \ldots, M_k \in \mathbb{F}_q^{m \times n}$, find x such that

$$\operatorname{rank}\left(M_0 + \sum_{i=1}^{k} x_i M_i\right) \leq r.$$

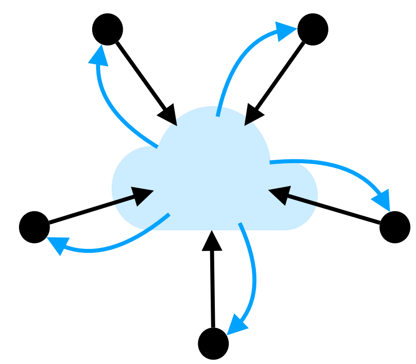
The case of MIRA

MinRank Problem

From $M_0, \ldots, M_k \in \mathbb{F}_q^{m \times n}$, find x such that

$$\operatorname{rank}\left(M_0 + \sum_{i=1}^{k} x_i M_i\right) \leq r.$$

Multiparty computation (MPC)



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

- Check the rank constraint

- *Input of the MPC protocol*: a sharing $[\![x]\!]$ of x.
- Let us denote $E = M_0 + \sum_{i=1}^{k} x_i M_i \in \mathbb{F}_q^{m \times n}$.

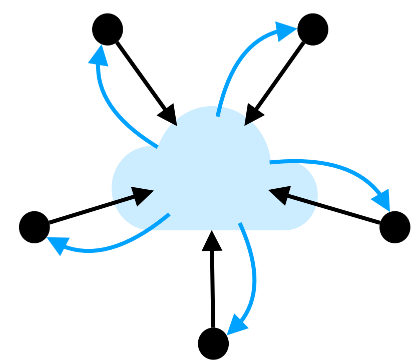
The case of MIRA

MinRank Problem

From $M_0, \ldots, M_k \in \mathbb{F}_q^{m \times n}$, find x such that

$$\text{rank}\left(M_0 + \sum_{i=1}^{k} x_i M_i\right) \leq r.$$

Multiparty computation (MPC)



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

- Check the rank constraint

- *Input of the MPC protocol*: a sharing $[\![x]\!]$ of x.
- Let us denote $E = M_0 + \sum_{i=1}^{k} x_i M_i \in \mathbb{F}_q^{m \times n}$.
- We decompose $E = \left(E_1 \mid E_2 \mid \ldots \mid E_n\right)$ by columns and write each column E_i as a field element $e_i \in \mathbb{F}_{q^m}$. We thus have

$$\text{rank}(E) \leq r \iff \dim \text{Span}_{\mathbb{F}_q}(e_1, \ldots, e_n) \leq r.$$

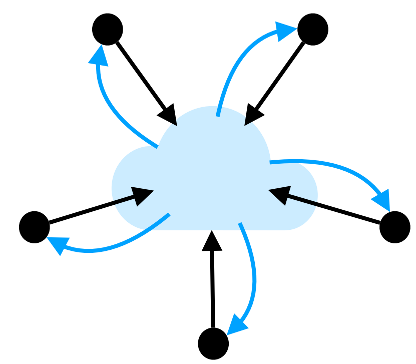
The case of MIRA

<u>MinRank Problem</u>

From $M_0, \ldots, M_k \in \mathbb{F}_q^{m \times n}$, find x such that

$$\operatorname{rank}\left(M_0 + \sum_{i=1}^{k} x_i M_i\right) \le r.$$

<u>Multiparty computation (MPC)</u>



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

- Check the rank constraint

- *<u>Input of the MPC protocol</u>*: a sharing $[\![x]\!]$ of x.
- Let us denote $E = M_0 + \sum_{i=1}^{k} x_i M_i \in \mathbb{F}_q^{m \times n}$.
- We decompose $E = \left(E_1 \mid E_2 \mid \ldots \mid E_n\right)$ by columns and write each column E_i as a field element $e_i \in \mathbb{F}_{q^m}$. We thus have
$$\operatorname{rank}(E) \le r \iff \dim \operatorname{Span}_{\mathbb{F}_q}(e_1, \ldots, e_n) \le r.$$
- We can use the *same protocol* than with RYDE.

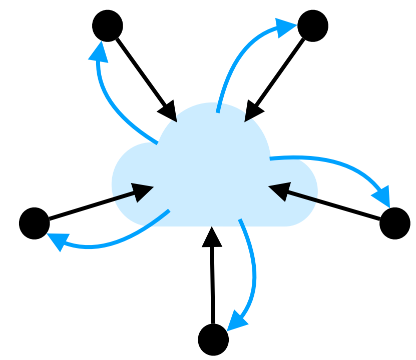
The case of MIRA

MinRank Problem

From $M_0, \ldots, M_k \in \mathbb{F}_q^{m \times n}$, find x such that

$$\operatorname{rank}\left(M_0 + \sum_{i=1}^{k} x_i M_i\right) \le r.$$

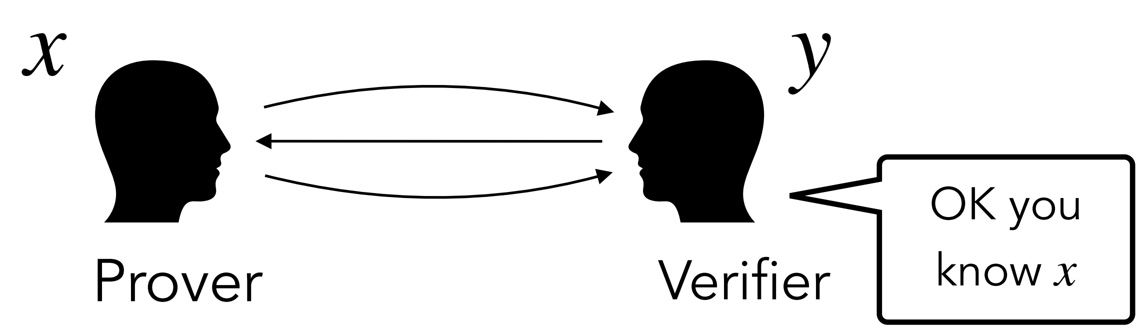
Multiparty computation (MPC)



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

- Check the rank constraint

Zero-knowledge proof



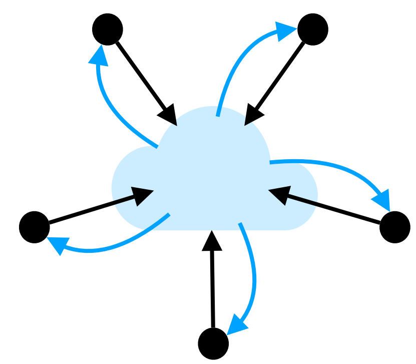
The case of MIRA

MinRank Problem

From $M_0, \ldots, M_k \in \mathbb{F}_q^{m \times n}$, find x such that

$$\operatorname{rank}\left(M_0 + \sum_{i=1}^{k} x_i M_i\right) \leq r.$$

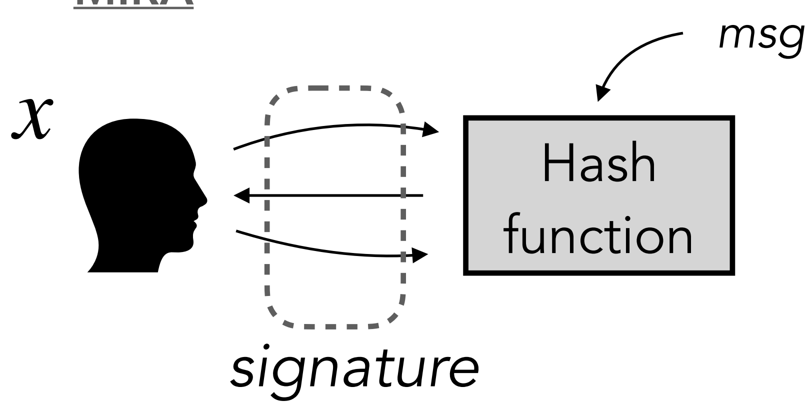
Multiparty computation (MPC)



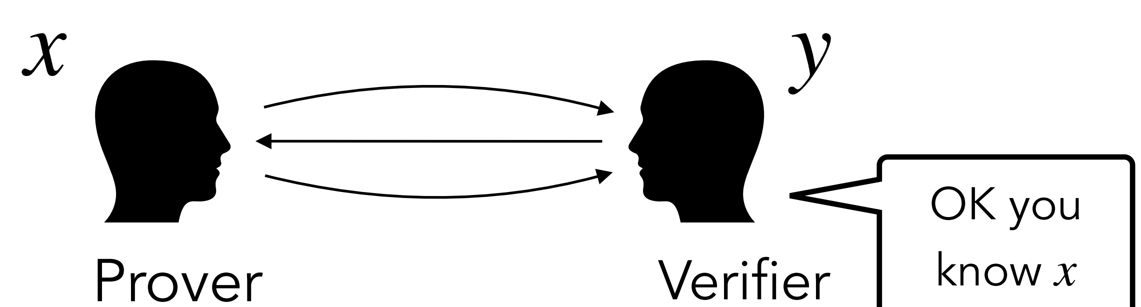
Input sharing $[\![x]\!]$

- Check the rank constraint

MIRA



Zero-knowledge proof



Performances

Performances

	Short Instance			Fast Instance		
	$\|sig\|$	t_{sign}	t_{verify}	$\|sig\|$	t_{sign}	t_{verify}
RYDE L1	6.0	23.4	20.1	7.4	5.4	4.4
MIRA L1	5.6	46.8	43.9	7.4	37.4	36.7
RYDE L3	12.9	49.6	44.8	16.4	12.2	10.7
MIRA L3	11.8	119.7	116.2	15.5	107.2	107.0
RYDE L5	22.8	105.5	94.9	29.1	26.0	22.7
MIRA L5	20.8	337.7	331.4	27.7	322.3	323.2

All public keys are smaller than 200 bytes.

Isochronous implementations
Size in kilobytes, timing in Mcycles
@2.60GHz: 1 millisecond ≈ 2.6 Mcycles

Advantages and limitations

- <u>Limitations</u>
  - Relatively ***slow*** *(few milliseconds)*
    - Greedy use of symmetric cryptography
  - Relatively ***large*** signatures (*5.5-7.5 KB for L1*)
  - Signature size: ***quadratic*** growth in the security level

Advantages and limitations

- Limitations
  - Relatively ***slow*** *(few milliseconds)*
    - Greedy use of symmetric cryptography
  - Relatively ***large*** signatures (*5.5-7.5 KB for L1*)
  - Signature size: ***quadratic*** growth in the security level

- Advantages
  - ***Conservative*** hardness assumption:
    - No structure, no trapdoor
  - ***Small*** (public) keys
  - ***Good*** public key + signature size
  - Adaptive and ***tunable*** parameters

Rank Metric in the Head

RYDE

N. Aragon, M. Bardet, L. Bidoux, J.-J. Chi-Domínguez, V. Dyseryn, T. Feneuil, P. Gaborit, A. Joux, M. Rivain, J.-P. Tillich, A. Vincotte

`https://pqc-ryde.org`

MIRA

N. Aragon, M. Bardet, L. Bidoux, J.-J. Chi-Domínguez, V. Dyseryn, T. Feneuil, P. Gaborit, R. Neveu, M. Rivain, J.-P. Tillich

`https://pqc-mira.org`

Thank you for your attention.