Post-Quantum Signatures from Secure Multiparty Computation

Thibauld Feneuil

Winter Research School

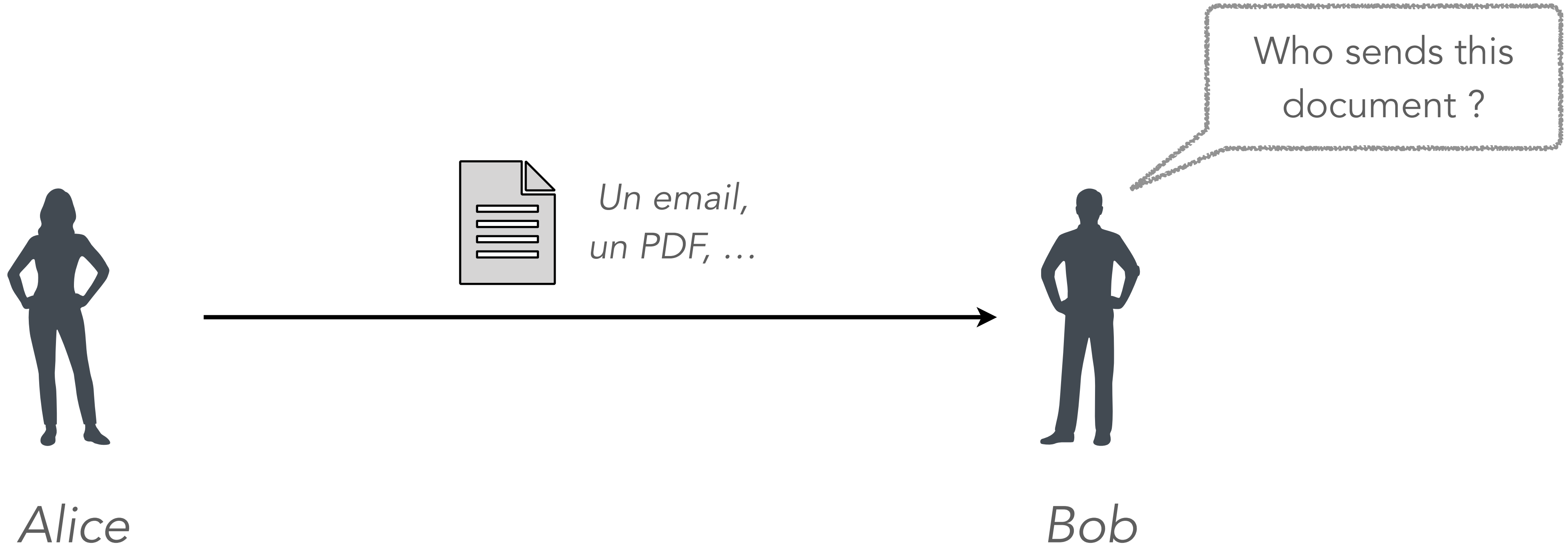
February 20, 2024 — Rennes (France)

CryptoExperts
WE INNOVATE TO SECURE YOUR BUSINESS

Table of Contents

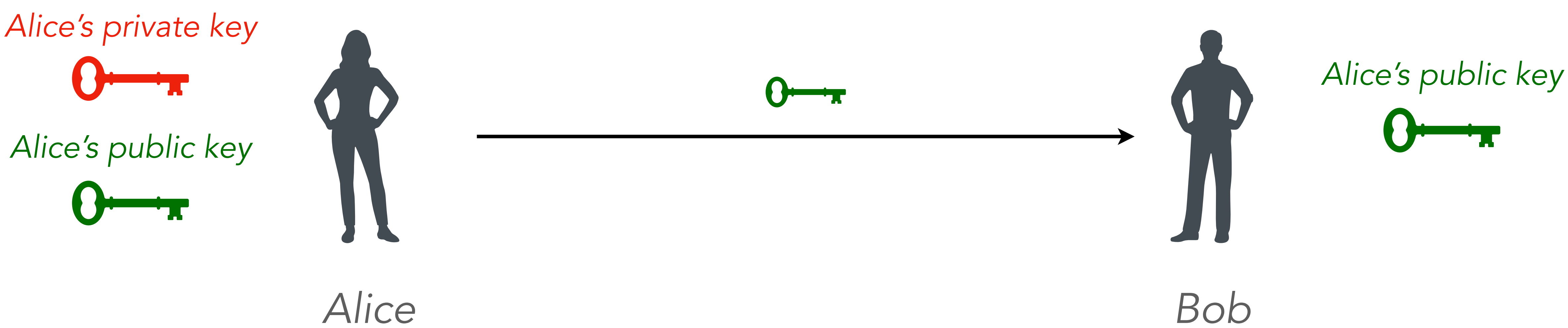
Introduction

Digital signatures



Digital signatures

Digital signatures

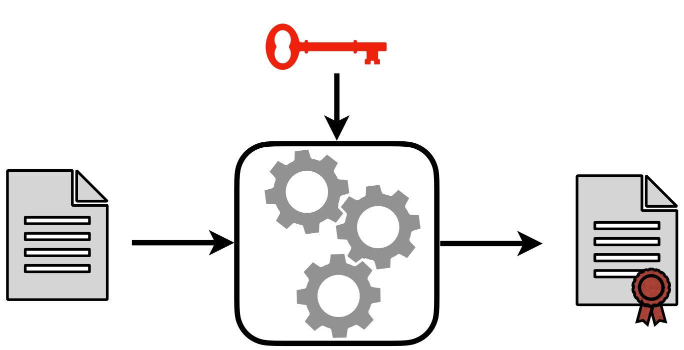


Digital signatures

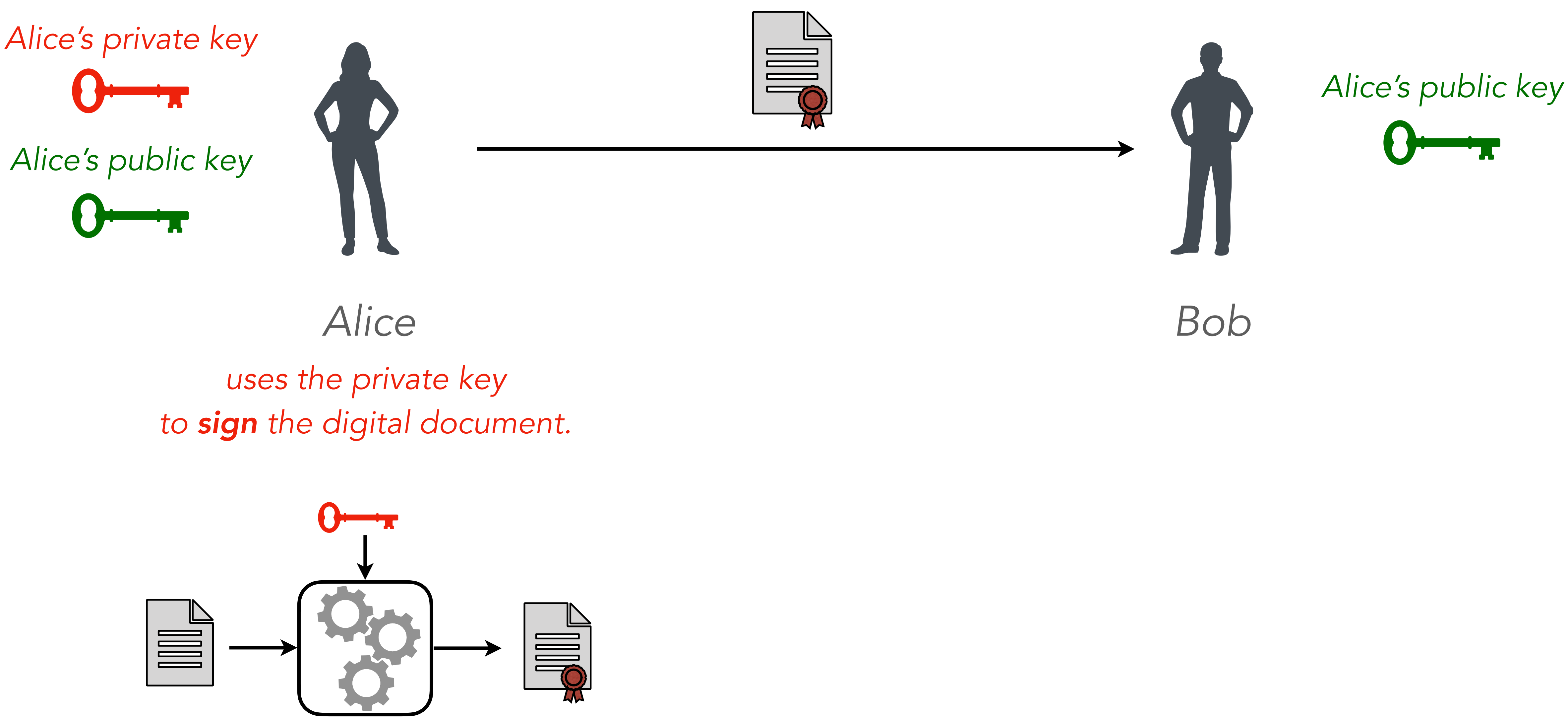
Alice

Bob

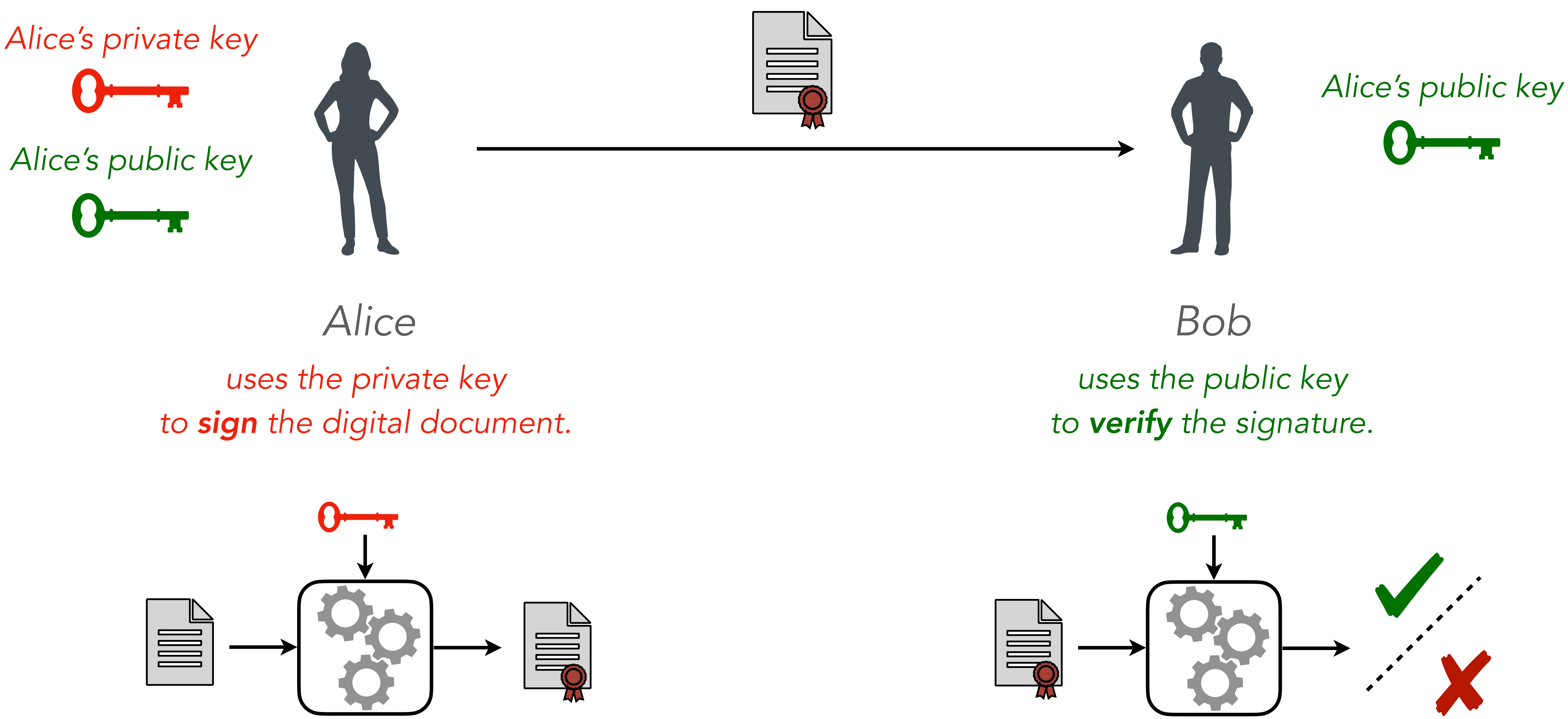
uses the private key
to ***sign*** *the digital document.*



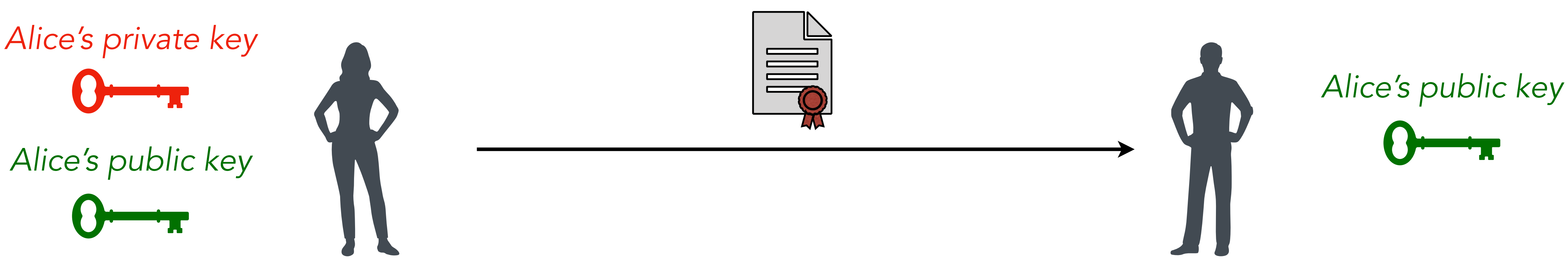
Digital signatures



Digital signatures



Digital signatures



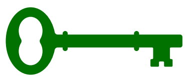
Alice
uses the private key
to ***sign*** *the digital document.*

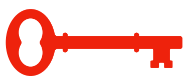
Bob
uses the public key
to ***verify*** *the signature.*

Security Notion: Should be ***impossible*** *to forge a valid signature* ***without*** *the corresponding private key.*

Digital signatures

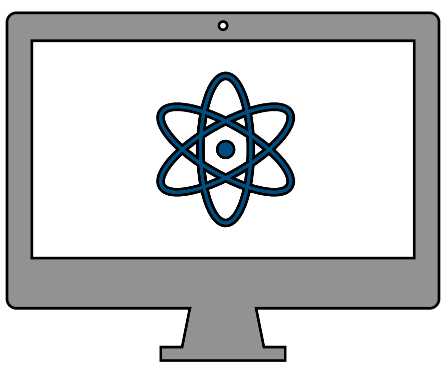
Example


A problem which is very hard to solve


The solution of the above problem

Given N, find non-trivial (p, q) such that $N = pq$.

(p, q)



*Existing signature schemes will be **broken** by the future quantum computers.*

*<u>Problematic</u>: build new signature schemes which would be **secure** even **against quantum computers**.*

How to build signature schemes?

Hash & Sign

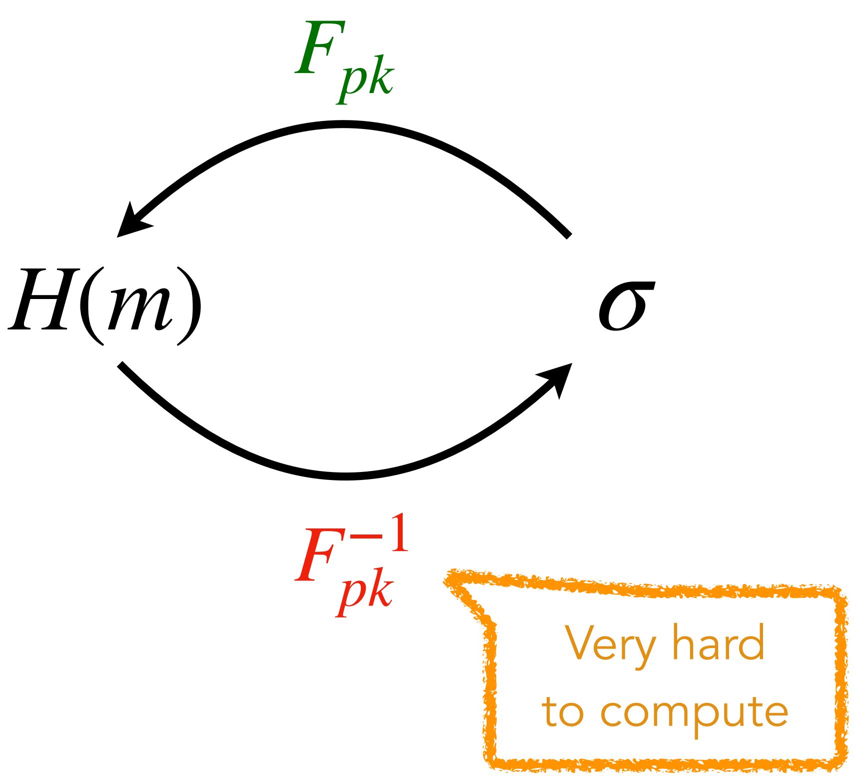
$$\begin{tikzcd} H(m) \arrow[r, bend right, "F_{pk}^{-1}"'] & \sigma \arrow[l, bend right, "F_{pk}"'] \end{tikzcd}$$

Very hard to compute

- Short signatures
- “Trapdoor” in the public key

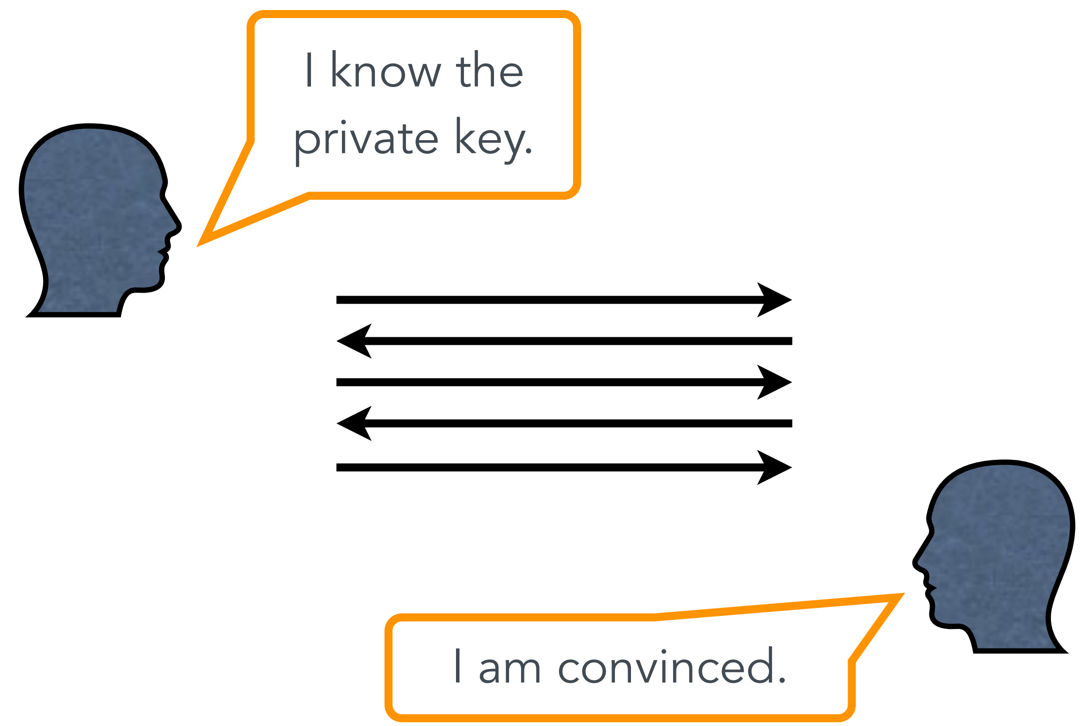
How to build signature schemes?

Hash & Sign



- Short signatures
- “Trapdoor” in the public key

From an identification scheme



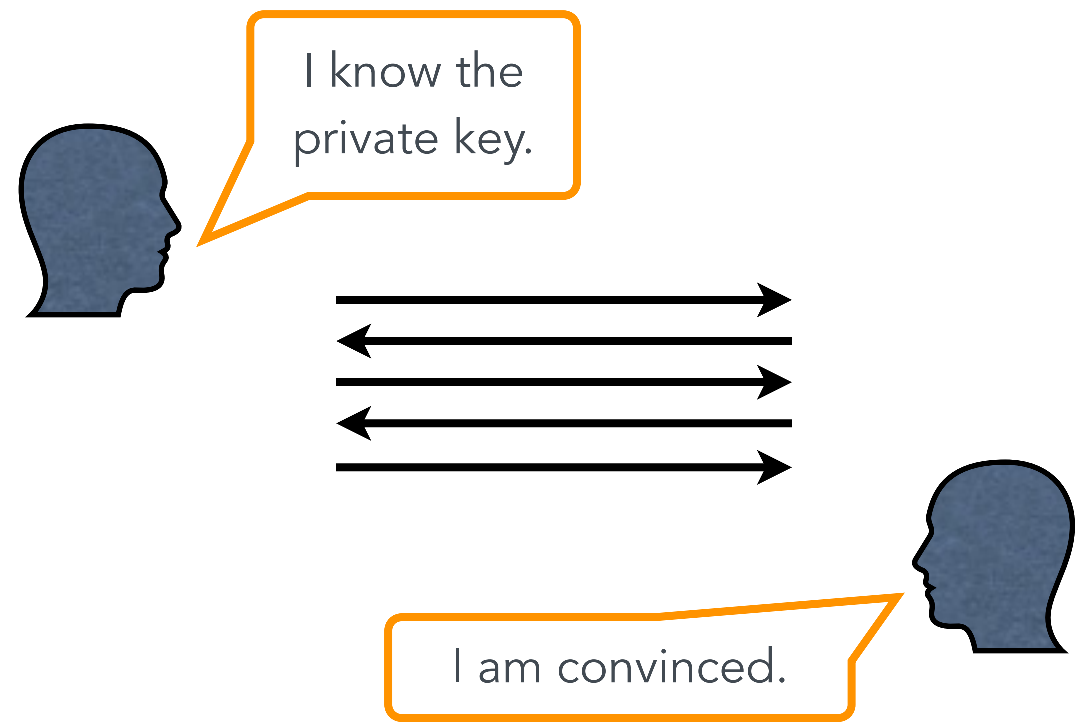
- Large(r) signatures
- Short public key

How to build signature schemes?

Hash & Sign

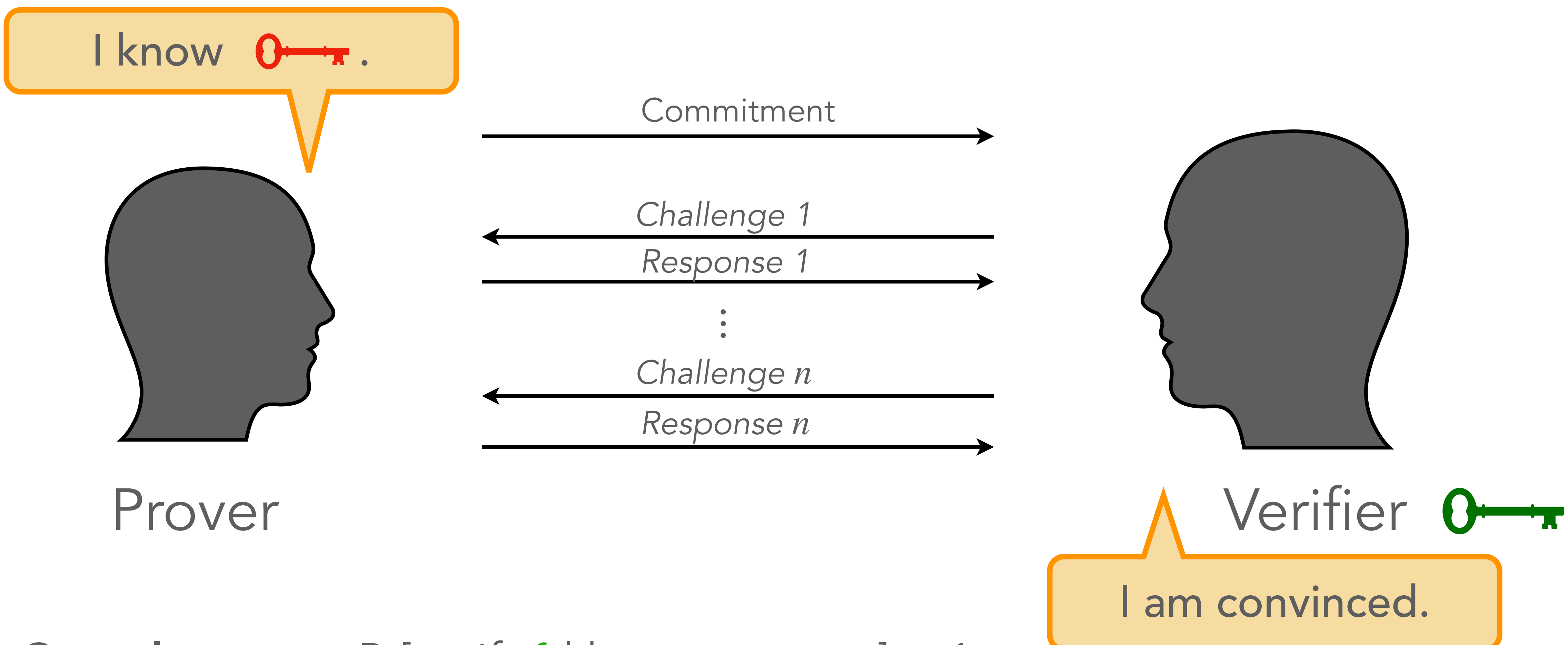
- Short signatures
- "Trapdoor" in the public key

From an identification scheme



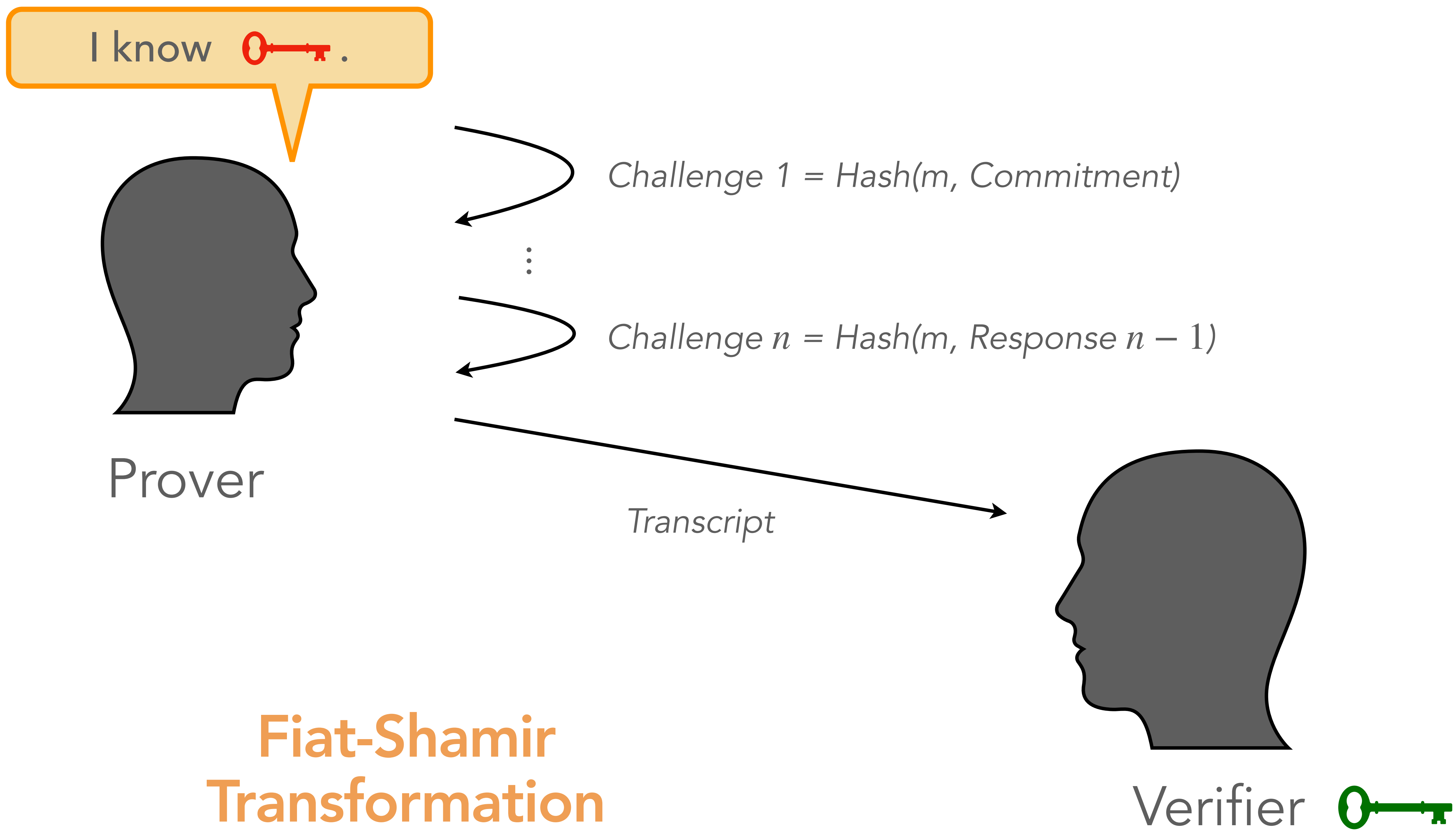
- Large(r) signatures
- Short public key

Identification Scheme



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

Identification Scheme



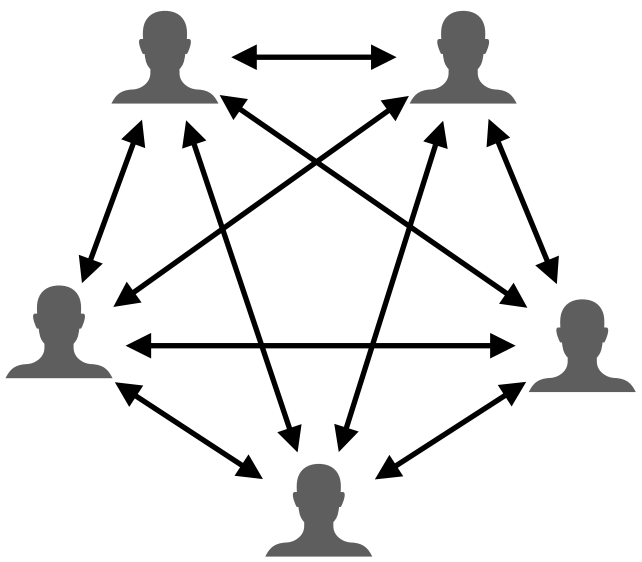
Fiat-Shamir Transformation

m: message to sign

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 an identification scheme



Multiparty Computation (MPC)

(t, N)-threshold Secret Sharing Scheme:

$$s$$

Share the secret s into N parts

$$[\![s]\!]_1 \quad [\![s]\!]_2 \quad [\![s]\!]_3 \quad \cdots \quad [\![s]\!]_N$$

- **Privacy:** Revealing $t-1$ shares leak no information about the secret s
- **Reconstruction:** The secret can be restored from any t shares.

Multiparty Computation (MPC)

Additive Sharing Scheme (modulo p):

- Sample $[\![s]\!]_1, \ldots, [\![s]\!]_{N-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_N$ as

$$[\![s]\!]_N = s - [\![s]\!]_1 - \ldots - [\![s]\!]_{N-1} \pmod{p}.$$

Revealing $N-1$ shares leaks no information about the secret s.

Multiparty Computation (MPC)

Additive Sharing Scheme (modulo p):

- Sample $[\![s]\!]_1, \ldots, [\![s]\!]_{N-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_N$ as

$$[\![s]\!]_N = s - [\![s]\!]_1 - \ldots - [\![s]\!]_{N-1} \pmod{p}.$$

Revealing $N-1$ shares leaks no information about the secret s.

Example: I want to share 835 (modulo 1021) into 5 parts.

$[\![s]\!]_1 = ?$ $[\![s]\!]_2 = ?$ $[\![s]\!]_3 = ?$ $[\![s]\!]_4 = ?$ $[\![s]\!]_5 = ?$

Multiparty Computation (MPC)

Additive Sharing Scheme (modulo p):

- Sample $[\![s]\!]_1, \ldots, [\![s]\!]_{N-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_N$ as

$$[\![s]\!]_N = s - [\![s]\!]_1 - \ldots - [\![s]\!]_{N-1} \pmod{p}.$$

Revealing $N-1$ shares leaks no information about the secret s.

Example: I want to share 835 (modulo 1021) into 5 parts.

$[\![s]\!]_1 = 325$ $[\![s]\!]_2 = 393$ $[\![s]\!]_3 = 847$ $[\![s]\!]_4 = 752$ $[\![s]\!]_5 = ?$

Multiparty Computation (MPC)

Additive Sharing Scheme (modulo p):

- Sample $[\![s]\!]_1, \ldots, [\![s]\!]_{N-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_N$ as

$$[\![s]\!]_N = s - [\![s]\!]_1 - \ldots - [\![s]\!]_{N-1} \pmod{p}.$$

Revealing $N-1$ shares leaks no information about the secret s.

Example: I want to share 835 (modulo 1021) into 5 parts.

$[\![s]\!]_1 = 325$ $[\![s]\!]_2 = 393$ $[\![s]\!]_3 = 847$ $[\![s]\!]_4 = 752$ $[\![s]\!]_5 = 560$

$= 835 - 325 - 393 - 847 - 752$

Multiparty Computation (MPC)

Additive Sharing Scheme (modulo p):

- Sample $[\![s]\!]_1, \ldots, [\![s]\!]_{N-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_N$ as

$$[\![s]\!]_N = s - [\![s]\!]_1 - \ldots - [\![s]\!]_{N-1} \pmod{p}.$$

Revealing $N-1$ shares leaks no information about the secret s.

Example: I want to share $?$ (modulo 1021) into 5 parts.

$[\![s]\!]_1 = 429$ $\quad$ $[\![s]\!]_2 = 19$ $\quad$ $[\![s]\!]_3 = 583$ $\quad$ $[\![s]\!]_4 = ?$ $\quad$ $[\![s]\!]_5 = 822$

Multiparty Computation (MPC)

Additive Sharing Scheme (modulo p):

- Sample $[\![s]\!]_1, \ldots, [\![s]\!]_{N-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_N$ as

$$[\![s]\!]_N = s - [\![s]\!]_1 - \ldots - [\![s]\!]_{N-1} \pmod{p}.$$

Revealing $N-1$ shares leaks no information about the secret s.

Example: I want to share $?$ (modulo 1021) into 5 parts.

$[\![s]\!]_1 = 429$ $\quad$ $[\![s]\!]_2 = 19$ $\quad$ $[\![s]\!]_3 = 583$ $\quad$ $[\![s]\!]_4 = ?$ $\quad$ $[\![s]\!]_5 = 822$

Impossible to deduce the shared value!

Multiparty Computation (MPC)

Additive Sharing Scheme (modulo p):

- Sample $[\![s]\!]_1, \ldots, [\![s]\!]_{N-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_N$ as

$$[\![s]\!]_N = s - [\![s]\!]_1 - \ldots - [\![s]\!]_{N-1} \pmod{p}.$$

Revealing $N-1$ shares leaks no information about the secret s.

Example: I want to share ? (modulo 1021) into 5 parts.

$[\![s]\!]_1 = 429$ $[\![s]\!]_2 = 19$ $[\![s]\!]_3 = 583$ $[\![s]\!]_4 = 231$ $[\![s]\!]_5 = 822$

$$s = [\![s]\!]_1 + \ldots + [\![s]\!]_N = 42$$

Multiparty Computation (MPC)

Shamir's Sharing Scheme (modulo p):

- Sample $r_1, \ldots, r_{t-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_1, \ldots, [\![s]\!]_N$ as

$$\forall i \in \{1, \ldots, N\},\ [\![s]\!]_i = P(i)$$

where $P(X) := s + \sum_{j=1}^{t-1} r_j \cdot X^j$.

Revealing $t-1$ shares leaks no information about the secret s.

Revealing t shares enables to restore the secret s.

Multiparty Computation (MPC)

Shamir's Sharing Scheme (modulo p):

- Sample $r_1, \ldots, r_{t-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_1, \ldots, [\![s]\!]_N$ as

$$\forall i \in \{1, \ldots, N\},\ [\![s]\!]_i = P(i)$$

where $P(X) := s + \sum_{j=1}^{t-1} r_j \cdot X^j$.

<u>Example</u>: I want to share 835 (modulo 1021) into 5 parts, which $t = 3$.

$r_1 = ?$

$r_2 = ?$

$P = ?$

$[\![s]\!]_1 = P(1) = ?$

$[\![s]\!]_2 = P(2) = ?$

$[\![s]\!]_3 = P(3) = ?$

$[\![s]\!]_4 = P(4) = ?$

$[\![s]\!]_5 = P(5) = ?$

Multiparty Computation (MPC)

Shamir's Sharing Scheme (modulo p):

- Sample $r_1, \ldots, r_{t-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_1, \ldots, [\![s]\!]_N$ as

$$\forall i \in \{1, \ldots, N\},\ [\![s]\!]_i = P(i)$$

where $P(X) := s + \sum_{j=1}^{t-1} r_j \cdot X^j$.

Example: I want to share 835 (modulo 1021) into 5 parts, which $t = 3$.

$$r_1 = 644$$
$$r_2 = 943$$
$$P(X) = 835 + 644 \cdot X + 943 \cdot X^2$$

$$[\![s]\!]_1 = P(1) = ?$$
$$[\![s]\!]_2 = P(2) = ?$$
$$[\![s]\!]_3 = P(3) = ?$$

$$[\![s]\!]_4 = P(4) = ?$$
$$[\![s]\!]_5 = P(5) = ?$$

Multiparty Computation (MPC)

Shamir's Sharing Scheme (modulo p):

- Sample $r_1, \ldots, r_{t-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_1, \ldots, [\![s]\!]_N$ as

$$\forall i \in \{1, \ldots, N\},\ [\![s]\!]_i = P(i)$$

where $P(X) := s + \sum_{j=1}^{t-1} r_j \cdot X^j$.

Example: I want to share 835 (modulo 1021) into 5 parts, which $t = 3$.

$$r_1 = 644$$
$$r_2 = 943$$
$$P(X) = 835 + 644 \cdot X + 943 \cdot X^2$$

$$[\![s]\!]_1 = P(1) = 380$$
$$[\![s]\!]_2 = P(2) = 790$$
$$[\![s]\!]_3 = P(3) = 23$$

$$[\![s]\!]_4 = P(4) = 121$$
$$[\![s]\!]_5 = P(5) = 63$$

Multiparty Computation (MPC)

Shamir's Sharing Scheme (modulo p):

- Sample $r_1, \ldots, r_{t-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_1, \ldots, [\![s]\!]_N$ as

$$\forall i \in \{1,\ldots,N\},\ [\![s]\!]_i = P(i)$$

where $P(X) := s + \sum_{j=1}^{t-1} r_j \cdot X^j$.

Example: I want to share $?$ (modulo 1021) into 5 parts, which $t = 3$.

$r_1 = ?$

$r_2 = ?$

$P = ?$

$[\![s]\!]_1 = P(1) = ?$

$[\![s]\!]_2 = P(2) = 63$

$[\![s]\!]_3 = P(3) = ?$

$[\![s]\!]_4 = P(4) = ?$

$[\![s]\!]_5 = P(5) = 311$

Multiparty Computation (MPC)

Shamir's Sharing Scheme (modulo p):

- Sample $r_1, \ldots, r_{t-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_1, \ldots, [\![s]\!]_N$ as

$$\forall i \in \{1,\ldots,N\},\ [\![s]\!]_i = P(i)$$

where $P(X) := s + \sum_{j=1}^{t-1} r_j \cdot X^j$.

Example: I want to share $?$ (modulo 1021) into 5 parts, which $t = 3$.

$$r_1 = ?$$
$$r_2 = ?$$
$$P = ?$$

$$[\![s]\!]_1 = P(1) = ?$$
$$[\![s]\!]_2 = P(2) = 63$$
$$[\![s]\!]_3 = P(3) = ?$$

$$[\![s]\!]_4 = P(4) = ?$$
$$[\![s]\!]_5 = P(5) = 311$$

Impossible to deduce the shared value!

Multiparty Computation (MPC)

Shamir's Sharing Scheme (modulo p):

- Sample $r_1, \ldots, r_{t-1}$ uniformly at random (modulo p)
- Compute $[\![s]\!]_1, \ldots, [\![s]\!]_N$ as

$$\forall i \in \{1,\ldots,N\},\ [\![s]\!]_i = P(i)$$

where $P(X) := s + \sum_{j=1}^{t-1} r_j \cdot X^j$.

<u>Example</u>: I want to share ? (modulo 1021) into 5 parts, which $t = 3$.

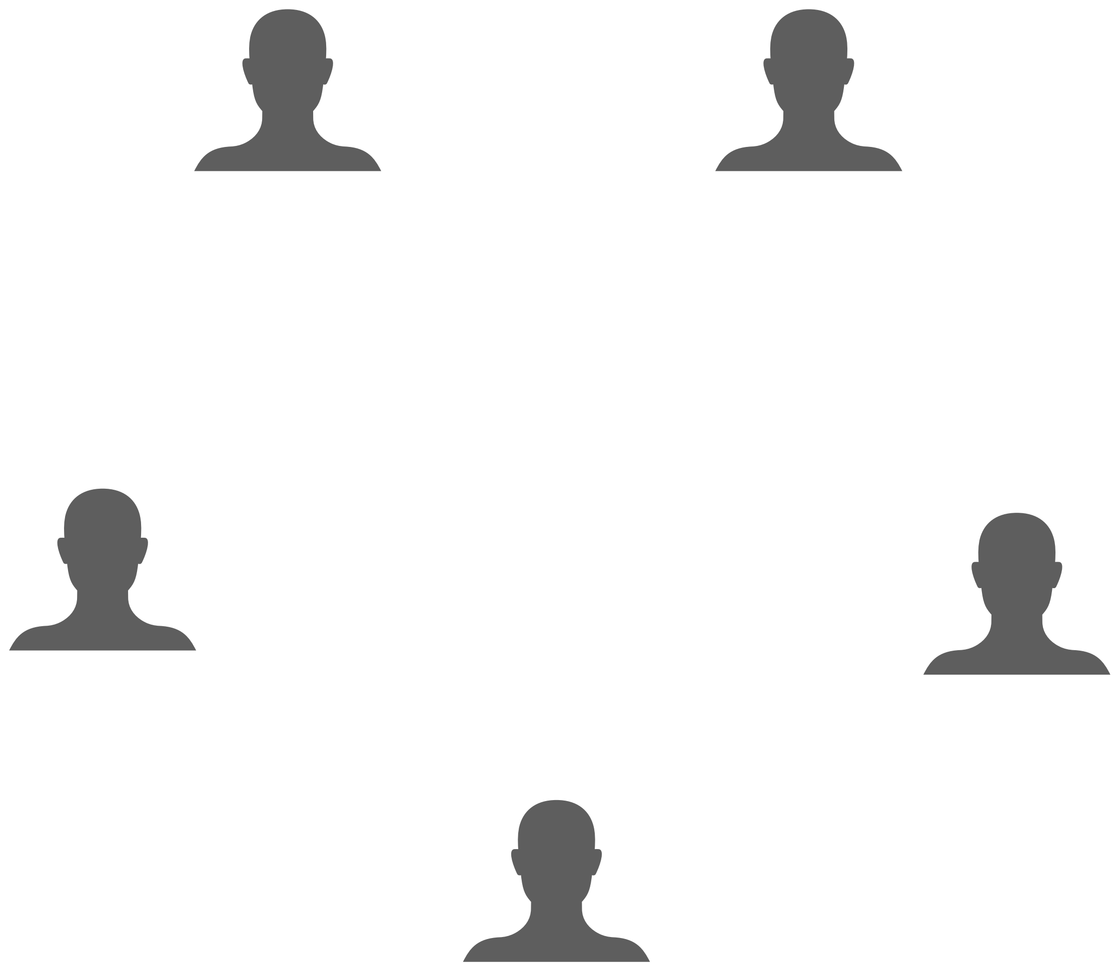
$$r_1 = 574$$
$$r_2 = 416$$
$$P(X) = \boxed{314} + 574 \cdot X + 416 \cdot X^2$$

$$[\![s]\!]_1 = P(1) = ?$$
$$[\![s]\!]_2 = P(2) = 63$$
$$[\![s]\!]_3 = P(3) = 675$$

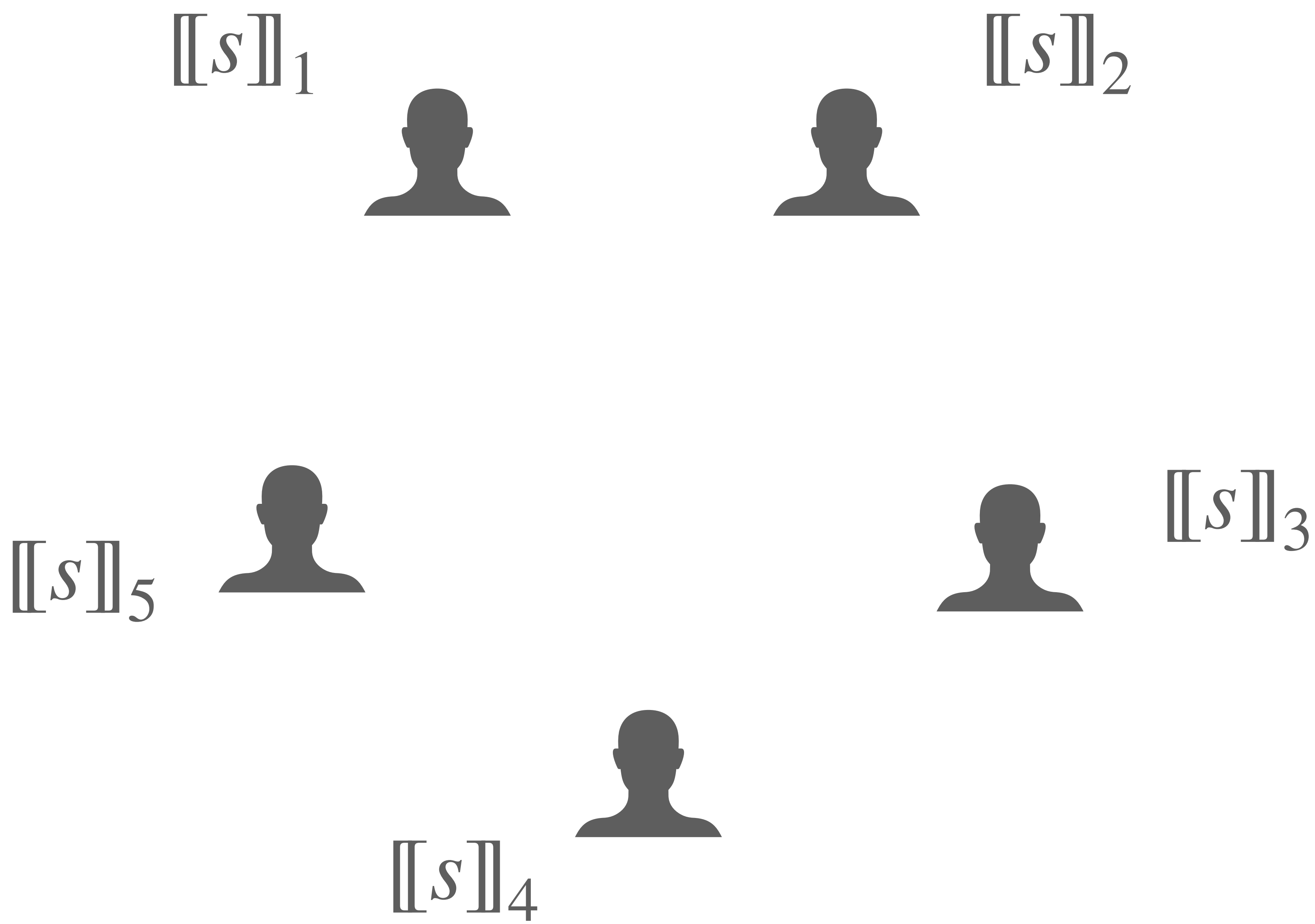
$$[\![s]\!]_4 = P(4) = ?$$
$$[\![s]\!]_5 = P(5) = 311$$

Polynomial interpolation

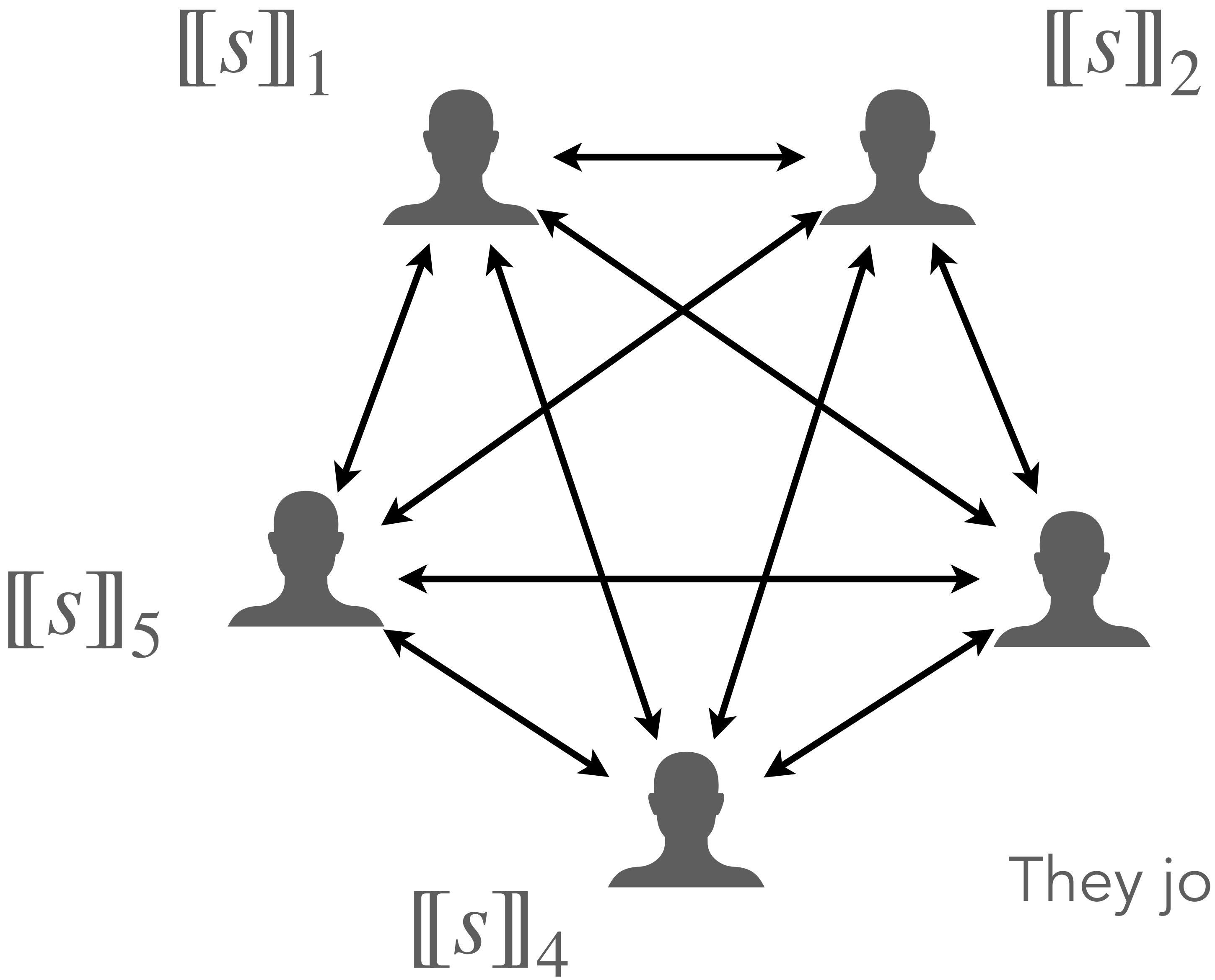
Multiparty Computation (MPC)



Multiparty Computation (MPC)



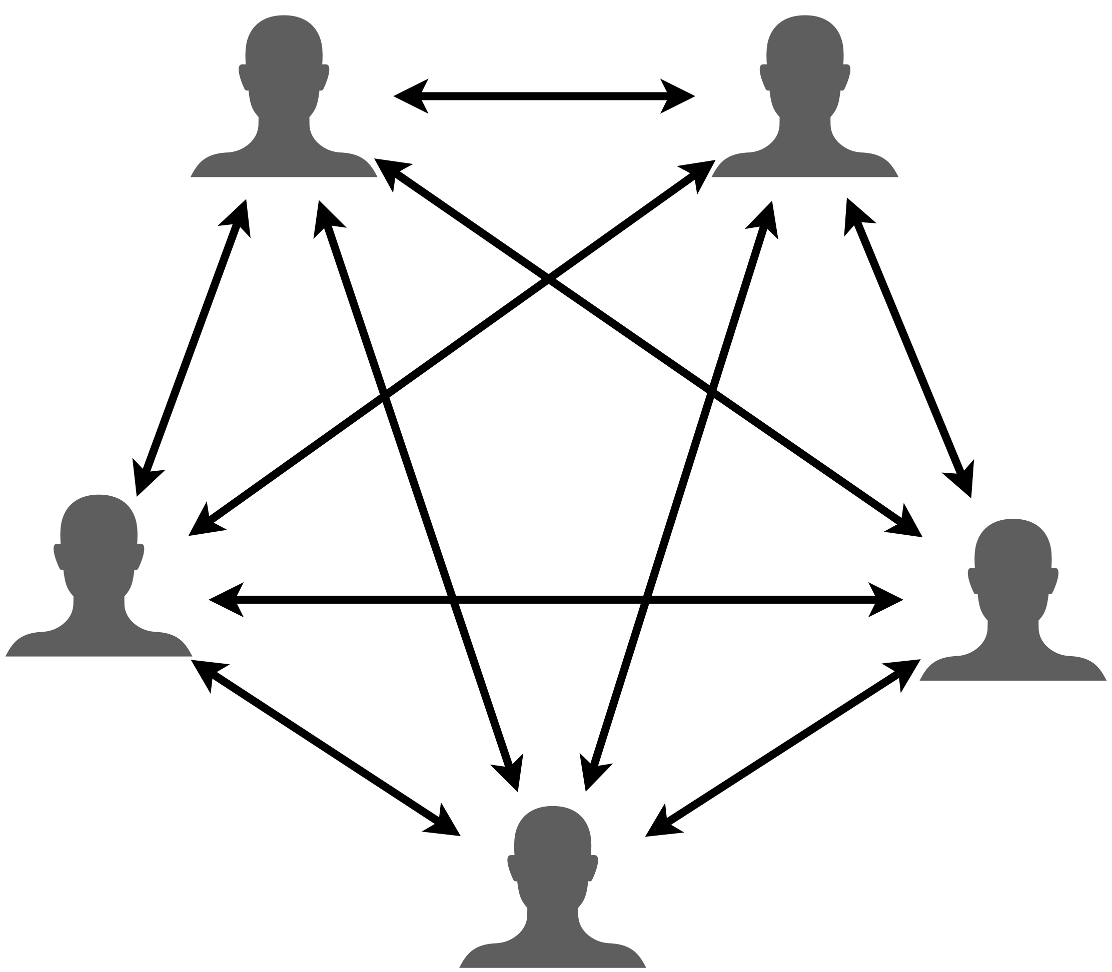
Multiparty Computation (MPC)



They jointly compute

$$y \leftarrow C(s)$$

Multiparty Computation (MPC)

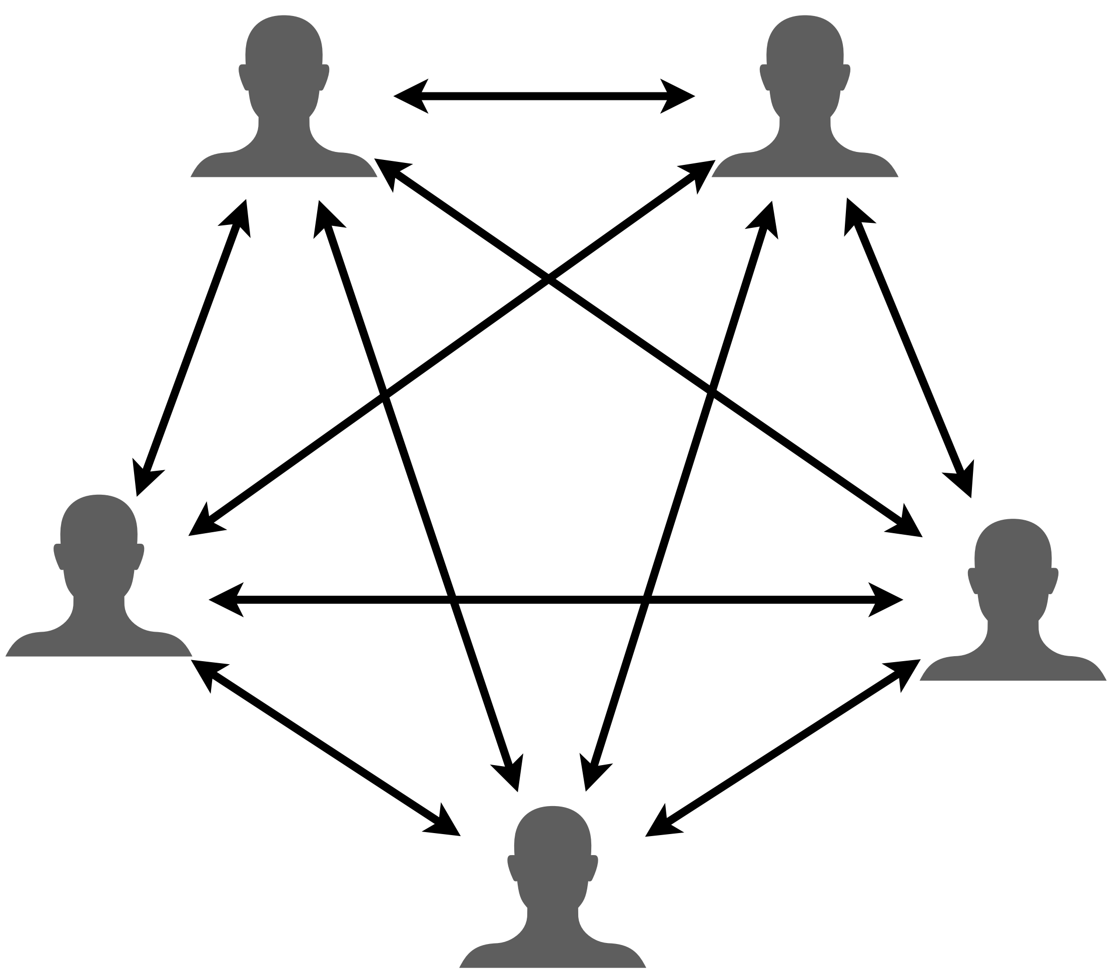


Input: $[\![a]\!]$ and $[\![b]\!]$, a public constant c

- They can compute $[\![a+b]\!]$:

$$[\![a+b]\!]_1 \leftarrow [\![a]\!]_1 + [\![b]\!]_1$$
$$\vdots$$
$$[\![a+b]\!]_N \leftarrow [\![a]\!]_N + [\![b]\!]_N$$

Multiparty Computation (MPC)



Input: $[\![a]\!]$ and $[\![b]\!]$, a public constant c

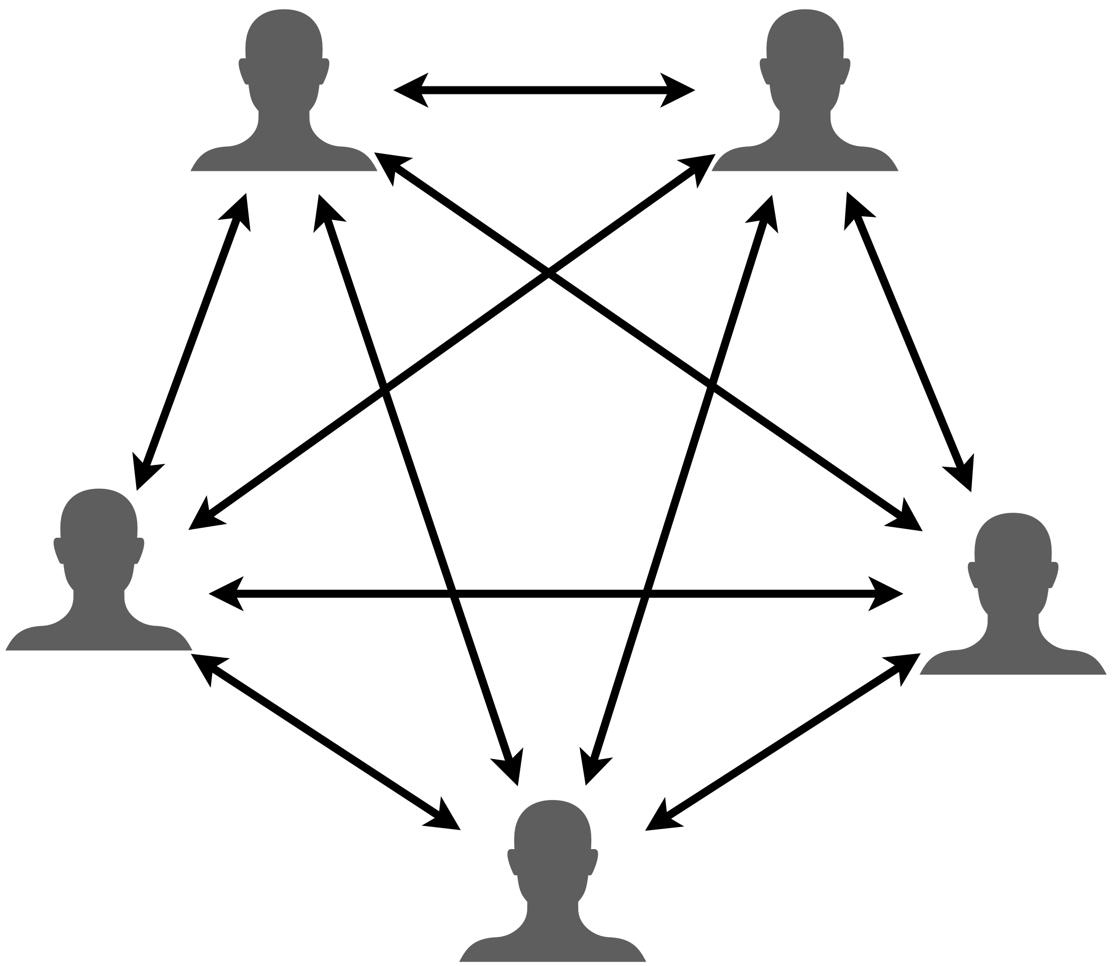
- They can compute $[\![a+b]\!]$:

$$\begin{aligned} [\![a+b]\!]_1 &\leftarrow [\![a]\!]_1 + [\![b]\!]_1 \\ &\vdots \\ [\![a+b]\!]_N &\leftarrow [\![a]\!]_N + [\![b]\!]_N \end{aligned}$$

- They can compute $[\![a+c]\!]$:

$$\begin{aligned} [\![a+c]\!]_1 &\leftarrow [\![a]\!]_1 + c \\ [\![a+c]\!]_2 &\leftarrow [\![a]\!]_2 \\ &\vdots \\ [\![a+c]\!]_N &\leftarrow [\![a]\!]_N \end{aligned}$$

Multiparty Computation (MPC)

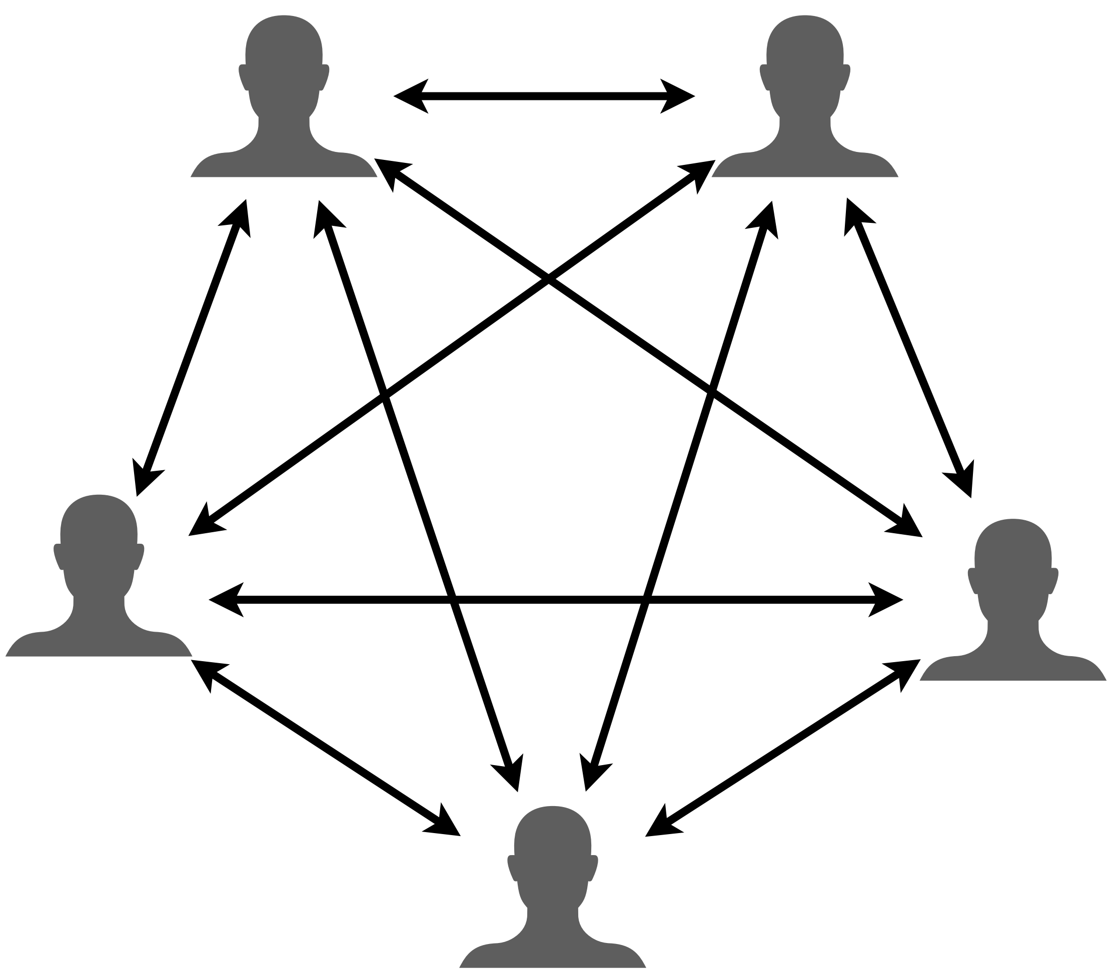


Input: $[\![a]\!]$ and $[\![b]\!]$, a public constant c

- They can compute $[\![c \cdot a]\!]$:

$$[\![c \cdot a]\!]_1 \leftarrow c \cdot [\![a]\!]_1$$
$$\vdots$$
$$[\![c \cdot a]\!]_N \leftarrow c \cdot [\![a]\!]_N$$

Multiparty Computation (MPC)



Input: $[\![a]\!]$ and $[\![b]\!]$, a public constant c

- They can compute $[\![c \cdot a]\!]$:

$$\begin{gathered}[\![c \cdot a]\!]_1 \leftarrow c \cdot [\![a]\!]_1 \\ \vdots \\ [\![c \cdot a]\!]_N \leftarrow c \cdot [\![a]\!]_N\end{gathered}$$

- They can compute $[\![a \cdot b]\!]$…

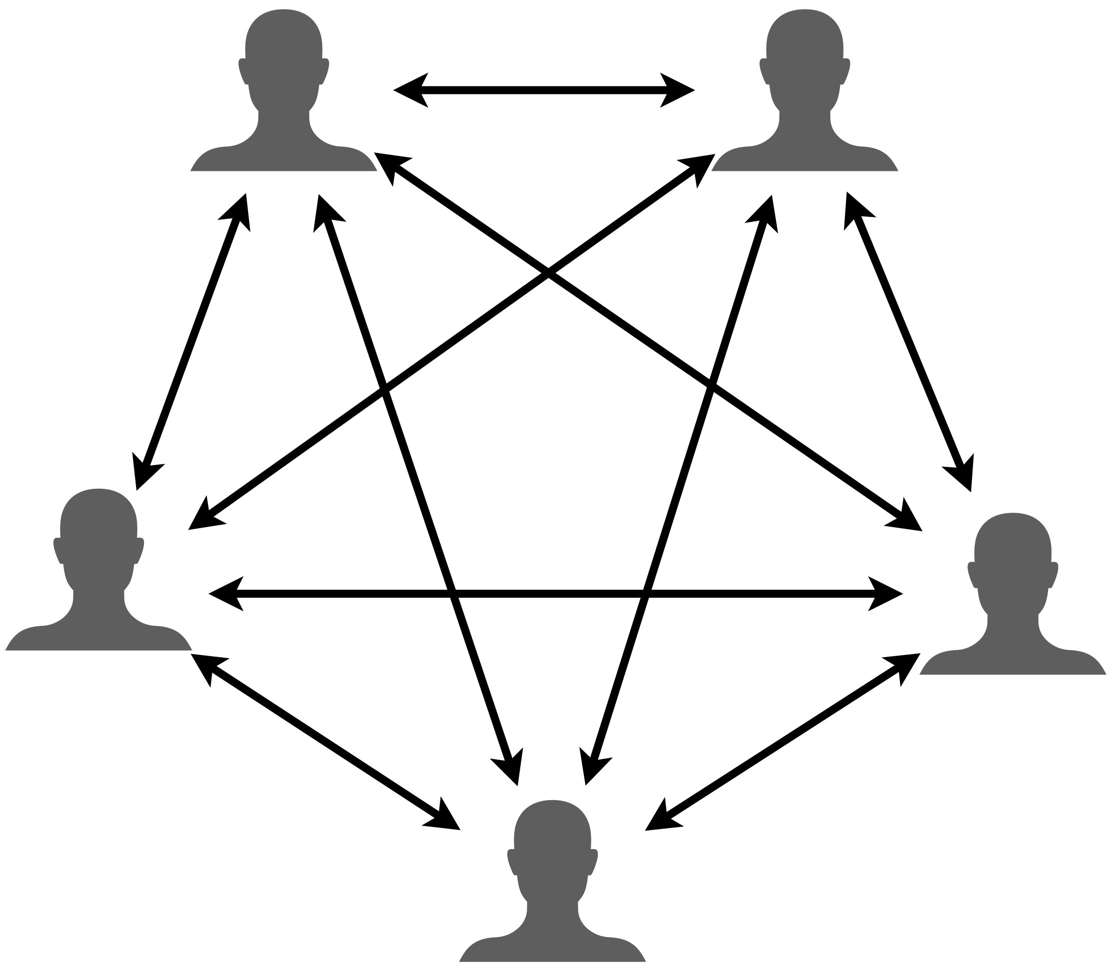
…but it is not trivial.

It requires ***communication*** between the parties.

Multiparty Computation (MPC)

- Given a matrix H and a sharing $[\![x]\!]$ of a vector x, they can compute $[\![Hx]\!]$.

- Given two sharings $[\![A]\!], [\![B]\!]$ of two matrices A and B, they can compute $[\![A \cdot B]\!]$.

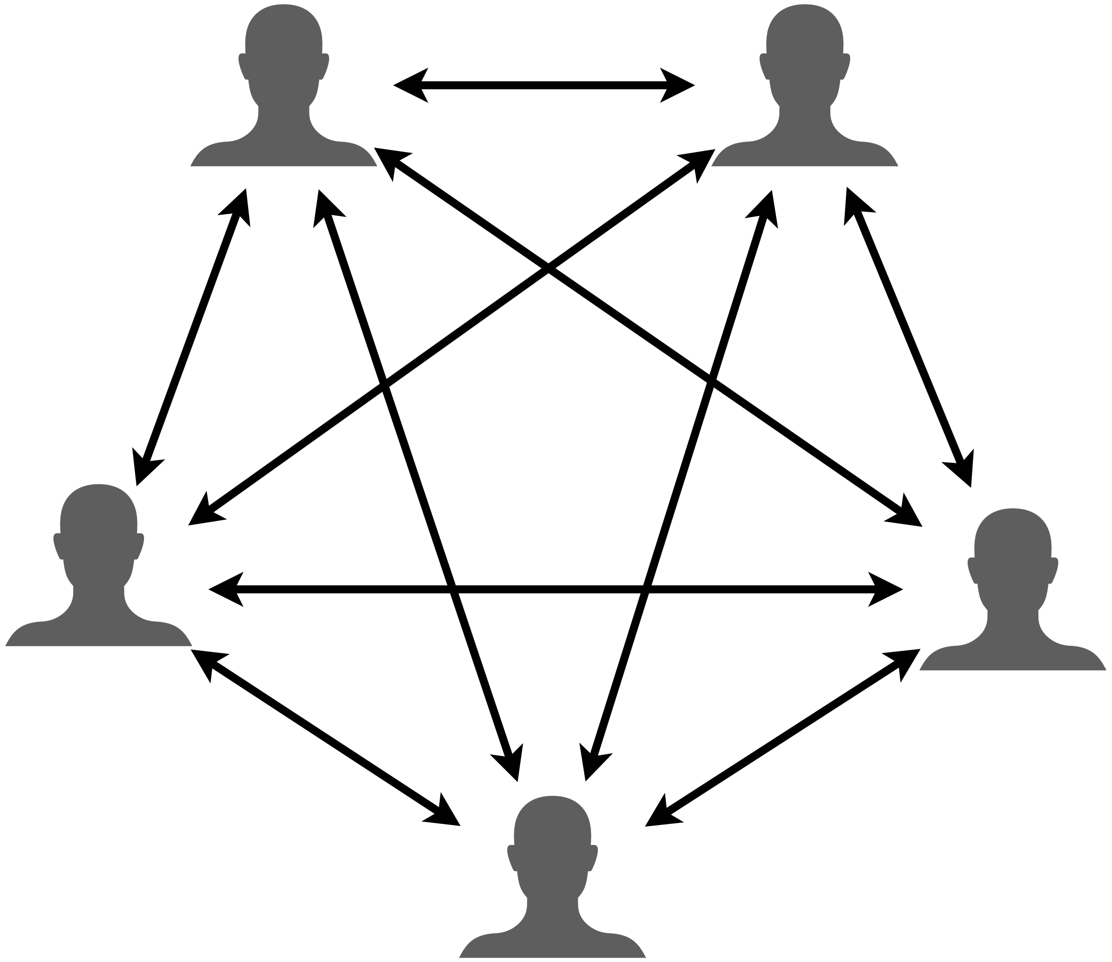


Multiparty Computation (MPC)

- Given a matrix H and a sharing $[\![x]\!]$ of a vector x, they can compute $[\![Hx]\!]$.

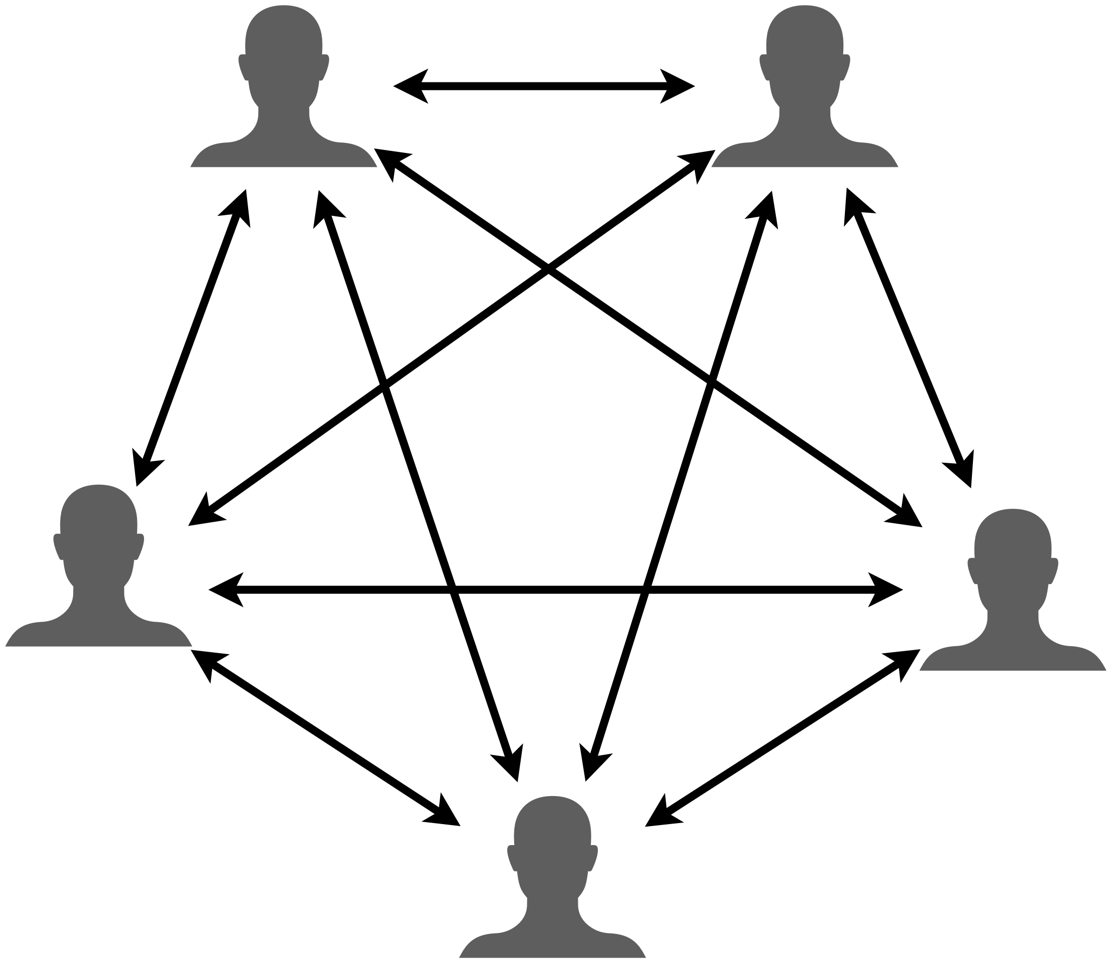
- Given two sharings $[\![A]\!], [\![B]\!]$ of two matrices A and B, they can compute $[\![A \cdot B]\!]$.

- Given a sharing $[\![x]\!]$ of a value x, they can check that $x \in \{0,1\}$ by computing and revealing $[\![x \cdot (x-1)]\!]$.



Multiparty Computation (MPC)

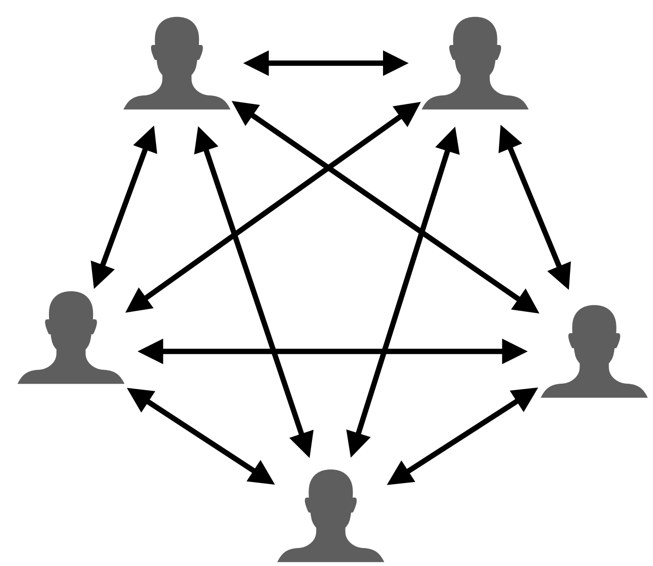
- Given a matrix H and a sharing $[\![x]\!]$ of a vector x, they can compute $[\![Hx]\!]$.
- Given two sharings $[\![A]\!]$, $[\![B]\!]$ of two matrices A and B, they can compute $[\![A \cdot B]\!]$.
- Given a sharing $[\![x]\!]$ of a value x, they can check that $x \in \{0,1\}$ by computing and revealing $[\![x \cdot (x-1)]\!]$.
- Given a sharing $[\![M]\!]$ of a matrix M, they can check that the rank of M is smaller than a public constant r.
- …



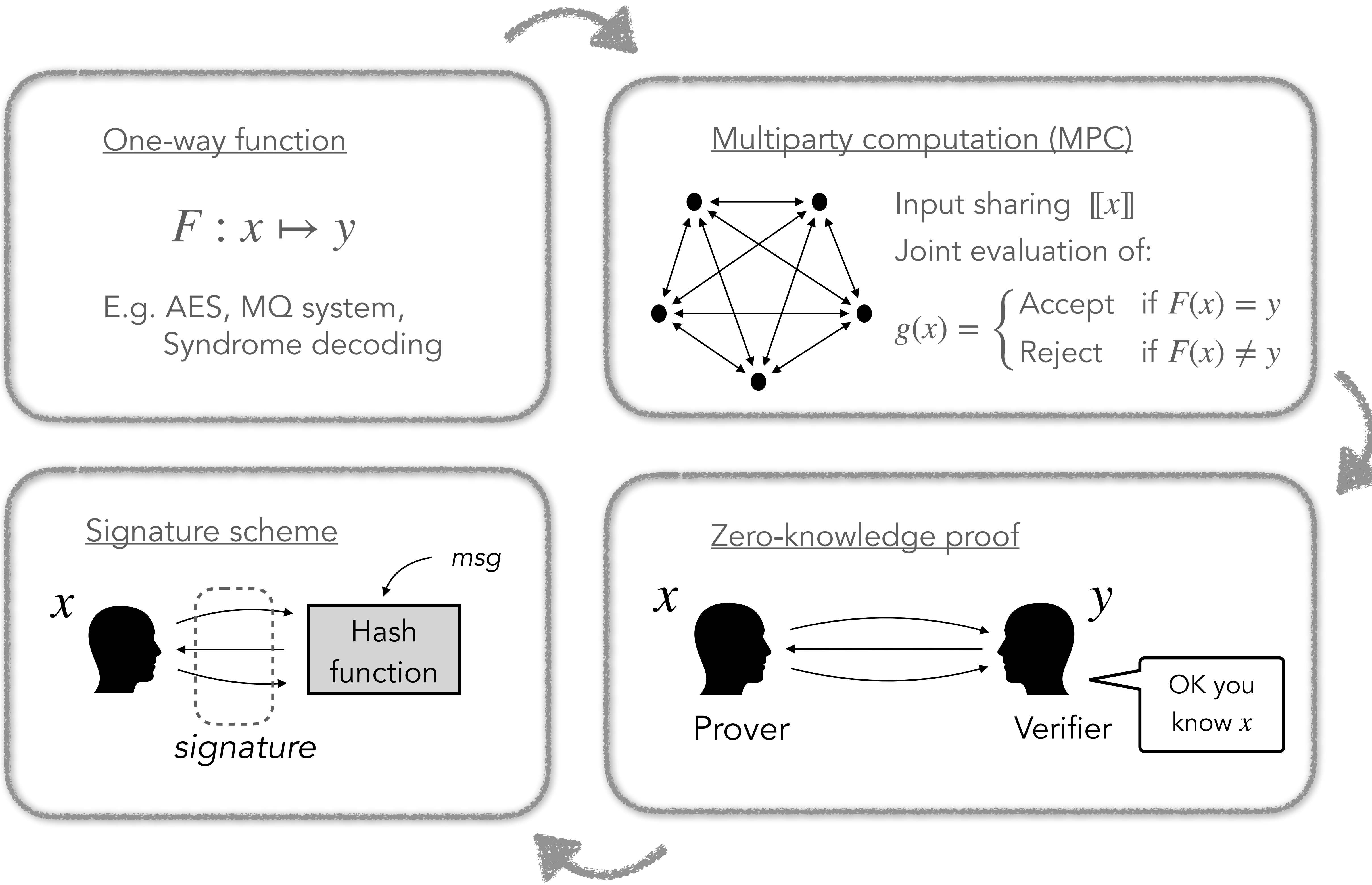
MPC in the Head

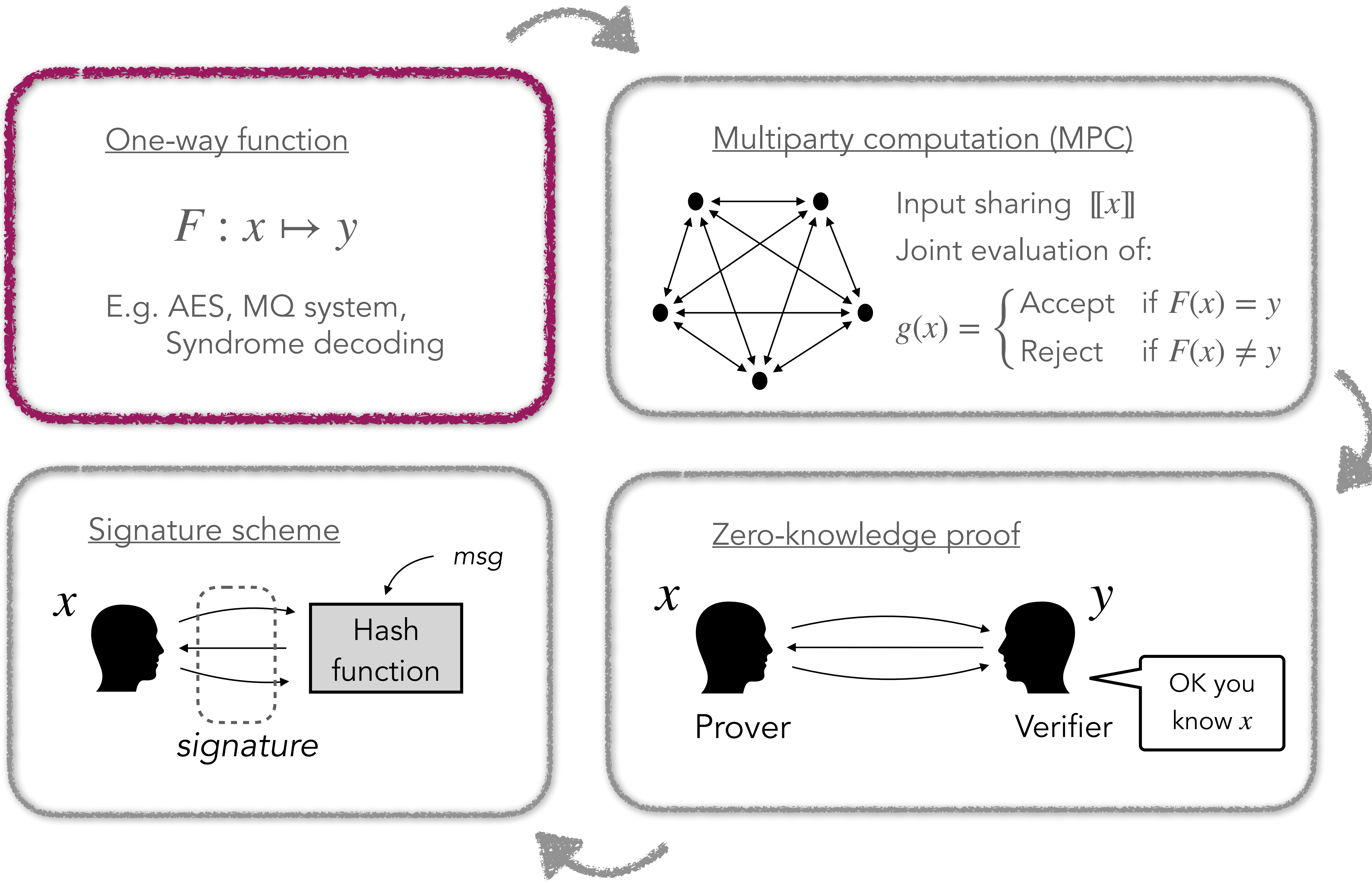
- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)

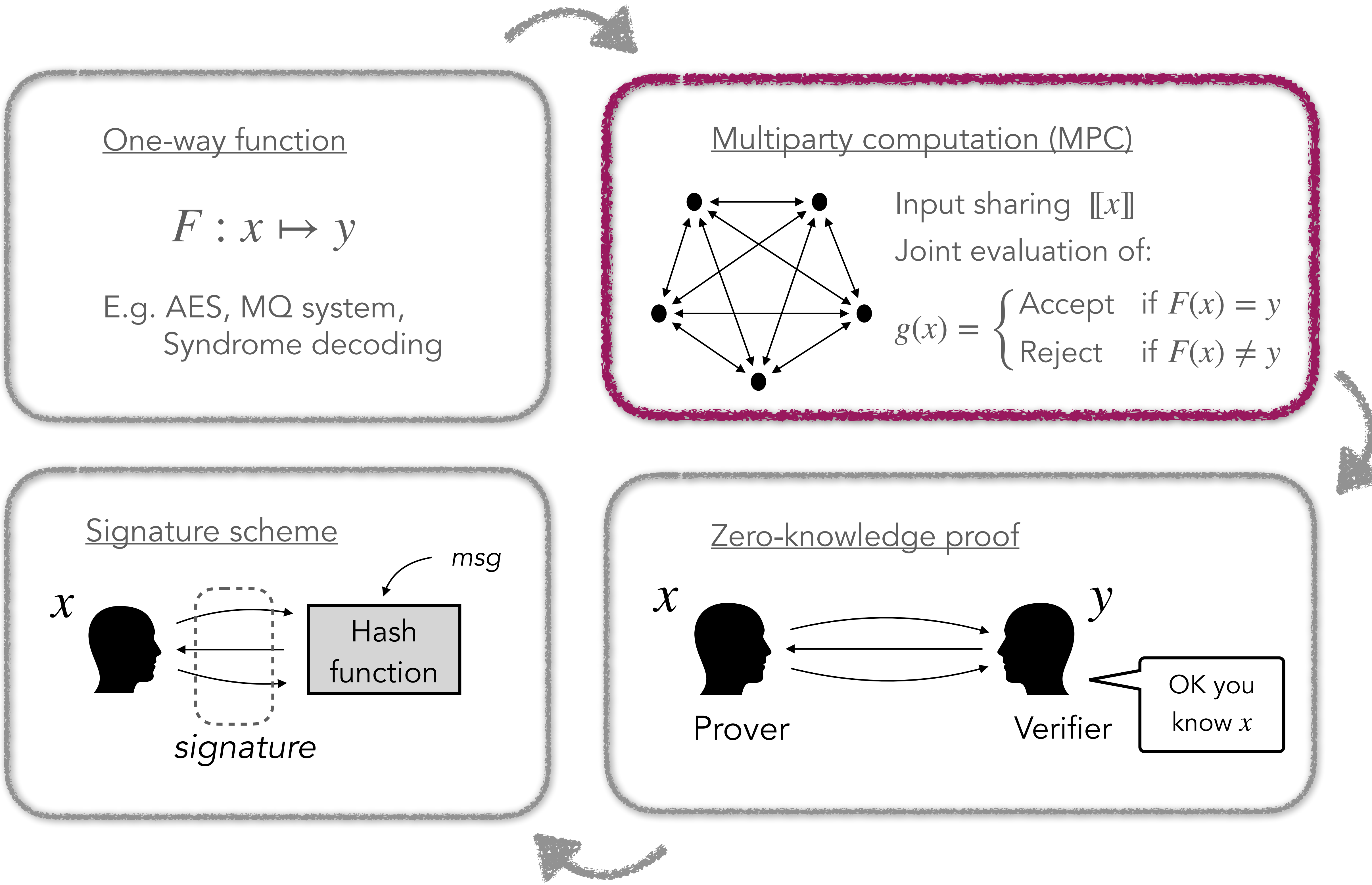
- Turn a *multiparty computation* (MPC) into an identification scheme



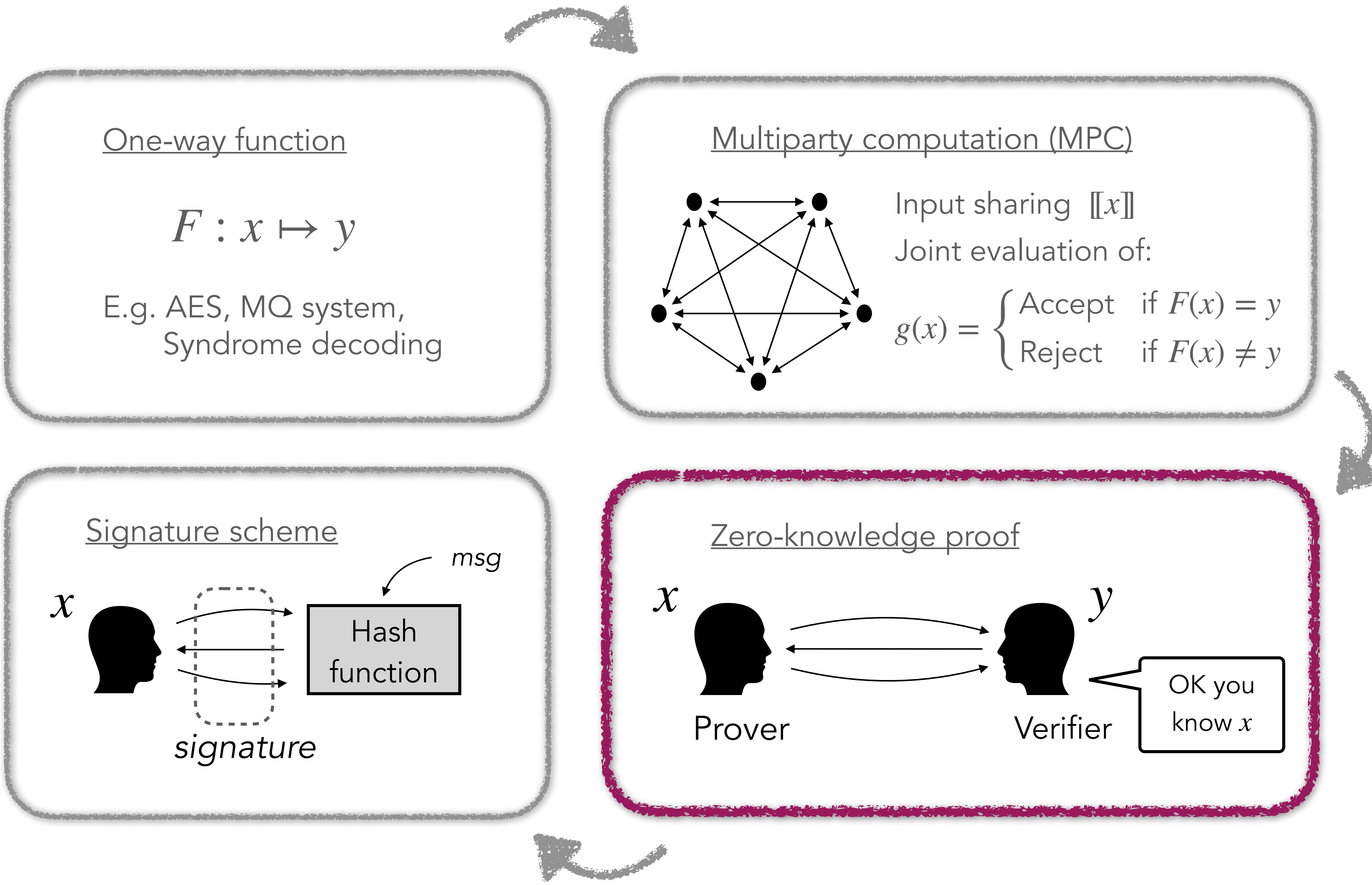
- **Generic**: can be apply to any cryptographic problem


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

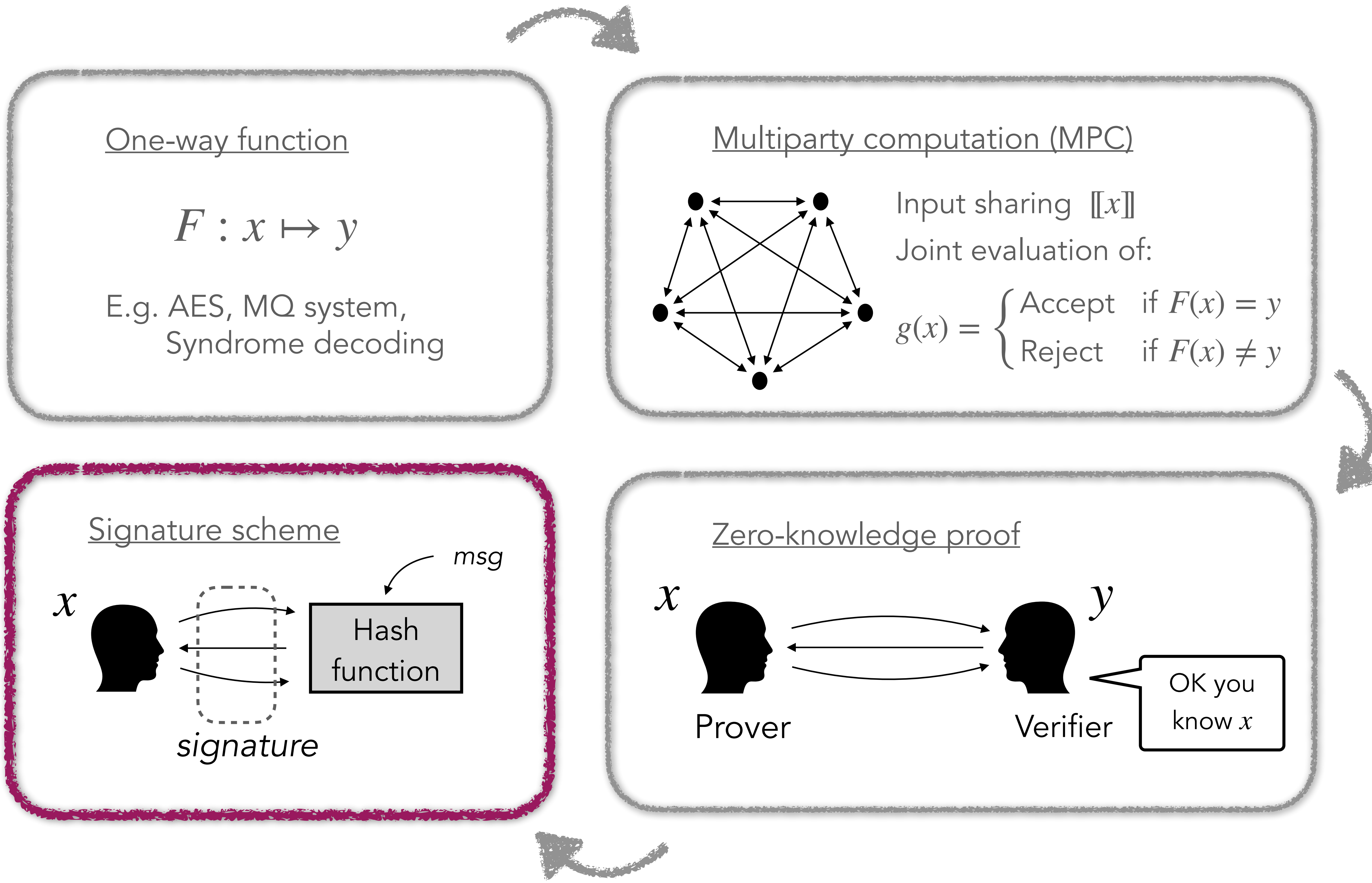

One-way function
$F : x \mapsto y$
E.g. AES, MQ system, Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Signature scheme
x
msg
Hash function
signature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

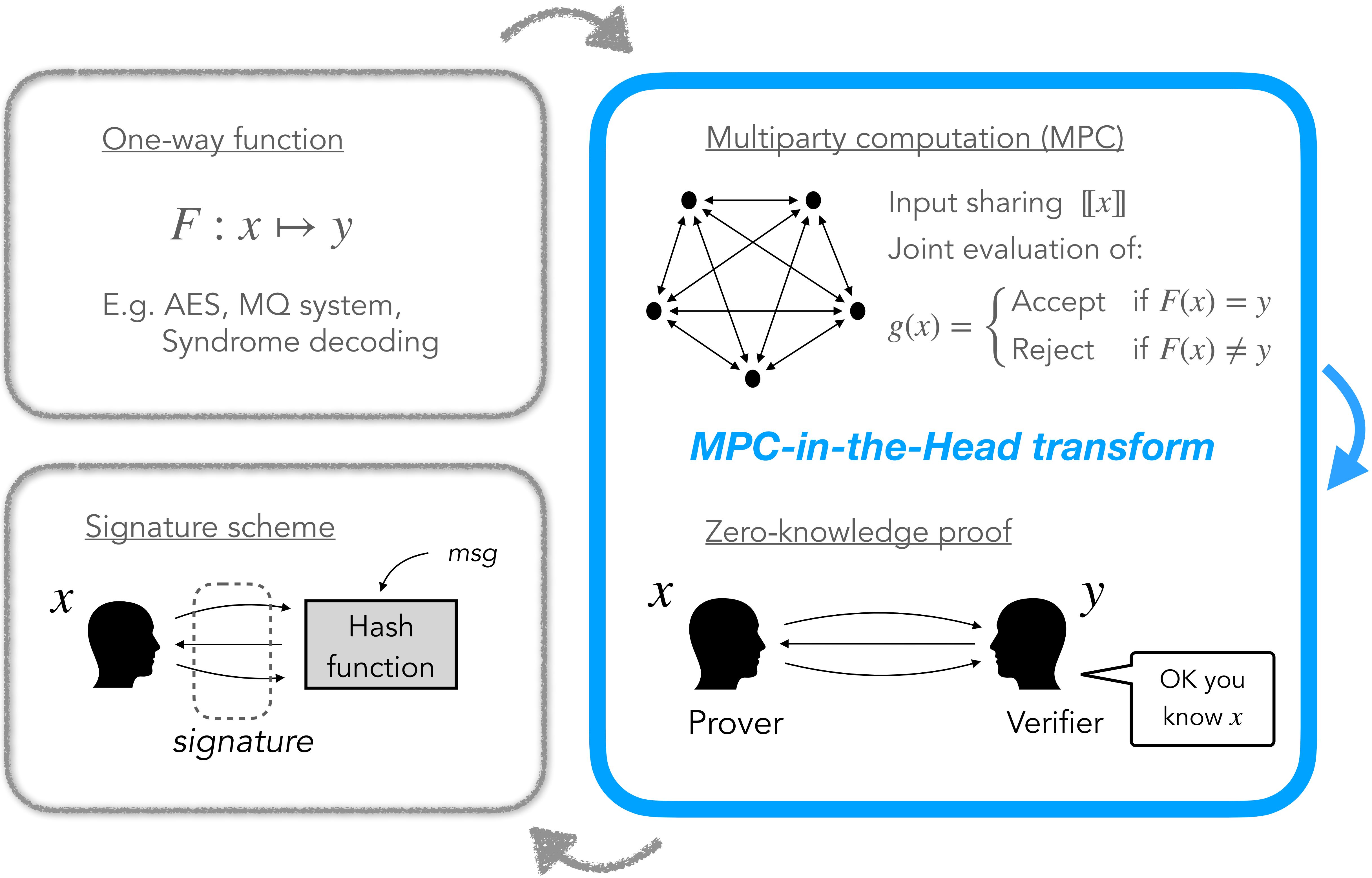


One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Signature scheme
x
msg
Hash function
signature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

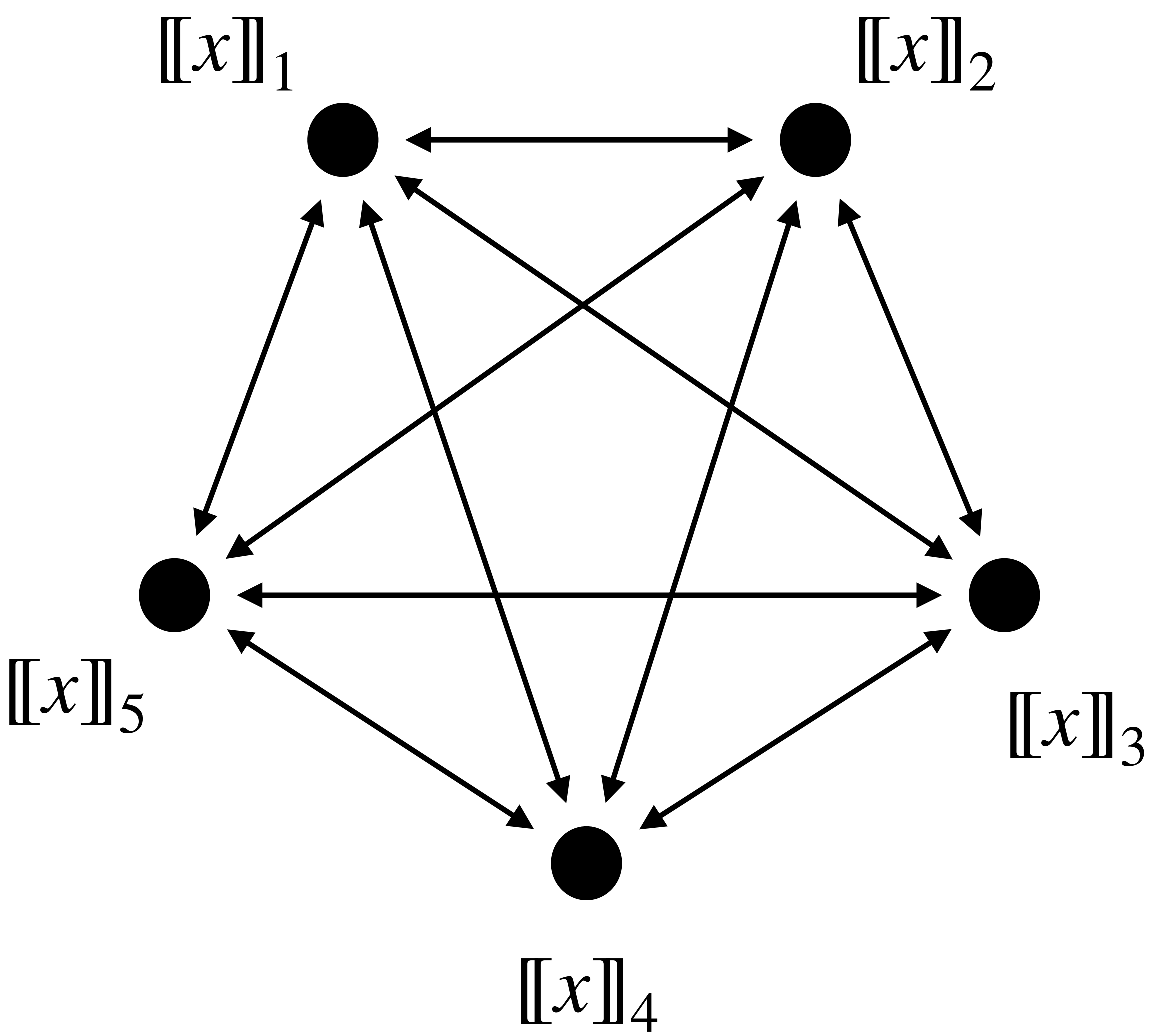

One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Zero-knowledge proof
x
y
Prover
Verifier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



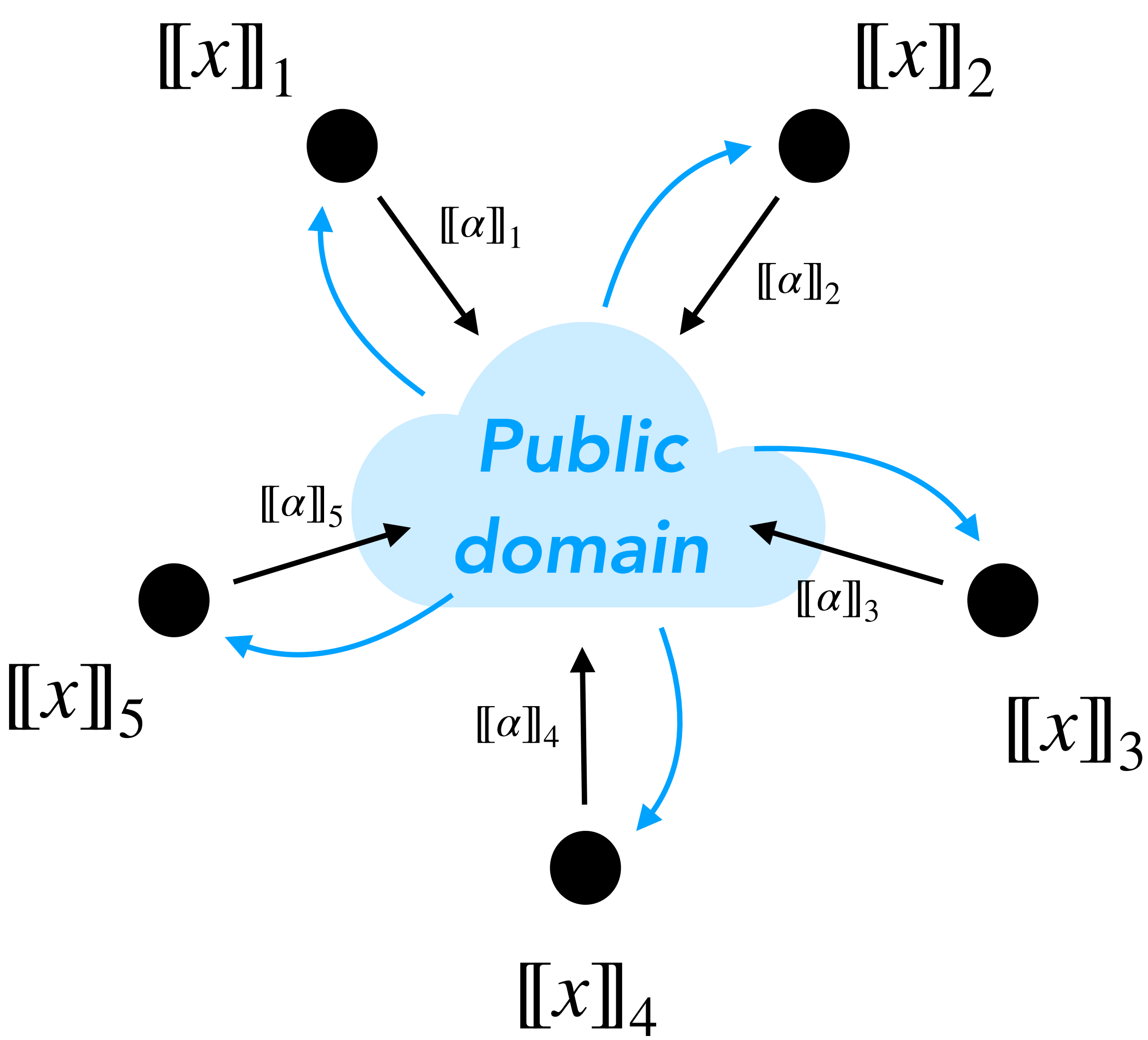
$$x = [\![x]\!]_1 + [\![x]\!]_2 + \ldots + [\![x]\!]_N$$

- **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

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

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

$\cdots$

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

$\longrightarrow$

Prover

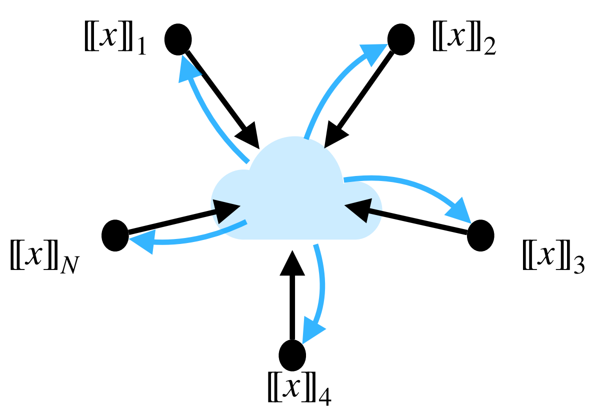
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\ldots$

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

send broadcast

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

Prover

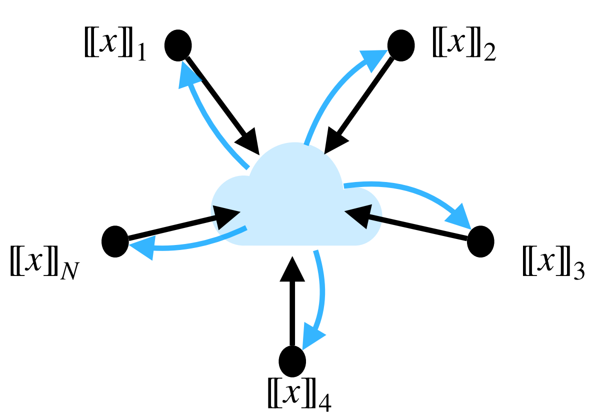
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\ldots$

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

i^*

$\longleftarrow$

③ Choose a random party

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

Prover

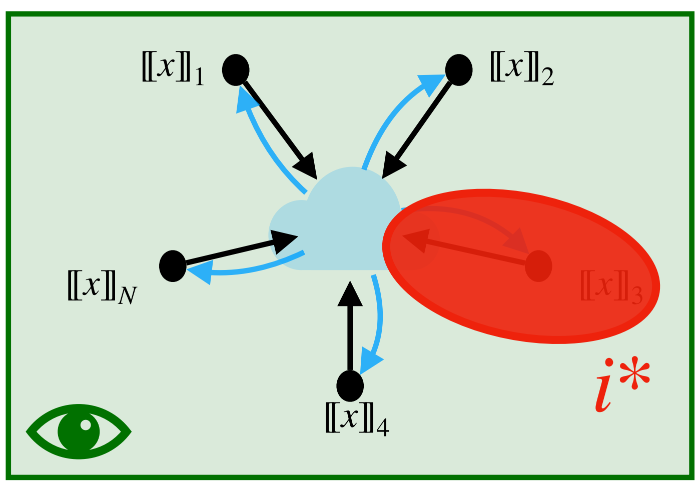
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

$\dots$

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

send broadcast

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

$\longleftarrow$ i^*

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

③ Choose a random party

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

Prover

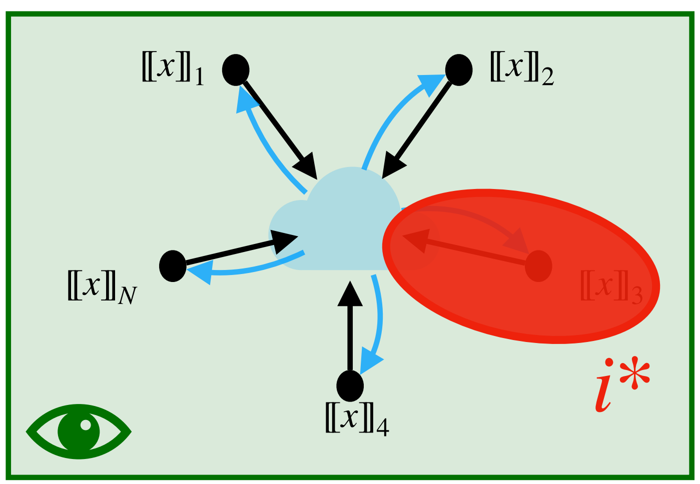
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

$\ldots$

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

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 $g(y, \alpha) =$ 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

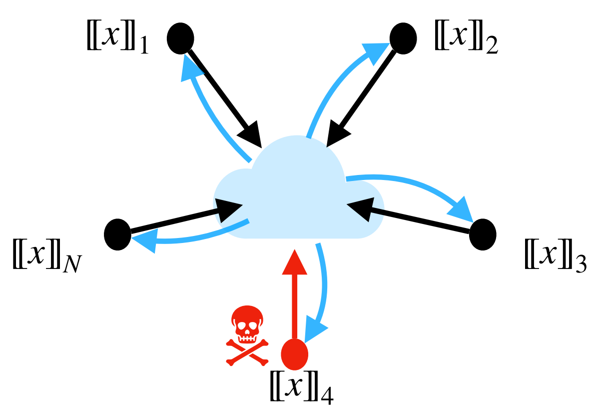
① Generate and commit shares

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

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

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

② Run MPC in their head



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

$\ldots$

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

Malicious Prover

Verifier

MPCitH transform

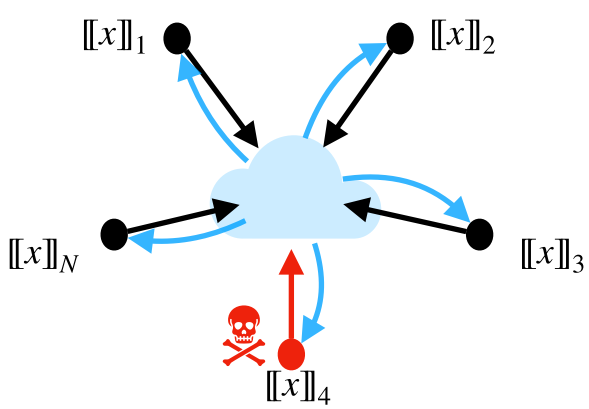
① Generate and commit shares

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

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

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

← i^*

③ Choose a random party

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

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

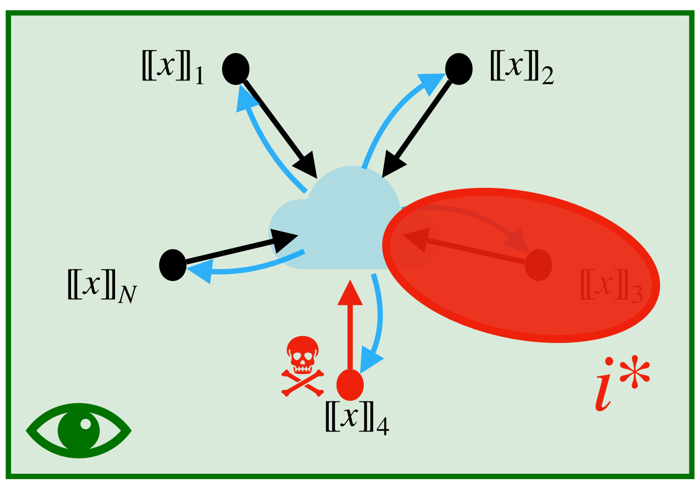
$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

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

$\cdots$

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

② Run MPC in their head



send broadcast

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

③ Choose a random party

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

i^*

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

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

Malicious Prover

Verifier

MPCitH transform

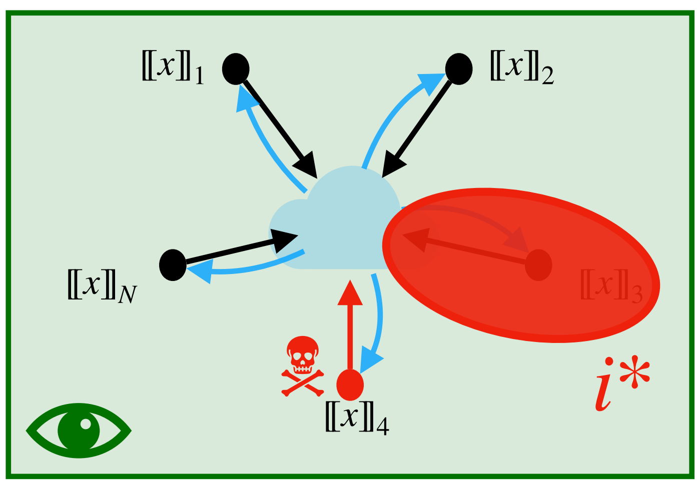
① Generate and commit shares

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

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

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

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 $g(y, \alpha) =$ Accept

Malicious Prover

Verifier

Cheating detected!

MPCitH transform

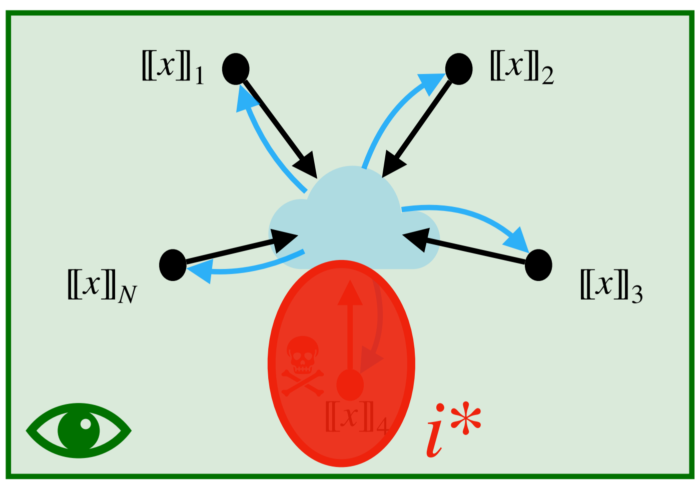
① Generate and commit shares

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

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

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

② Run MPC in their head



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

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

$\dots$

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

send broadcast

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

← i^*

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

③ Choose a random party

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

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

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

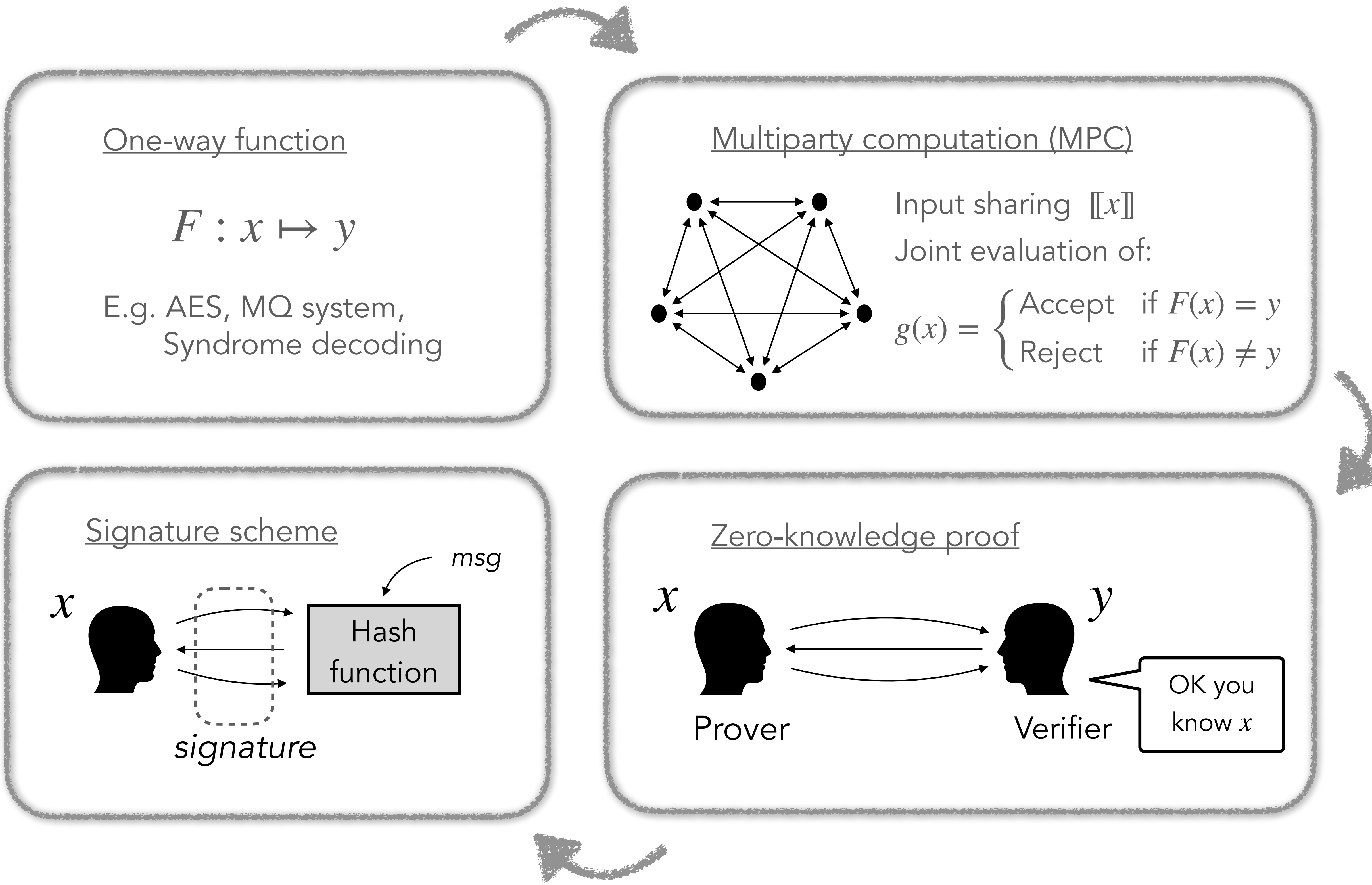
- **Zero-knowledge** $\iff$ MPC protocol is $(N-1)$-private
- **Soundness:**

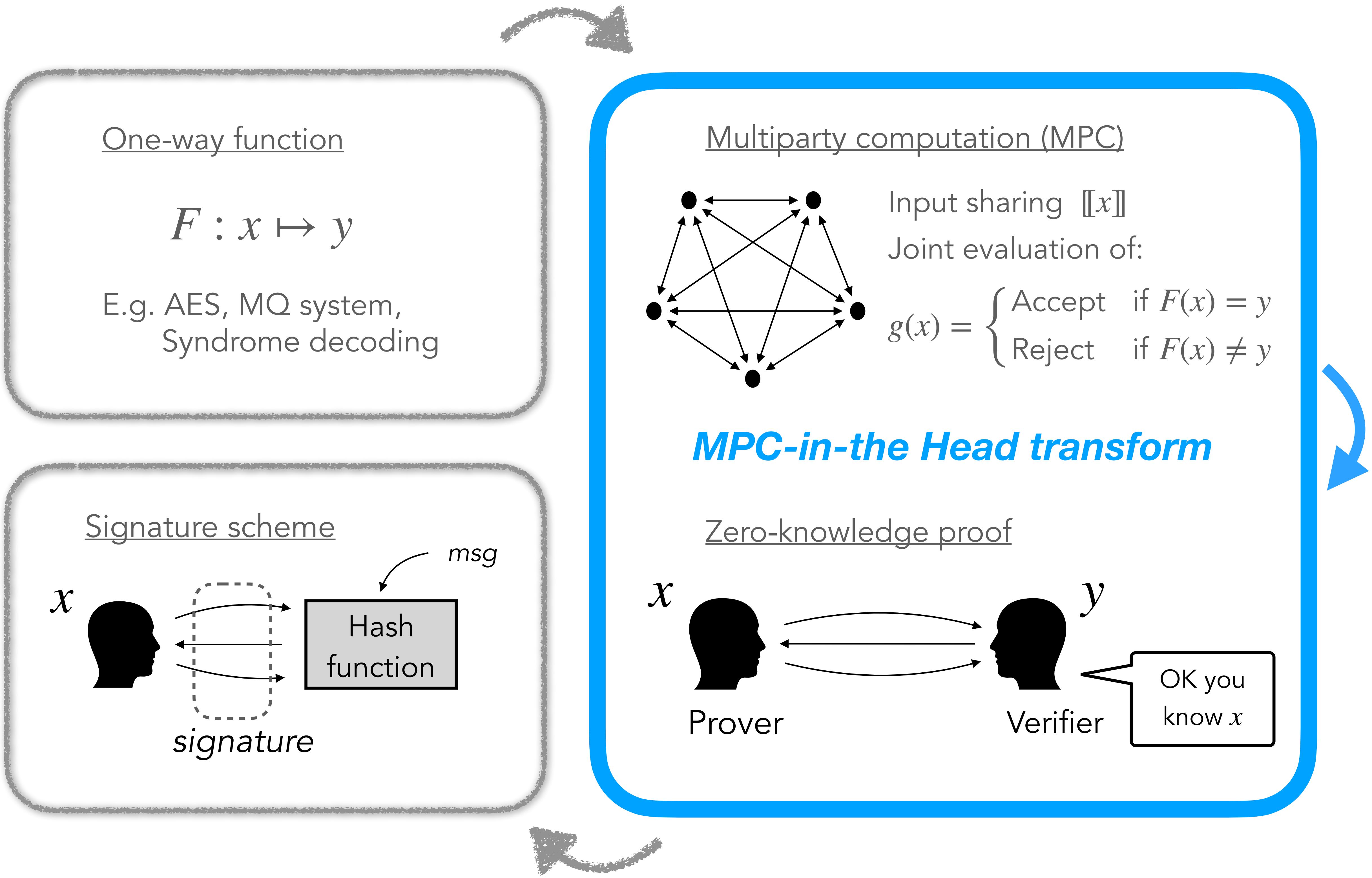
$$\begin{aligned}\mathbb{P}(\text{malicious prover convinces the verifier}) \\ &= \mathbb{P}(\text{corrupted party remains hidden}) \\ &= \frac{1}{N}\end{aligned}$$

- **Parallel repetition**

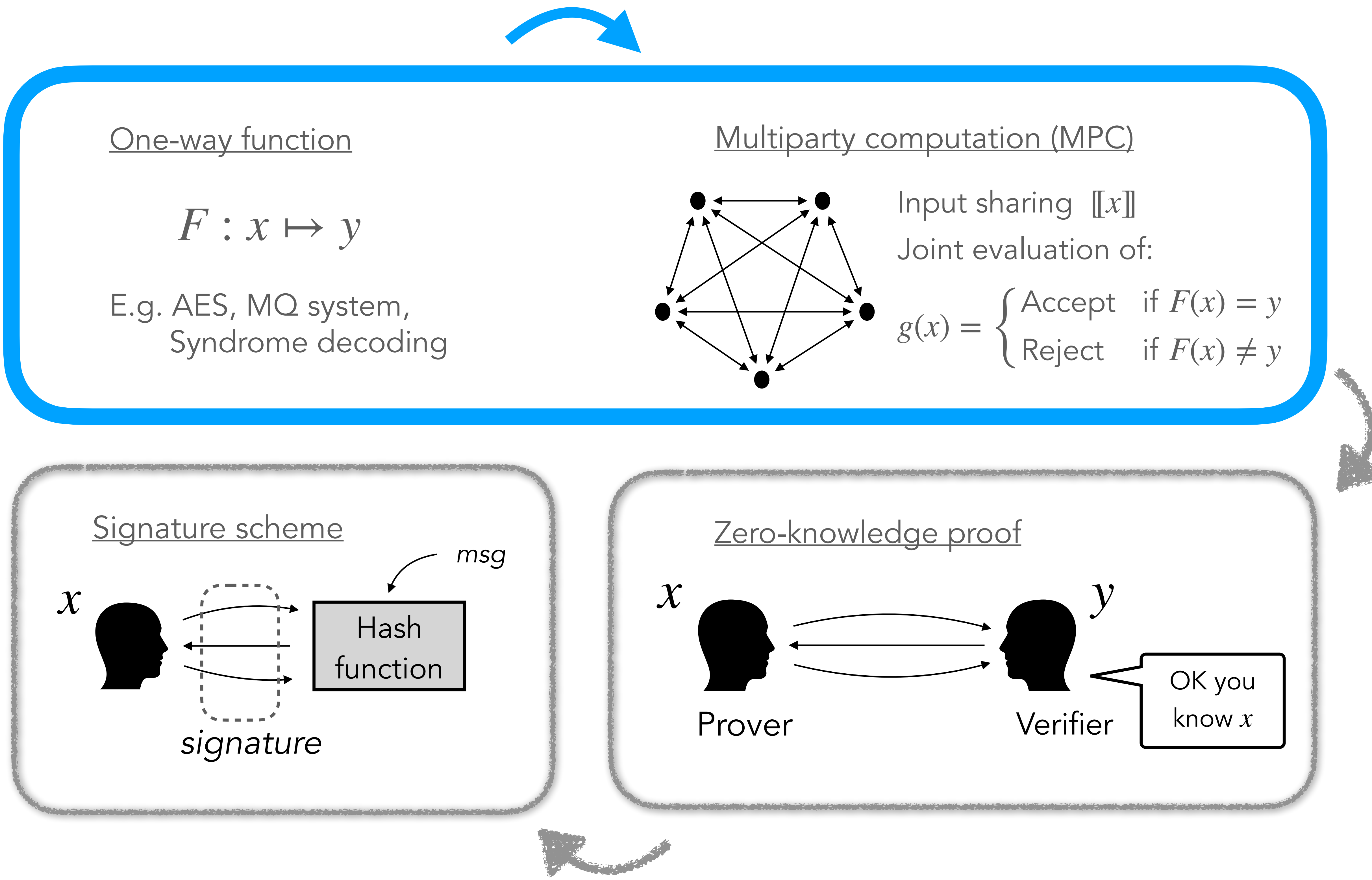
Protocol repeated τ times in parallel, soundness error $\left(\frac{1}{N}\right)^{\tau}$

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
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
Signature scheme
x
msg
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



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

The problem of factorisation:

$$(p, q) \mapsto N := pq$$

Very hard to invert !

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

The problem of factorisation:

$$(p, q) \mapsto N := pq$$

Very hard to invert !

1. Build a MPC protocol that takes $[\![p]\!]$ and $[\![q]\!]$ and checks that $p \cdot q = N$.

2. Using the MPC-in-the-Head transformation, we get a zero-knowledge proof of knowledge for the factorisation problem.

3. Using the Fiat-Shamir transformation, we get a signature scheme relying on the hardness to solve to factorize a composite number.

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Not secure against quantum computers!

The problem of factorisation:

$$(p, q) \mapsto N := pq$$

Very hard to invert !

1. Build a MPC protocol that takes $[\![p]\!]$ and $[\![q]\!]$ and checks that $p \cdot q = N$.

2. Using the MPC-in-the-Head transformation, we get a zero-knowledge proof of knowledge for the factorisation problem.

3. Using the Fiat-Shamir transformation, we get a signature scheme relying on the hardness to solve to factorize a composite number.

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Quantum-resilient hard problems:

- *Lattice-based cryptography*
- *Code-based cryptography*
- *Multivariate cryptography*
- *Symmetric cryptography*
- *…*

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Quantum-resilient hard problems:

- ***Lattice-based cryptography***
- *Code-based cryptography*
- *Multivariate cryptography*
- *Symmetric cryptography*
- …

- Lattice-based cryptography
  - The **Short Integer Solution** (SIS) problem: from (A, t), find a vector s such that

    $$t = As \quad \text{and} \quad \|s\| \text{ small.}$$

  - The **Learning With Errors** (LWE) problem: from (A, t), find two vectors s, e such that

    $$t = As + e \quad \text{and} \quad \|e\| \text{ small.}$$

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Quantum-resilient hard problems:

- *Lattice-based cryptography*
- ***Code-based cryptography***
- *Multivariate cryptography*
- *Symmetric cryptography*
- …

- Code-based cryptography
  - The **Syndrome Decoding** (SD) problem: from (H, y), find a vector x such that
  $$y = Ax$$
  and x has w non-zero coordinates.
  - The **MinRank** problem: from $k+1$ matrices $M_0, \dots M_k$, find a linear combination x such that
  $$E := M_0 + \sum_{j=1}^{k} x_j M_j$$
  has a rank smaller than some public constant r.

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Quantum-resilient hard problems:

- *Lattice-based cryptography*
- *Code-based cryptography*
- ***Multivariate cryptography***
- *Symmetric cryptography*
- …

- Multivariate cryptography
  - The **Multivariate Quadratic** (MQ) problem: find a solution x of the system of m quadratic equations

$$\begin{cases} y_1 & = \sum_{i \le j} a_{1,i,j} \cdot x_i x_j + \sum_i b_{1,i} \cdot x_i \\ & \vdots \\ y_m & = \sum_{i \le j} a_{m,i,j} \cdot x_i x_j + \sum_i b_{m,i} \cdot x_i \end{cases}$$

where $\{a_{k,i,j}\}$ and $\{b_{k,i}\}$ are the coefficients of the system.

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Quantum-resilient hard problems:

- *Lattice-based cryptography*
- *Code-based cryptography*
- *Multivariate cryptography*
- ***Symmetric cryptography***
- …

- Symmetric cryptography
  - Hash functions.
  - AES cipher: given (x, y), find an AES key k for which the ciphertext of x is y:

$$y = \mathrm{AES}_k(x)$$

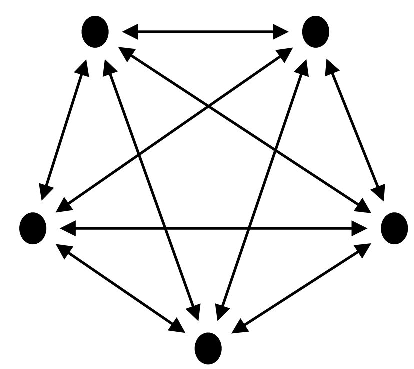
  - Any other cipher scheme.

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Quantum-resilient hard problems:

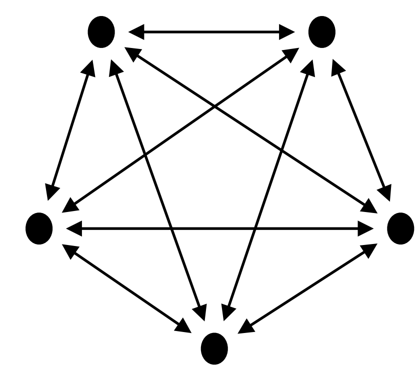
- *Lattice-based cryptography*
- *Code-based cryptography*
- *Multivariate cryptography*
- *Symmetric cryptography*
- …

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

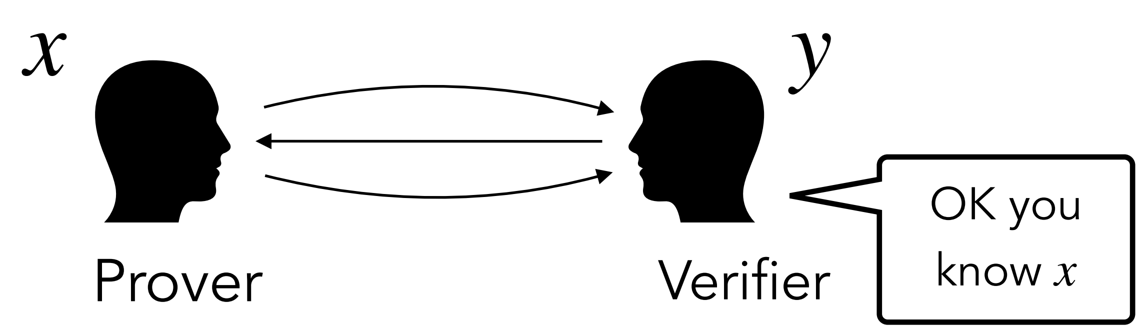
Joint evaluation of:

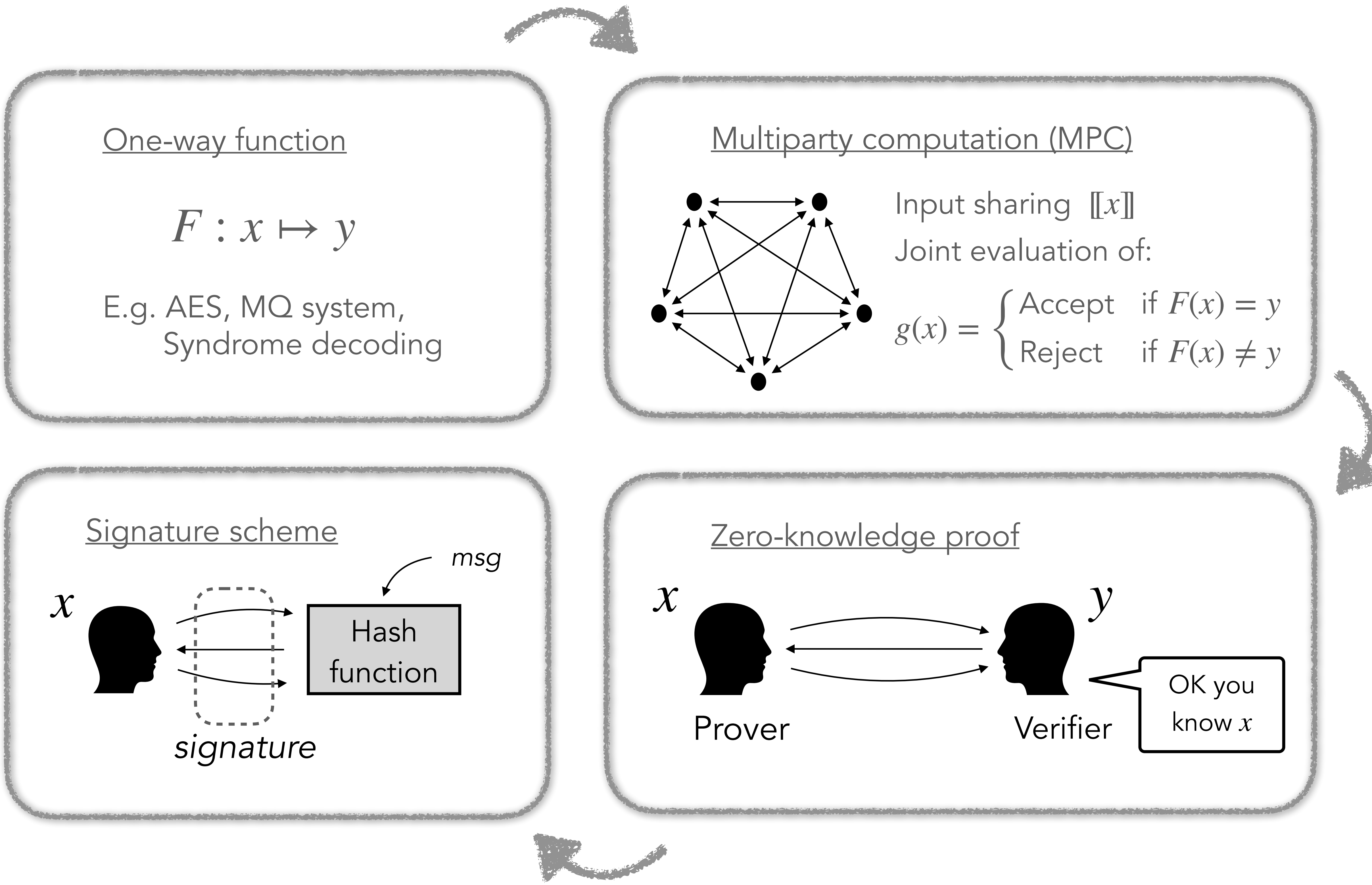
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Quantum-resilient hard problems:

- *Lattice-based cryptography*
- *Code-based cryptography*
- *Multivariate cryptography*
- *Symmetric cryptography*
- …

Zero-knowledge proof





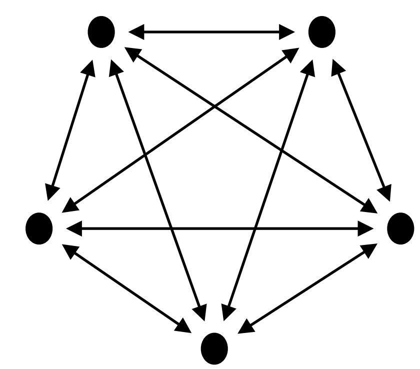
One-way function
$F : x \mapsto y$
E.g. AES, MQ system, Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Signature scheme
x
msg
Hash function
signature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

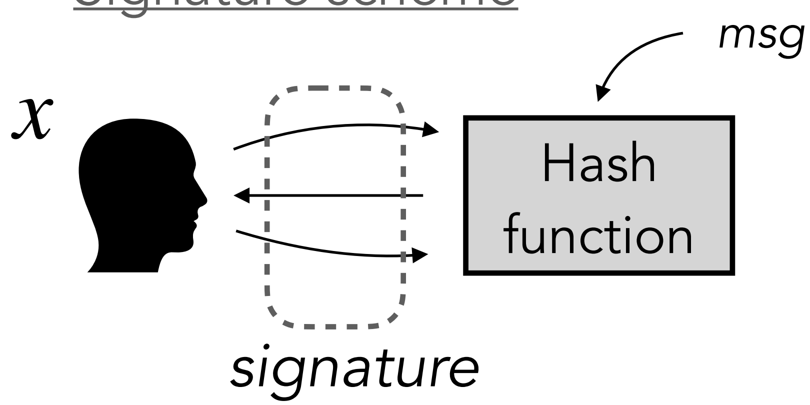


Input sharing $[\![x]\!]$

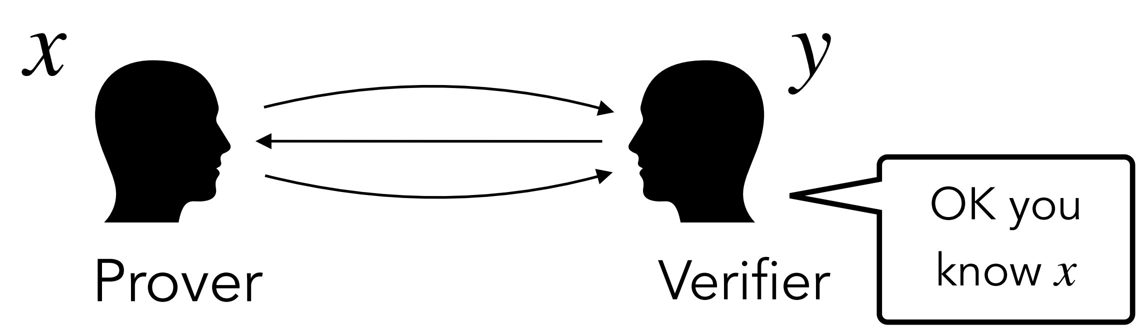
Joint evaluation of:

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

Signature scheme



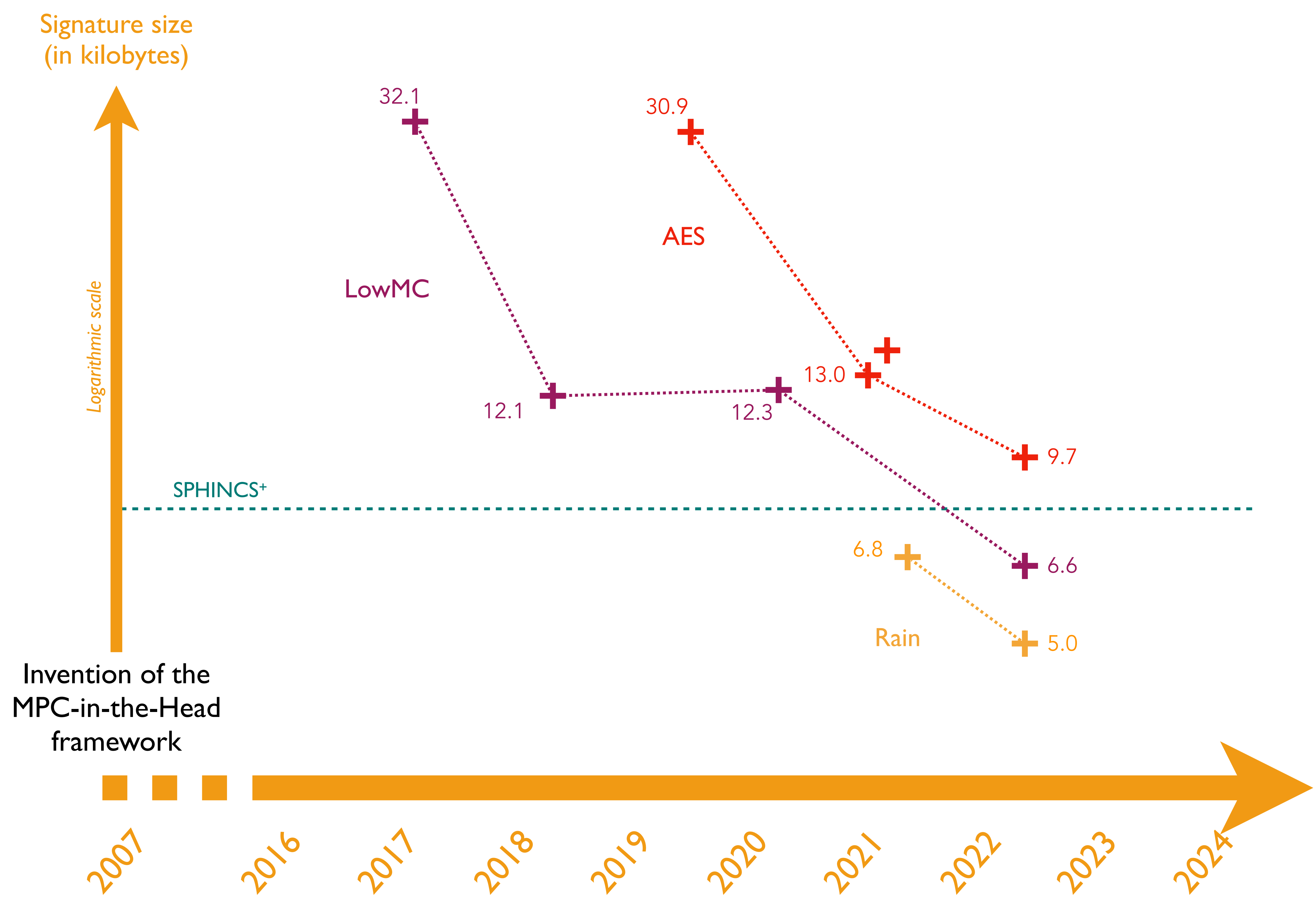
Zero-knowledge proof

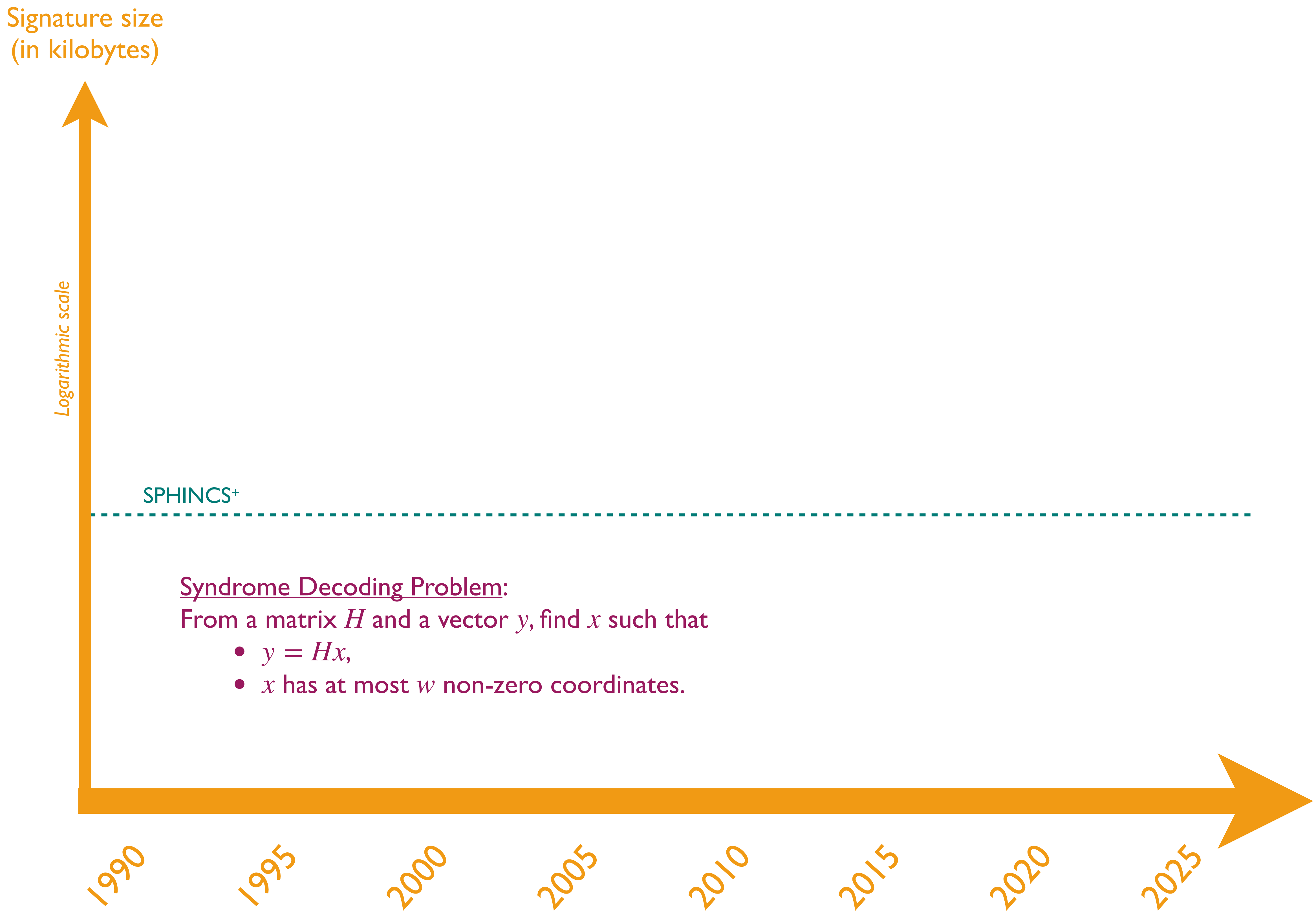


Fiat-Shamir transform

Should take [KZ20] attack into account (when there are more than 3 rounds)!

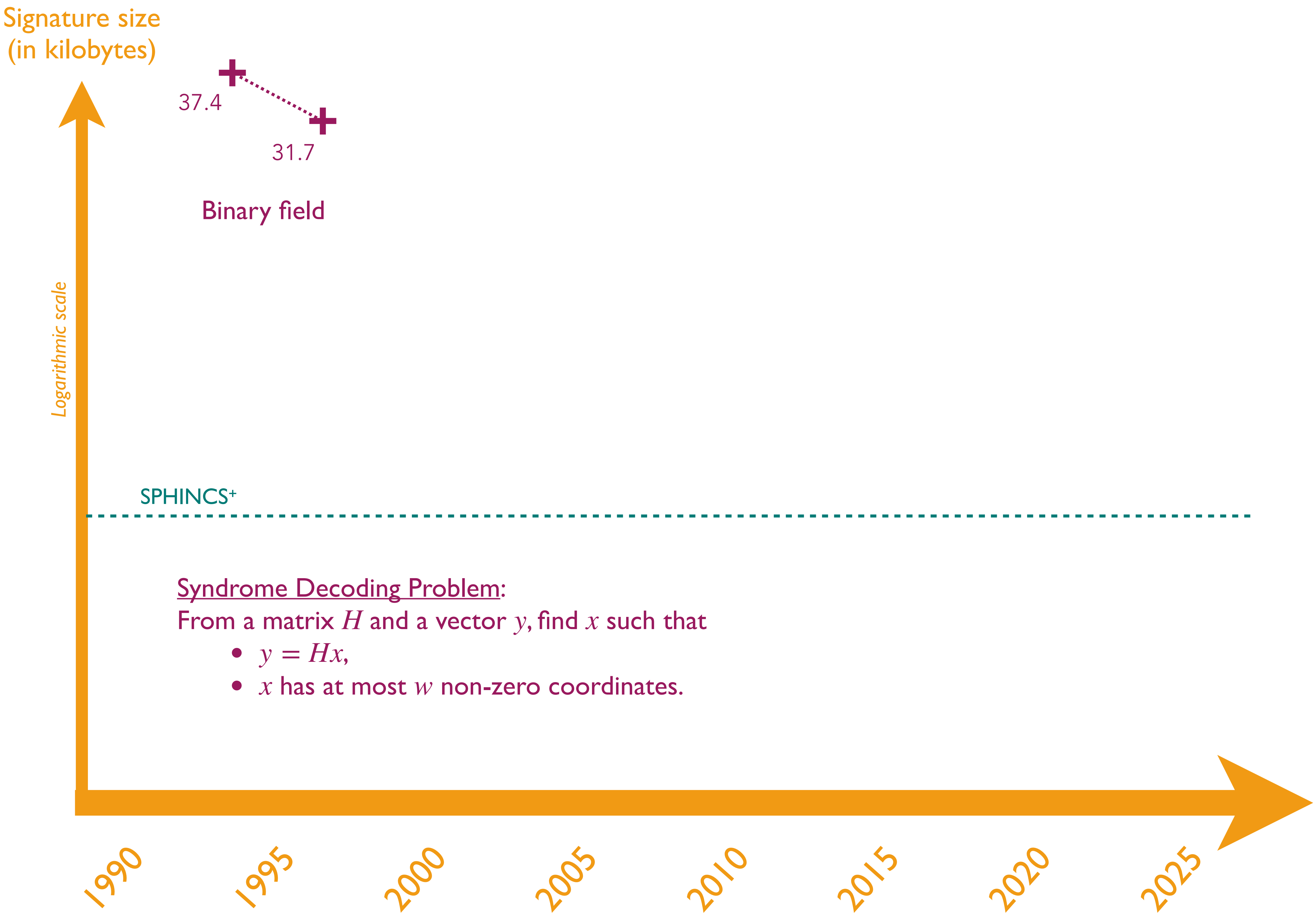
[KZ20] Kales, Zaverucha. "An attack on some signature schemes constructed from five-pass identification schemes" (CANS20)


Signature size
(in kilobytes)
Logarithmic scale
32.1
30.9
AES
LowMC
13.0
12.1
12.3
9.7
SPHINCS+
6.8
6.6
Rain
5.0
Invention of the
MPC-in-the-Head
framework
2007
2016
2017
2018
2019
2020
2021
2022
2023
2024



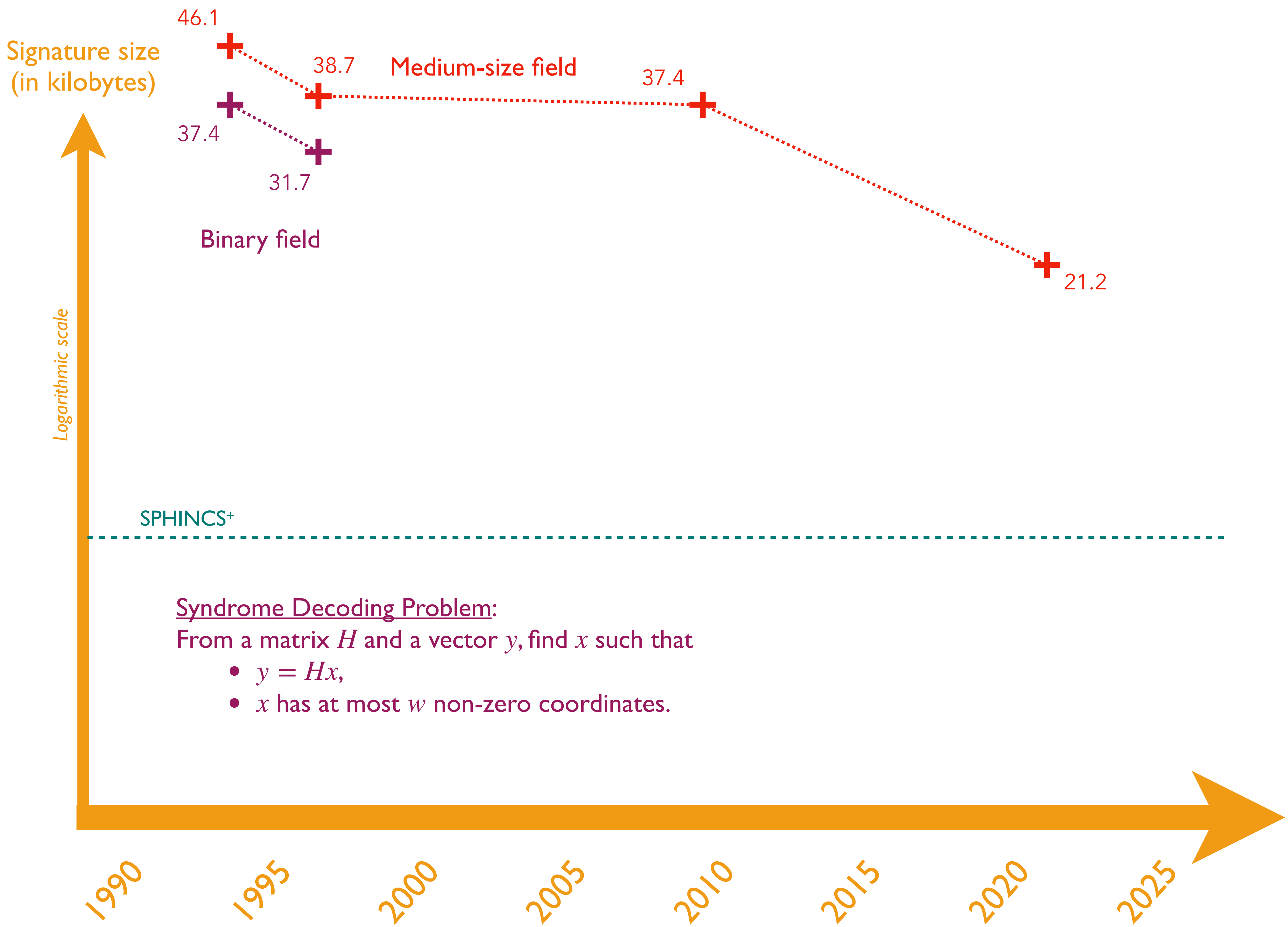
Syndrome Decoding Problem:
From a matrix H and a vector y, find x such that

- $y = Hx$,
- x has at most w non-zero coordinates.



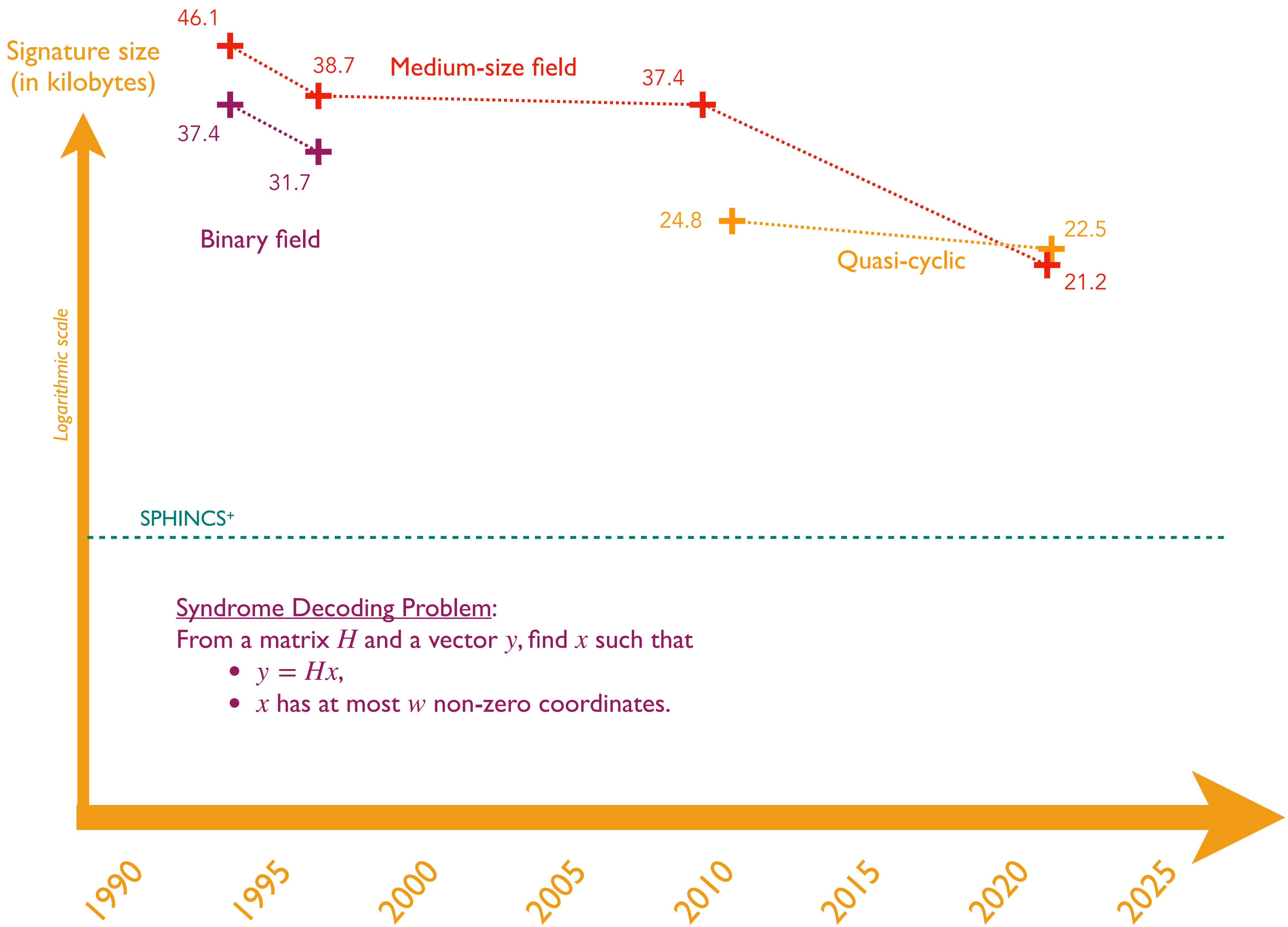
<u>Syndrome Decoding Problem</u>:
From a matrix H and a vector y, find x such that

- $y = Hx$,
- x has at most w non-zero coordinates.



Syndrome Decoding Problem:
From a matrix H and a vector y, find x such that

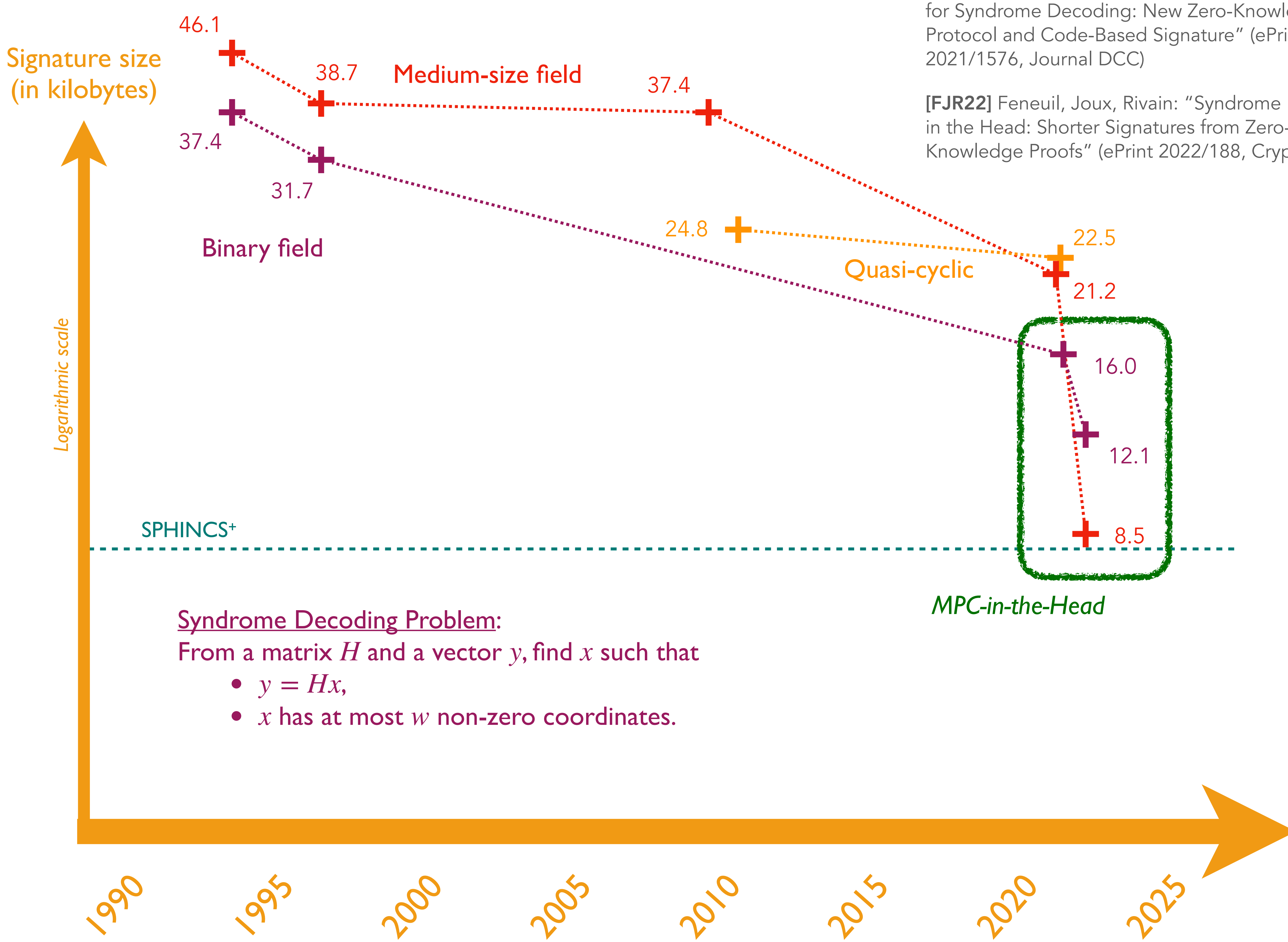
- $y = Hx$,
- x has at most w non-zero coordinates.



Signature size
(in kilobytes)
Logarithmic scale
46.1
38.7
Medium-size field
37.4
37.4
31.7
Binary field
24.8
22.5
Quasi-cyclic
21.2
SPHINCS+
Syndrome Decoding Problem:
From a matrix H and a vector y, find x such that
- $y = Hx$,
- x has at most w non-zero coordinates.
1990
1995
2000
2005
2010
2015
2020
2025

[FJR23] Feneuil, Joux, Rivain: "Shared Permutation for Syndrome Decoding: New Zero-Knowledge Protocol and Code-Based Signature" (ePrint 2021/1576, Journal DCC)

[FJR22] Feneuil, Joux, Rivain: "Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs" (ePrint 2022/188, Crypto 2022)



<u>Syndrome Decoding Problem</u>:
From a matrix H and a vector y, find x such that

- $y = Hx$,
- x has at most w non-zero coordinates.

Exploring other assumptions

- **Subset Sum Problem:** ≥ 100 KB $\Rightarrow$ 19.1 KB
- **Multivariate Quadratic Problem:** $6.3 - 7.3$ KB
- **MinRank Problem:** $\approx 5 - 6$ KB
- **Rank Syndrome Decoding Problem:** $\approx 5 - 6$ KB
- **Permuted Kernel Problem (or variant):** ≈ 6 KB
- $\ldots$

MPCitH-based NIST Candidates

1st June 2023:

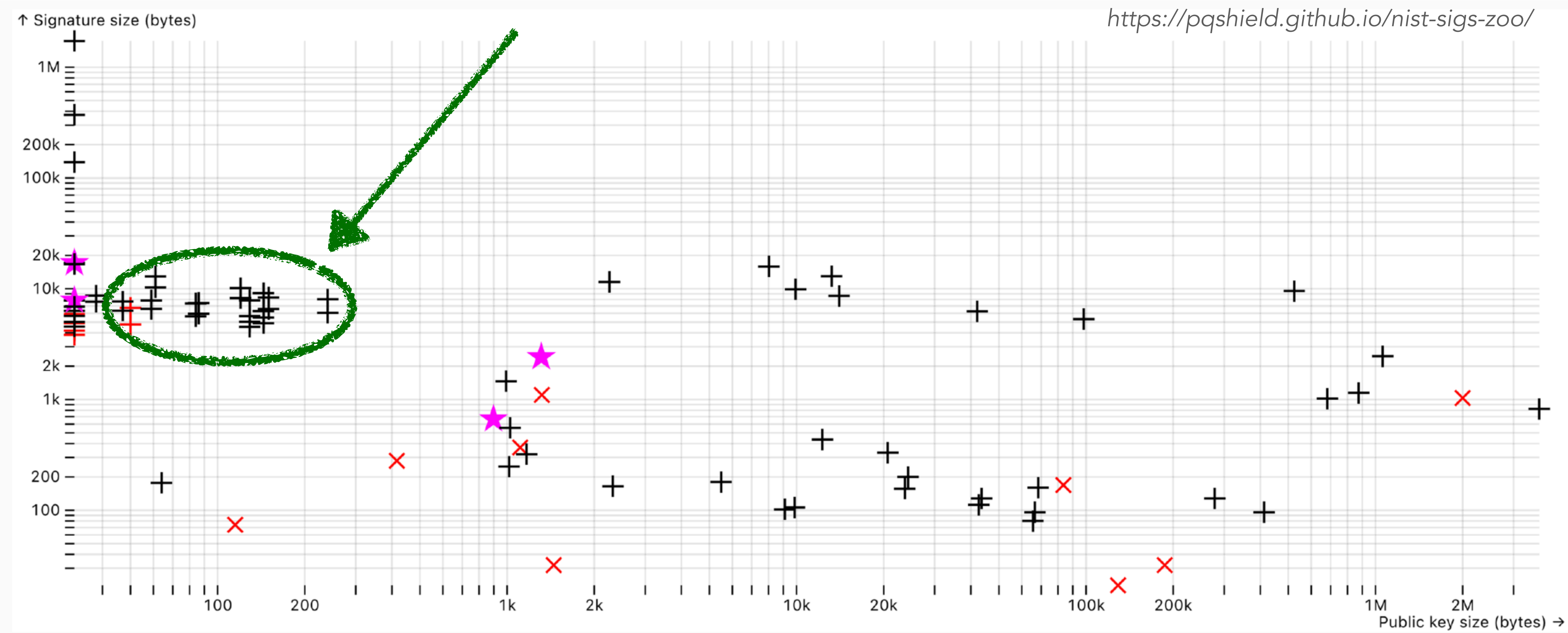
Deadline for the NIST call
for additional post-quantum signatures

MPCitH-based NIST Candidates

	Assumption	Size (in KB)
AIMer	AIM (MPC-friendly one-way function)	4.2
Biscuit	Structured MQ problem (PowAff2)	4.7
MIRA	MinRank problem	5.6
MiRitH	MinRank problem	5.7
RYDE	Syndrome decoding problem in rank metric	6.0
PERK*	Permuted Kernel problem (variant)	6.1
MQOM	Unstructured MQ problem	6.3
SDitH	Syndrome decoding problem in Hamming	8.2

MPCitH-based NIST Candidates

Figure extracted from PQ Signatures Zoo
https://pqshield.github.io/nist-sigs-zoo/



- Medium signature sizes (4-10 KB)
- Small public keys

Optimisations and variants

Field $GF(251)$

With `SDitH-L1-gf251` as example.

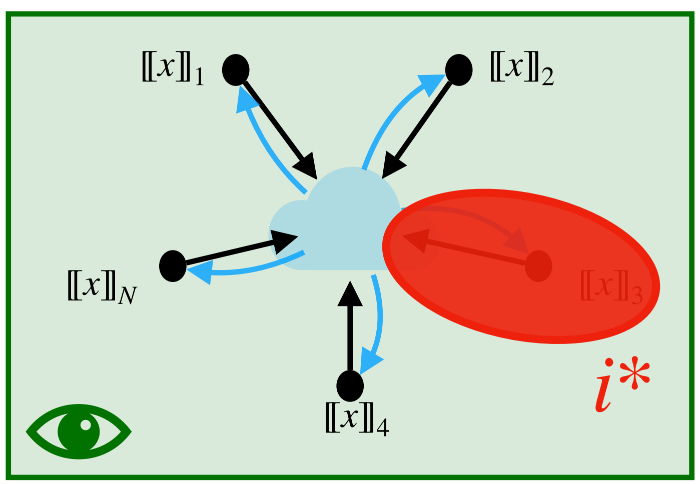
NIST Category I

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

$\ldots$

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

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

Prover

Verifier

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)

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

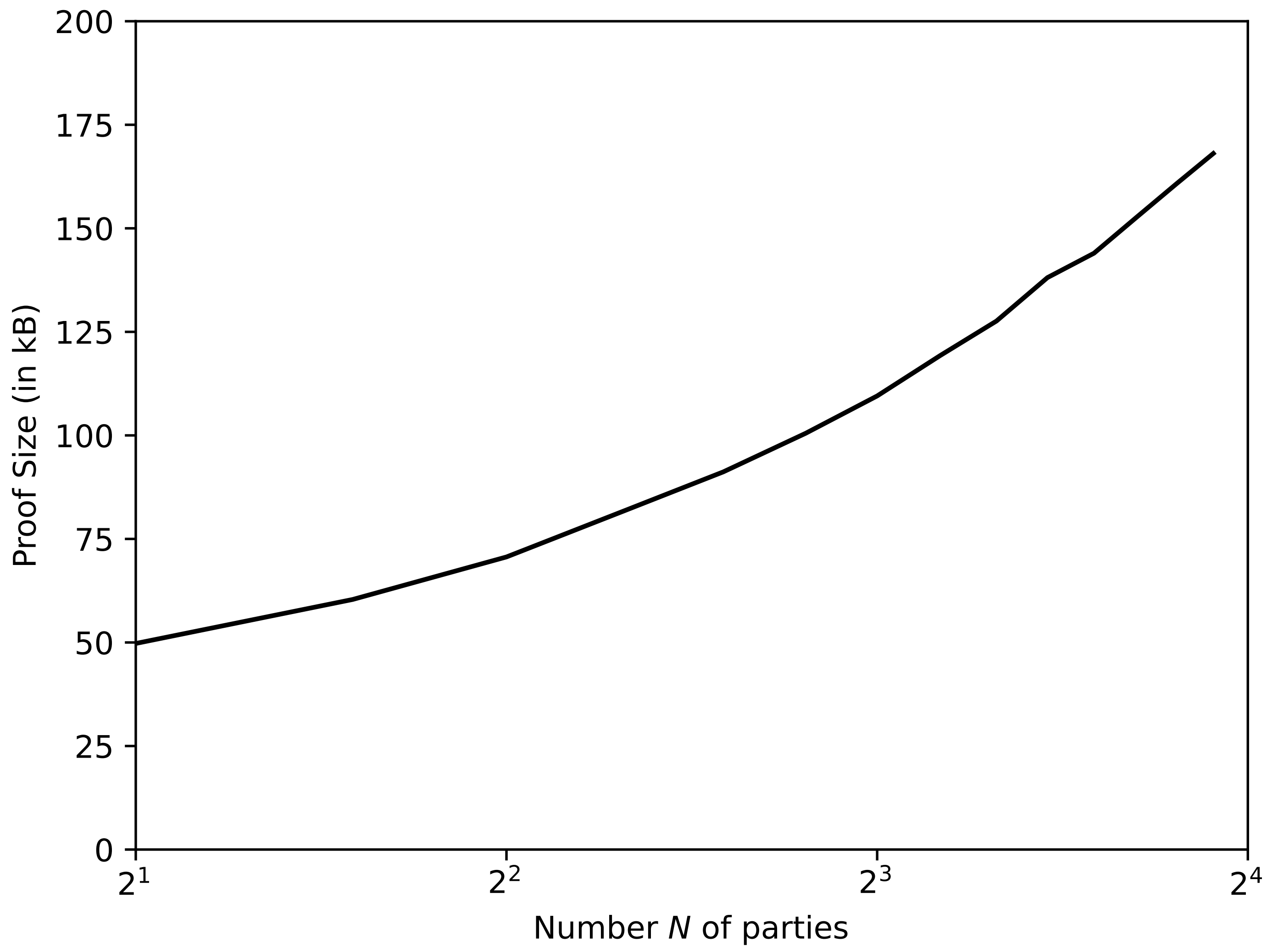
Number of repetitions to achieve the desired security level

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

`SDitH-L1-gf251:`

the input x of the MPC protocol is around **323** bytes,
The broadcast value α of the MPC protocol is around **36** bytes.

Naive MPCitH transformation



`SDitH-L1-gf251:`

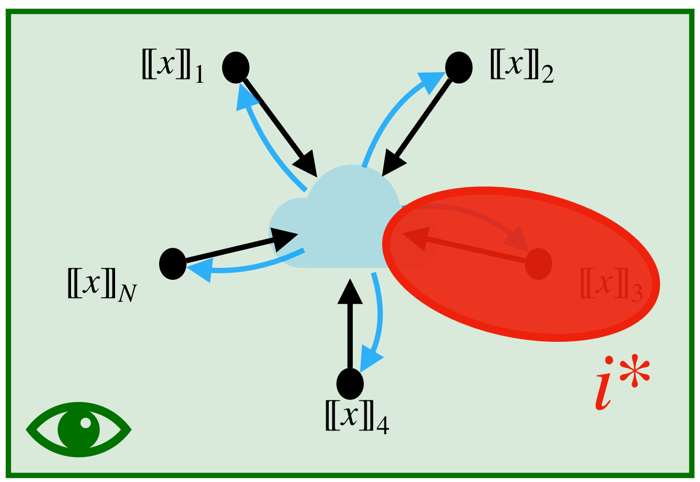
the input x of the MPC protocol is around **323** bytes,
The broadcast value α of the MPC protocol is around **36** bytes

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

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

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

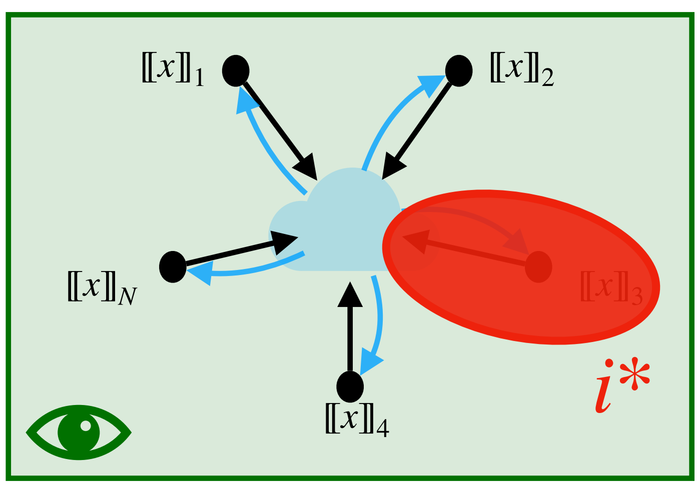
① Generate and commit shares

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

Compute

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

② Run MPC in their head



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

Prover

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

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

← i^*

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

③ Choose a random party

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

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

Verifier

MPCitH transform

① Generate and commit shares

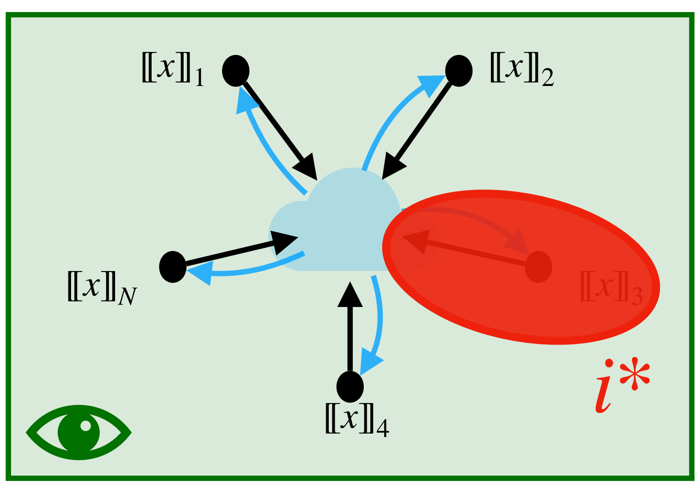
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

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

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

Using a Seed Tree

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

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

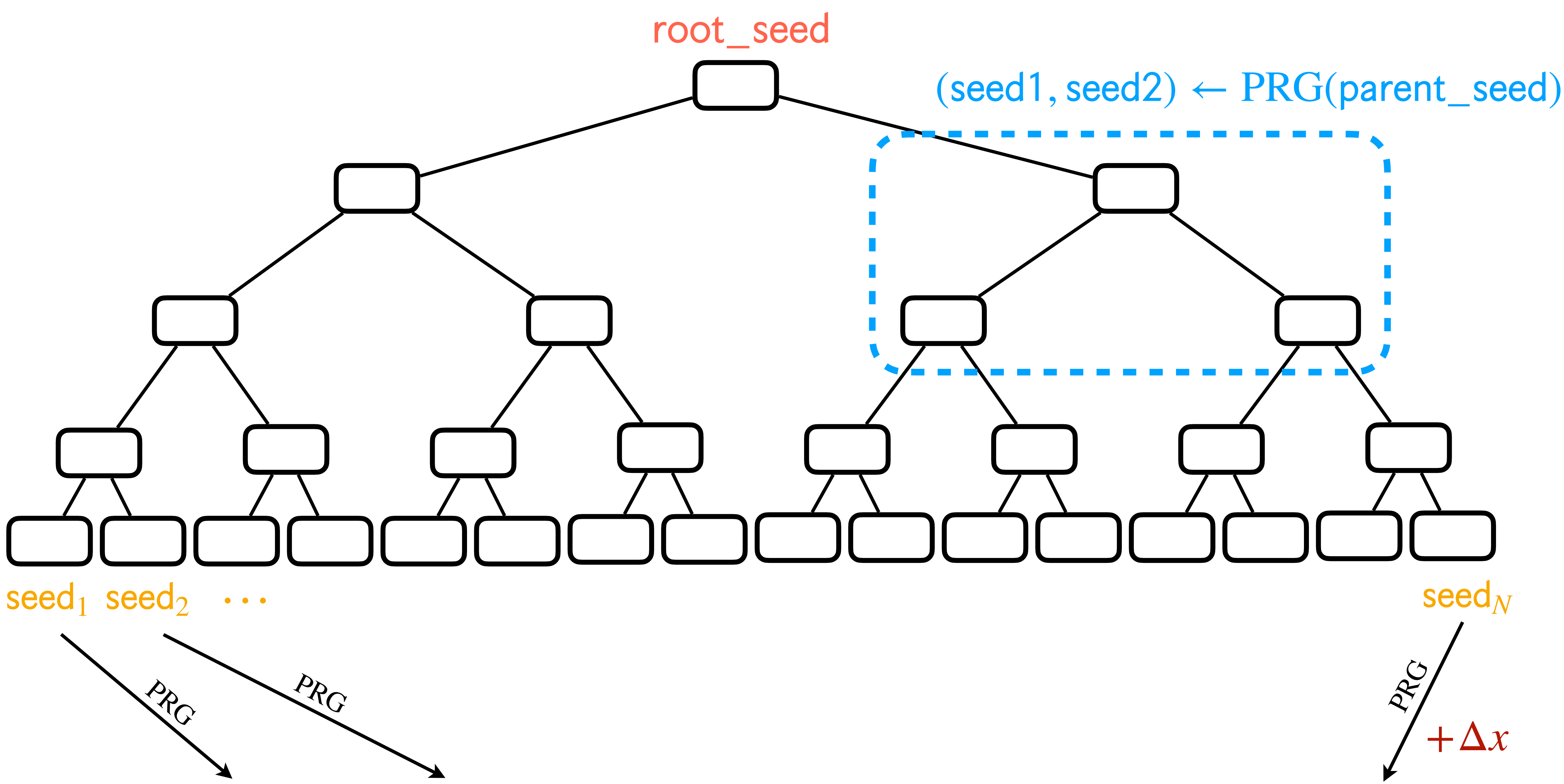
Using a Seed Tree

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

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

Using a Seed Tree

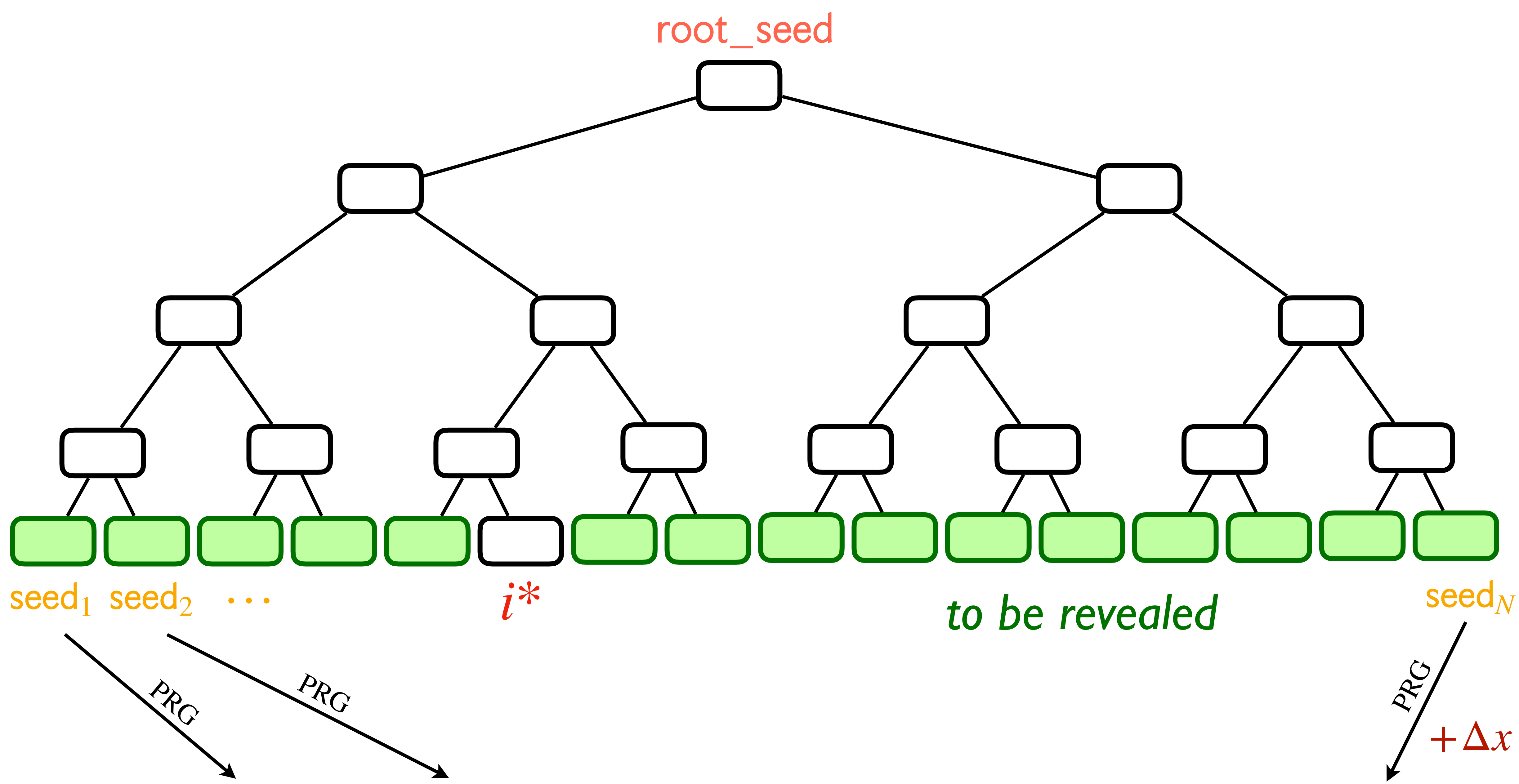
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



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

Using a Seed Tree

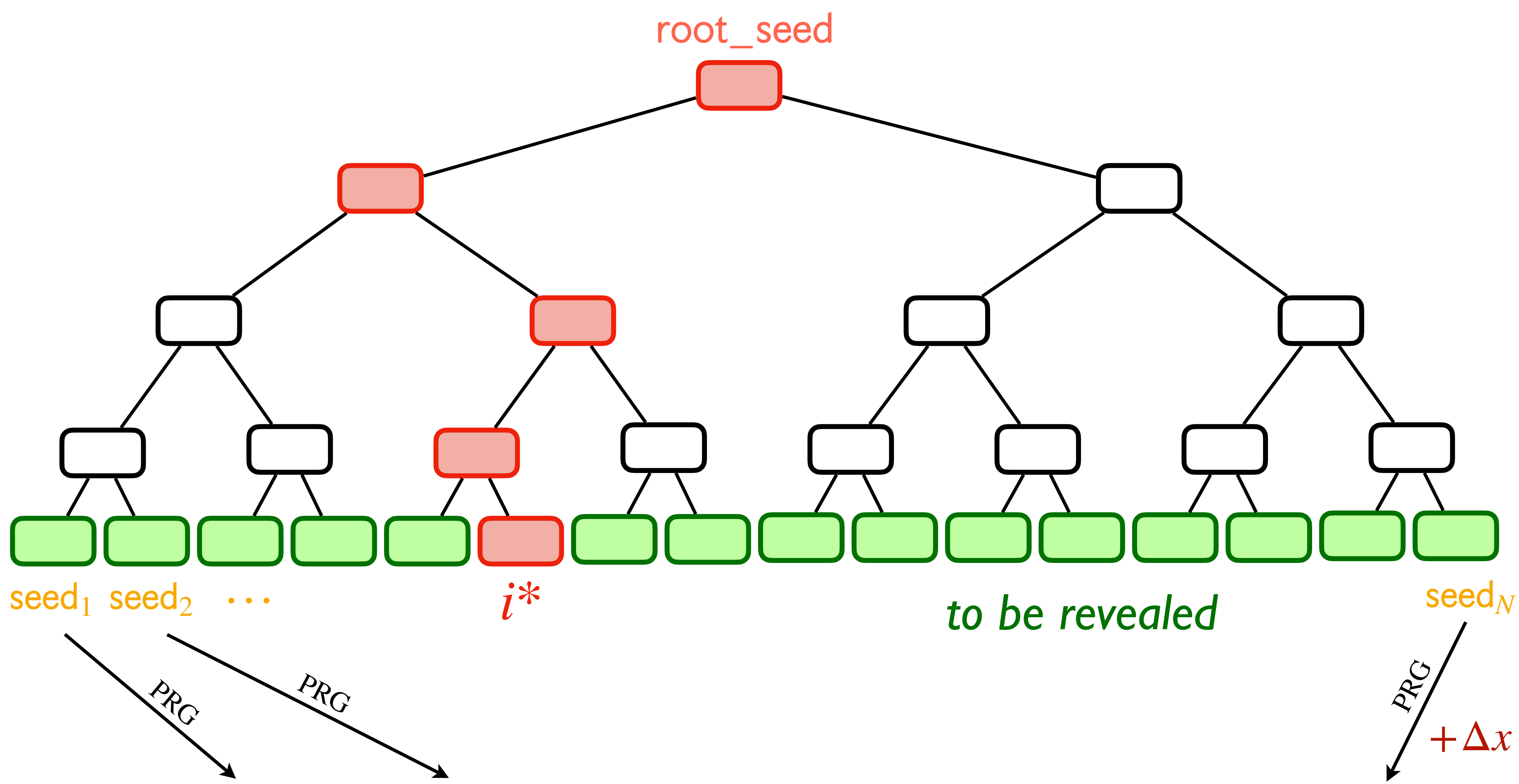
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



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

Using a Seed Tree

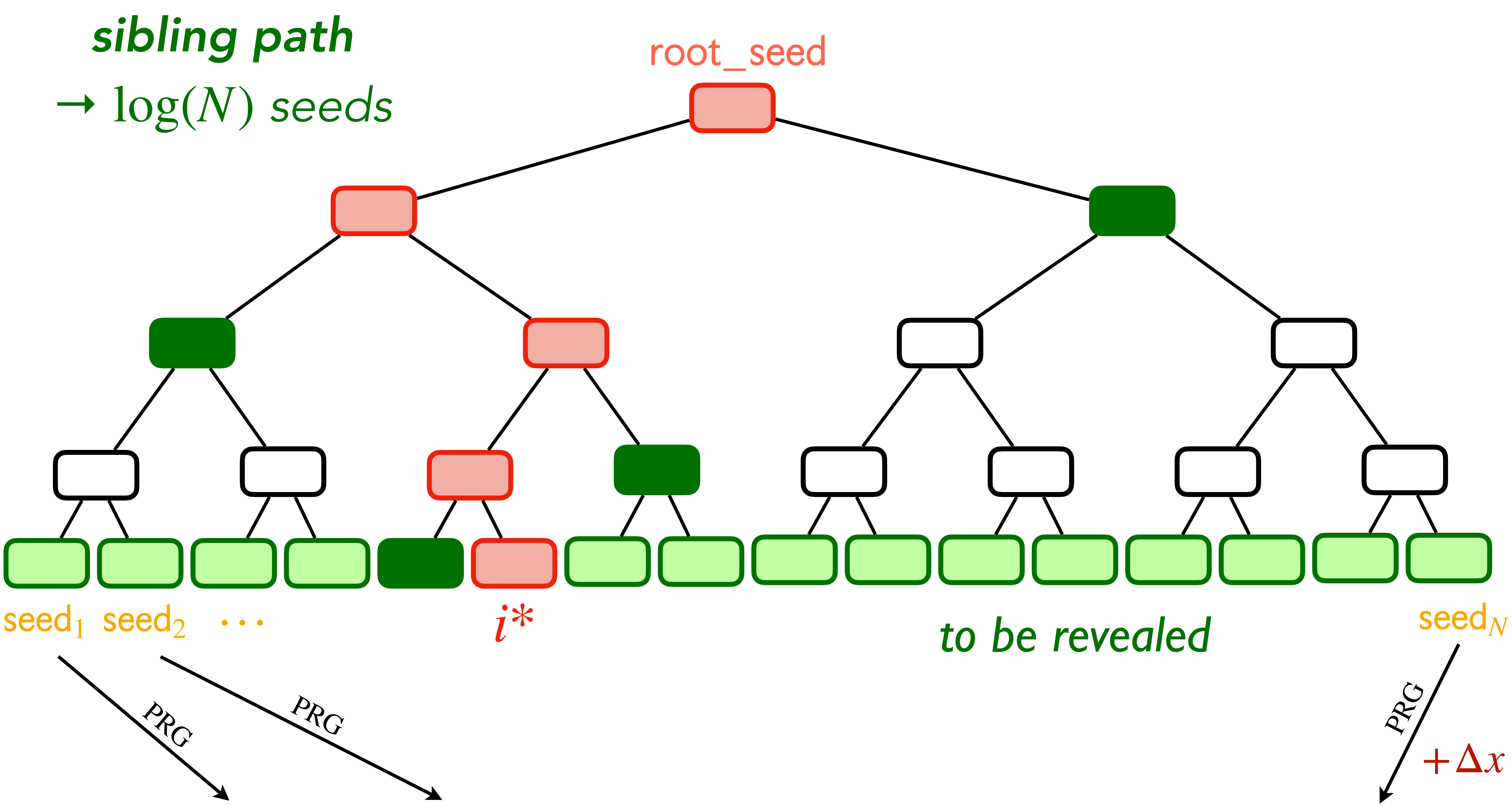
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



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

Using a Seed Tree

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



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

Traditional MPCitH transformation

Size of the broadcast (of the hidden party)

Size of the auxiliary value

Path in the seed (GGM) tree

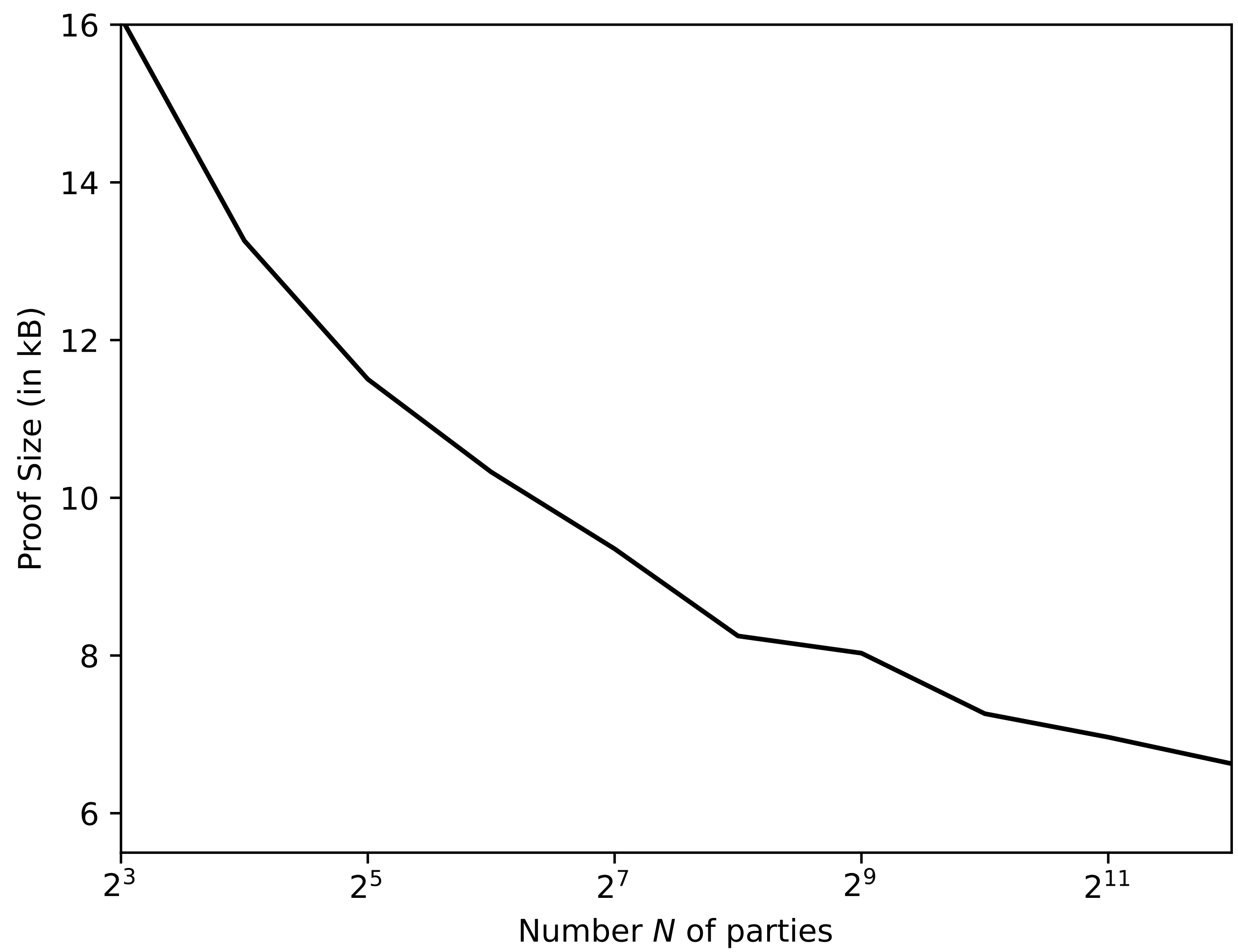
$$\text{Size} \approx \tau \cdot \big(|\Delta x| + |\alpha| + \lambda \cdot \log_2 N + 2\lambda \big)$$

Commitment
of the hidden party

Number of repetitions to achieve the desired security level

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

Traditional MPCitH transformation

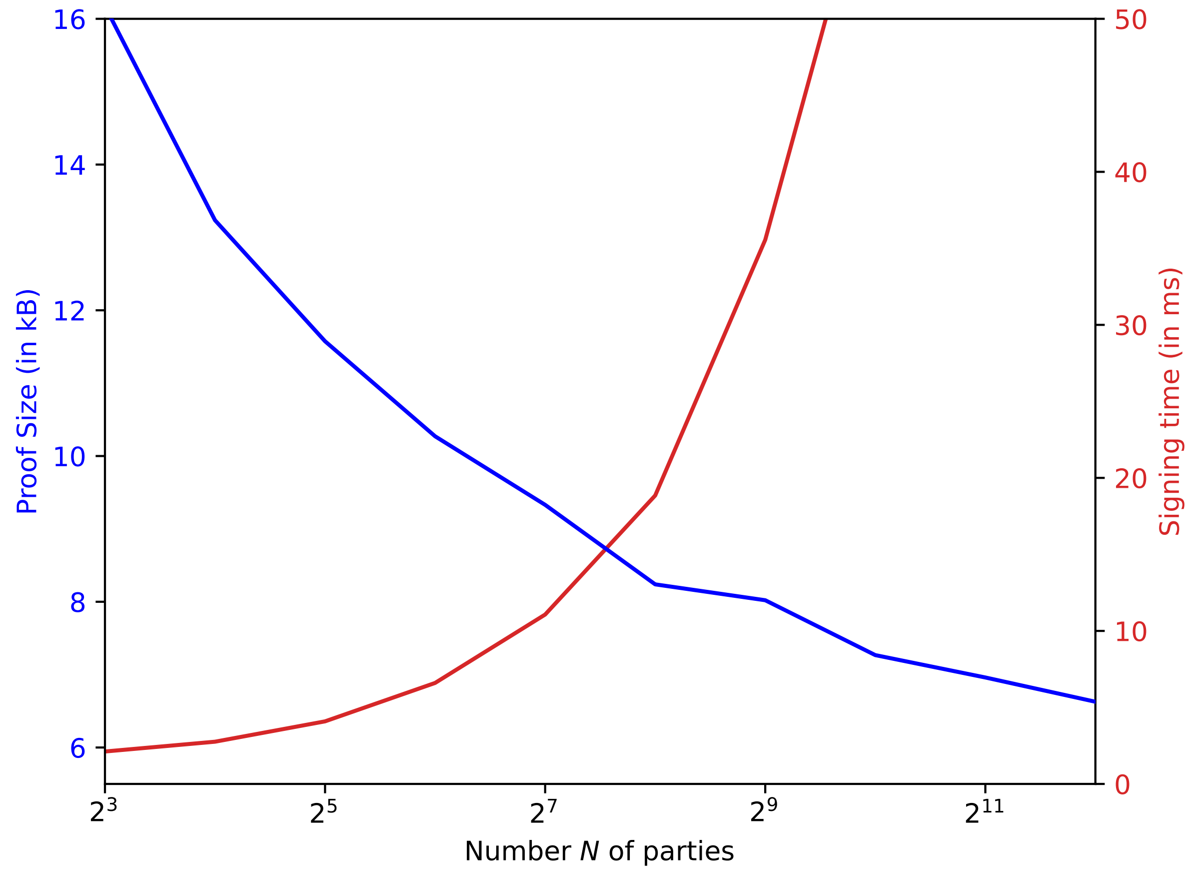


`SDitH-L1-gf251:`

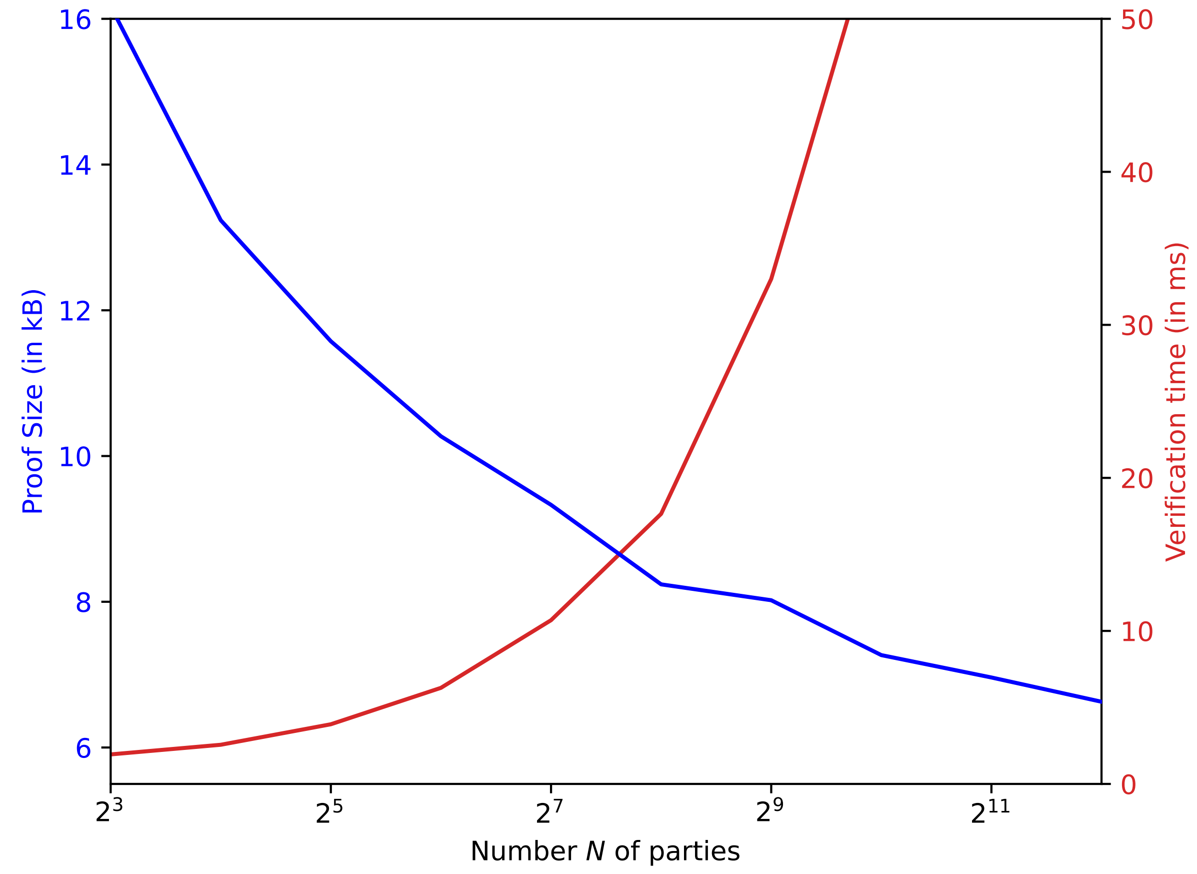
the input x of the MPC protocol is around **323** bytes,
The broadcast value α of the MPC protocol is around **36** bytes.

Traditional MPCitH transformation

Signing algorithm



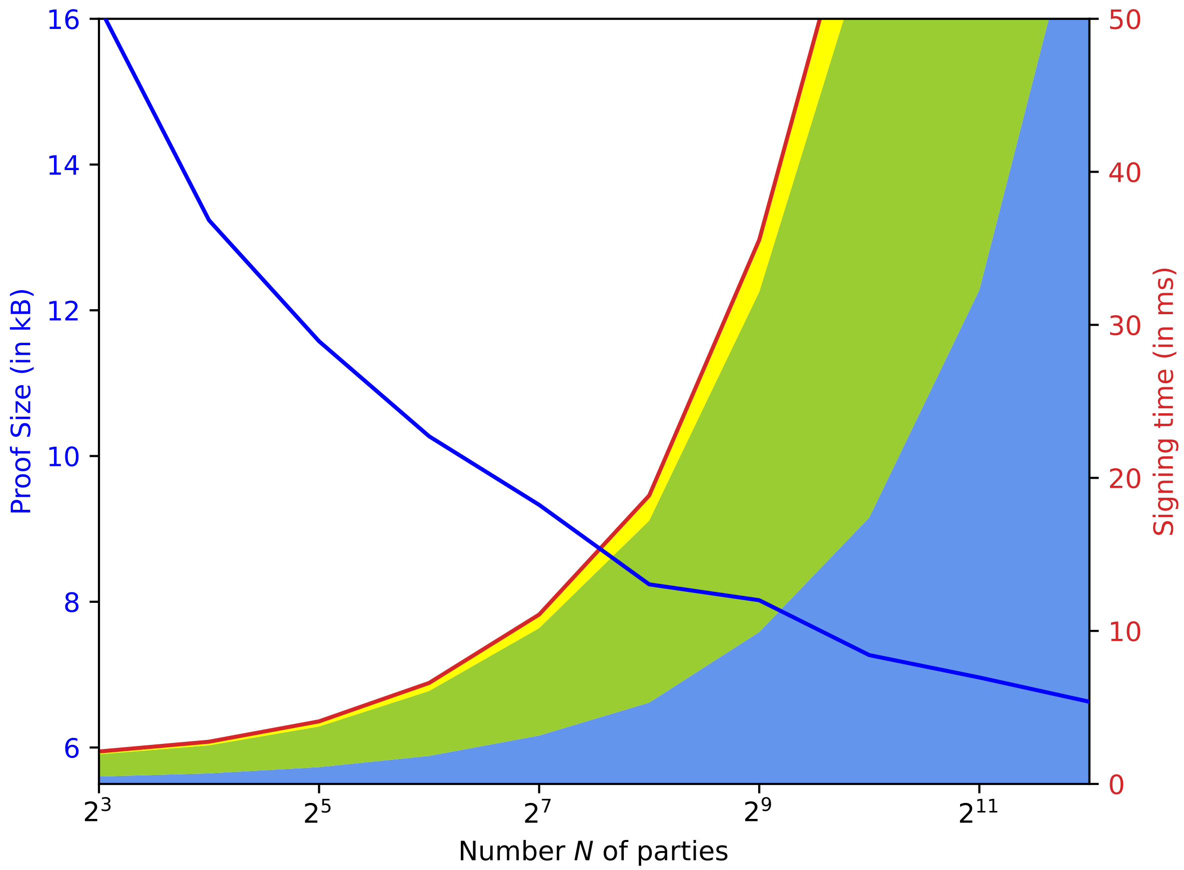
Verification algorithm



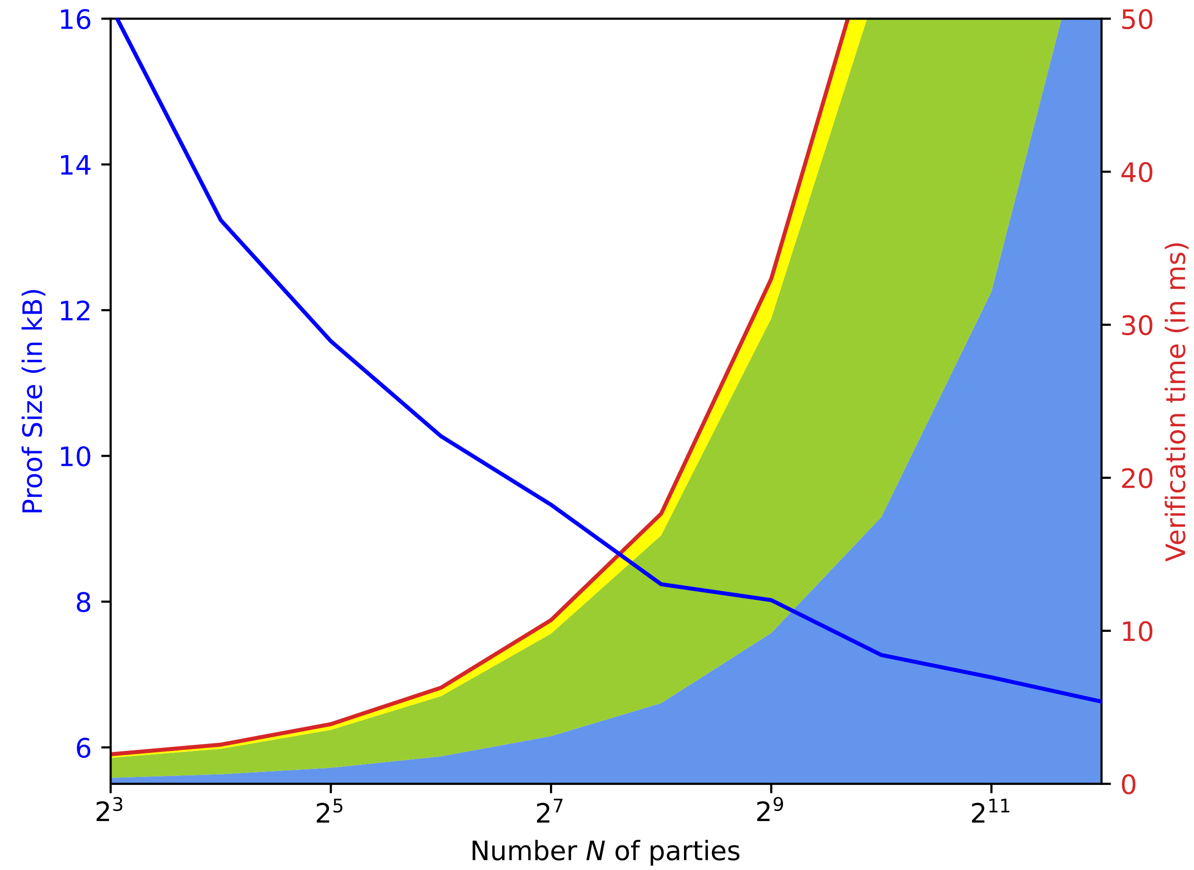
Running times @3.80Ghz

Traditional MPCitH transformation

Signing algorithm



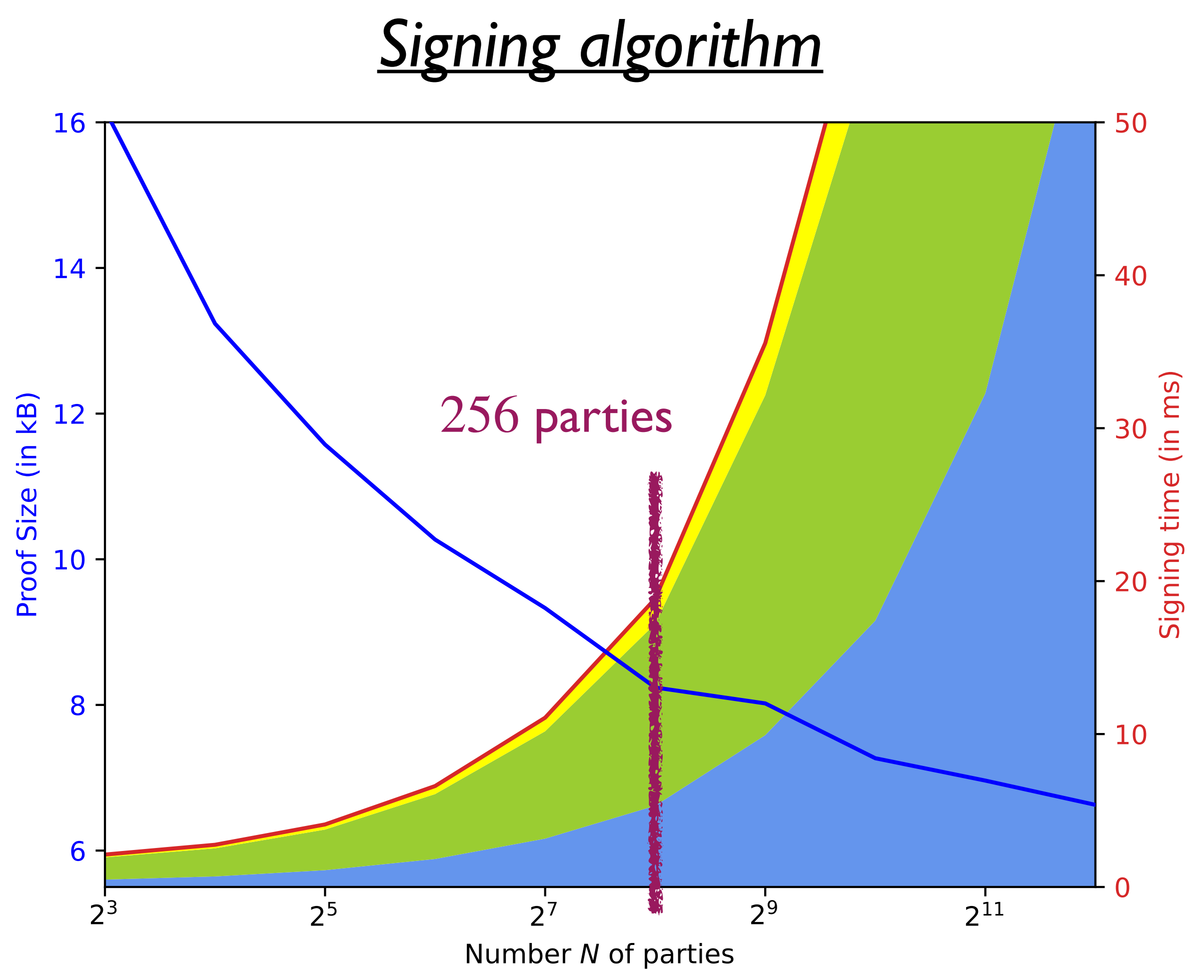
Verification algorithm

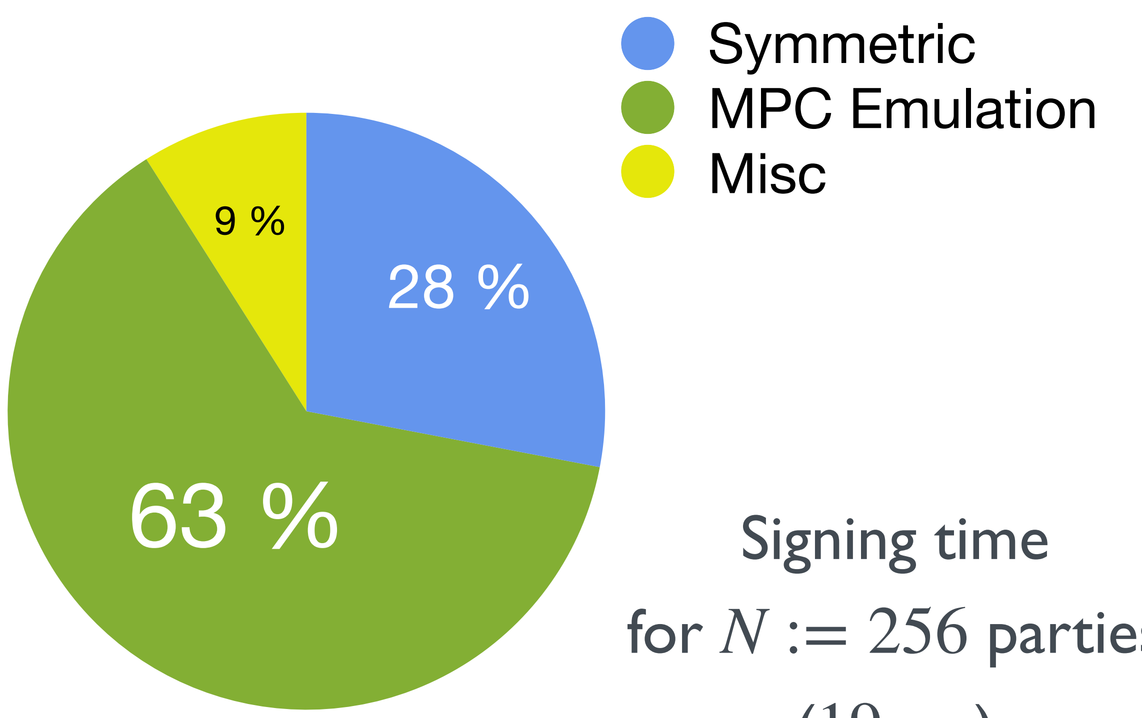


Symmetric
MPC Emulation
Misc

Running times @3.80Ghz

Traditional MPCitH transformation





Signing time
for $N := 256$ parties
(19 ms)

Running times @3.80Ghz

The Hypercube Technique

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

Traditional: N party emulations per repetition

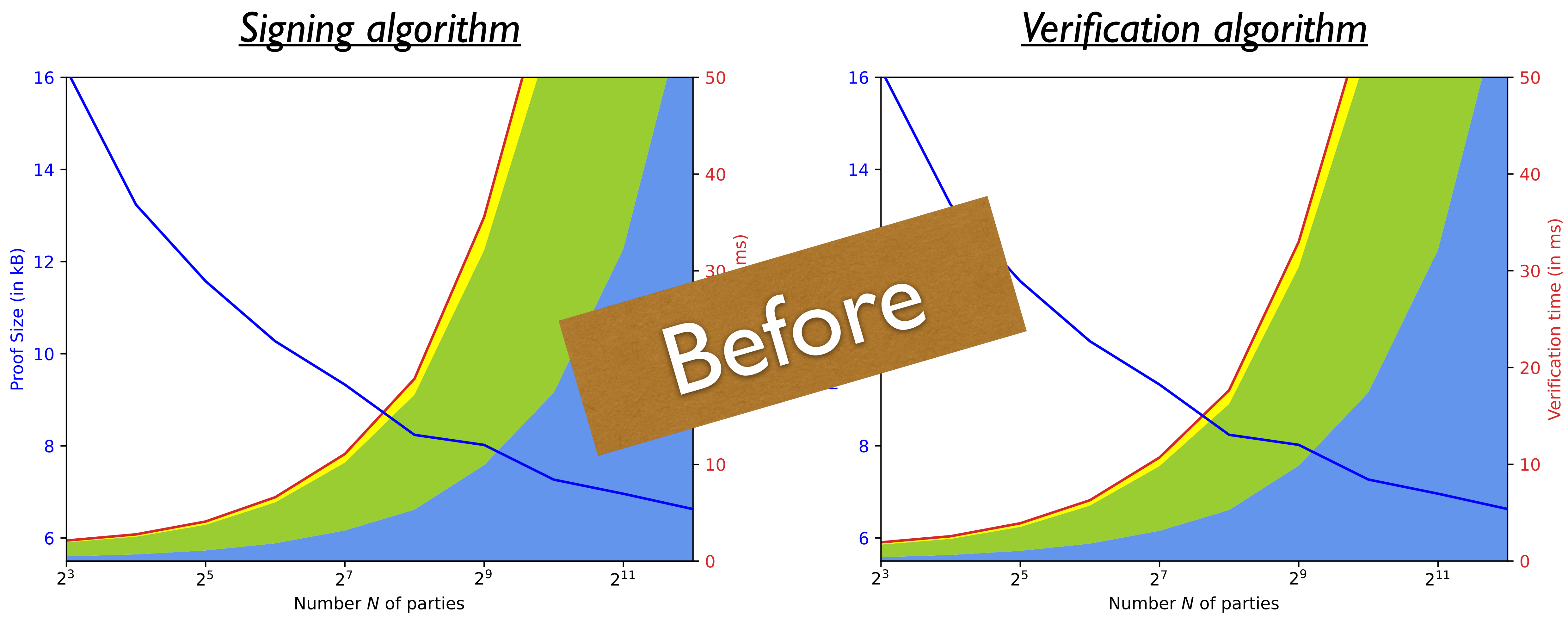
$$N = 256$$

↓

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

$$1 + \log_2 N = 9$$

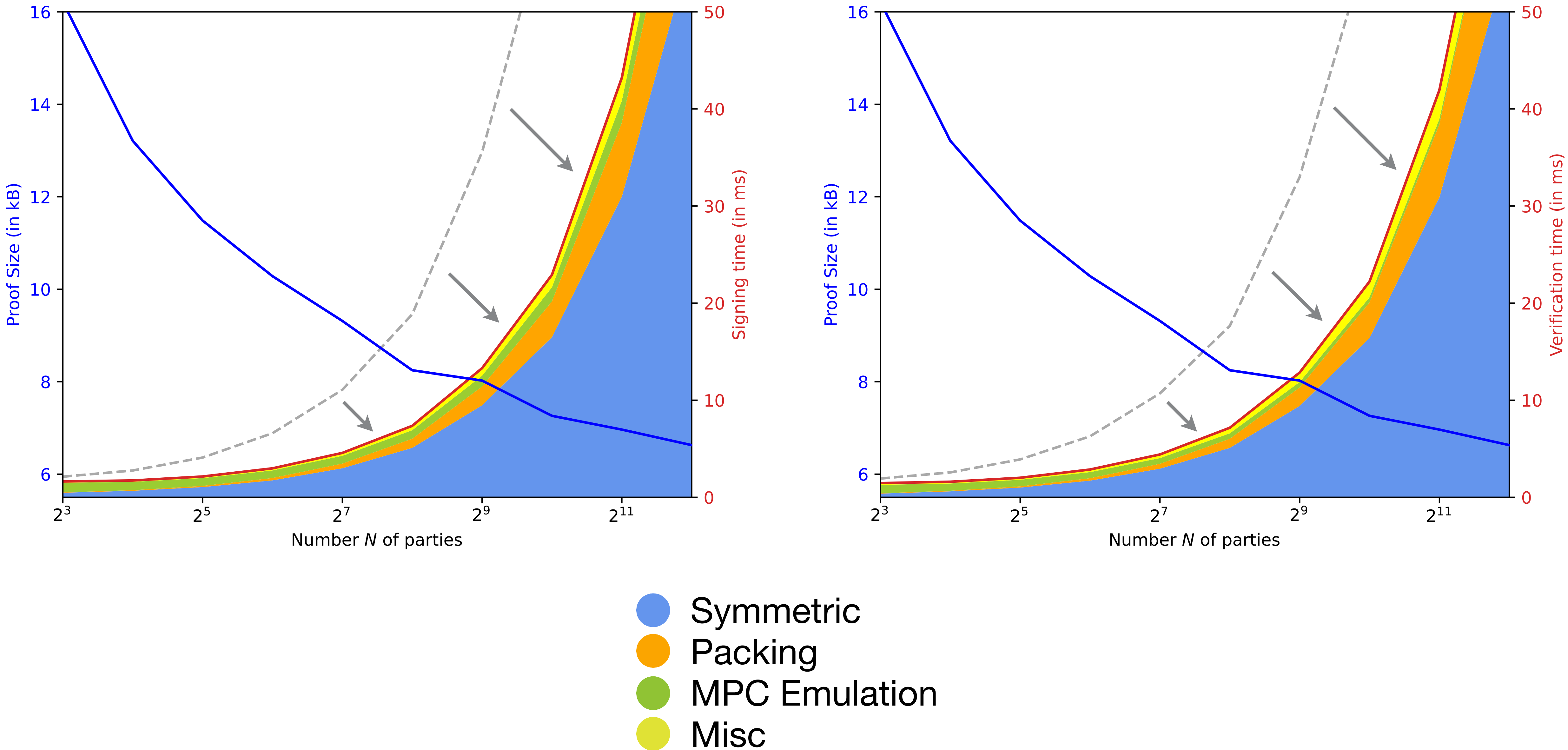
The Hypercube Technique



Running times @3.80Ghz

The Hypercube Technique

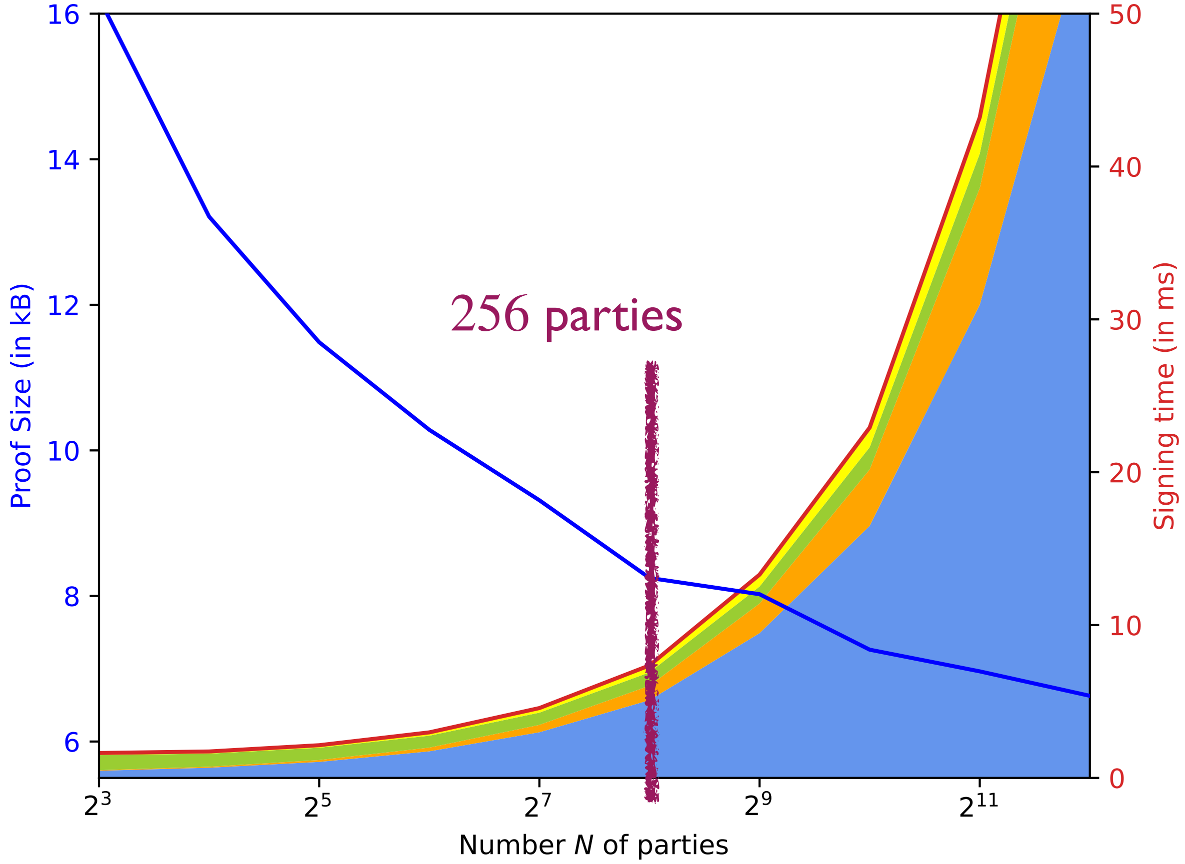
Signing algorithm | *Verification algorithm*

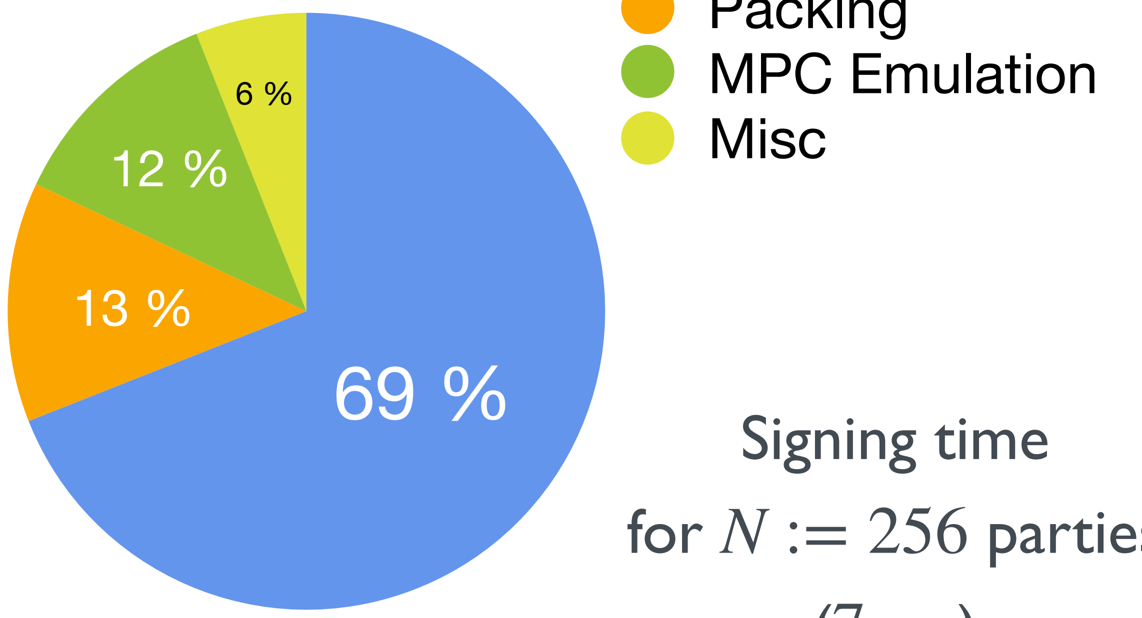


Running times @3.80Ghz

The Hypercube Technique

Signing algorithm





Signing time
for $N := 256$ parties
(7 ms)

Running times @3.80Ghz

The Threshold Approach (Original)

[FR22] Feneuil, Rivain: "Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head" (ePrint 2022/1407)

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

To share a value x,

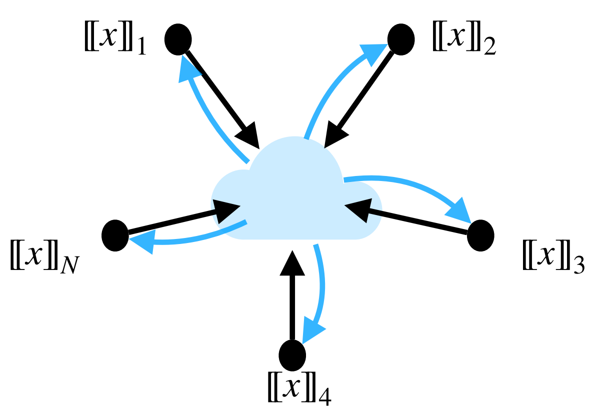
- sample $r_1, r_2, \ldots, r_\ell$ uniformly at random,
- build the polynomial $P(X) = x + \sum_{k=0}^{\ell} r_k \cdot X^k$,
- Set the share $[\![x]\!]_i \leftarrow P(e_i)$, where e_i is publicly known.

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

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

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

← I

$([\![x]\!]_i, \rho_i)_{i \in I}$ →

③ Choose a random set of parties

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

⑤ Check $\forall i \in I$

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

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

Prover

Verifier

MPCitH Transform with Threshold LSSS

① Generate and commit shares

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

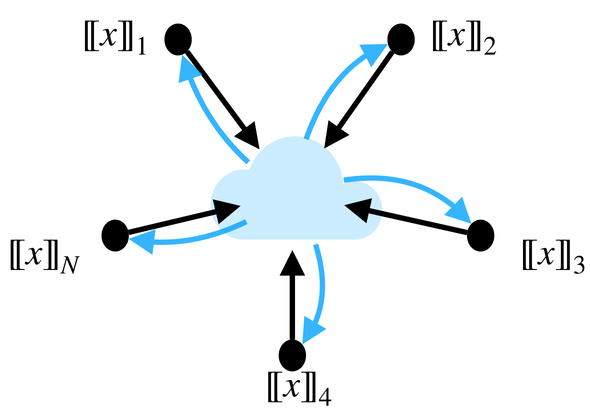
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

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

Threshold LSSS ⇒ 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

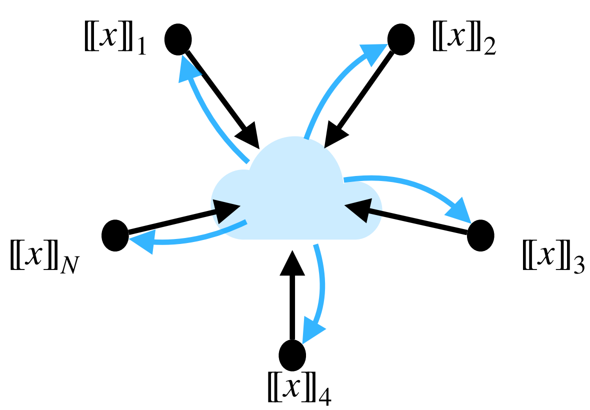
Prover

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

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

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

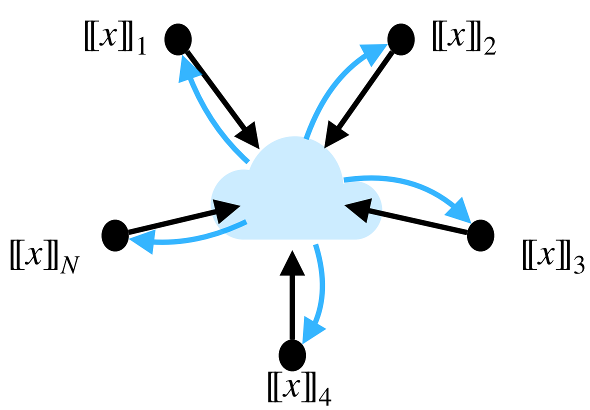
Verifier

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

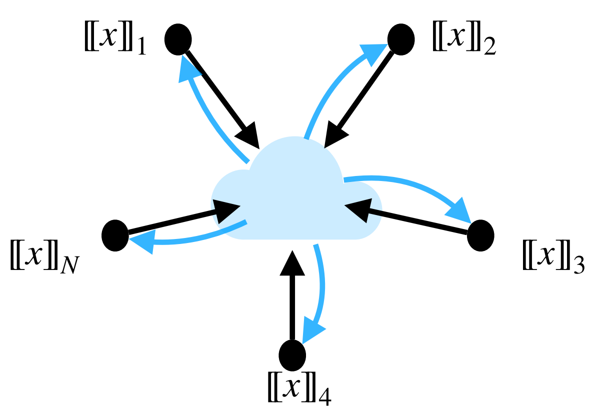
Prover

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
$\Rightarrow$ **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 $g(y, \alpha) = \text{Accept}$

Prover

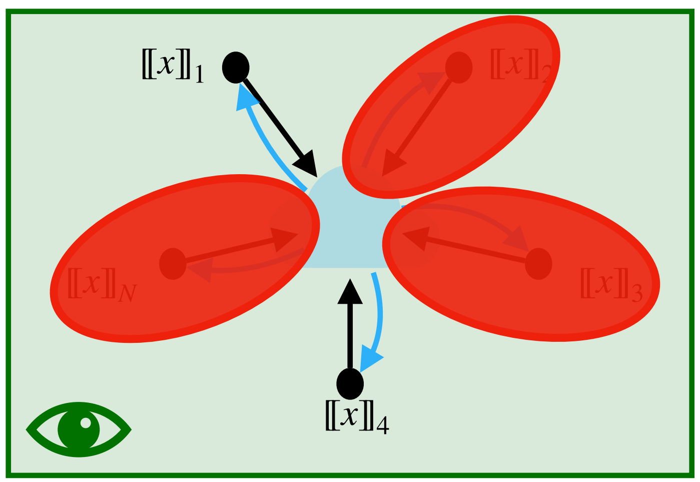
Verifier

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
$\Rightarrow$ **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

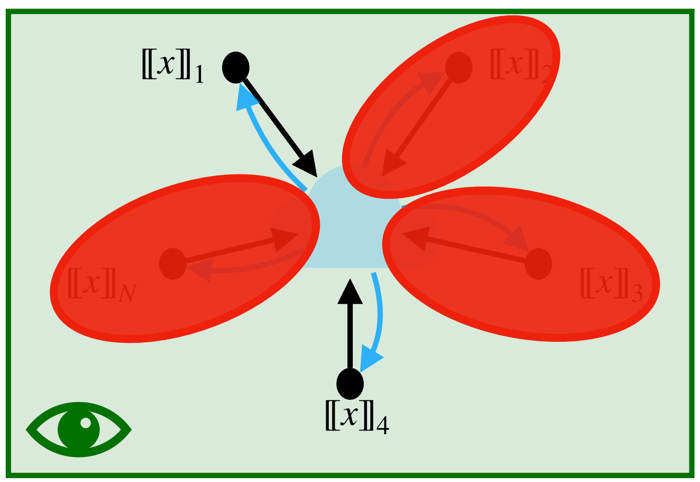
Verifier

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

ℓ parties opened instead of $N - 1$

only ℓ party computations required

The Threshold Approach (Original)

[FR22] Feneuil, Rivain: "Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head" (ePrint 2022/1407)

***Traditional*:** N party emulations per repetition

$$N = 256$$

***Threshold*:** $1 + \ell$ party emulations per repetition

$$1 + \ell = 2$$

The Threshold Approach (Original)

	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 (Original)

	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	$1 + \log_2 N$	$\boxed{2}$
Verifier # party computations	$\log_2 N$	$\boxed{1}$
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

Much cheaper emulation

The Threshold Approach (Original)

	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 (Original)

	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 (Original)

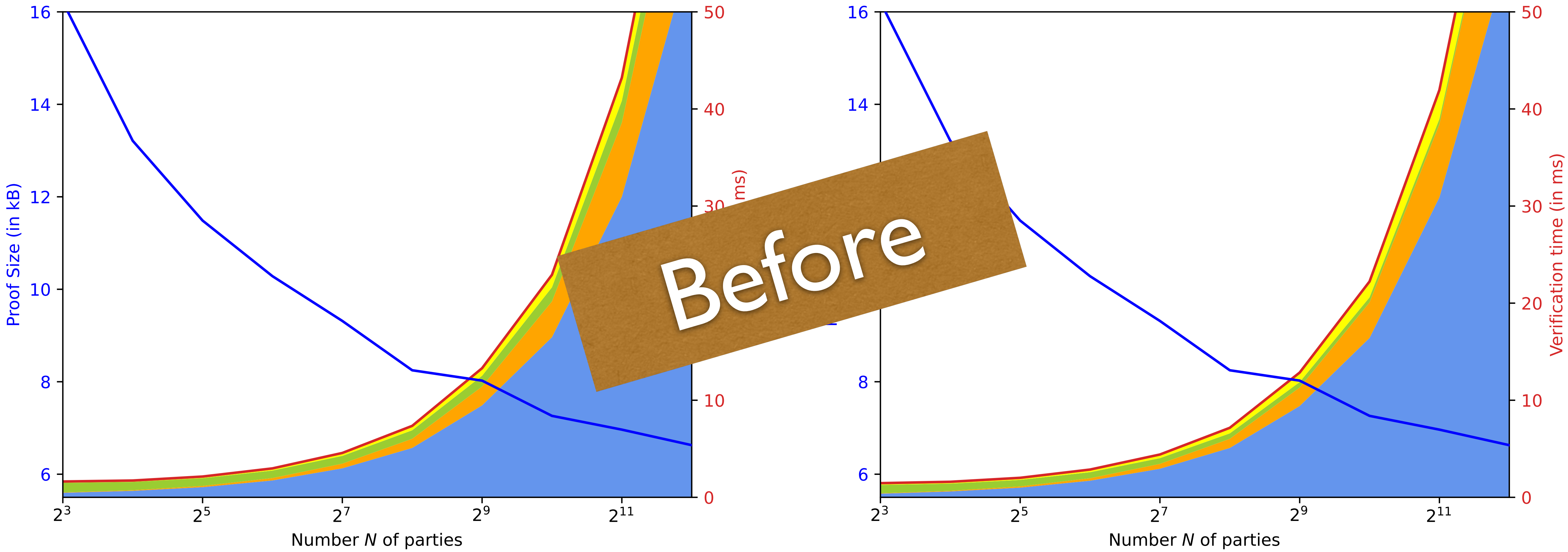
Require $N \leq |\mathbb{F}|$

	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 $2\lambda \cdot \log N$

The Threshold Approach (Original)

Signing algorithm

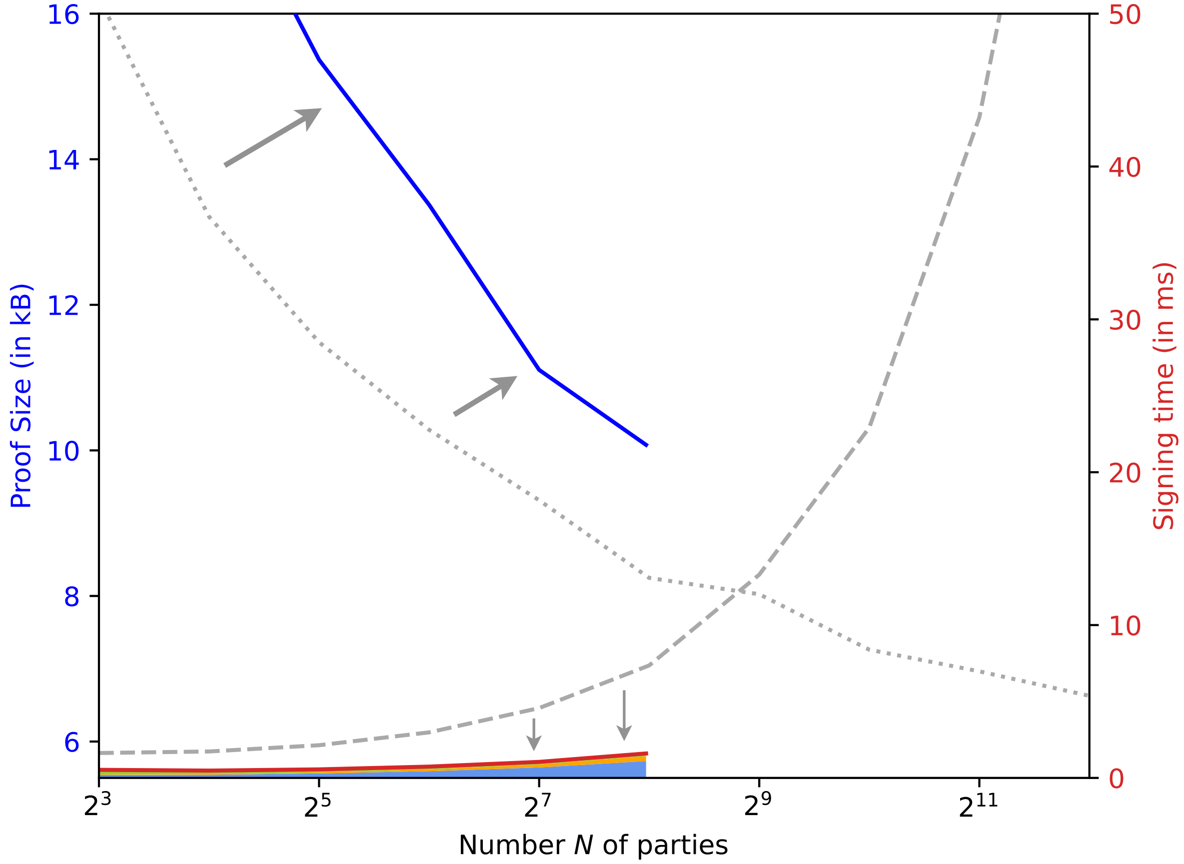
Verification algorithm



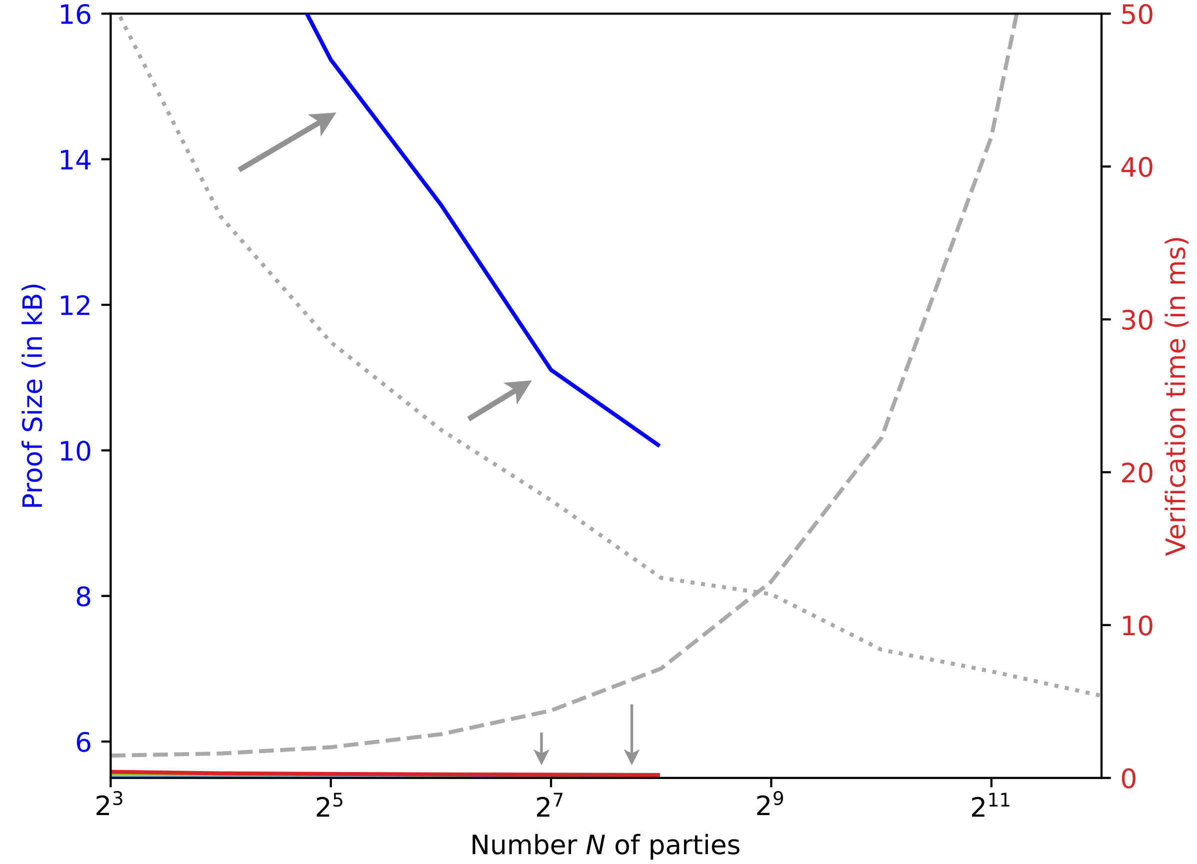
Running times @3.80Ghz

The Threshold Approach (Original)

Signing algorithm



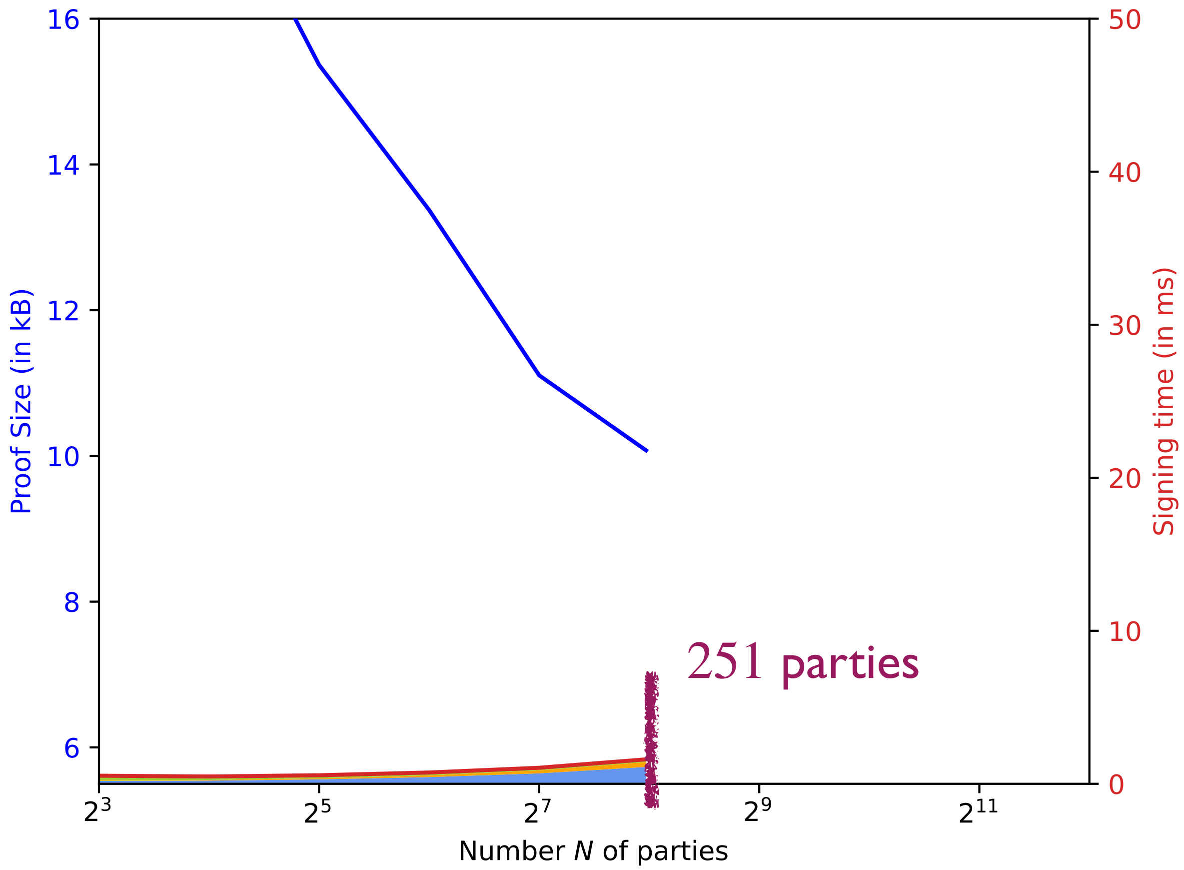
Verification algorithm

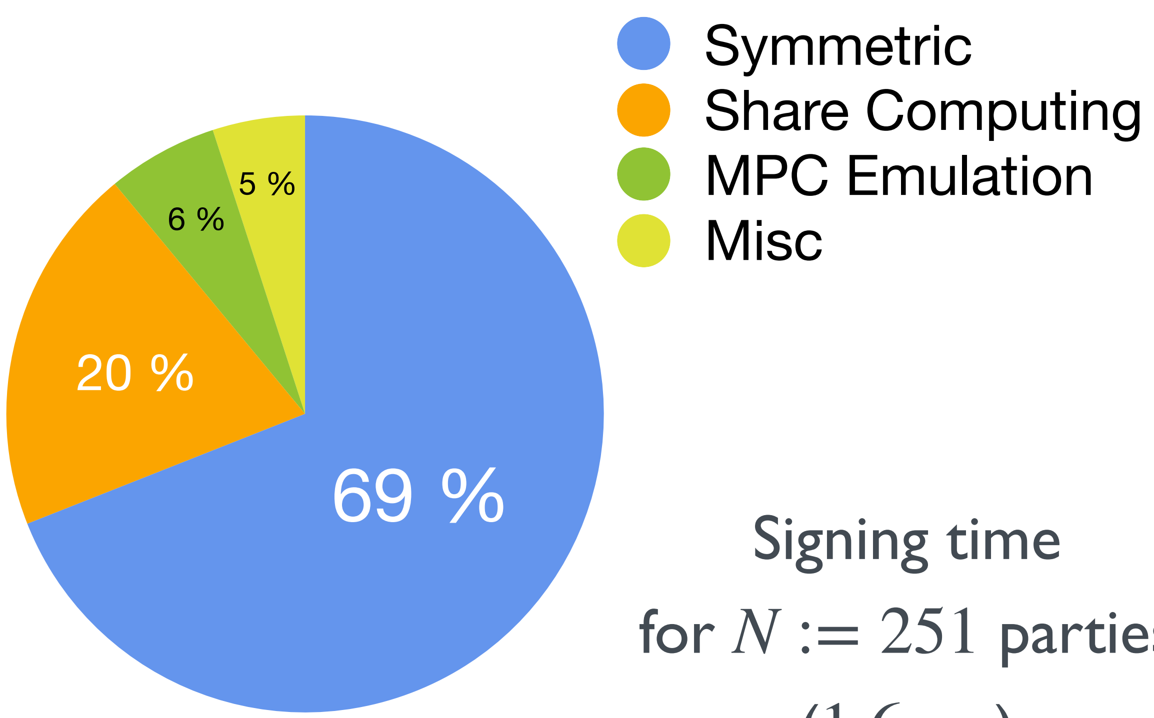


Running times @3.80Ghz

The Threshold Approach (Original)

Signing algorithm

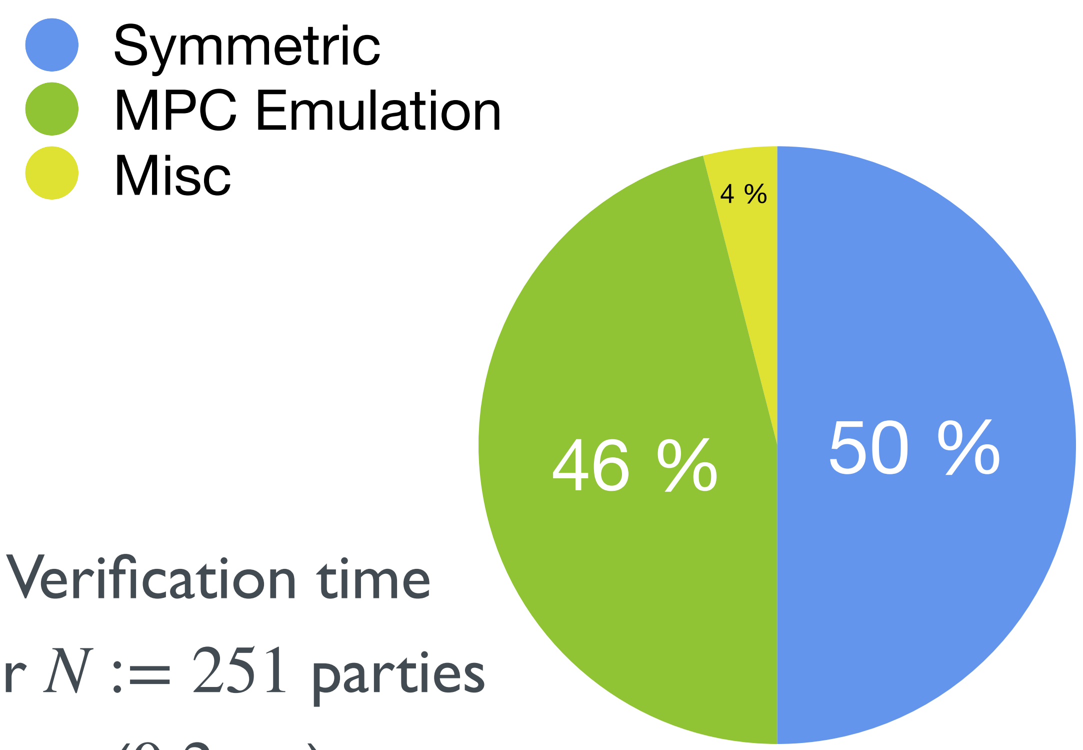




Signing time
for $N := 251$ parties
(1.6 ms)

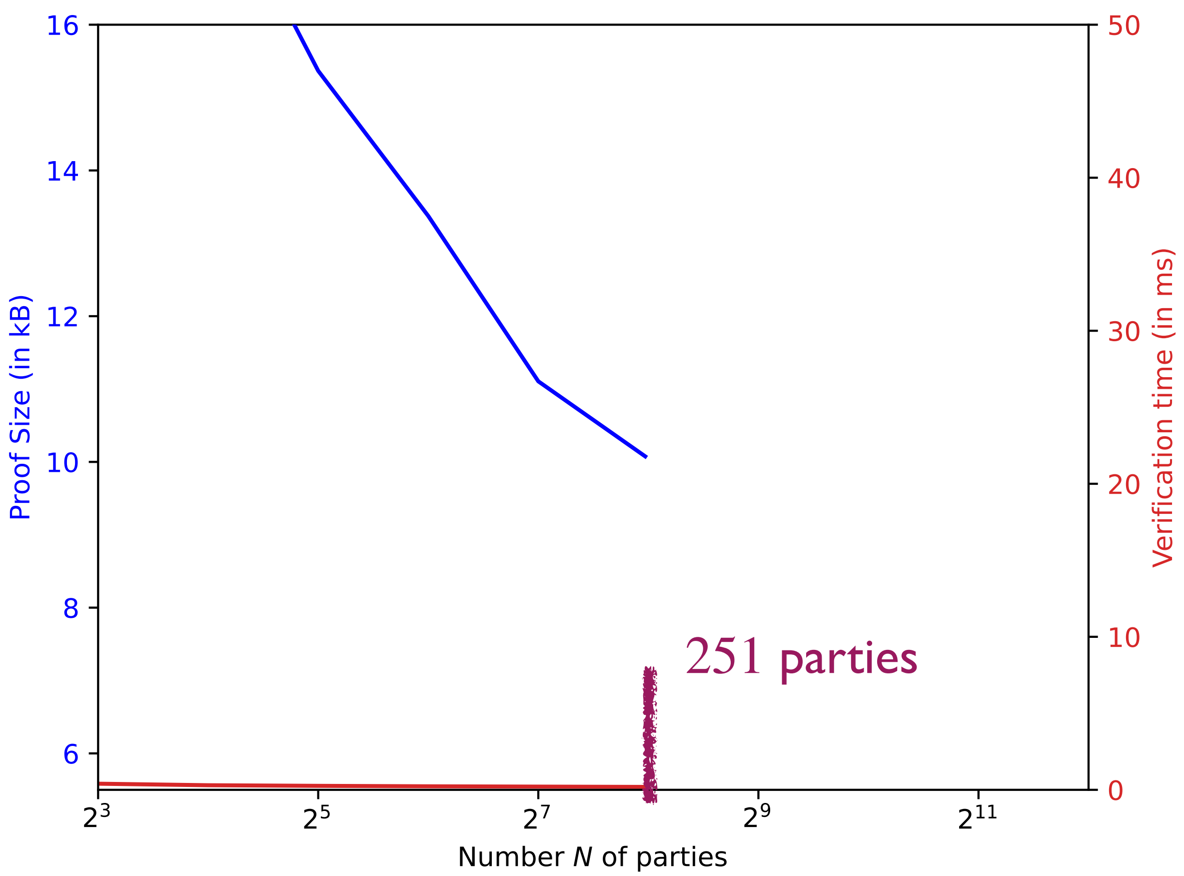
Running times @3.80Ghz

The Threshold Approach (Original)



Verification time
for $N := 251$ parties
(0.2 ms)

Verification algorithm



Running times @3.80Ghz

Traditional Transformation

(2018) Emulation : N parties

Traditional Transformation

(2018) Emulation : N parties

Hypercube Technique

(2022)

$1 + \log_2 N$ parties

No communication penalty

[AGHHJY23] Aguilar-Melchor, Gama, Howe, Hülsing, Joseph, Yue: "The Return of the SDitH" (ePrint 2022/1645, Eurocrypt 2023)

Traditional Transformation (2018)

Emulation : N parties

Hypercube Technique (2022)

$1 + \log_2 N$ parties

No communication penalty

Shamir's secret sharings

Threshold Approach (2022)

$1 + \ell$ parties

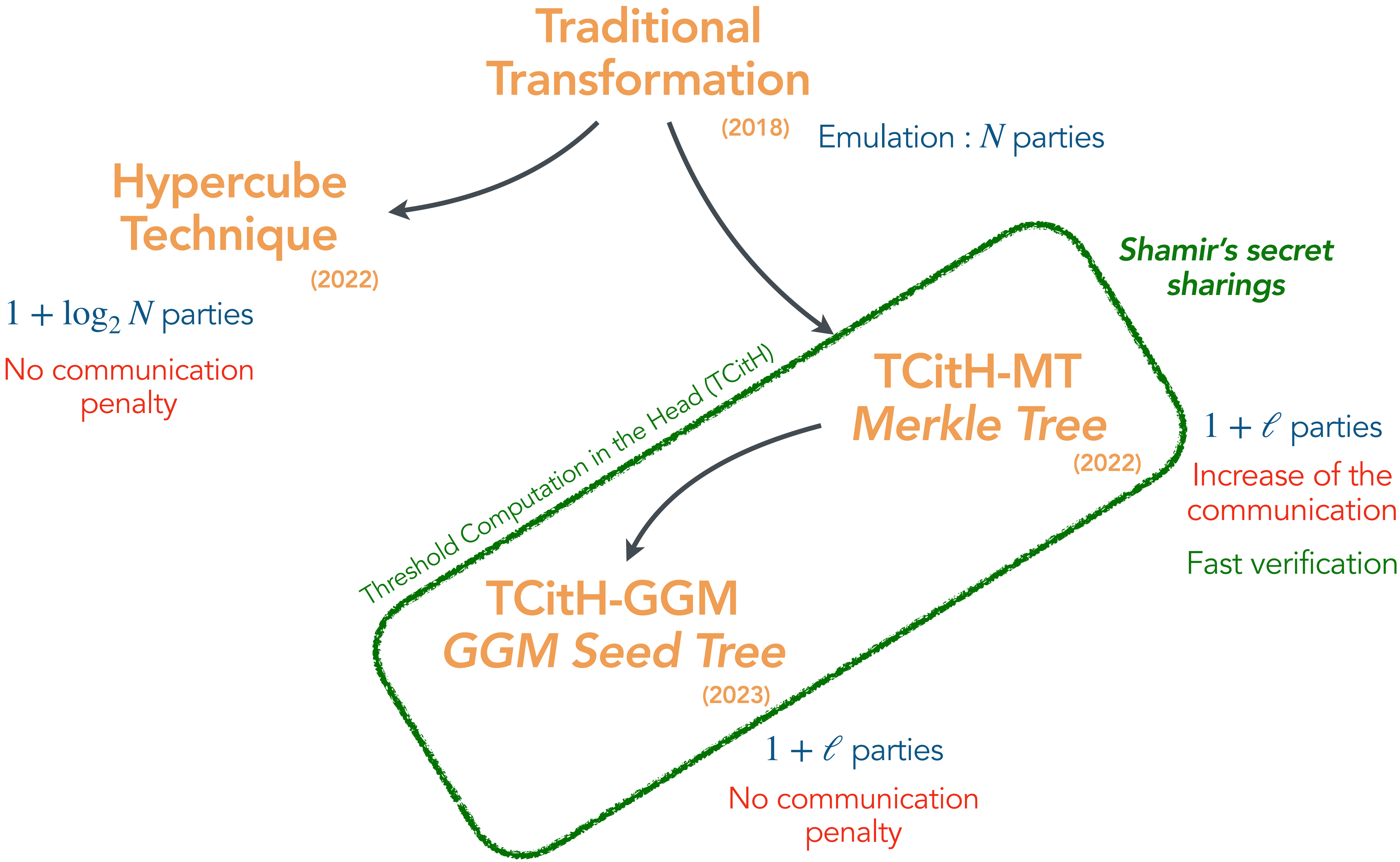
Increase of the communication

Fast verification

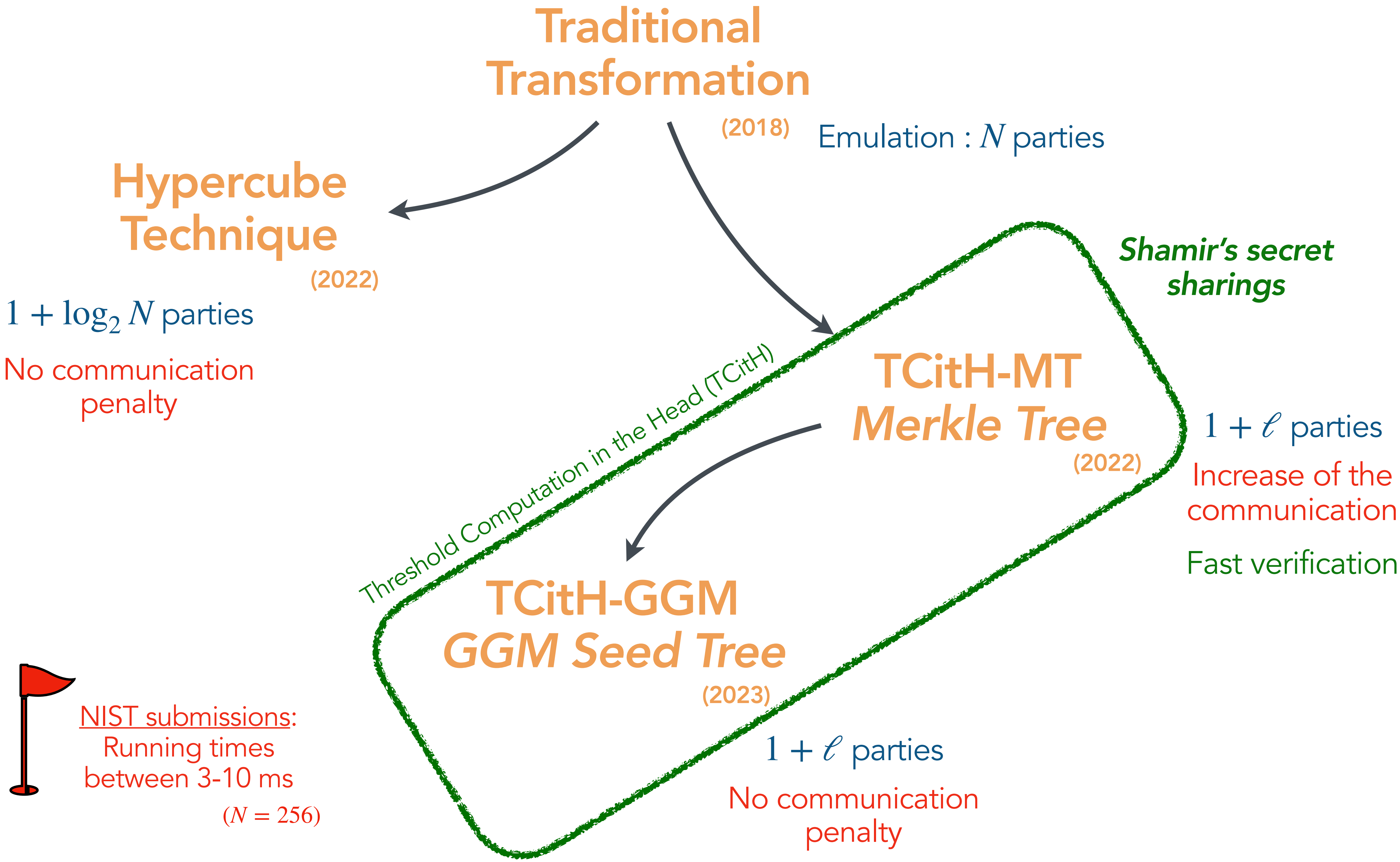
Shamir's secret sharing: to share a value s,

- Build a random degree-ℓ polynomial $P(X) := s + \sum_{j=1}^{\ell} r_j X^j$.
- Set the i^{th} share $[\![s]\!]_i$ as $[\![s]\!]_i := P(e_i)$, where $e_i \neq 0$.

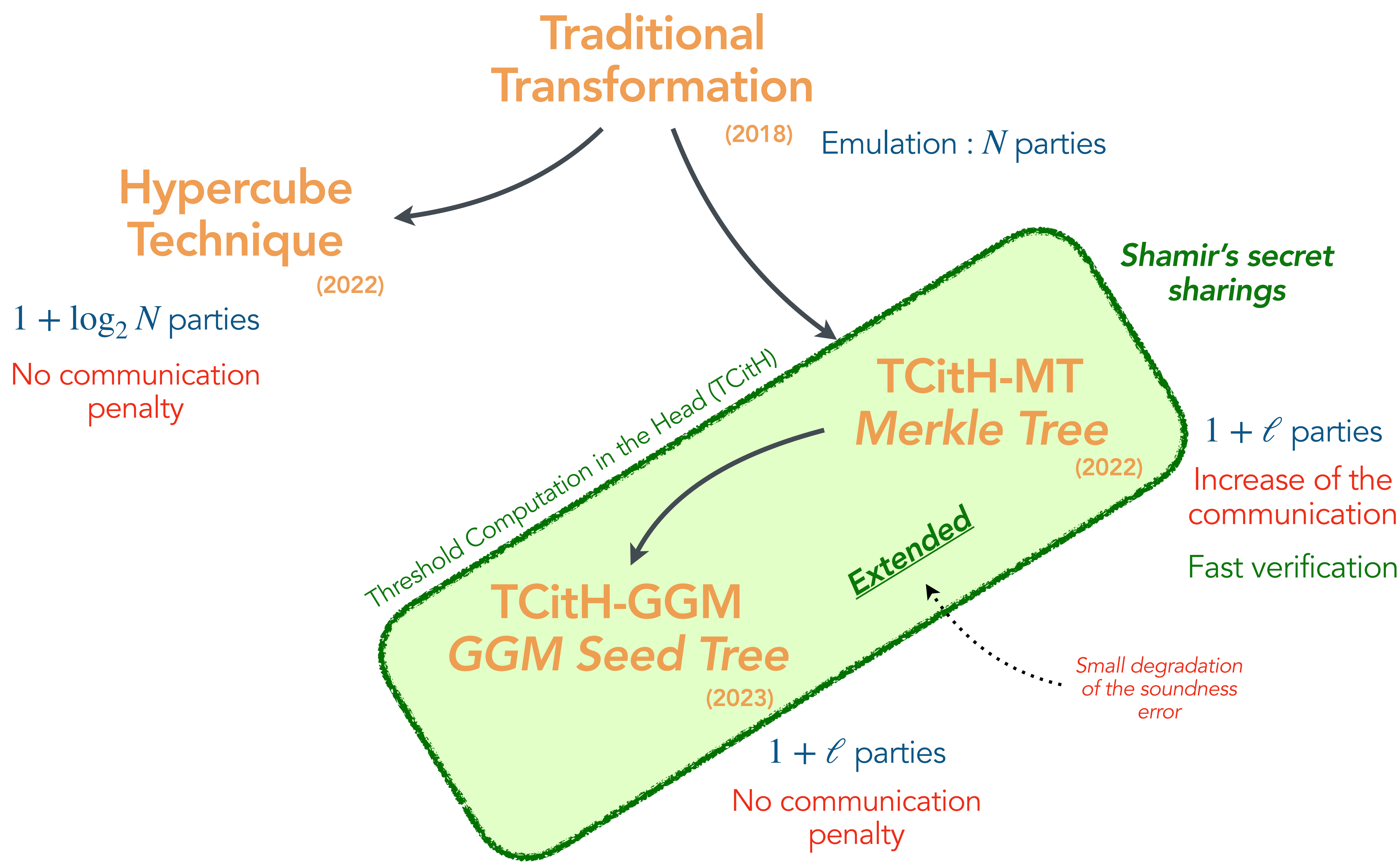
[FR22] Feneuil, Rivain: "Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head" (ePrint 2022/1407, Asiacrypt 2023)



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



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



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

Extended TCitH: some applications

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

Extended TCitH: some applications

- More efficient signature schemes
  - ***Unstructured multivariate quadratic (MQ) problem over $\mathbb{F}_{251}$***
    - MQOM: **6.5 KB**
    - Extended TCitH: **4.2 KB**

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

Extended TCitH: some applications

- More efficient signature schemes
  - ***Unstructured multivariate quadratic (MQ) problem over*** $\mathbb{F}_{251}$
    - MQOM: **6.5 KB**
    - Extended TCitH: **4.2 KB**

- Shorter post-quantum ring signature schemes
  - Extended TCitH with MQ: **5.8 KB** in around 8 ms, for 4000 users
  - Extended TCitH with SD: **10.30 KB** in around 10 ms, for 4000 users

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

Conclusion

Conclusion

- The MPC-in-the-Head framework is an active research field
  - Invented in **2007**
  - More and more popular since **2016** (first practical scheme)
    - Picnic: MPCitH-based signature in the first NIST call

Conclusion

- The MPC-in-the-Head framework is an active research field
  - Invented in **2007**
  - More and more popular since **2016** (first practical scheme)
    - Picnic: MPCitH-based signature in the first NIST call
  - **2016-2024**: shorter proof sizes
    - In 2016, the signature sizes was larger than 30 KB
    - Currently, the signature sizes are around 3–7 KB

Conclusion

- The MPC-in-the-Head framework is an active research field
  - Invented in **2007**
  - More and more popular since **2016** (first practical scheme)
    - Picnic: MPCitH-based signature in the first NIST call
  - **2016-2024**: shorter proof sizes
    - In 2016, the signature sizes was larger than 30 KB
    - Currently, the signature sizes are around 3–7 KB
  - **2022-2024**: faster schemes
    - Before 2022, we needed to emulate all the MPC parties
    - Currently, we just need to emulate a small value of parties
    - The computational bottleneck are becoming the symmetric part of the scheme, but some works are trying to mitigate it.

Conclusion

- A *versatile* tool to build signature schemes:
  - 7 NIST submissions relying on it in the new NIST call
  - Very competitive when focusing on minimizing

    Signature size + Public key size

    - Medium signature sizes (4-10 kilobytes)
    - Short public key ($\leq$ 200 bytes)
  - Transversal among the hardness assumptions
  - Can be convenient to build advanced signature schemes

Conclusion

- A *versatile* tool to build signature schemes:
  - 7 NIST submissions relying on it in the new NIST call
  - Very competitive when focusing on minimizing

    Signature size + Public key size

    - Medium signature sizes (4-10 kilobytes)
    - Short public key ($\leq$ 200 bytes)
  - Transversal among the hardness assumptions
  - Can be convenient to build advanced signature schemes

Thank you for your attention !