Constructions for digital signature Part I: Introduction to MPC-in-the-Head

Thibauld Feneuil

NIST PQC Seminar

May 21, 2024, online

CryptoExperts
WE INNOVATE TO SECURE YOUR BUSINESS

Table of Contents

Introduction

How to build signature schemes?

Hash & Sign

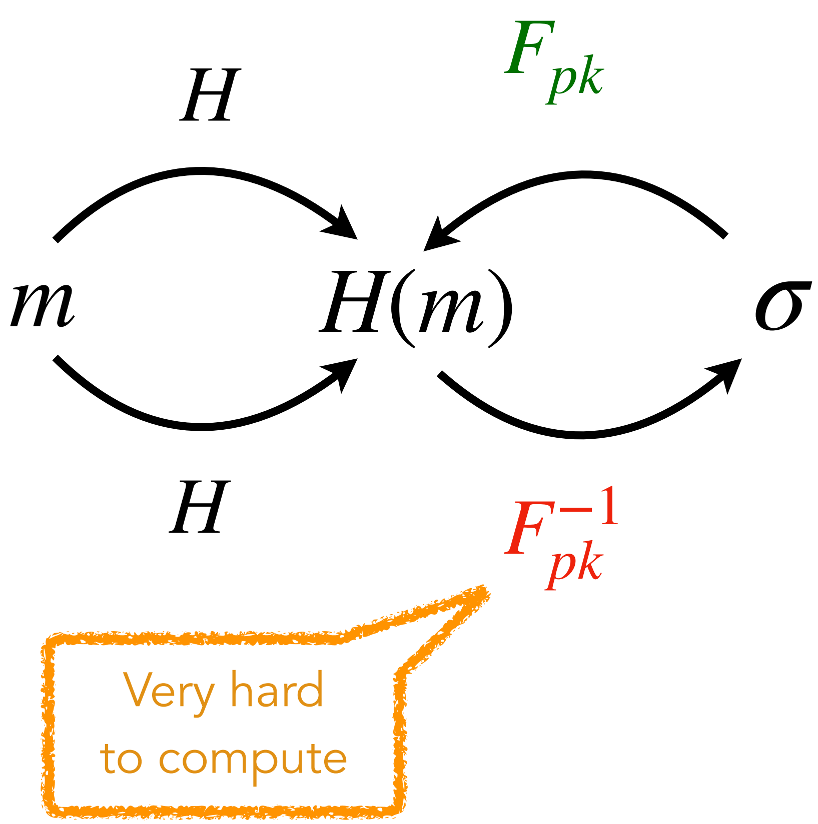
$$
\begin{tikzcd}
m \arrow[r, bend left, "H"] \arrow[r, bend right, "H"'] & H(m) \arrow[r, bend right, "F_{pk}^{-1}"'] & \sigma \arrow[l, bend right, "F_{pk}"']
\end{tikzcd}
$$

Very hard
to compute

- Short signatures
- “Trapdoor” in the public key

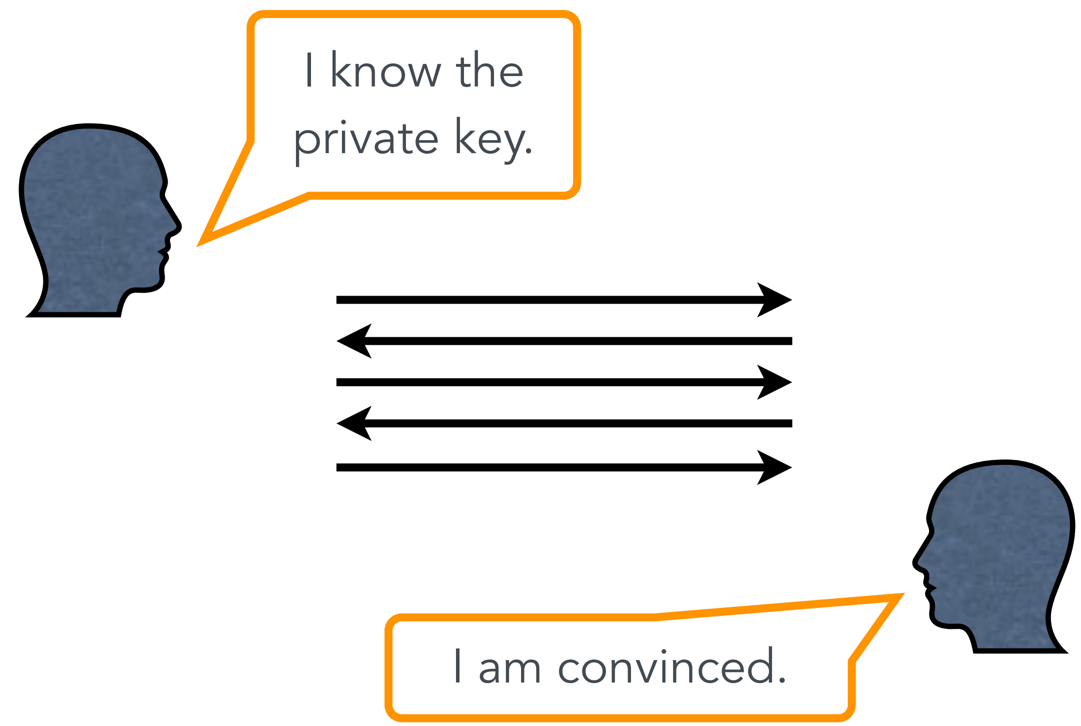
How to build signature schemes?

Hash & Sign



- Short signatures
- “Trapdoor” in the public key

From an identification scheme



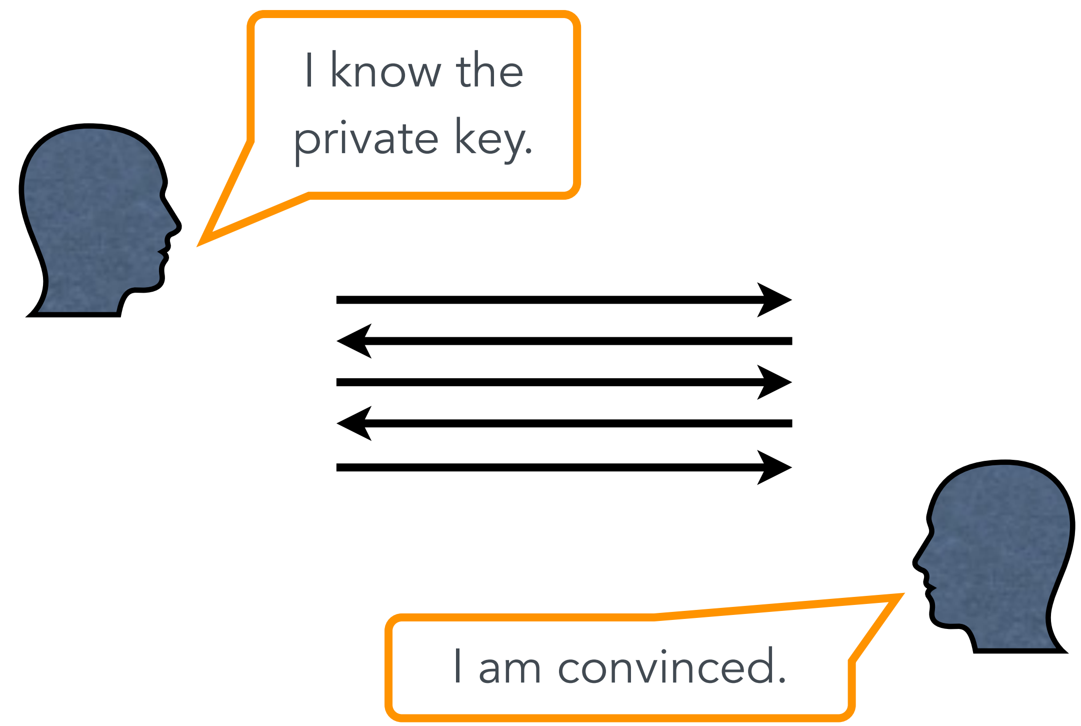
- Large(r) signatures
- Short public key

How to build signature schemes?

Hash & Sign

- Short signatures
- "Trapdoor" in the public key

From an identification scheme

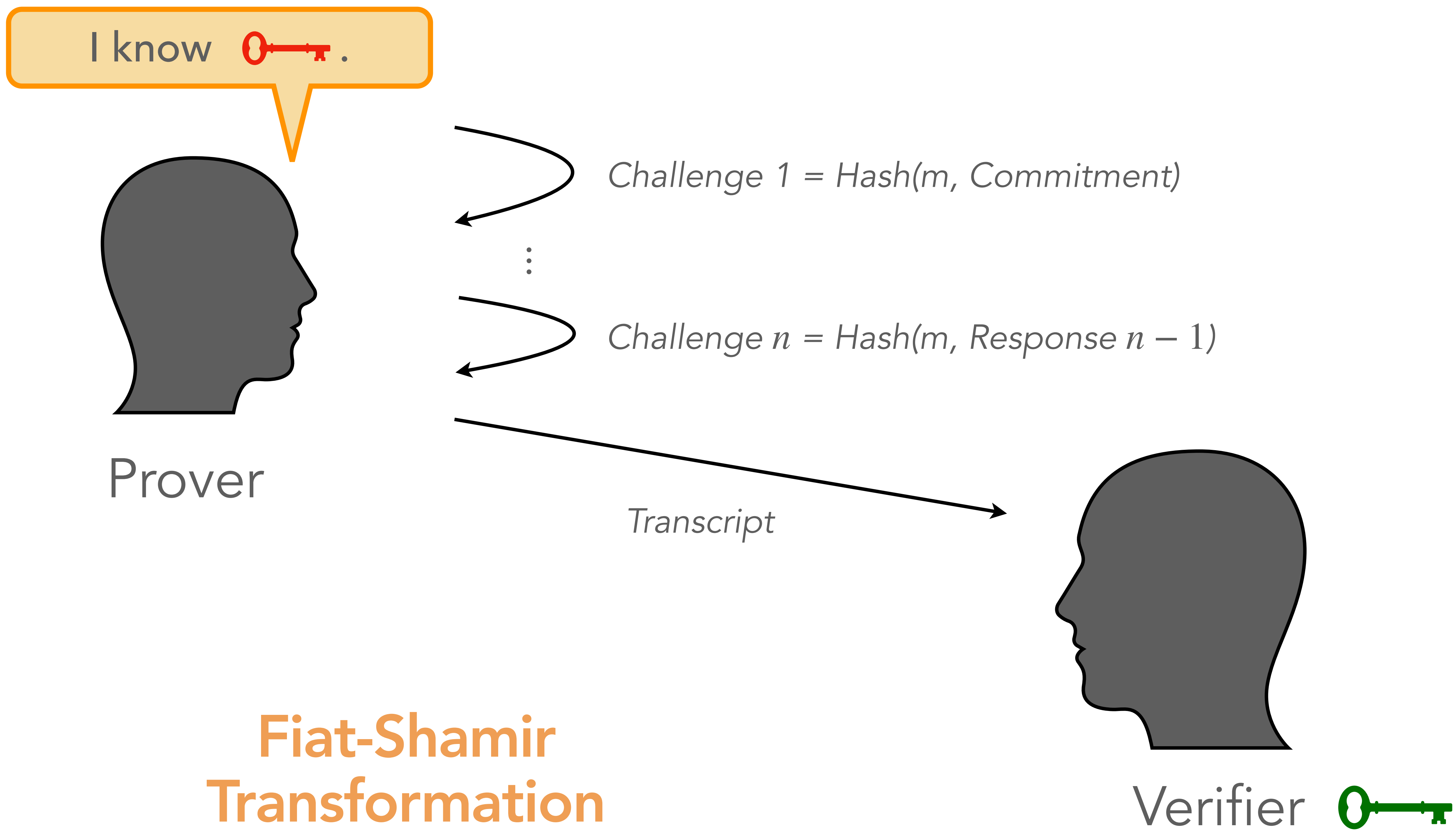


- Large(r) signatures
- Short public key

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



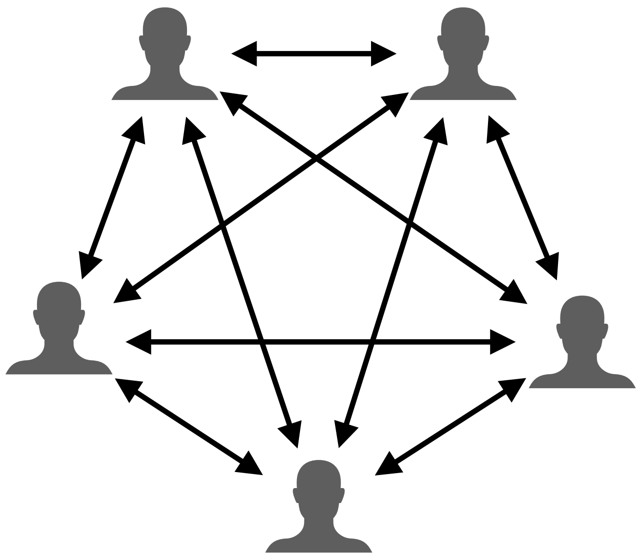
Fiat-Shamir Transformation

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 / zero-knowledge proof of knowledge

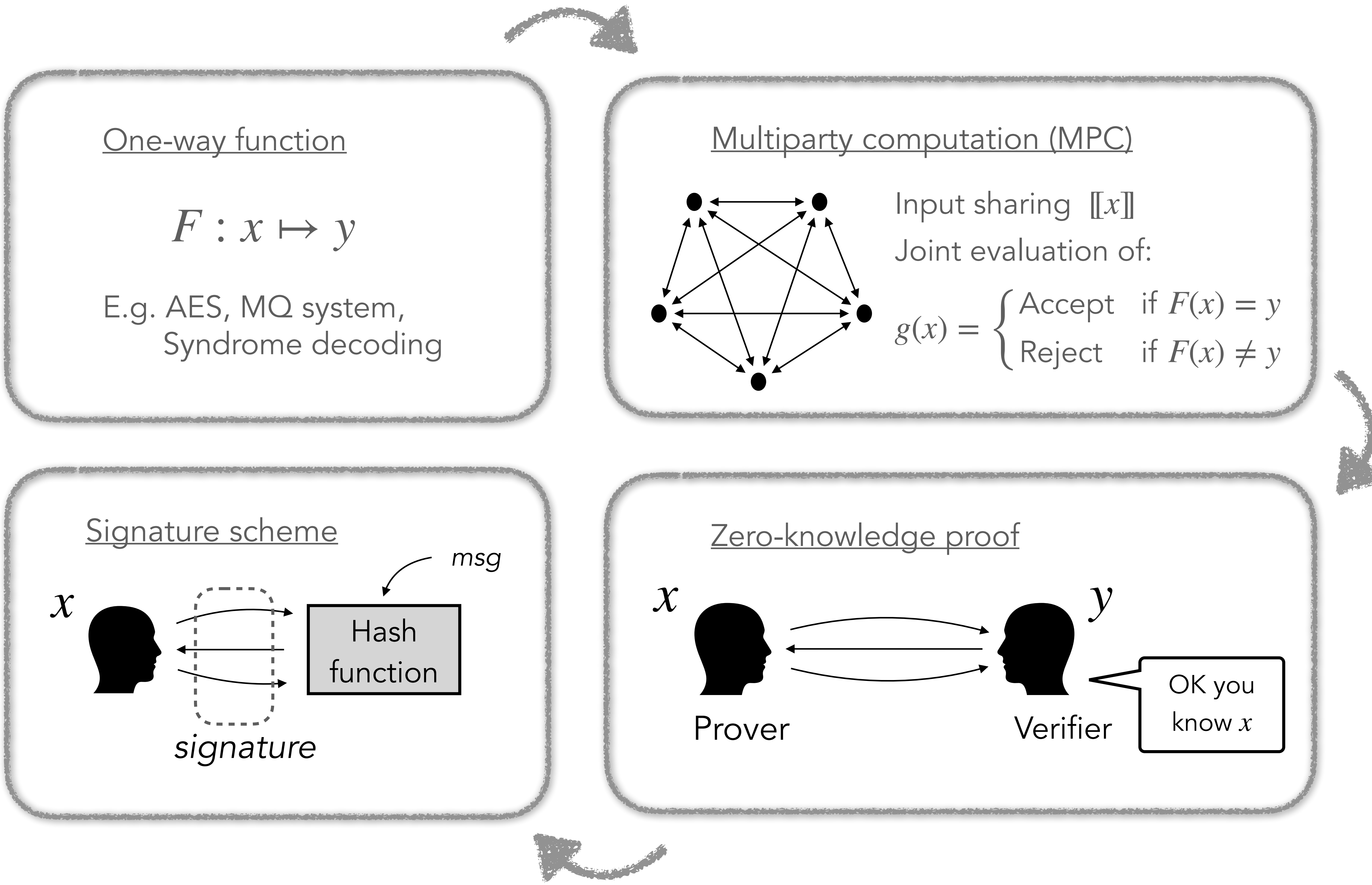


- **Generic**: can be applied to any cryptographic problem

MPC in the Head

- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)
- Convenient to build (candidate) **post-quantum signature** schemes
- **Picnic**: submission to NIST (2017)
- First round of recent NIST call: 7~9 MPCitH schemes / 40 submissions

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



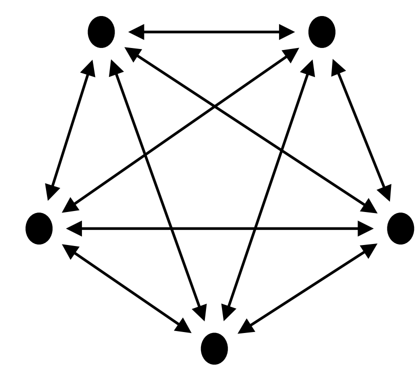
One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Signature scheme
x
msg
Hash function
signature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

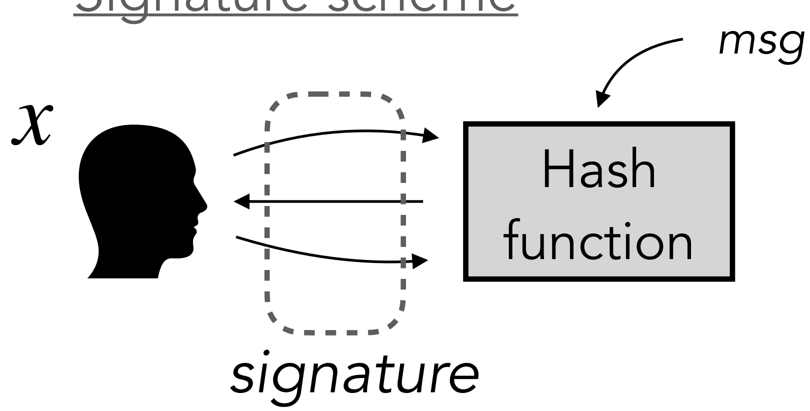


Input sharing $[\![x]\!]$

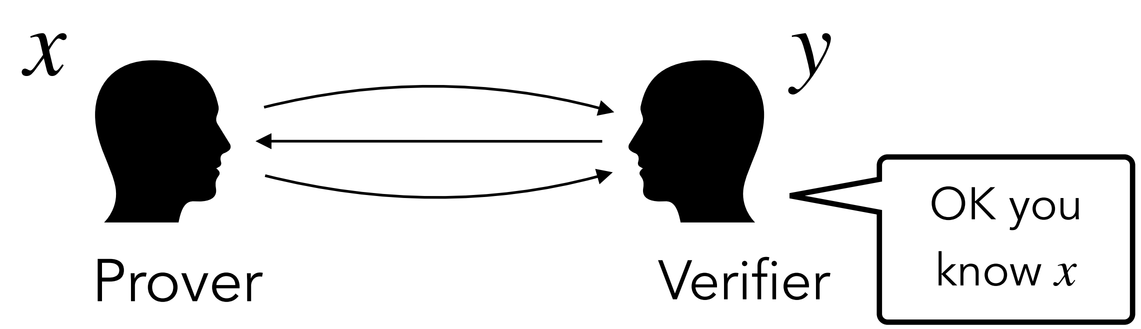
Joint evaluation of:

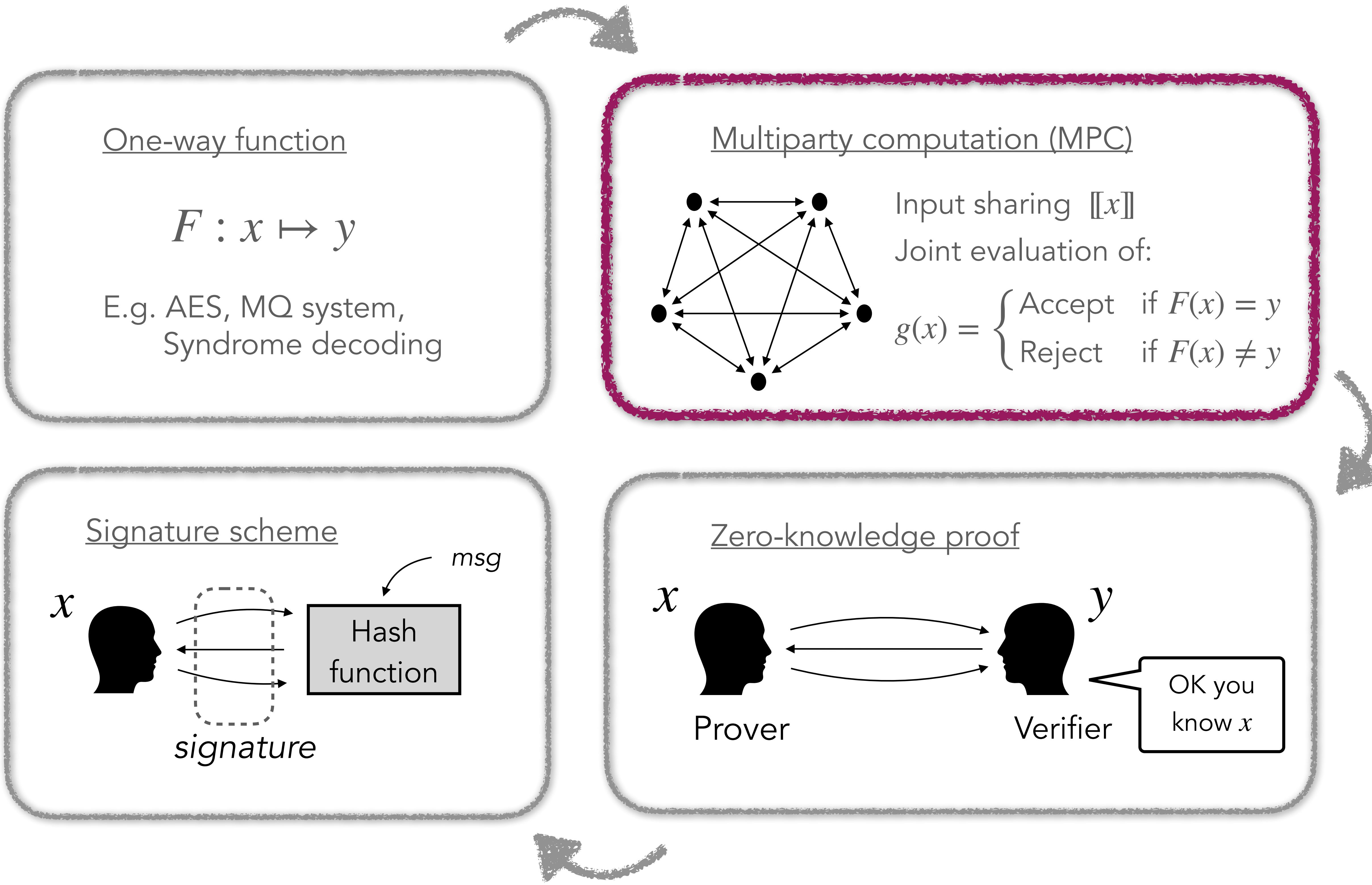
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

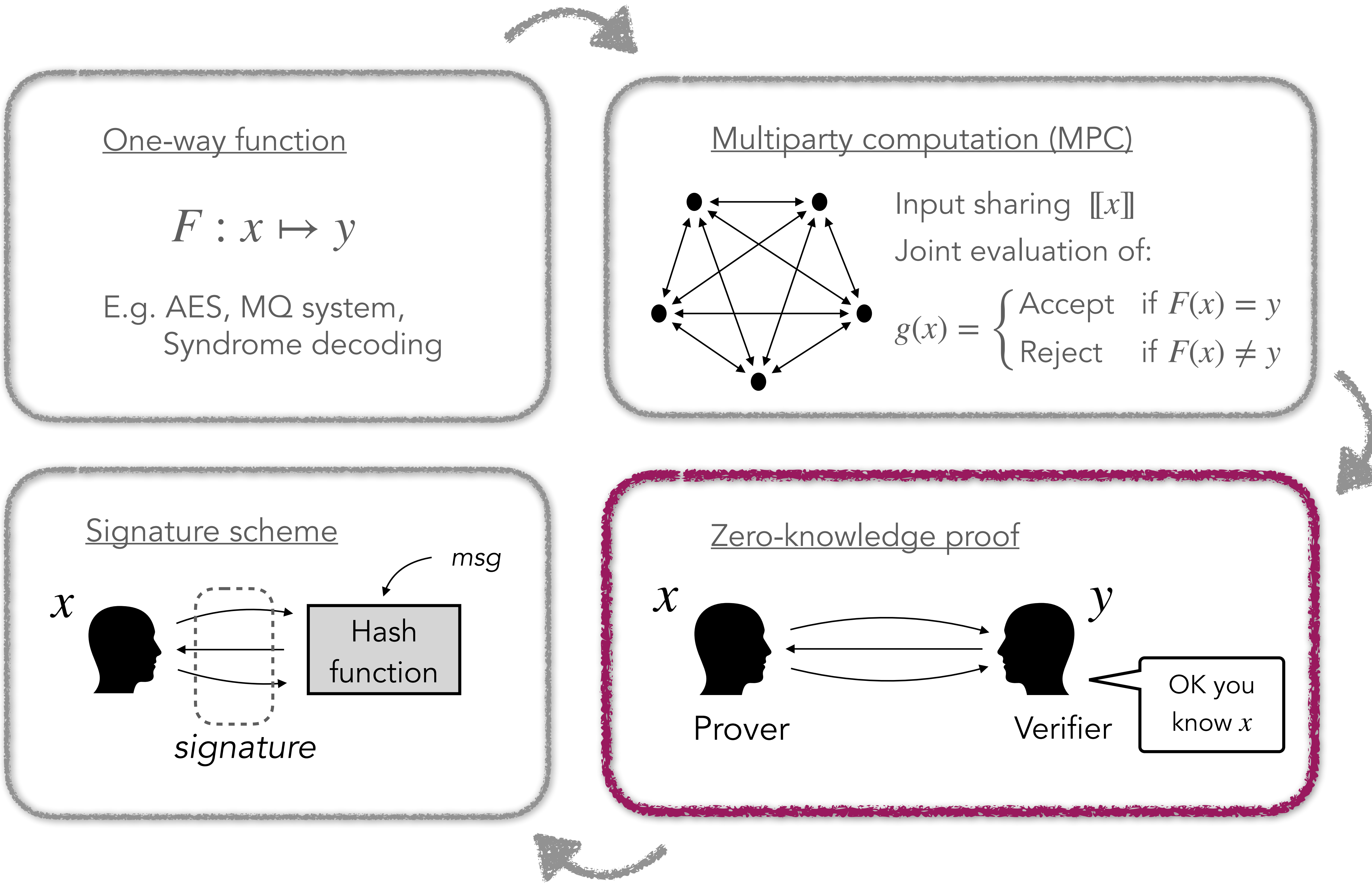
Signature scheme

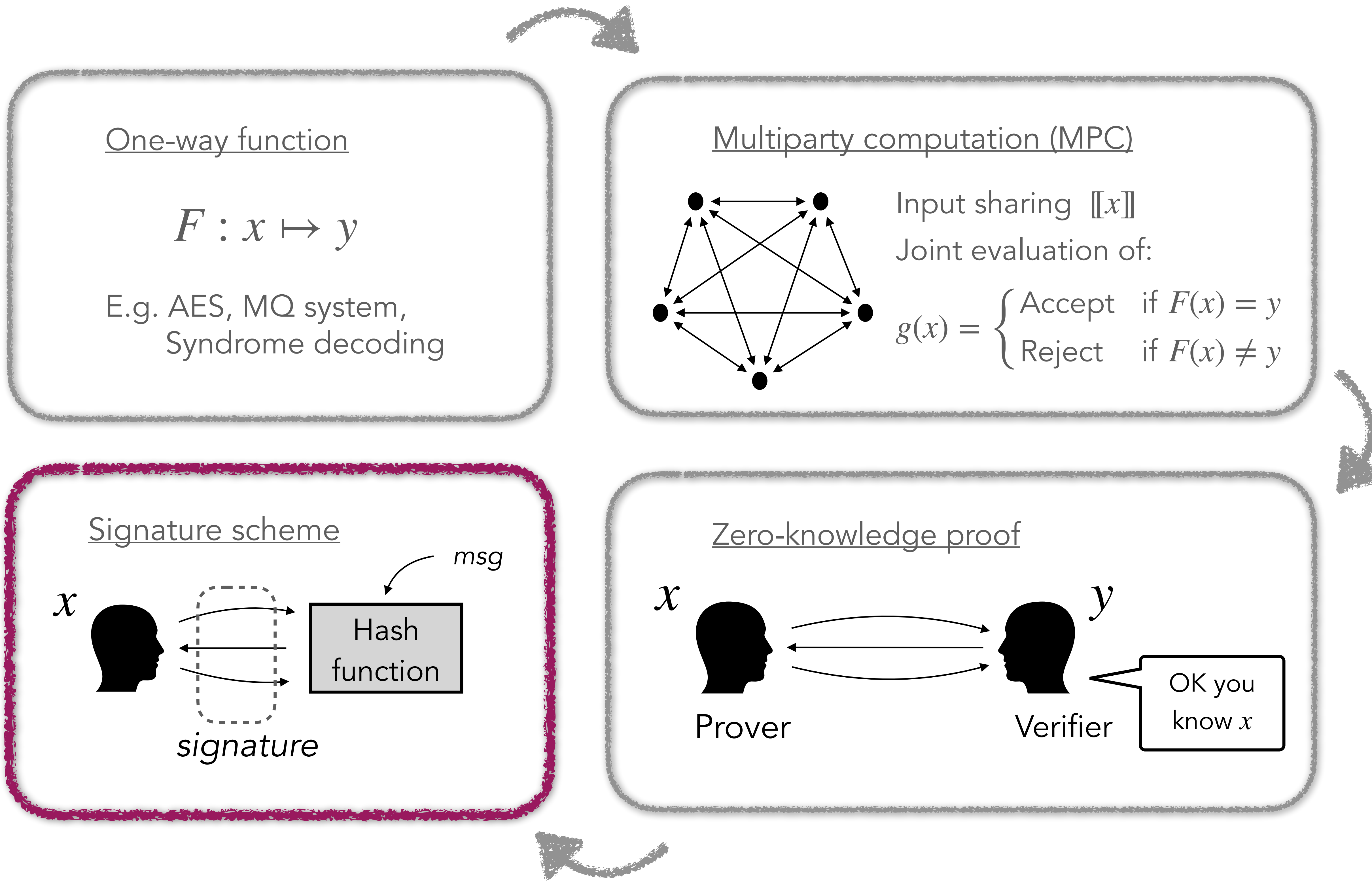


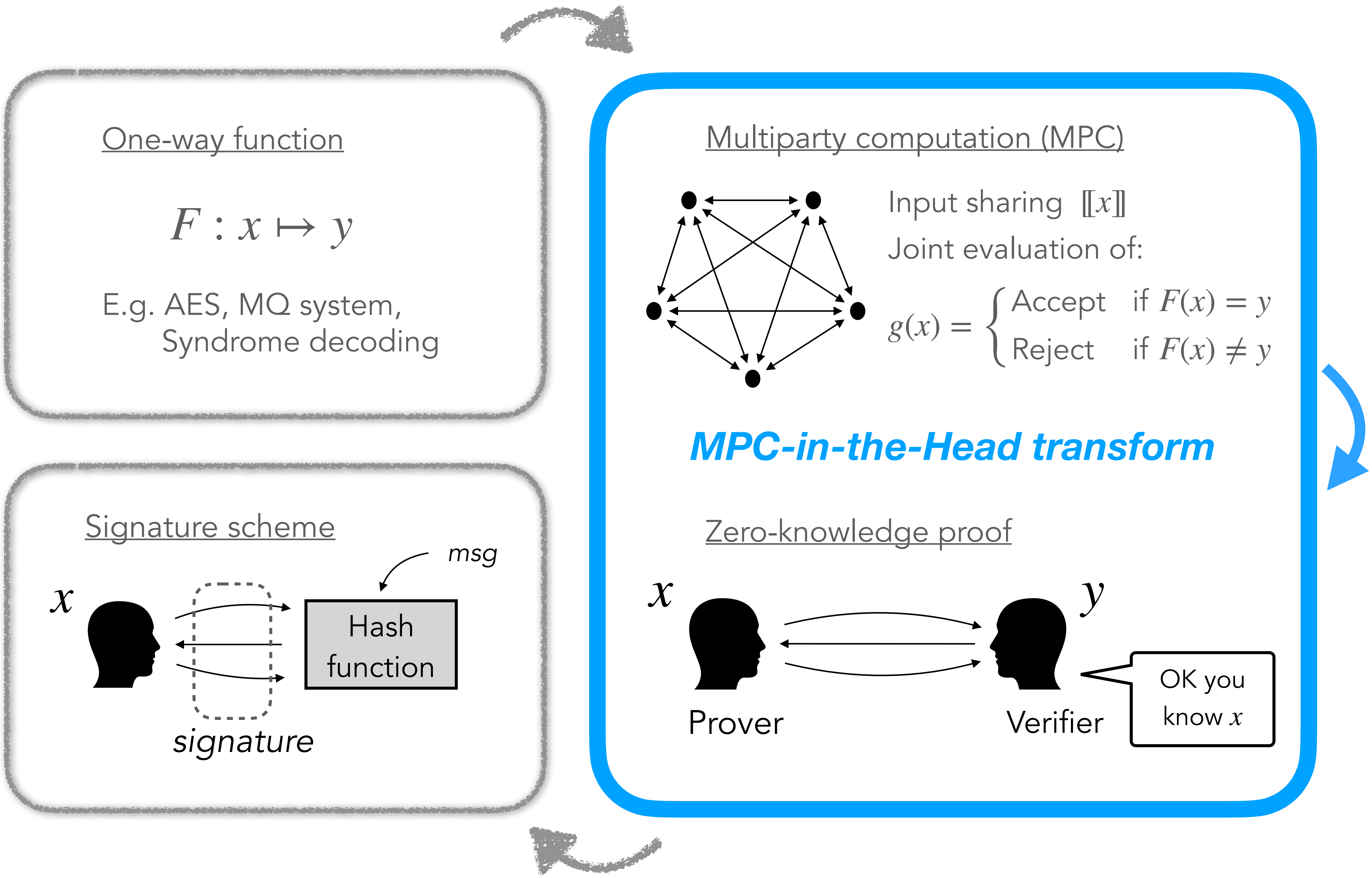
Zero-knowledge proof




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-in-the-Head Framework

Secret x which satisfies
some public relation $y = F(x)$

🤔 **How to build a zero-knowledge proof of knowledge for x?**

MPC-in-the-Head Framework

Secret x which satisfies

some public relation $y = F(x)$

$\downarrow$

Sharing $[\![x]\!]$ of the secret x

Additive secret sharing:

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

Shamir's secret sharing:

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

where P is a random degree-ℓ polynomial such that $P(0) = x$.

MPC-in-the-Head Framework

Secret x which satisfies
some public relation $y = F(x)$

$\downarrow$

Sharing $[\![x]\!]$ of the secret x

Additive secret sharing:
$x = [\![x]\!]_1 + [\![x]\!]_2 + \ldots + [\![x]\!]_N$

Shamir's secret sharing:
$\forall i, [\![x]\!]_i = P(e_i)$,
where P is a random degree-ℓ
polynomial such that $P(0) = x$.

MPC-in-the-Head Framework

Secret x which satisfies
some public relation $y = F(x)$

Sharing $[\![x]\!]$ of the secret x

Additive secret sharing:

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

Shamir's secret sharing:

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

where P is a random degree-ℓ polynomial such that $P(0) = x$.

If $x := 42$ lives in $\mathbb{F}_{1021}$, a possible sharing of x is

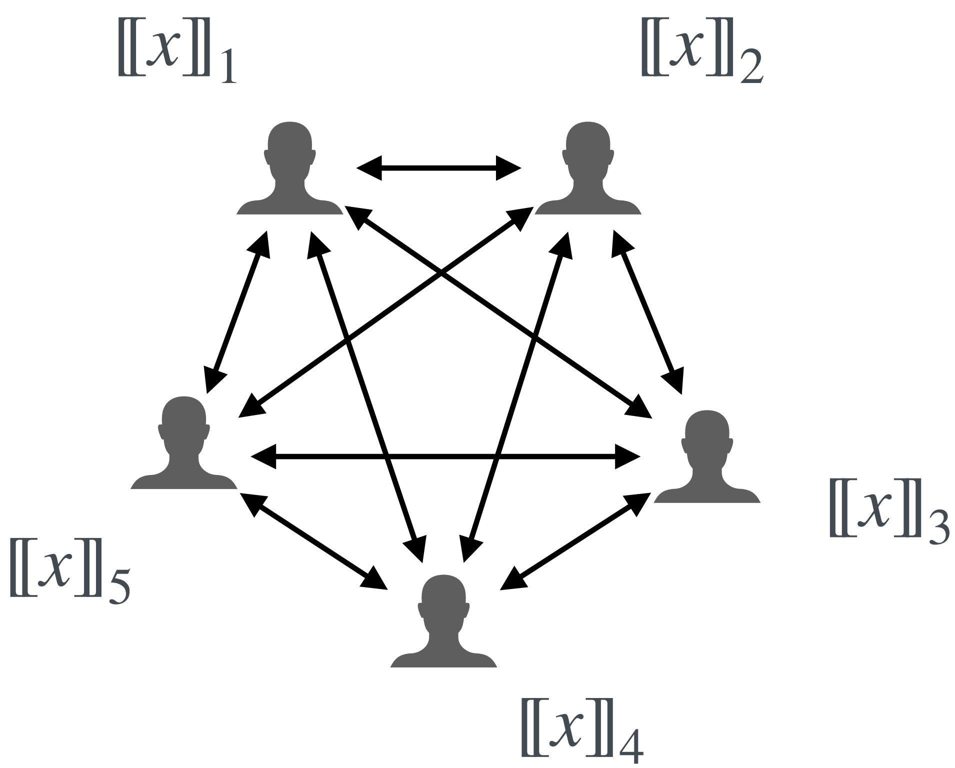
$$x = 429 + 19 + 583 + 231 + 822 \text{ over } \mathbb{F}_{1021}$$

MPC-in-the-Head Framework

Secret x which satisfies

some public relation $y = F(x)$

Sharing $[\![x]\!]$ of the secret x

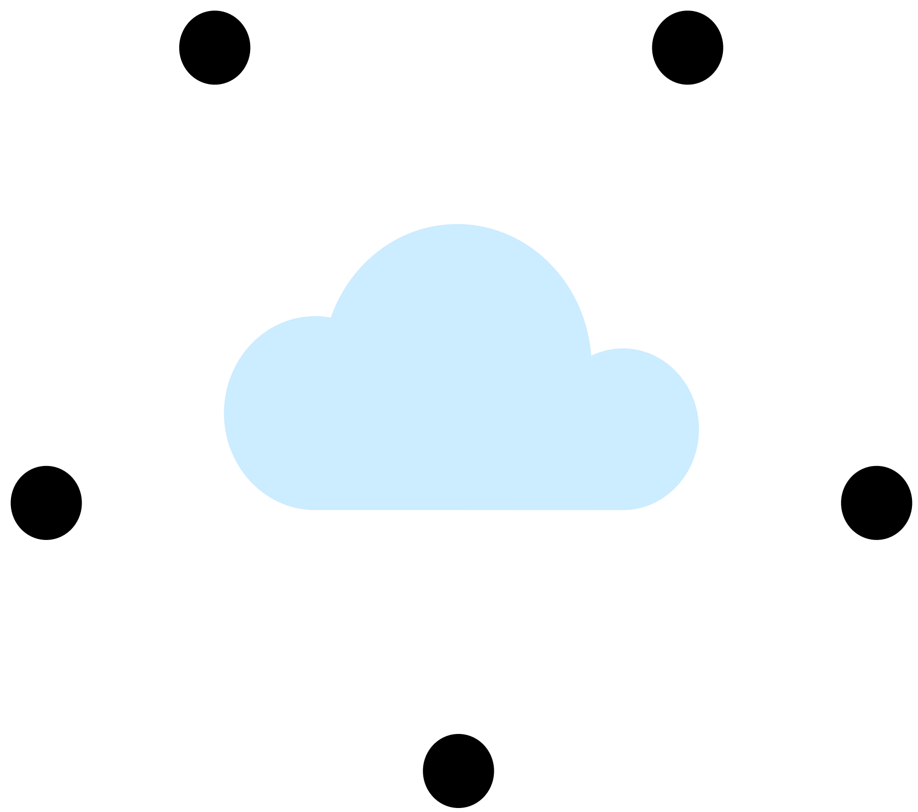


Input sharing $[\![x]\!]$

<u>Joint evaluation of</u>:

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

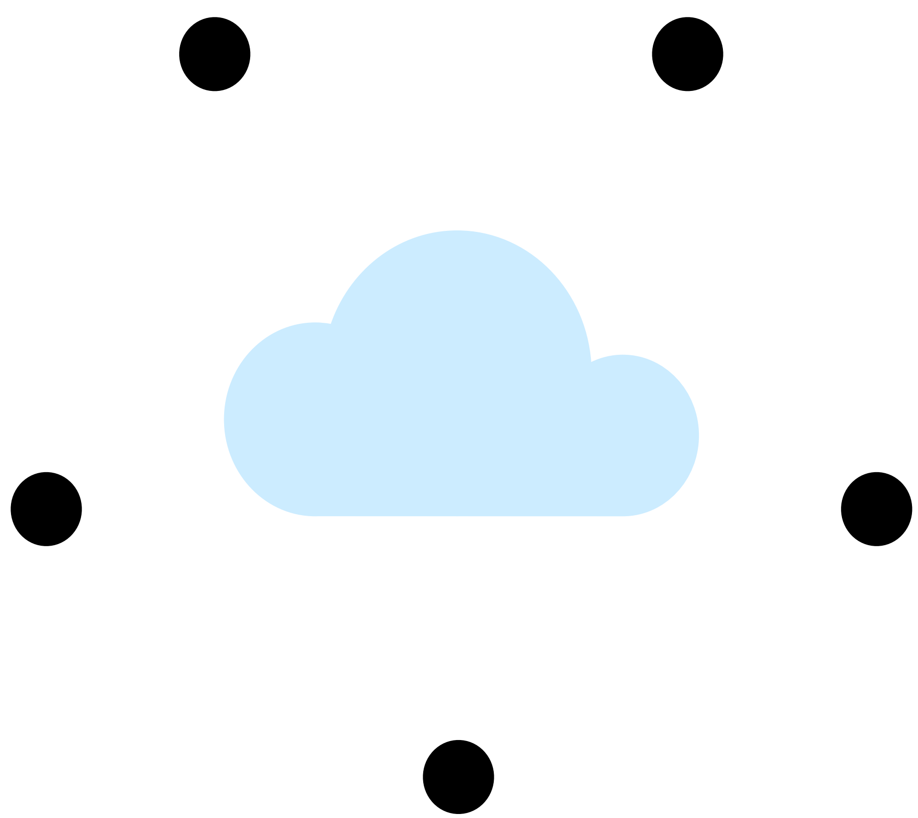
MPC model: discrete logarithm



- Secret x satisfies $y = z^x$, with z public.
- We want a multiparty computation that computes

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

MPC model: discrete logarithm

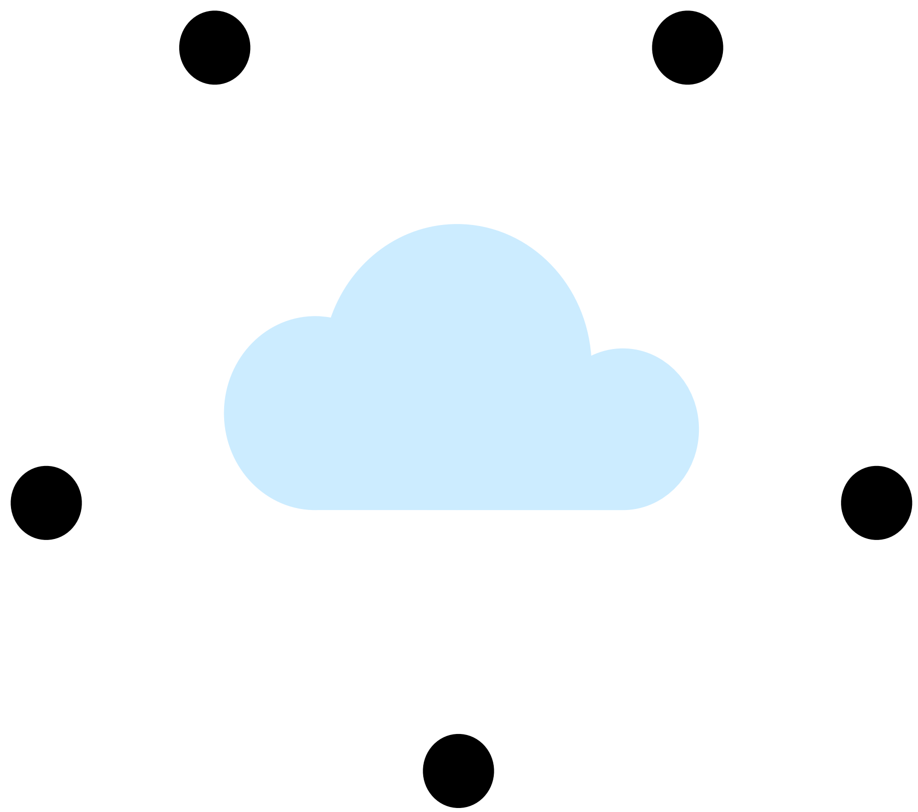


- Secret x satisfies $y = z^x$, with z public.
- We want a multiparty computation that computes

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

- **Party i:**
  - Receive the i^{th} share $[\![x]\!]_i$
  - Compute $[\![z^x]\!]_i \leftarrow z^{[\![x]\!]_i}$.
  - Broadcast $[\![z^x]\!]_i$.
  - Receive all the broadcasted values $[\![z^x]\!]_1, \ldots, [\![z^x]\!]_N$
  - Recover z^x and check that y.

MPC model: discrete logarithm



$z = 3 \pmod{1907}$ $\quad x = 575$ $\quad y = 1467 = z^x \pmod{1907}$

- Secret x satisfies $y = z^x$, with z public.
- We want a multiparty computation that computes

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

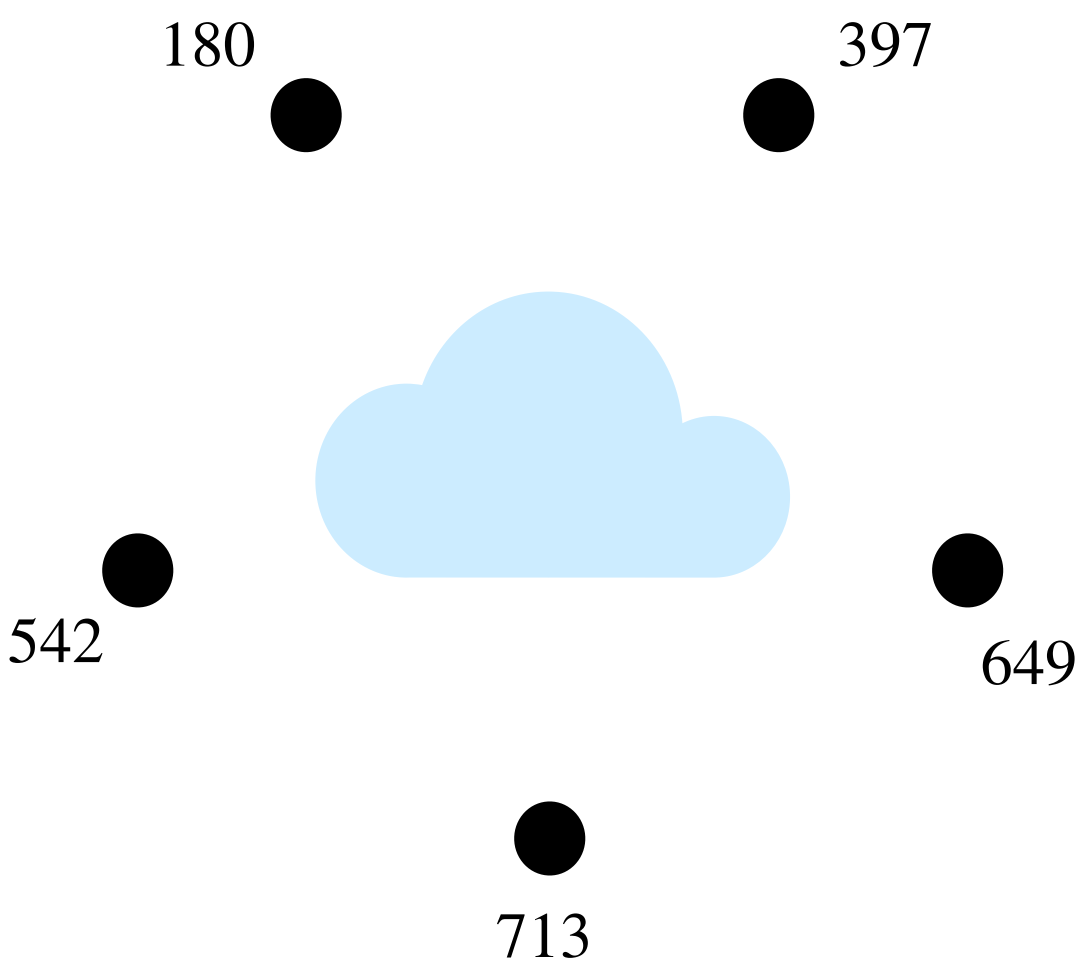
- **<u>Party i</u>:**
  - Receive the i^{th} share $[\![x]\!]_i$
  - Compute $[\![z^x]\!]_i \leftarrow z^{[\![x]\!]_i}$.
  - Broadcast $[\![z^x]\!]_i$.
  - Receive all the broadcasted values $[\![z^x]\!]_1, \ldots, [\![z^x]\!]_N$
  - Recover z^x and check that y.

MPC model: discrete logarithm

- Secret x satisfies $y = z^x$, with z public.
- We want a multiparty computation that computes

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

- **Party i:**
  - Receive the i^{th} share $[\![x]\!]_i$
  - Compute $[\![z^x]\!]_i \leftarrow z^{[\![x]\!]_i}$.
  - Broadcast $[\![z^x]\!]_i$.
  - Receive all the broadcasted values $[\![z^x]\!]_1, \ldots, [\![z^x]\!]_N$
  - Recover z^x and check that y.

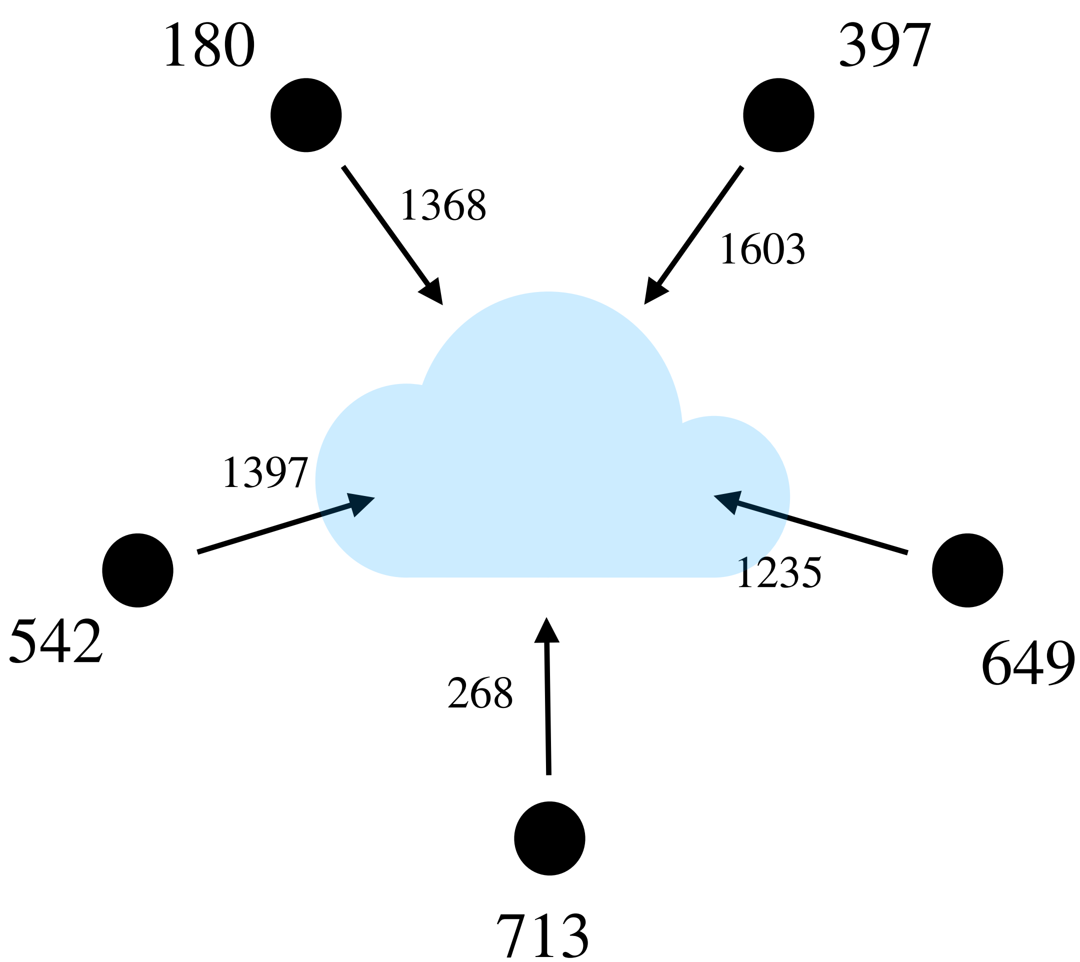


$z = 3 \pmod{1907}$ $\qquad x = 575 \qquad$ $y = 1467 = z^x \pmod{1907}$

$[\![x]\!]_1 = 180, \quad [\![x]\!]_2 = 397, \quad [\![x]\!]_3 = 649, \quad [\![x]\!]_4 = 713, \quad [\![x]\!]_5 = 542$

$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + [\![x]\!]_4 + [\![x]\!]_5 \pmod{953}$$

MPC model: discrete logarithm



$z = 3 \pmod{1907}$ $\quad x = 575 \quad$ $y = 1467 = z^x \pmod{1907}$

$[\![x]\!]_1 = 180, \quad [\![x]\!]_2 = 397, \quad [\![x]\!]_3 = 649, \quad [\![x]\!]_4 = 713, \quad [\![x]\!]_5 = 542$

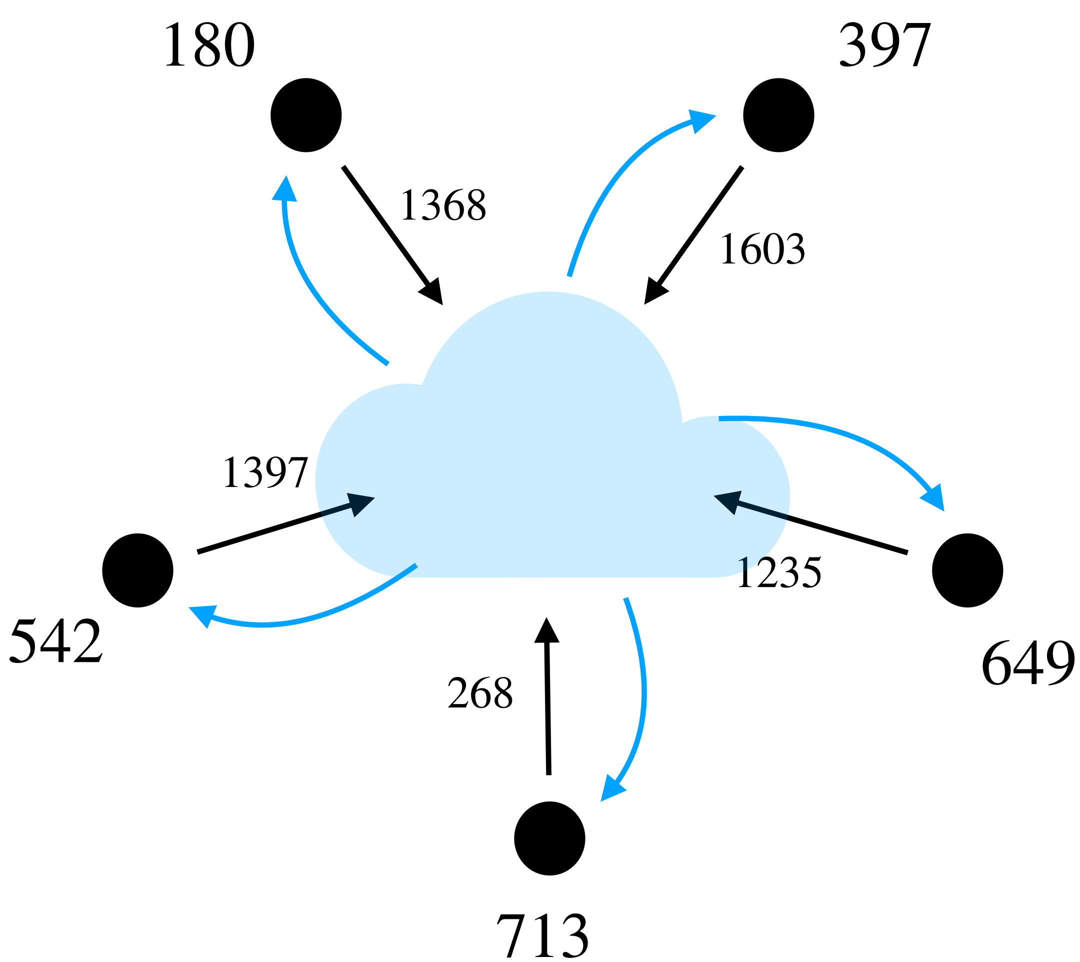
$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + [\![x]\!]_4 + [\![x]\!]_5 \pmod{953}$$

- Secret x satisfies $y = z^x$, with z public.
- We want a multiparty computation that computes

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

- **Party i:**
  - Receive the i^{th} share $[\![x]\!]_i$
  - Compute $[\![z^x]\!]_i \leftarrow z^{[\![x]\!]_i}$.
  - Broadcast $[\![z^x]\!]_i$.
  - Receive all the broadcasted values $[\![z^x]\!]_1, \ldots, [\![z^x]\!]_N$
  - Recover z^x and check that y.

MPC model: discrete logarithm



$$z = 3 \pmod{1907} \qquad x = 575 \qquad y = 1467 = z^x \pmod{1907}$$

$$[\![x]\!]_1 = 180, \quad [\![x]\!]_2 = 397, \quad [\![x]\!]_3 = 649, \quad [\![x]\!]_4 = 713, \quad [\![x]\!]_5 = 542$$

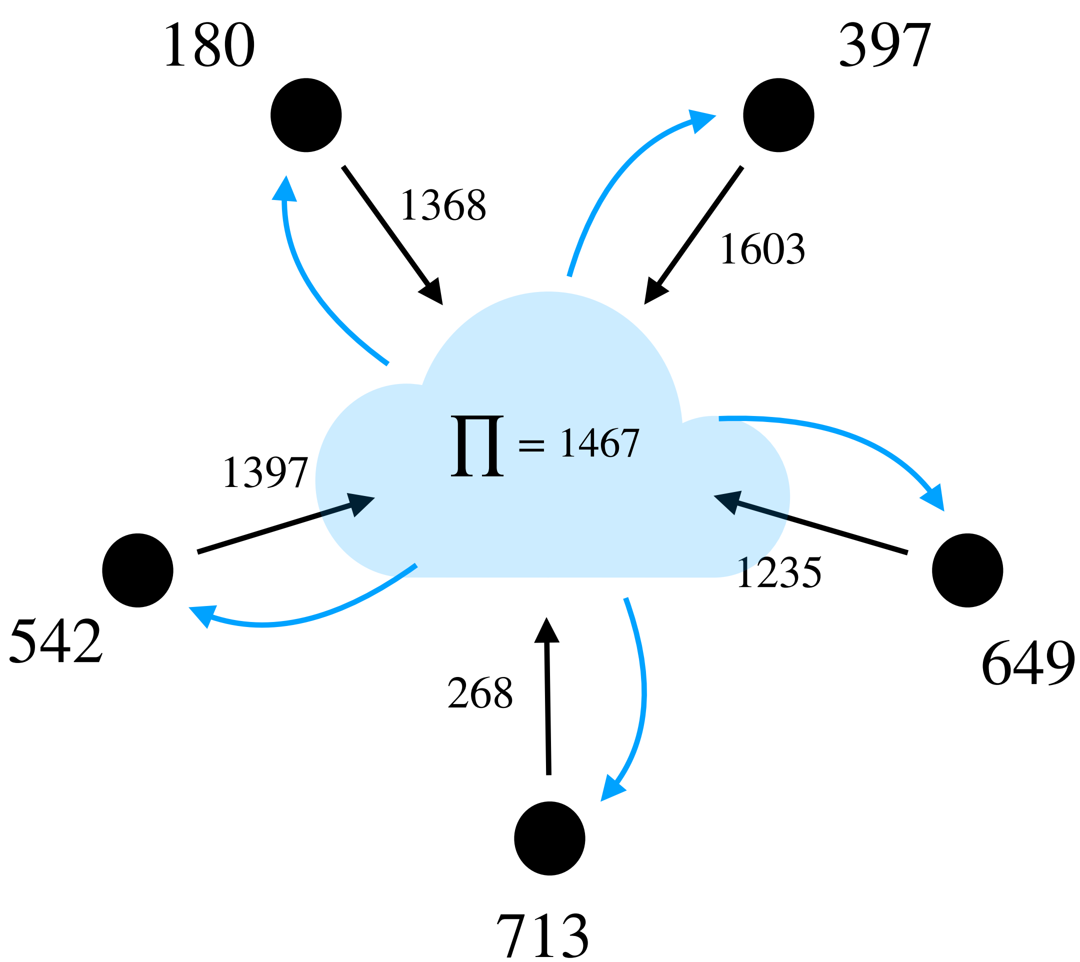
$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + [\![x]\!]_4 + [\![x]\!]_5 \pmod{953}$$

- Secret x satisfies $y = z^x$, with z public.
- We want a multiparty computation that computes

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

- **Party i:**
  - Receive the i^{th} share $[\![x]\!]_i$
  - Compute $[\![z^x]\!]_i \leftarrow z^{[\![x]\!]_i}$.
  - Broadcast $[\![z^x]\!]_i$.
  - Receive all the broadcasted values $[\![z^x]\!]_1, \ldots, [\![z^x]\!]_N$
  - Recover z^x and check that y.

MPC model: discrete logarithm



$z = 3 \pmod{1907}$ $\quad x = 575 \quad$ $y = 1467 = z^x \pmod{1907}$

$[\![x]\!]_1 = 180,\quad [\![x]\!]_2 = 397,\quad [\![x]\!]_3 = 649,\quad [\![x]\!]_4 = 713,\quad [\![x]\!]_5 = 542$

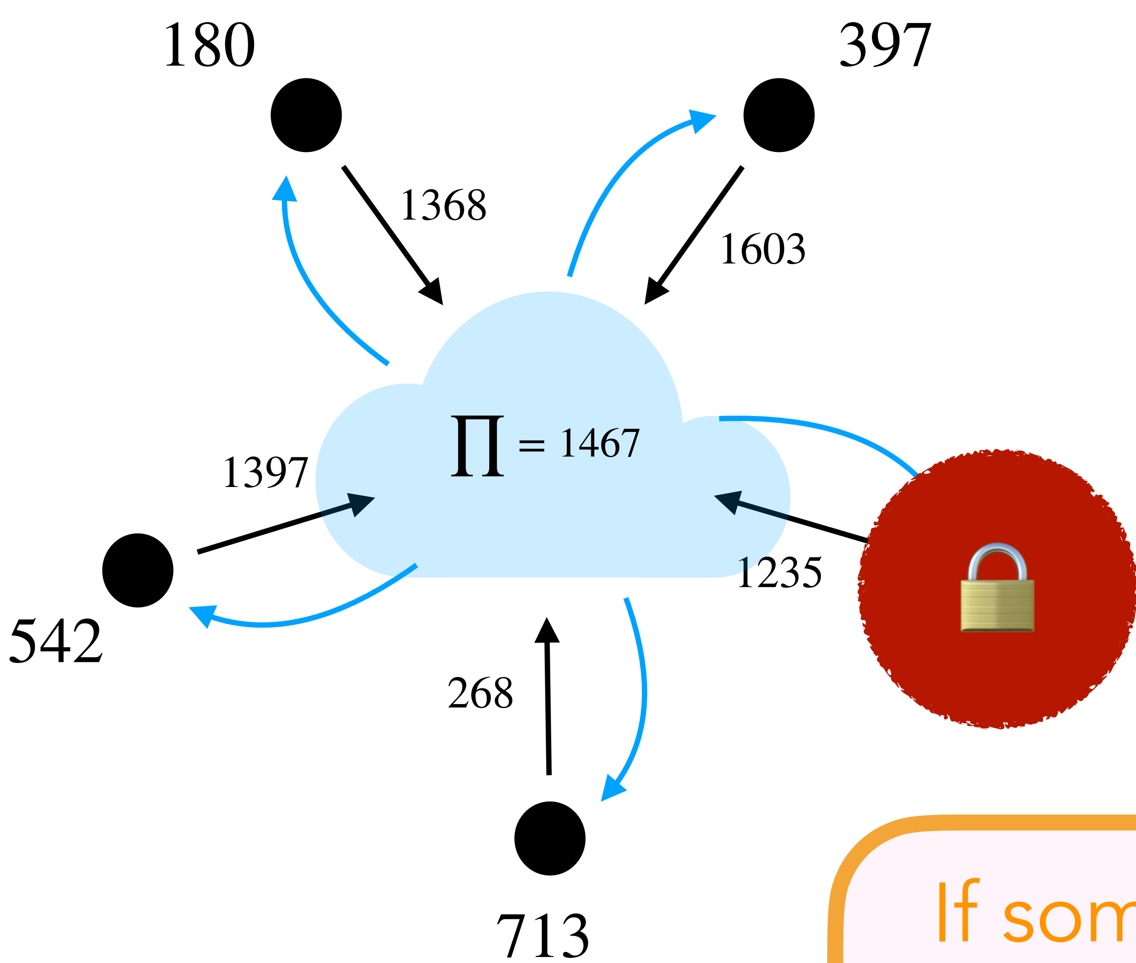
$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + [\![x]\!]_4 + [\![x]\!]_5 \pmod{953}$$

- Secret x satisfies $y = z^x$, with z public.
- We want a multiparty computation that computes

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

- **Party i:**
  - Receive the i^{th} share $[\![x]\!]_i$
  - Compute $[\![z^x]\!]_i \leftarrow z^{[\![x]\!]_i}$.
  - Broadcast $[\![z^x]\!]_i$.
  - Receive all the broadcasted values $[\![z^x]\!]_1, \ldots, [\![z^x]\!]_N$
  - Recover z^x and check that y.

MPC model: discrete logarithm



- Secret x satisfies $y = z^x$, with z public.
- We want a multiparty computation that computes

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

- **Party i:**
  - Receive the i^{th} share $[\![x]\!]_i$
  - Compute $[\![z^x]\!]_i \leftarrow z^{[\![x]\!]_i}$.

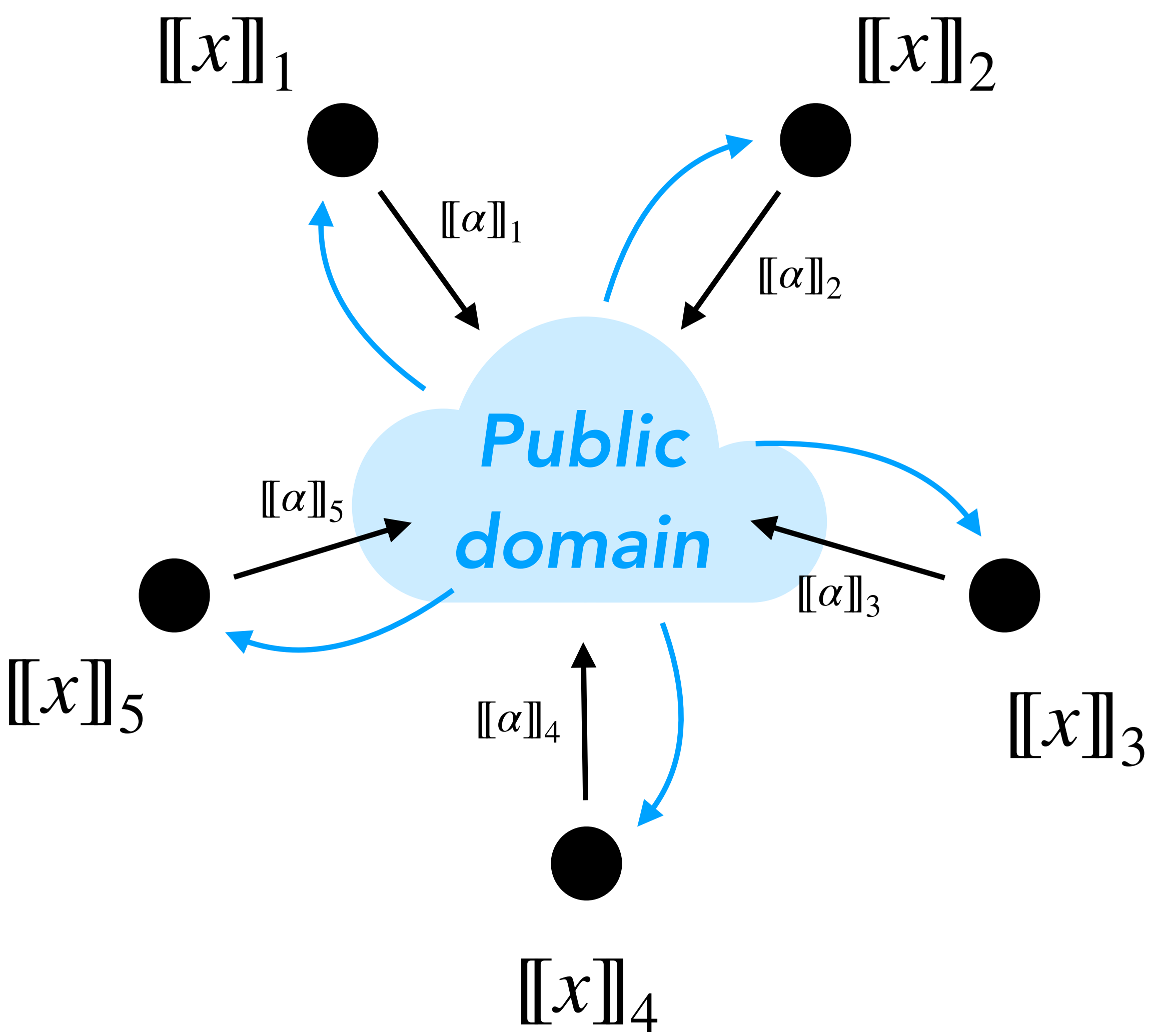
If someone sees the computation of all the parties except one, it leaks ***no information*** on x. 🧐

$z = 3 \pmod{1907}$ $x =$ $y = 1467$

$[\![x]\!]_1 = 180$, $[\![x]\!]_2 = 397$, $[\![x]\!]_3 =$, $[\![x]\!]_4 = 7$

$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + [\![x]\!]_4 + [\![x]\!]_5 \pmod{953}$$

MPC model



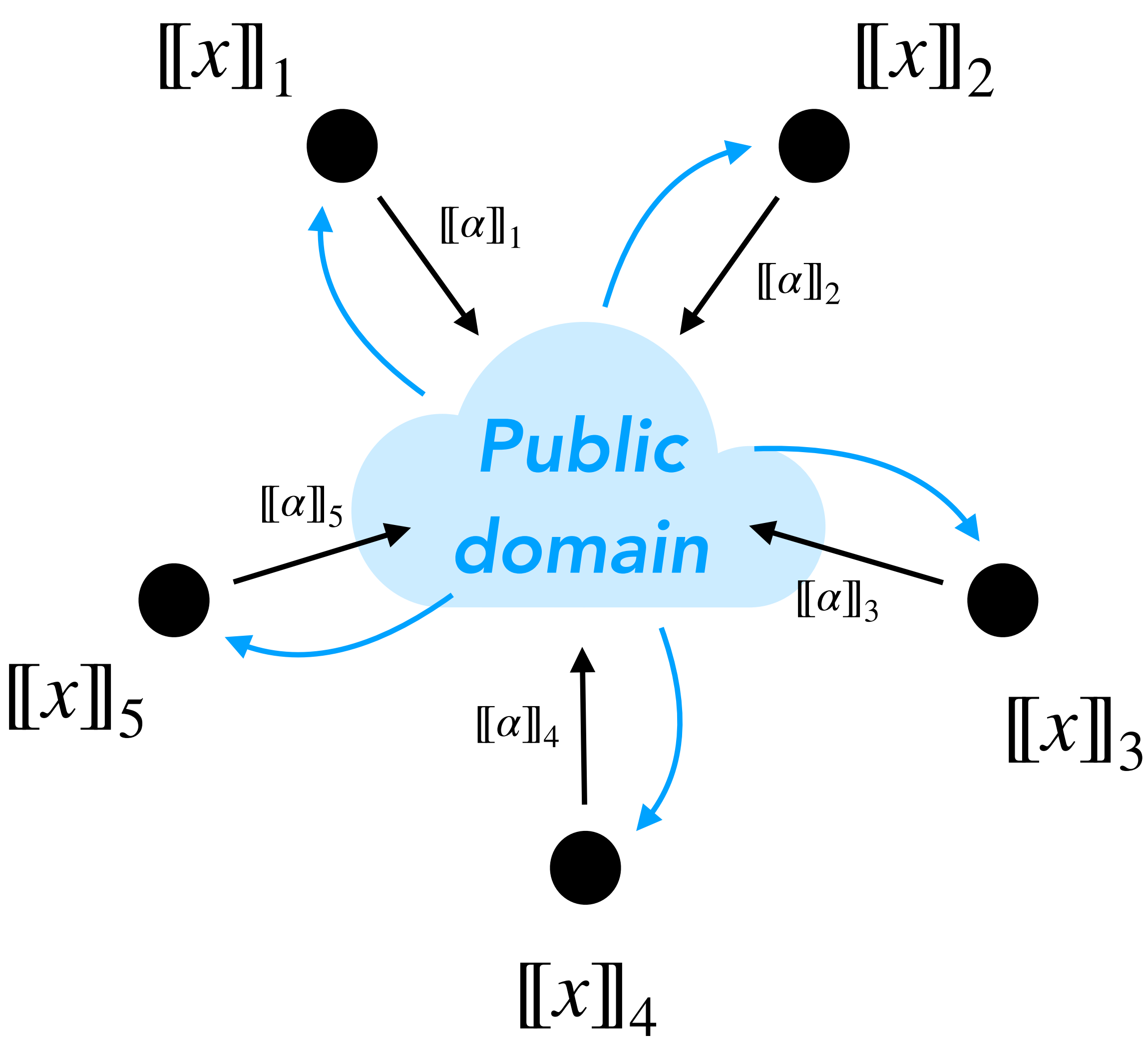
$$x = [\![x]\!]_1 + [\![x]\!]_2 + \ldots + [\![x]\!]_N$$

- **Jointly compute**

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

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

MPC model



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

- **Jointly compute**

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

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

MPCitH transform

Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

Prover

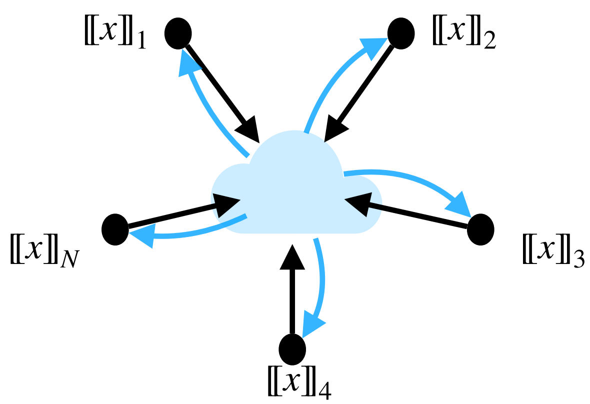
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\ldots$

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

send broadcast

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

Prover

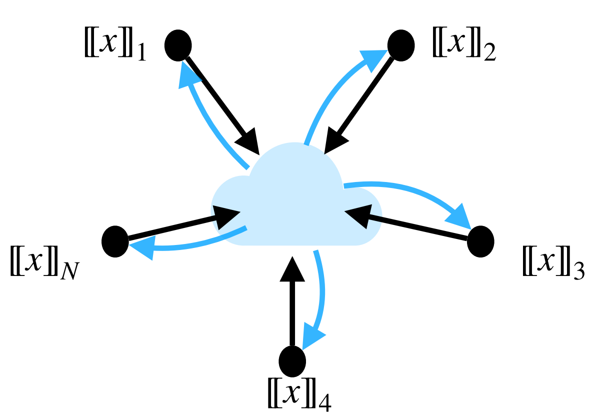
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

...

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

i^*

$\longleftarrow$

③ Choose a random party

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

Prover

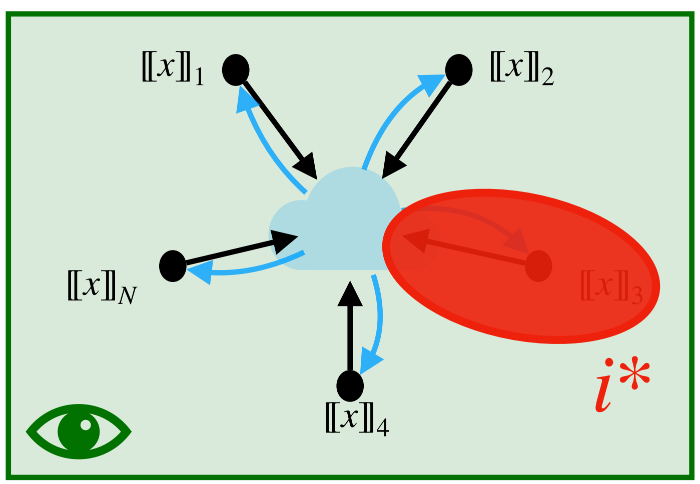
Verifier

MPCitH transform

① Generate and commit shares

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

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

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

③ Choose a random party

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

← i^*

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

Prover

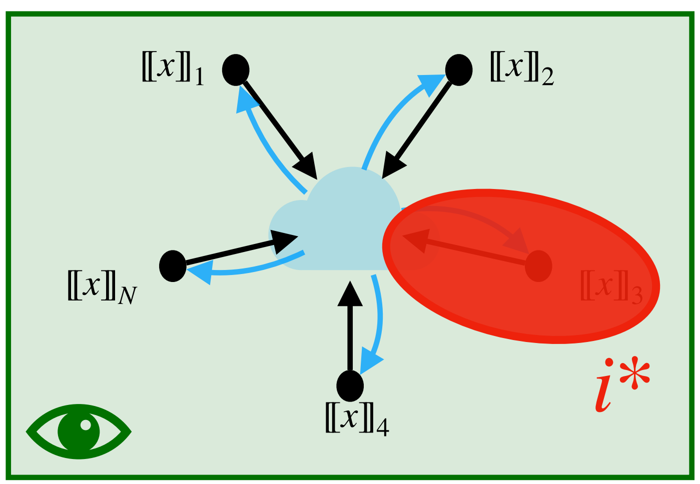
Verifier

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

Prover

Verifier

MPCitH transform

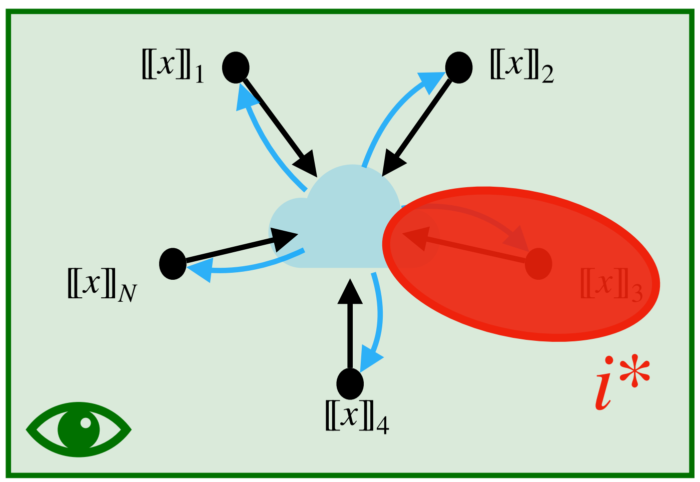
✅ Completeness
✅ Zero-Knowledge

① Generate and commit shares

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

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

② Run MPC in their head



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

③ Choose a random party

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

←
i^*

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

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

⑤ Check $\forall i \neq i^*$

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

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

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

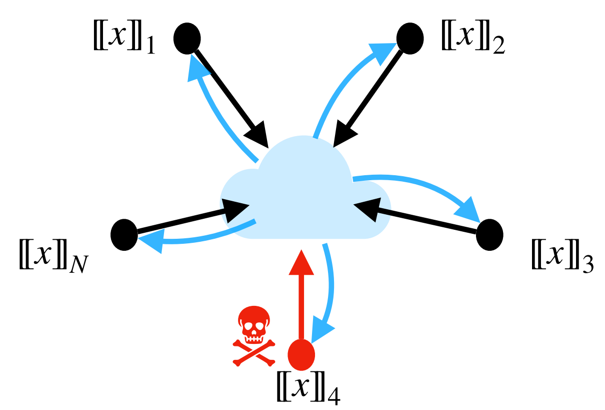
① Generate and commit shares

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

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

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

② Run MPC in their head



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

$\ldots$

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

send broadcast

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

Malicious Prover

Verifier

MPCitH transform

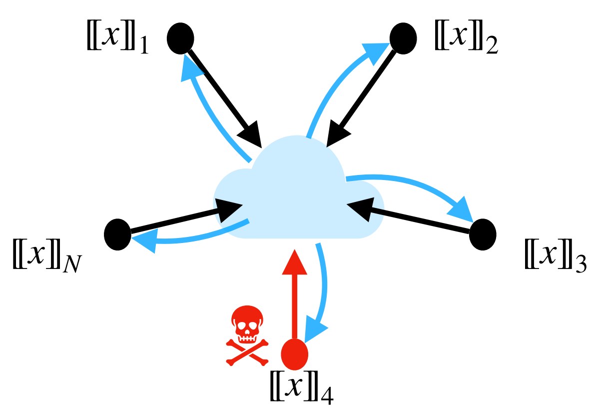
① Generate and commit shares

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

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

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

② Run MPC in their head



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

$\cdots$

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

③ Choose a random party

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

i^*

$\longleftarrow$

Malicious Prover

Verifier

MPCitH transform

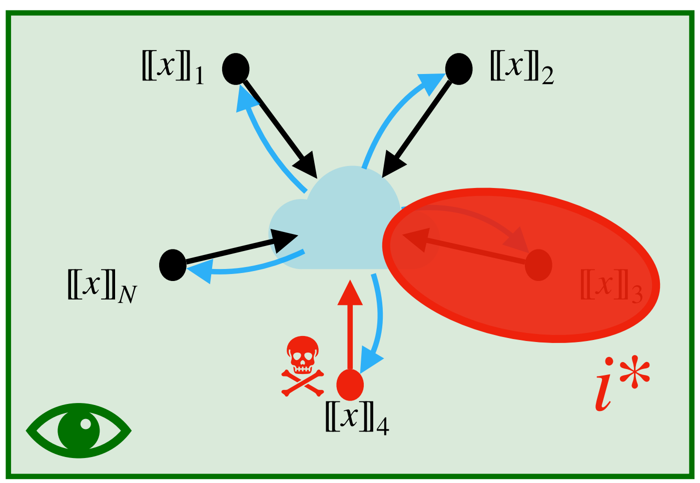
① Generate and commit shares

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

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

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

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

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

Malicious Prover

Verifier

MPCitH transform

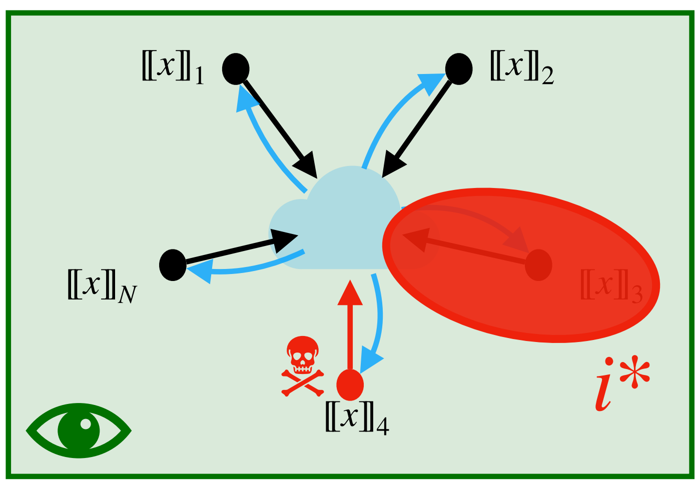
① Generate and commit shares

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

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

<u>Malicious Prover</u>

<u>Verifier</u>

Cheating detected!

MPCitH transform

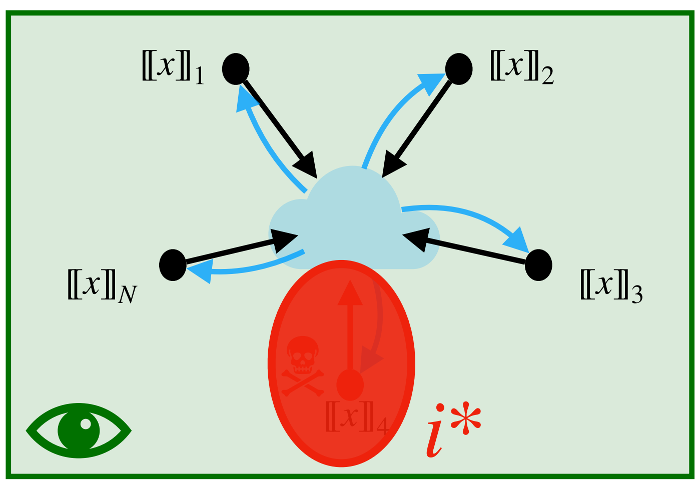
① Generate and commit shares

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

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

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

② Run MPC in their head



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

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

$\ldots$

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

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

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

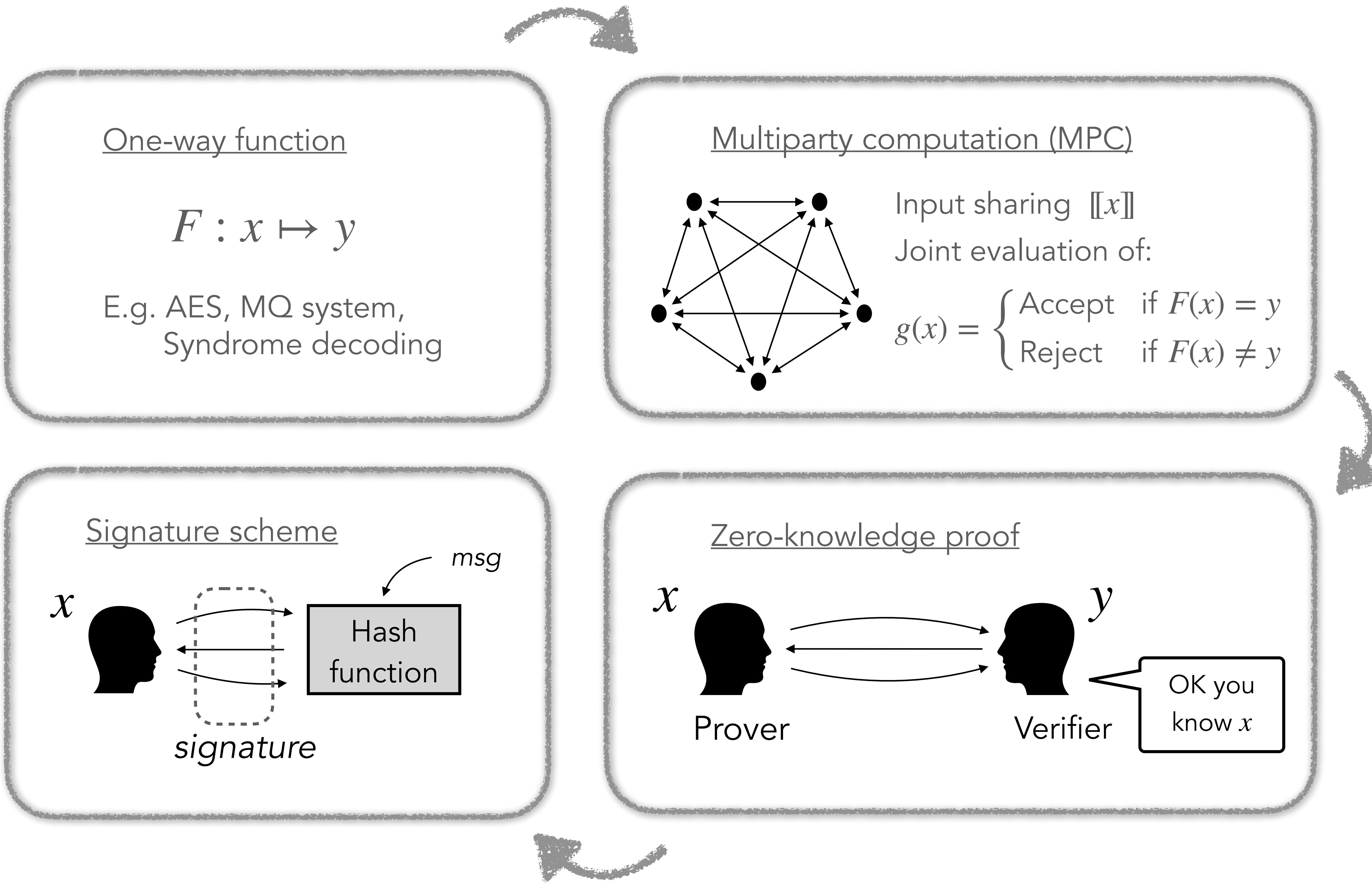
- **Soundness:**

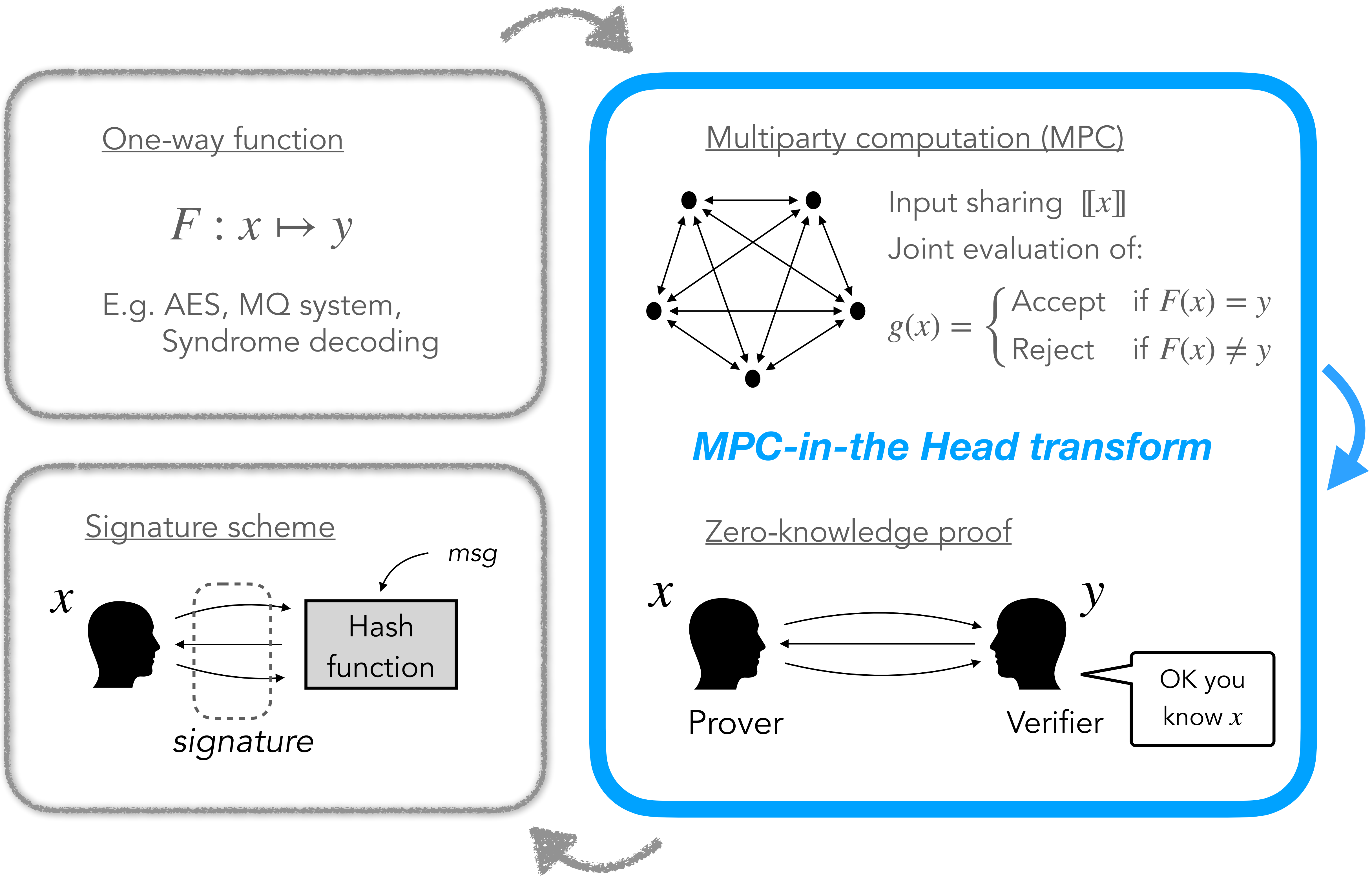
$$\begin{aligned}\mathbb{P}(\text{malicious prover convinces the verifier}) \\ &= \mathbb{P}(\text{corrupted party remains hidden}) \\ &= \frac{1}{N}\end{aligned}$$

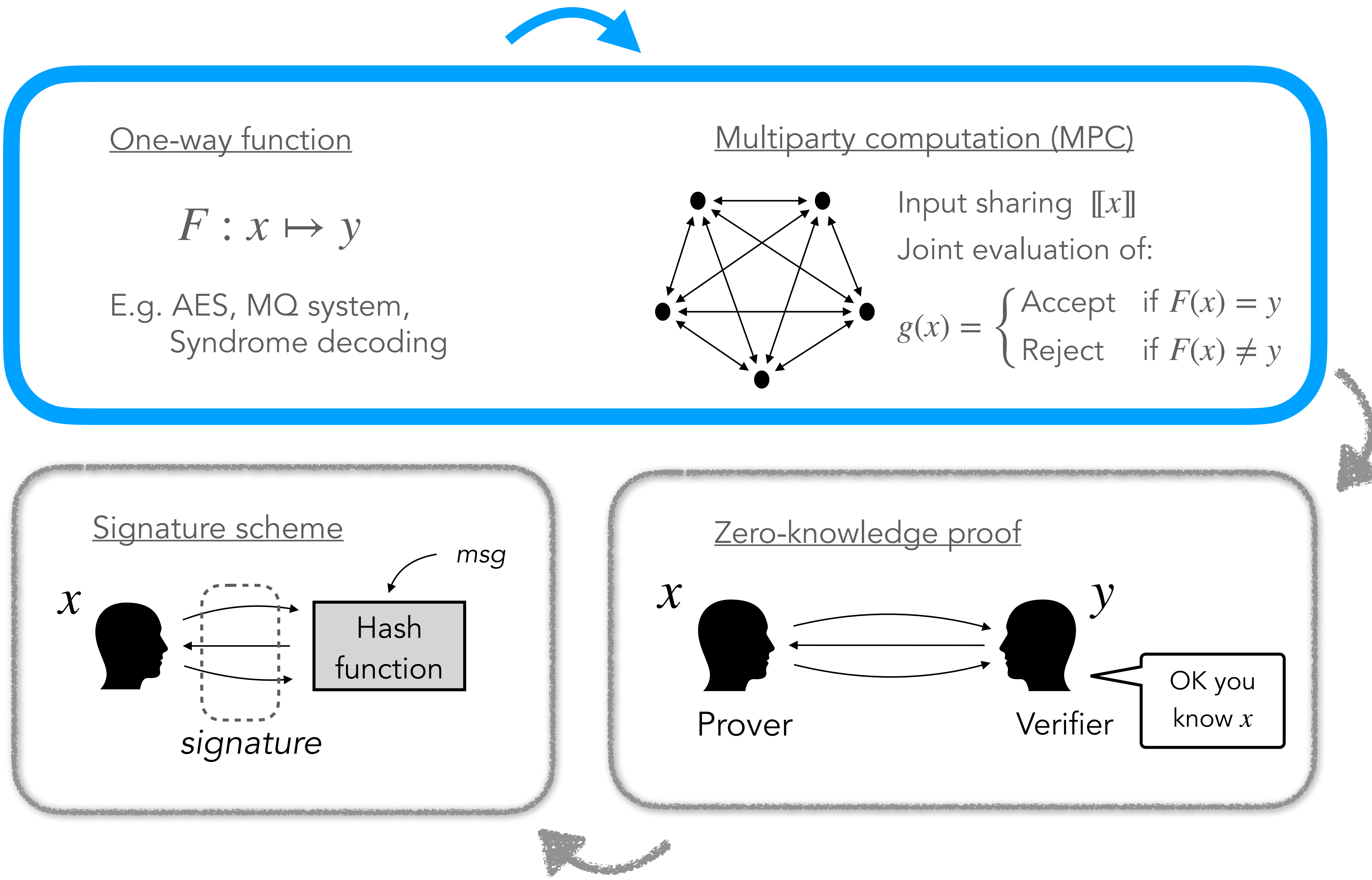
- **Parallel repetition**

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
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
Signature scheme
x
msg
Hash function
signature

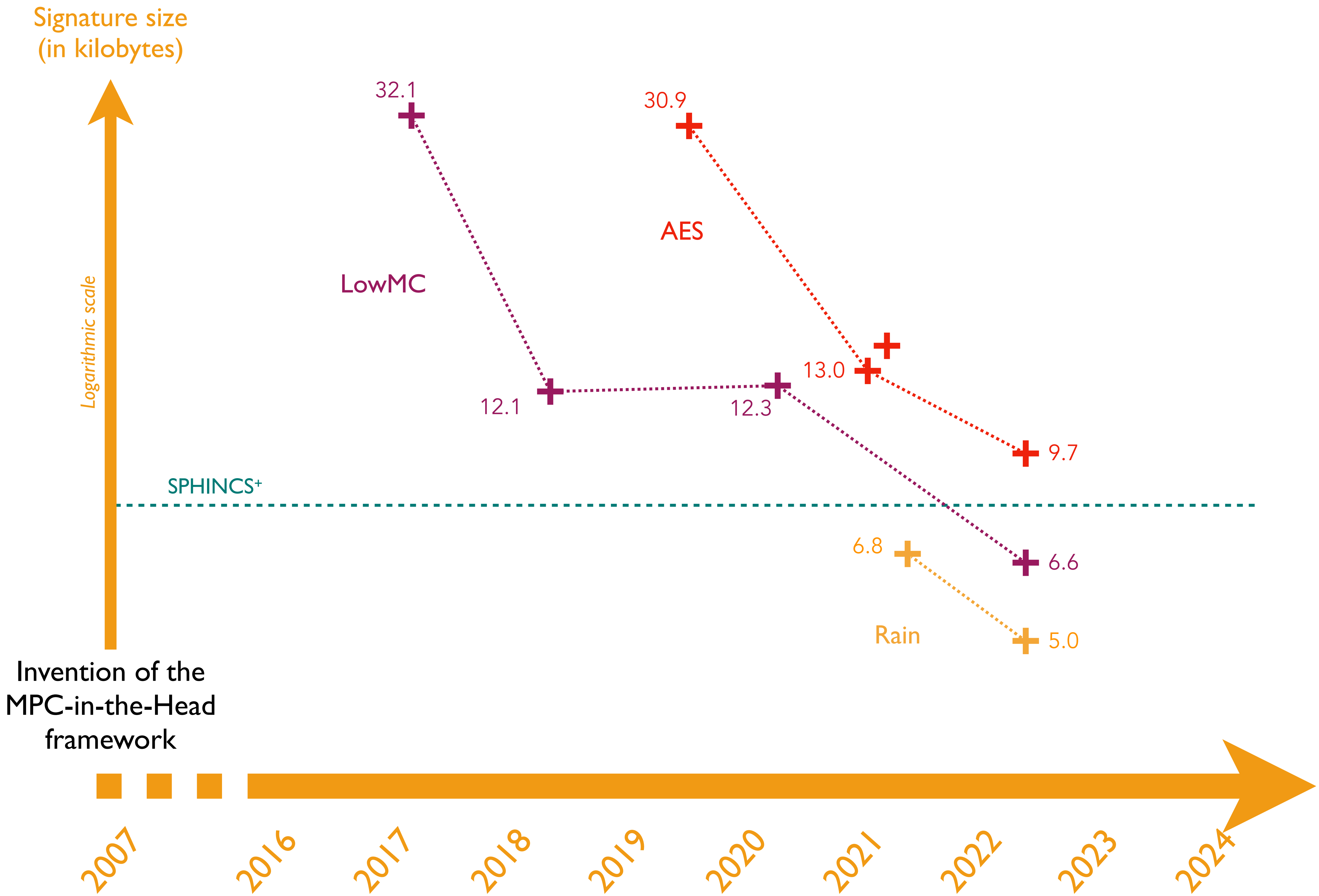


One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Signature scheme
x
msg
Hash function
signature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

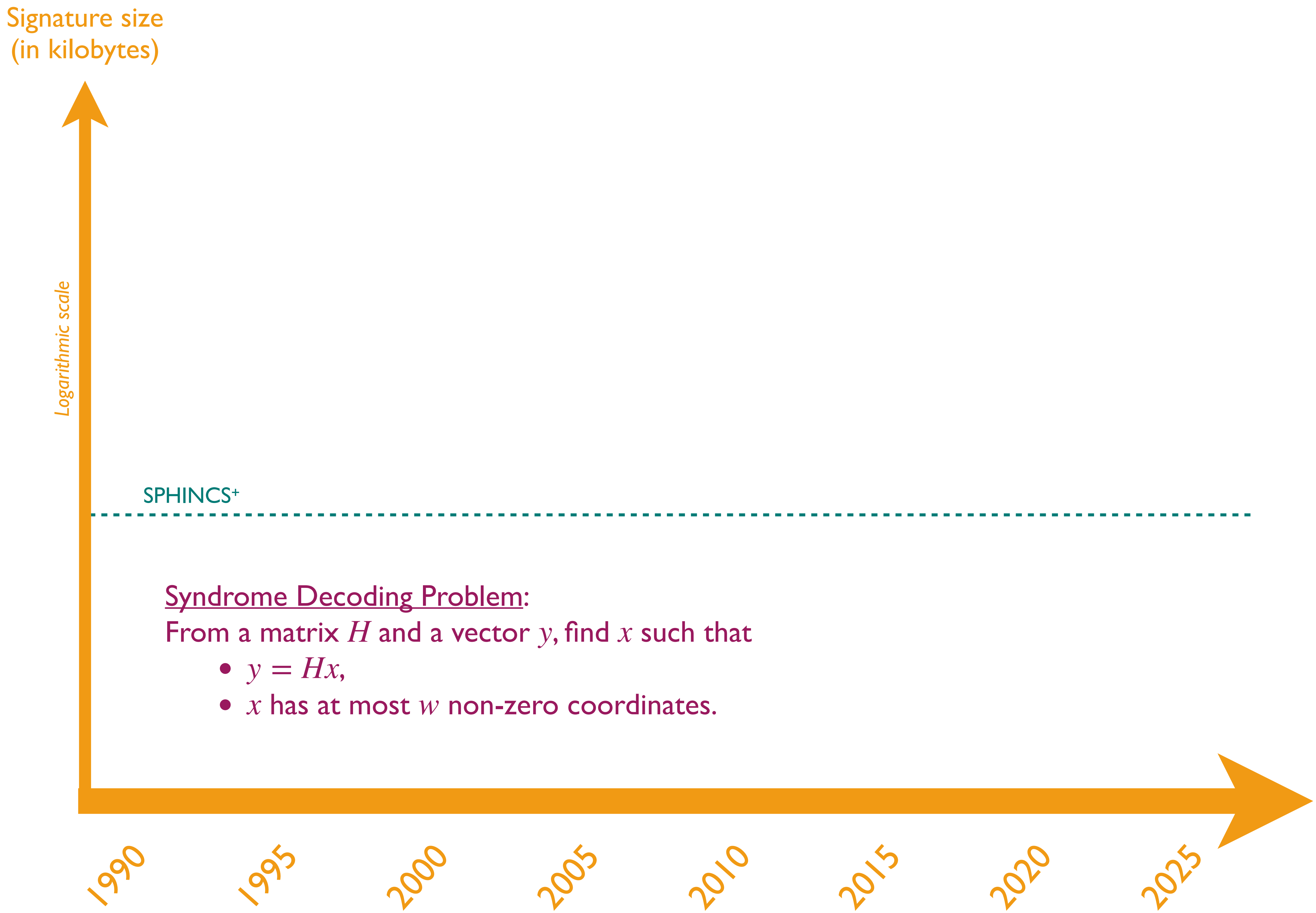
One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding



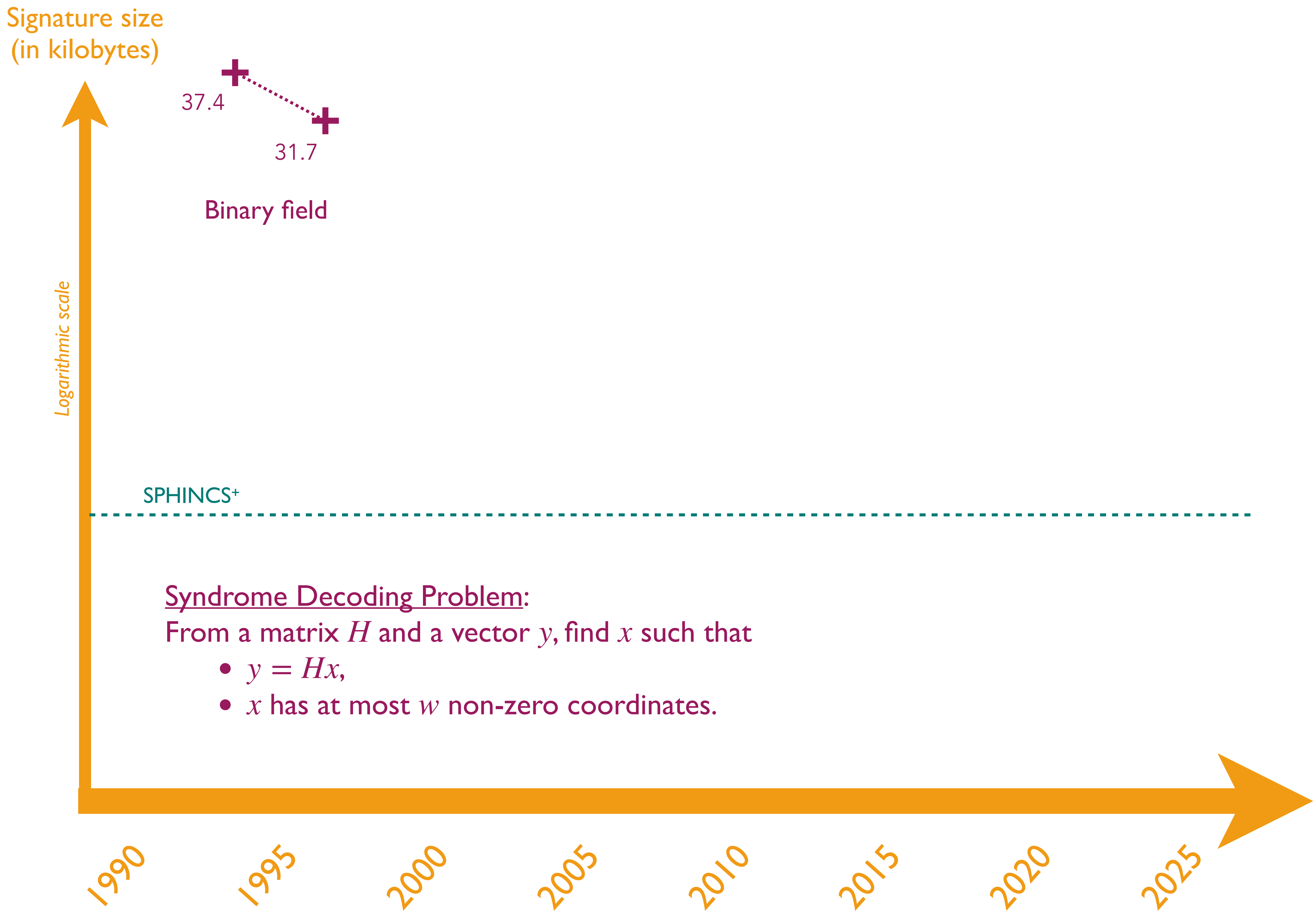
Signature size
(in kilobytes)
Logarithmic scale
32.1
30.9
AES
LowMC
13.0
12.1
12.3
9.7
SPHINCS+
6.8
6.6
Rain
5.0
Invention of the
MPC-in-the-Head
framework
2007
2016
2017
2018
2019
2020
2021
2022
2023
2024



<u>Syndrome Decoding Problem</u>:

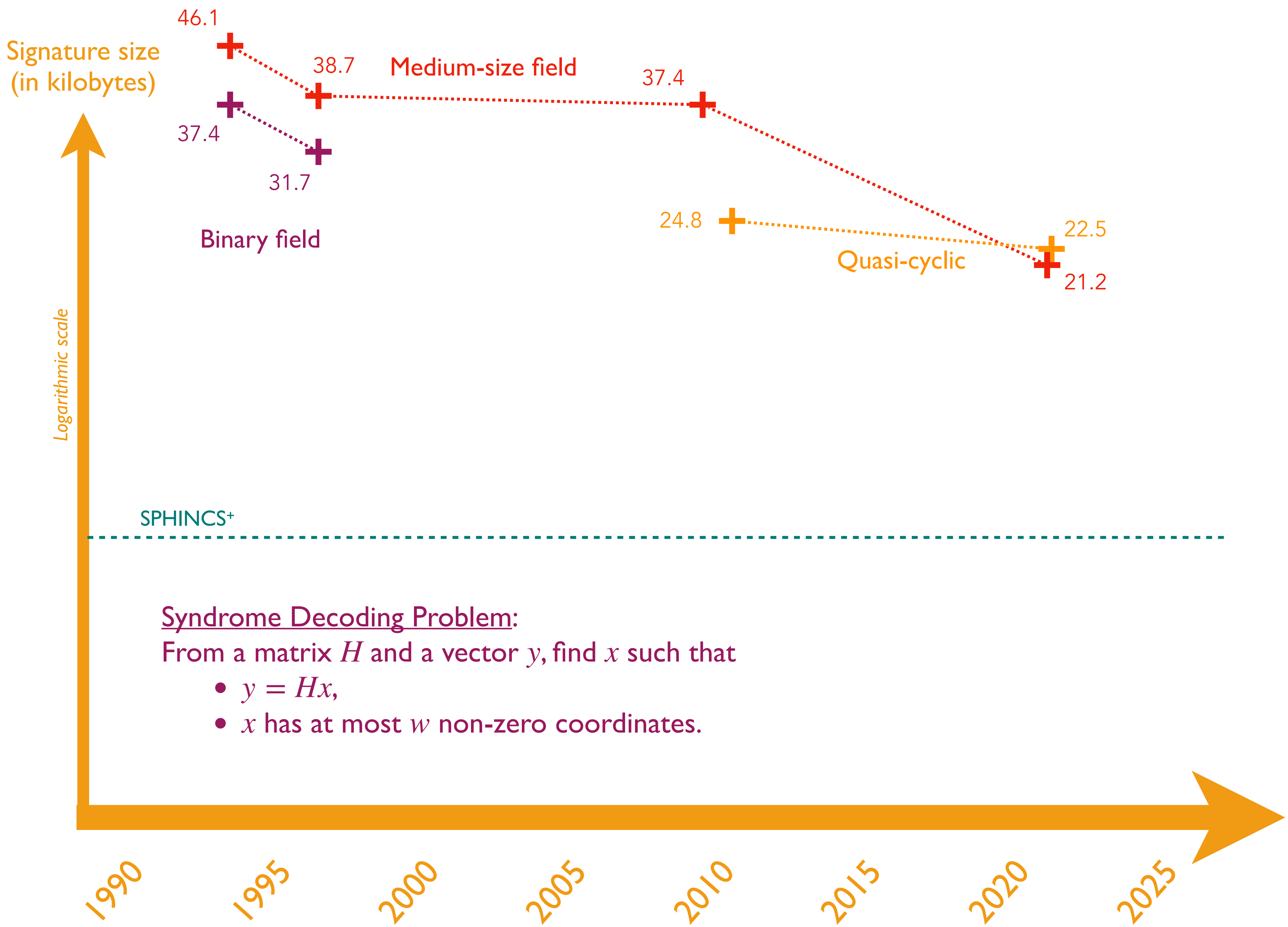
From a matrix H and a vector y, find x such that

- $y = Hx$,
- 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.



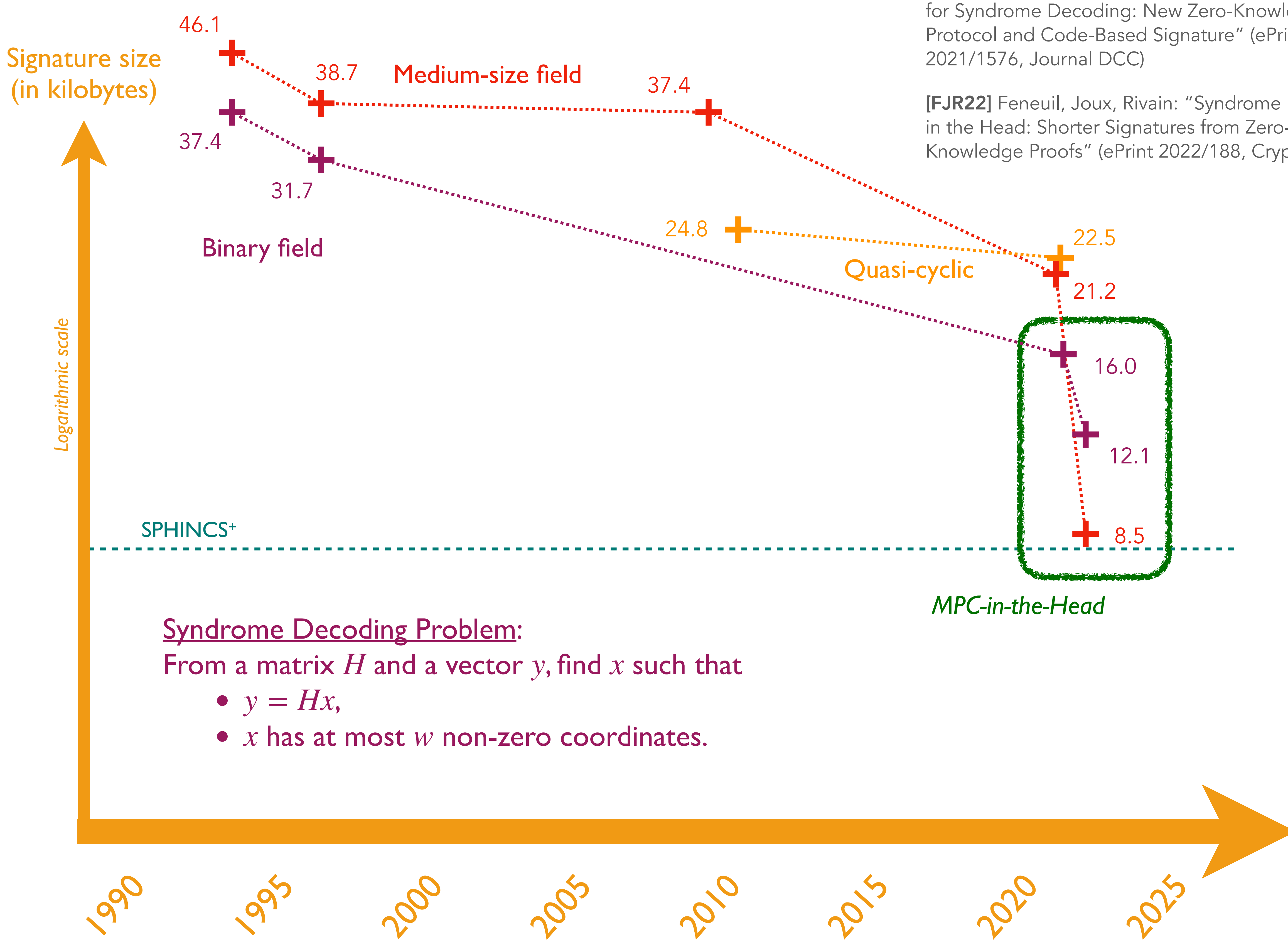
Syndrome Decoding Problem:

From a matrix H and a vector y, find x such that

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

[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 [FMRV22,Fen23]
- **Multivariate Quadratic Problem:** $6.3 - 7.3$ KB [Fen22,BFR23]
- **MinRank Problem:** $\approx 5 - 6$ KB [ARV22,Fen22,ABB+23]
- **Rank Syndrome Decoding Problem:** $\approx 5 - 6$ KB [Fen22]
- **Permuted Kernel Problem (or variant):** ≈ 6 KB [BG22,BBD+24]
- …

Remark: the displayed signature sizes correspond to the state-of-the-art for the NIST submission deadline of the call for additional post-quantum signatures, ***better sizes*** *can be achieved using newer results.*

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Three approaches:

- Rely on standard symmetric primitives
  - AES: *BBQ* (2019), *Banquet* (2021), *Limbo-Sign* (2021), *Helium+AES* (2022), *FAEST* (2023)

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Three approaches:

- Rely on standard symmetric primitives
- Rely on MPC-friendly symmetric primitives
  - LowMC: *Picnic1* (2017), *Picnic2* (2018), *Picnic3* (2020)
  - Rain: *Rainier* (2021), *BN++Rain* (2022)
  - AIM: *AIMer* (2022)

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Three approaches:

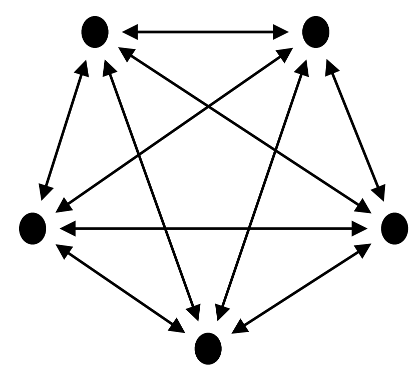
- Rely on standard symmetric primitives
- Rely on MPC-friendly symmetric primitives
- Rely on well-known hard problems *(non-exhaustive list)*
  - Syndrome Decoding: *SDitH* (2022), *RYDE* (2023)
  - MinRank: *MiRitH* (2022), *MIRA* (2023)
  - Multivariate Quadratic: *MQOM* (2023), *Biscuit* (2023)
  - Permuted Kernel: *PERK* (2023)

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Three approaches:

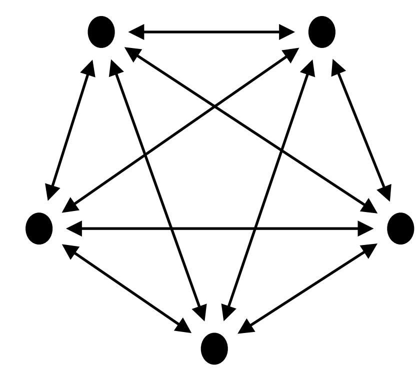
- Rely on standard symmetric primitives
- Rely on MPC-friendly symmetric primitives
- Rely on well-known hard problems

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Three approaches:

- Rely on standard symmetric primitives
- Rely on MPC-friendly symmetric primitives
- Rely on well-known hard problems

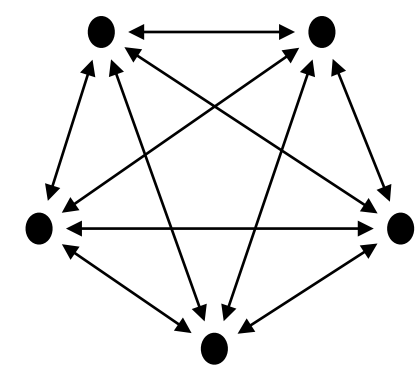
Expressed as an arithmetic circuit, enabling us to use existing MPCitH-based proof systems (as BN++)

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Three approaches:

- Rely on standard symmetric primitives
- Rely on MPC-friendly symmetric primitives
- Rely on well-known hard problems

Expressed as an arithmetic circuit, enabling us to use existing MPCitH-based proof systems (as BN++)

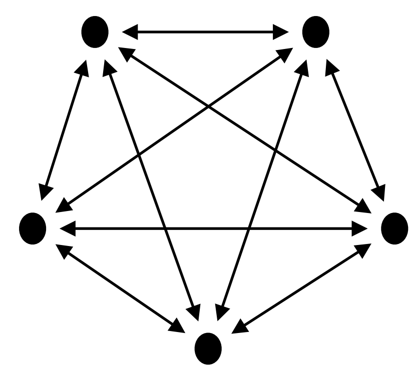
Should be rephrased to achieve interesting performances

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Three approaches:

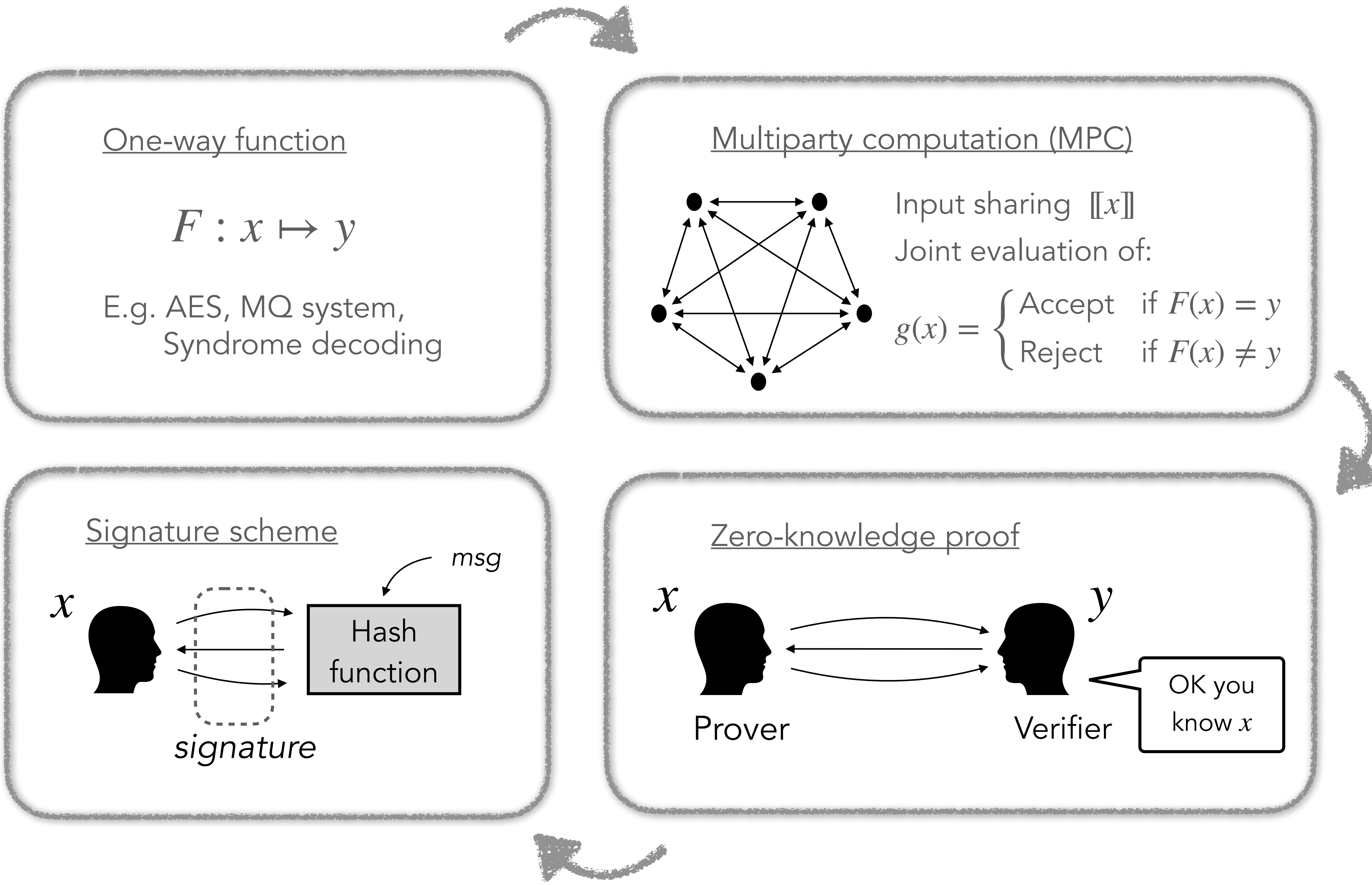
- Rely on standard symmetric primitives
- Rely on MPC-friendly symmetric primitives
- Rely on well-known hard problems

Expressed as an arithmetic circuit, enabling us to use existing MPCitH-based proof systems (as BN++)

Should be rephrased to achieve interesting performances

Example (RYDE): how to check that a vector $x \in \mathbb{F}_{q^m}^n$ has a rank weight smaller than some public bound r ?

By checking that $x_1, \ldots, x_n$ are roots of a degree-q^r q-polynomial $\sum_{i=0}^{r} a_i X^{q^i}$.

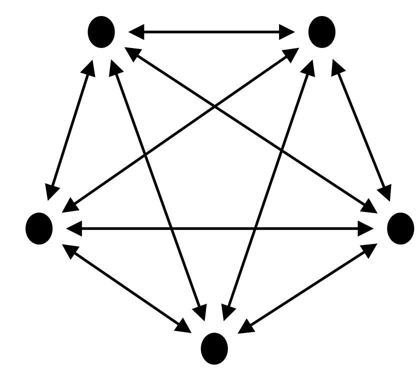

One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Zero-knowledge proof
x
y
Prover
Verifier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

Multiparty computation (MPC)

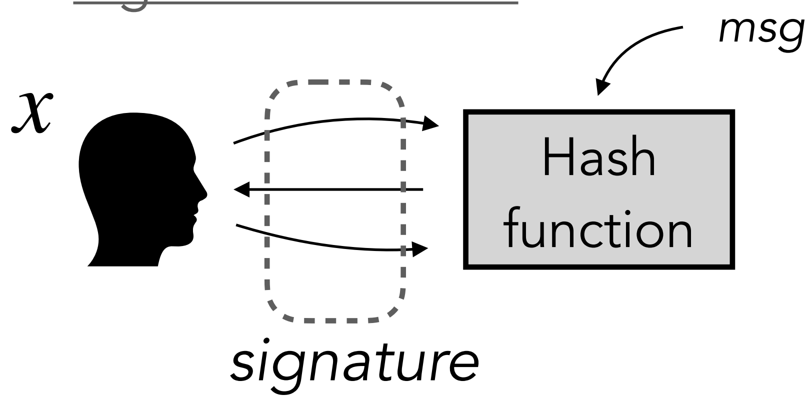


Input sharing $[\![x]\!]$

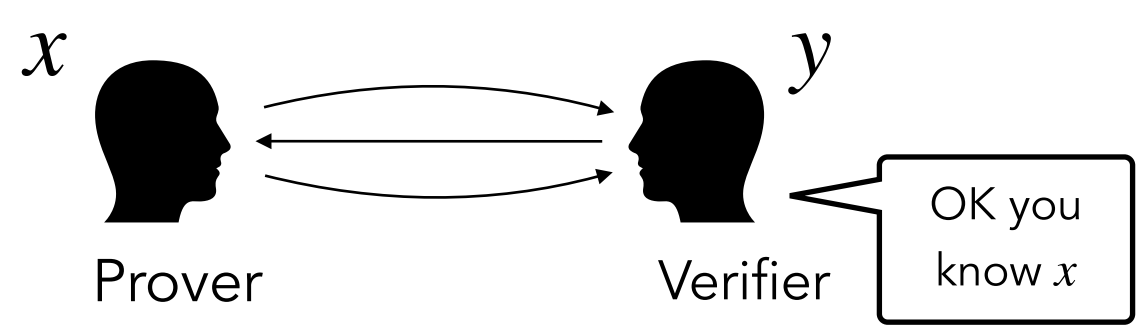
Joint evaluation of:

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

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)

MPCitH-based NIST Candidates

	Assumption	Size (in KB)
AIMer	AIM (MPC-friendly one-way function)	3.8-5.9
Biscuit	Structured MQ problem (PowAff2)	4.8-6.7
FAEST*	AES block cipher	4.6-6.3
MIRA	MinRank problem	5.6-7.4
MiRitH	MinRank problem	5.7-9.1
PERK	Permuted Kernel problem (variant)	6.8-8.4
MQOM	Unstructured MQ problem	6.3-7.8
RYDE	Syndrome decoding problem in rank metric	6.0-7.4
SDitH	Syndrome decoding problem in Hamming	8.3-10.4

** FAEST has not been formally introduced as an MPCitH-based scheme.*

Optimisations and variants

Optimisations and variants

Field $GF(251)$

With `SDitH-L1-gf251` as example.

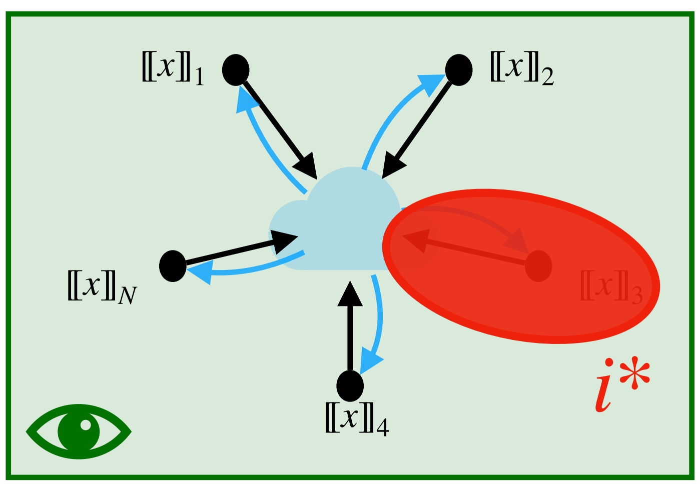
NIST Category I

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

$\cdots$

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

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

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)

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

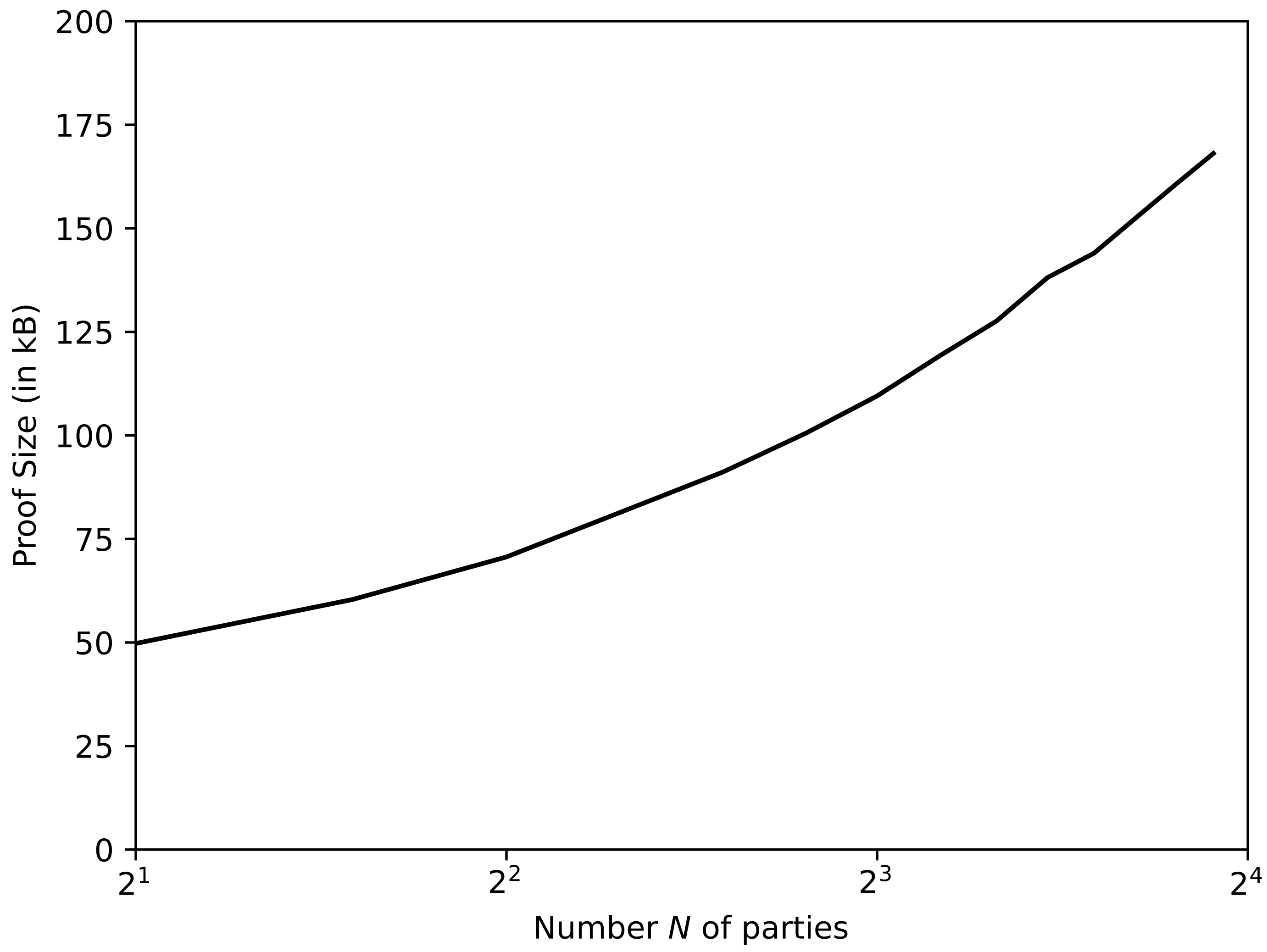
Number of repetitions to achieve the desired security level

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

`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:`

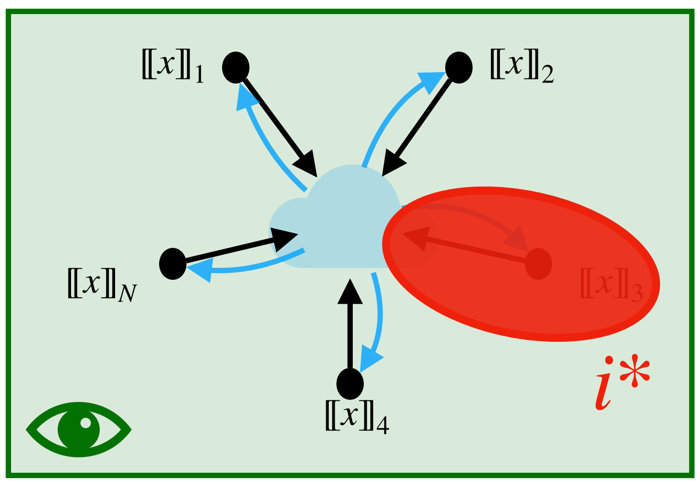
the input x of the MPC protocol is around **323** bytes,
The broadcast value α of the MPC protocol is around **36** bytes

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

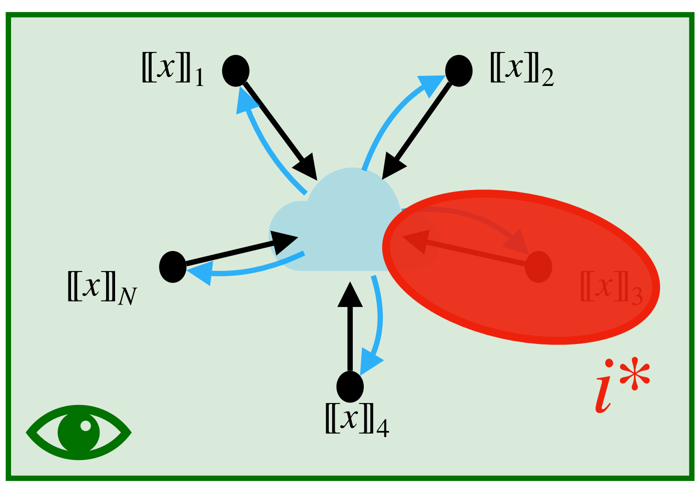
① Generate and commit shares

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

Compute

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

② Run MPC in their head



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

Prover

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

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

← i^*

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

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

Verifier

MPCitH transform

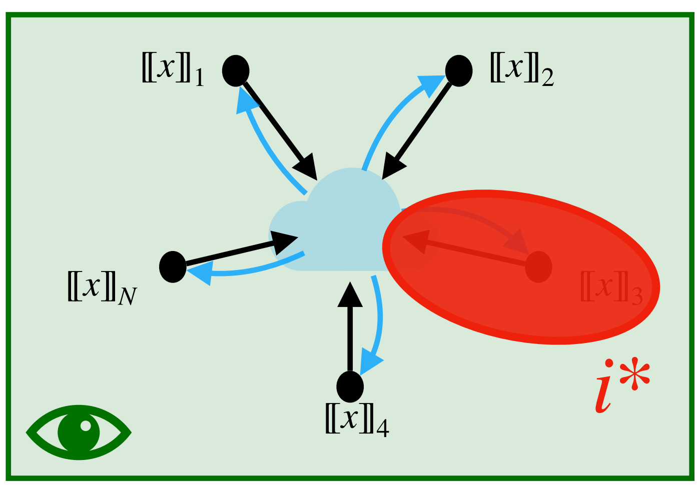
① Generate and commit shares

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

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

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

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

Verifier

Using a Seed Tree

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

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

Using a Seed Tree

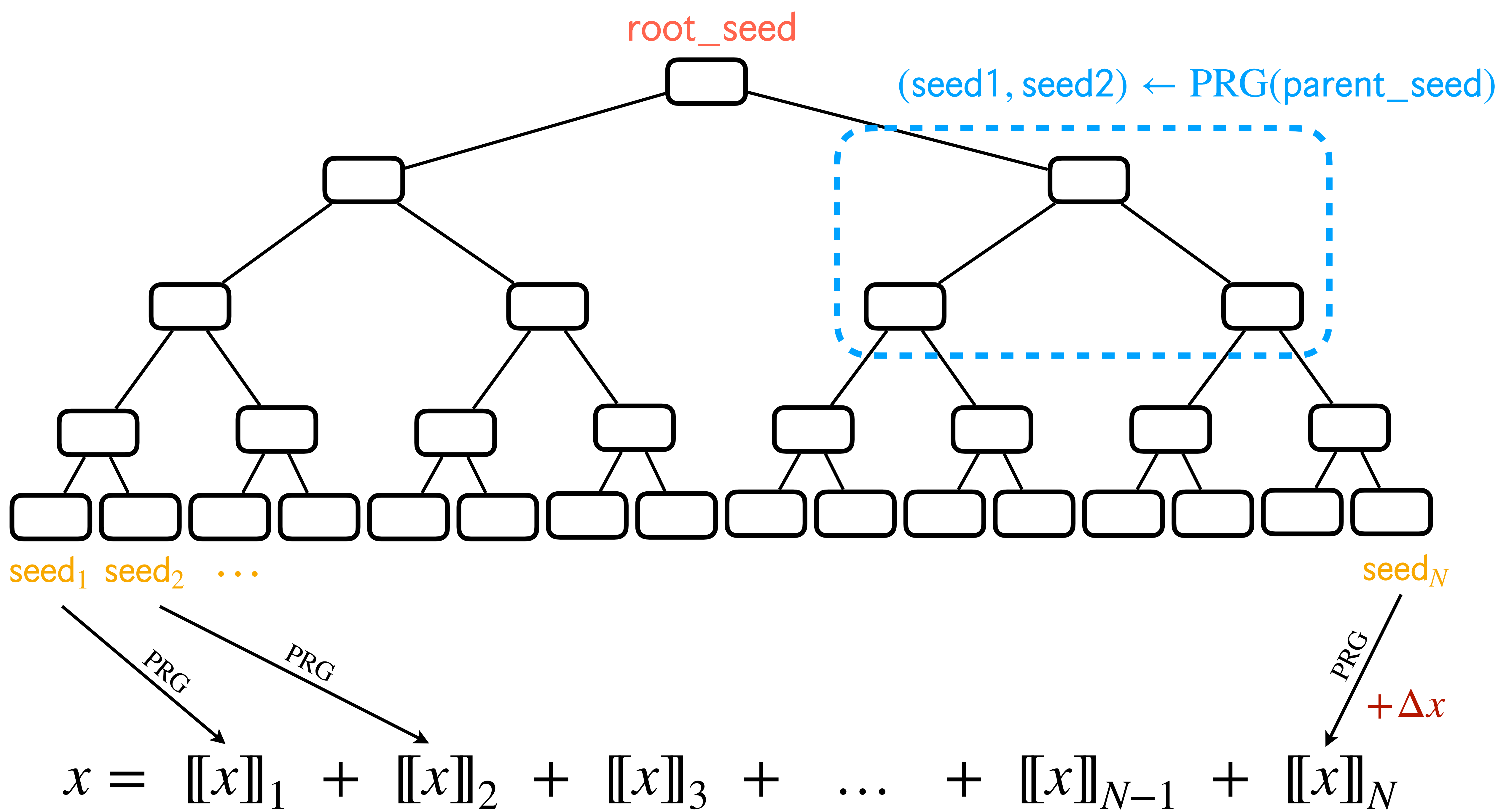
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)

$$\mathrm{seed}_1 \xrightarrow{\mathrm{PRG}} [\![x]\!]_1, \quad \mathrm{seed}_2 \xrightarrow{\mathrm{PRG}} [\![x]\!]_2, \quad \mathrm{seed}_3 \xrightarrow{\mathrm{PRG}} [\![x]\!]_3, \quad \ldots, \quad \mathrm{seed}_{N-1} \xrightarrow{\mathrm{PRG}} [\![x]\!]_{N-1}, \quad \mathrm{seed}_N \xrightarrow{\mathrm{PRG}\ +\Delta x} [\![x]\!]_N$$

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

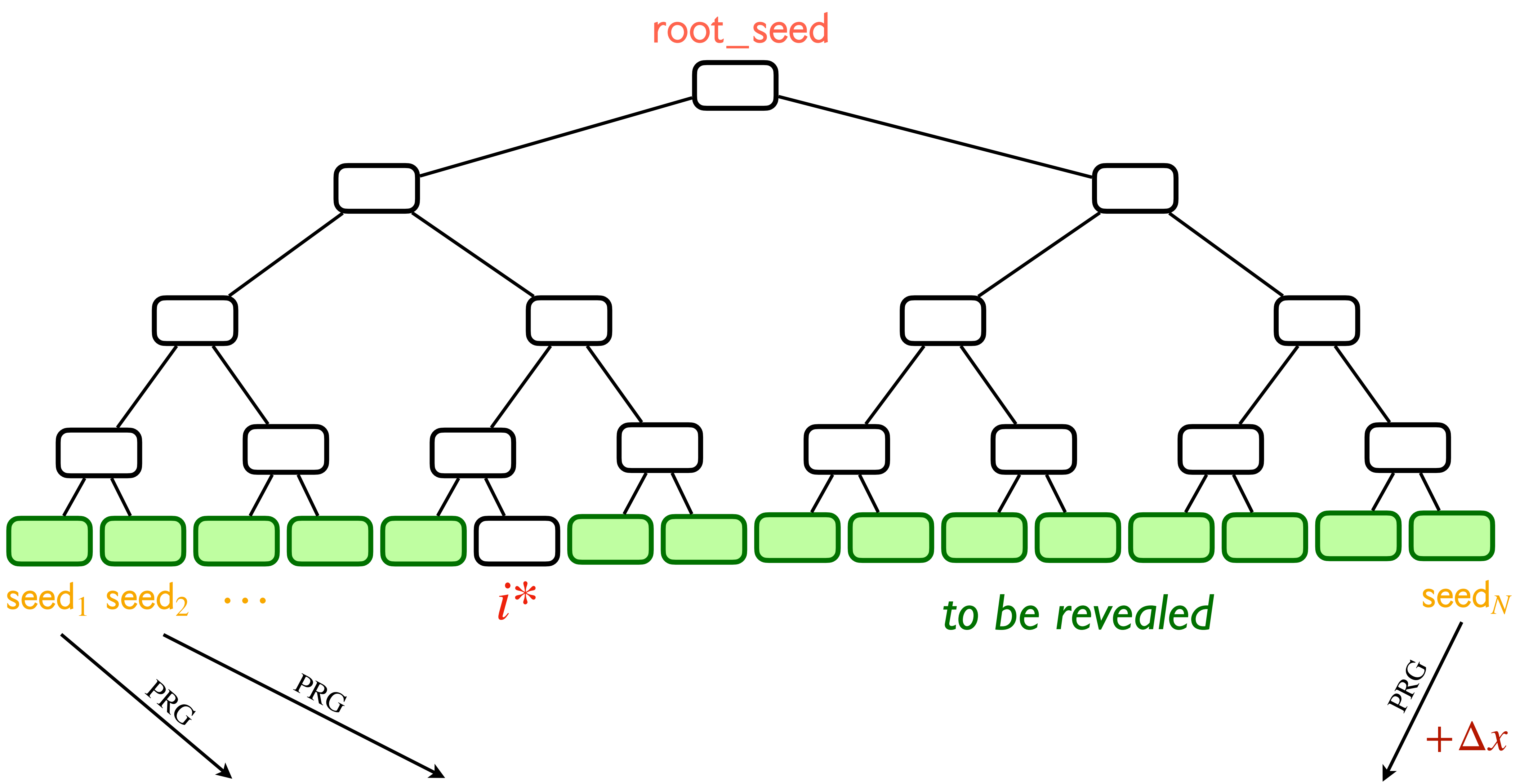
Using a Seed Tree

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



Using a Seed Tree

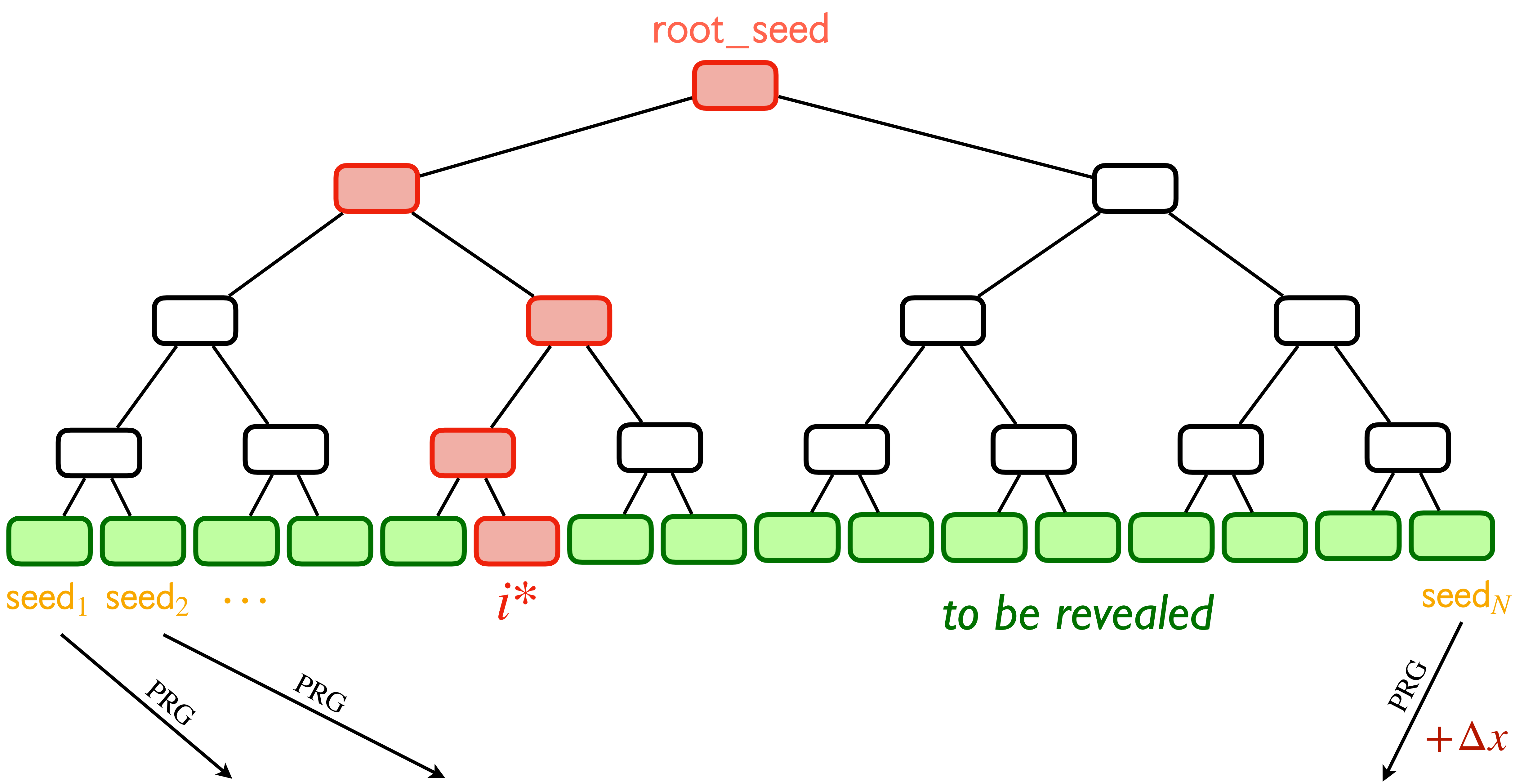
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



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

Using a Seed Tree

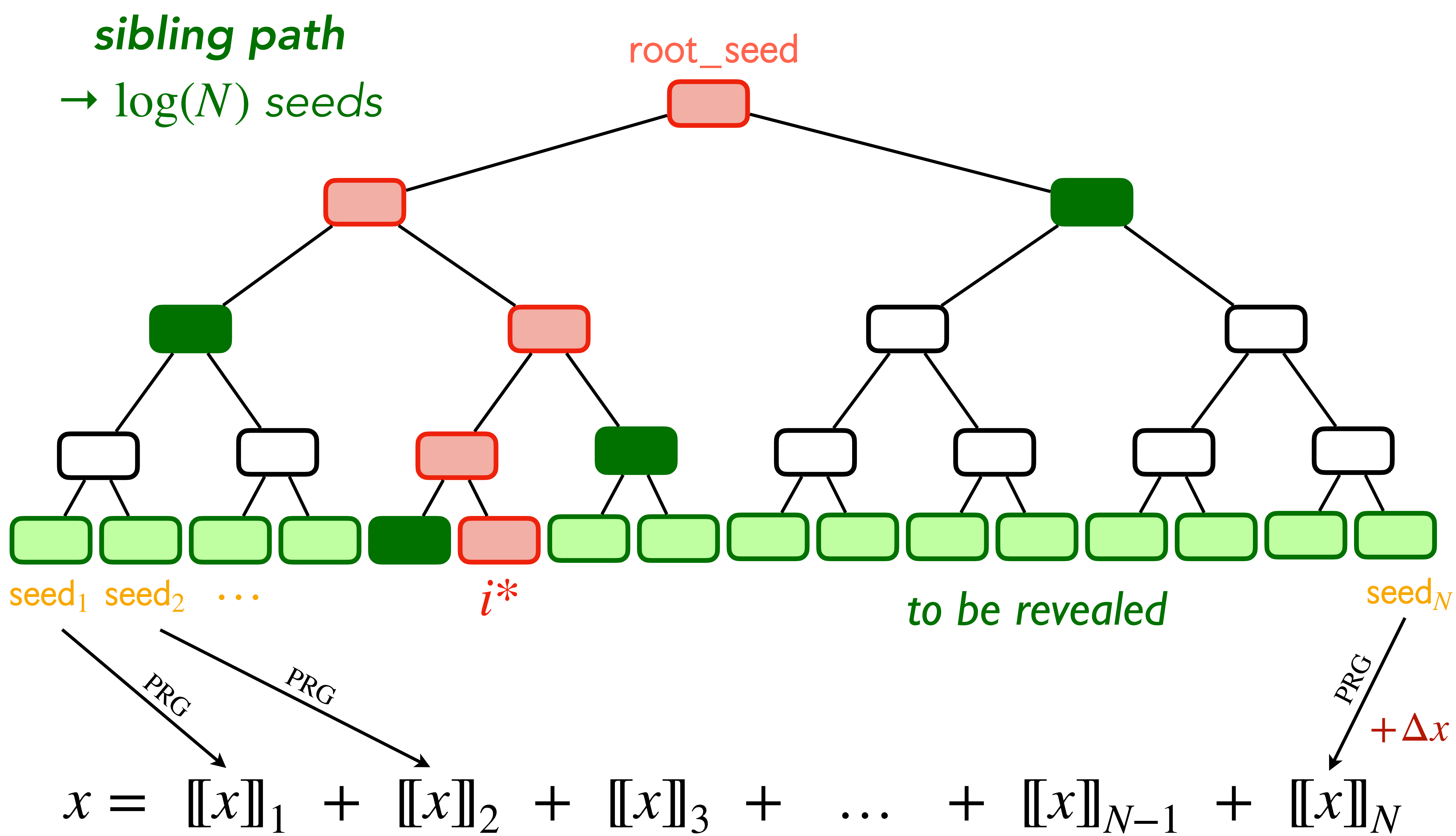
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



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

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 (of the hidden party)

Size of the auxiliary value

Path in the seed (GGM) tree

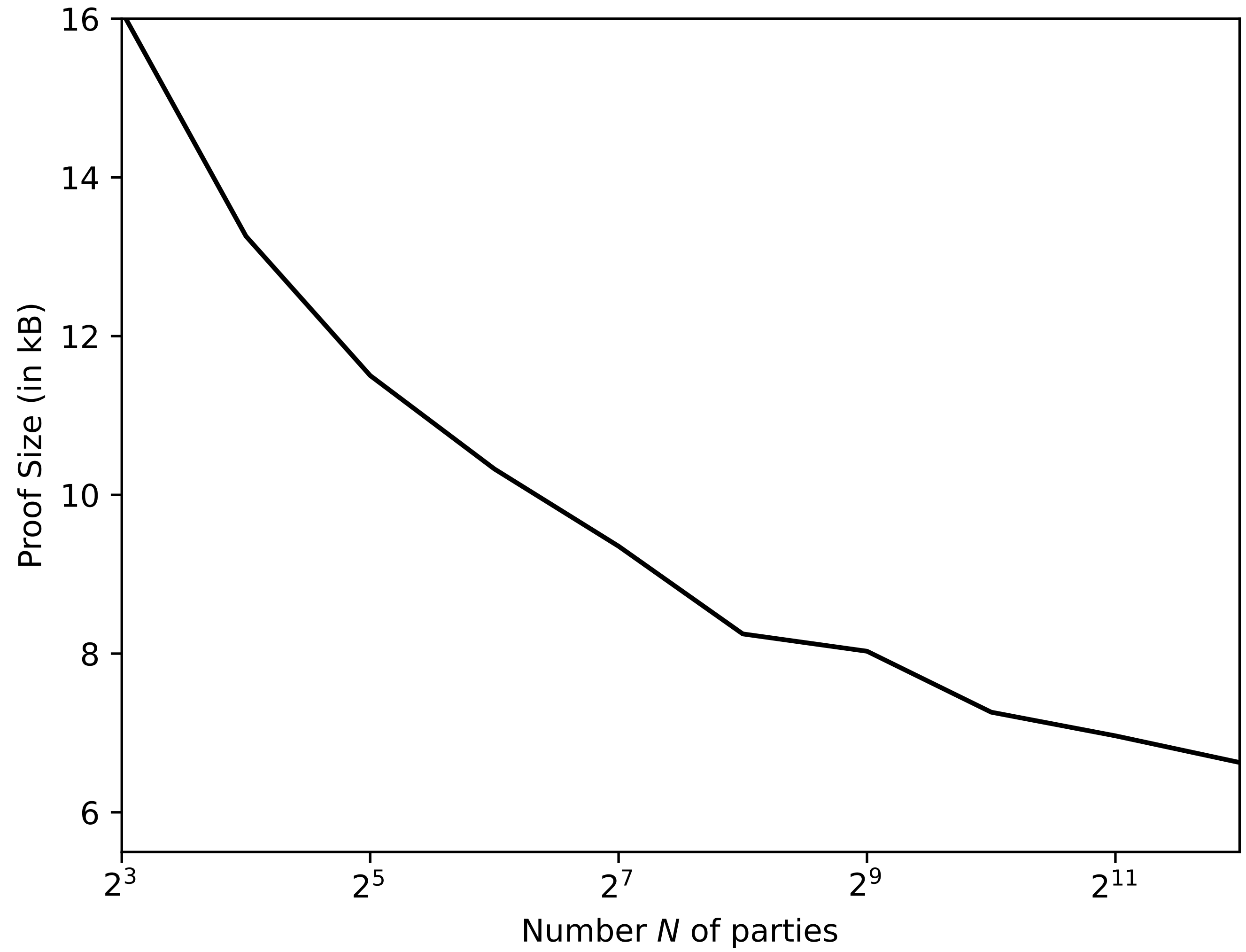
$$\text{Size} \approx \tau \cdot \left(|\Delta x| + |\alpha| + \lambda \cdot \log_2 N + 2\lambda \right)$$

Commitment of the hidden party

Number of repetitions to achieve the desired security level

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

Traditional MPCitH transformation

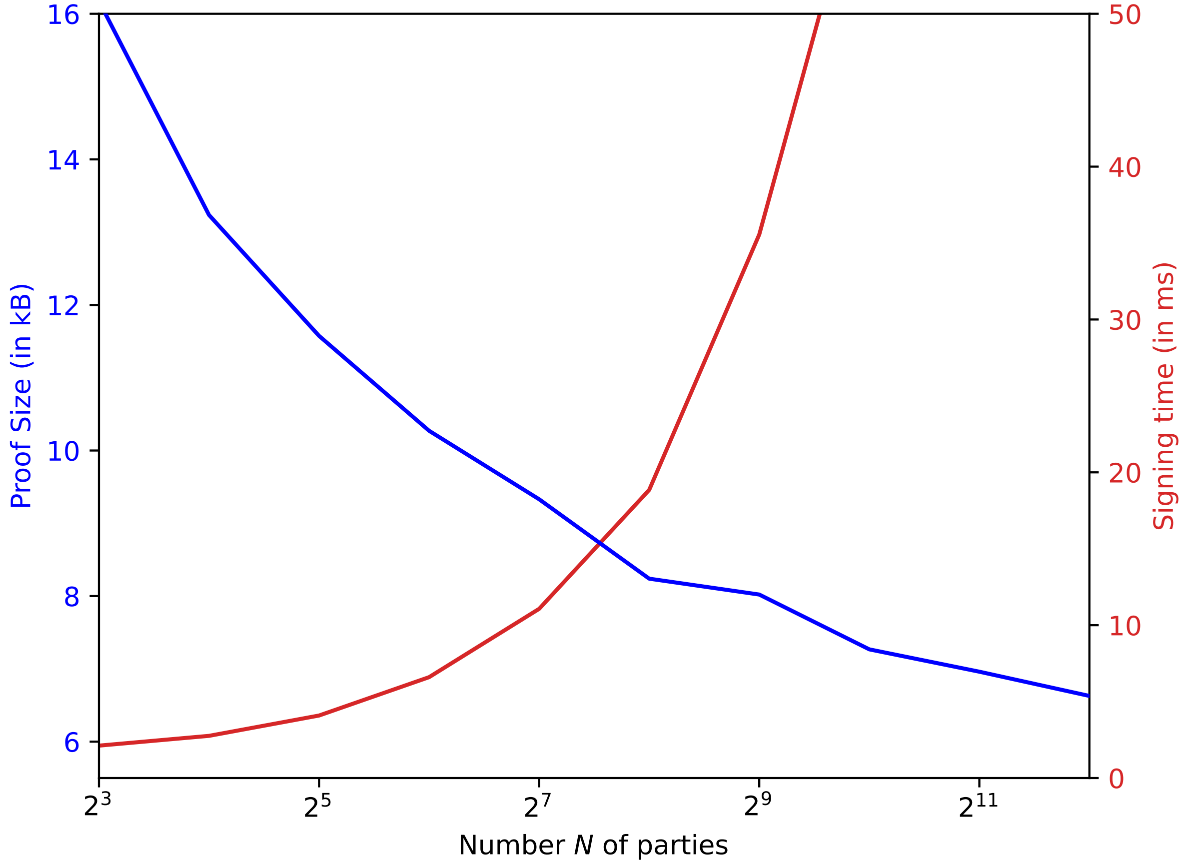


`SDitH-L1-gf251:`

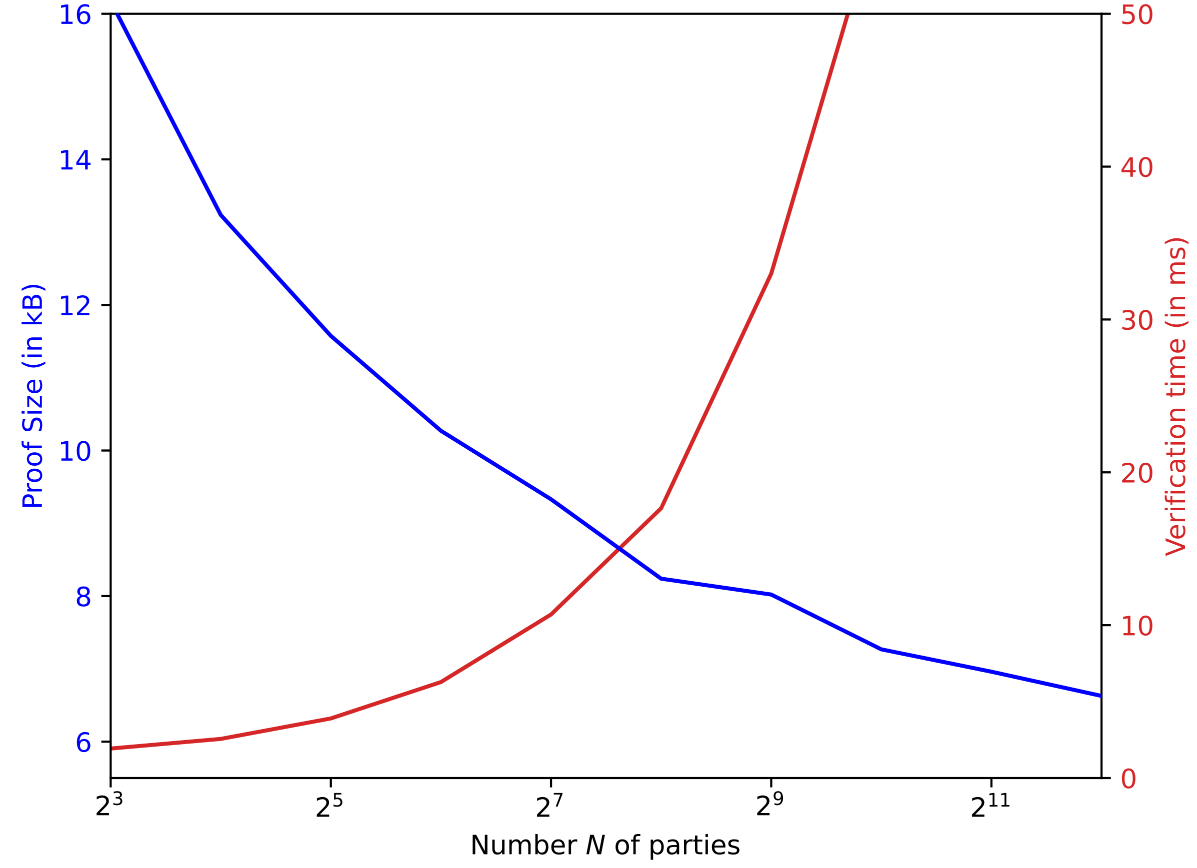
the input x of the MPC protocol is around **323** bytes,
The broadcast value α of the MPC protocol is around **36** bytes.

Traditional MPCitH transformation

Signing algorithm

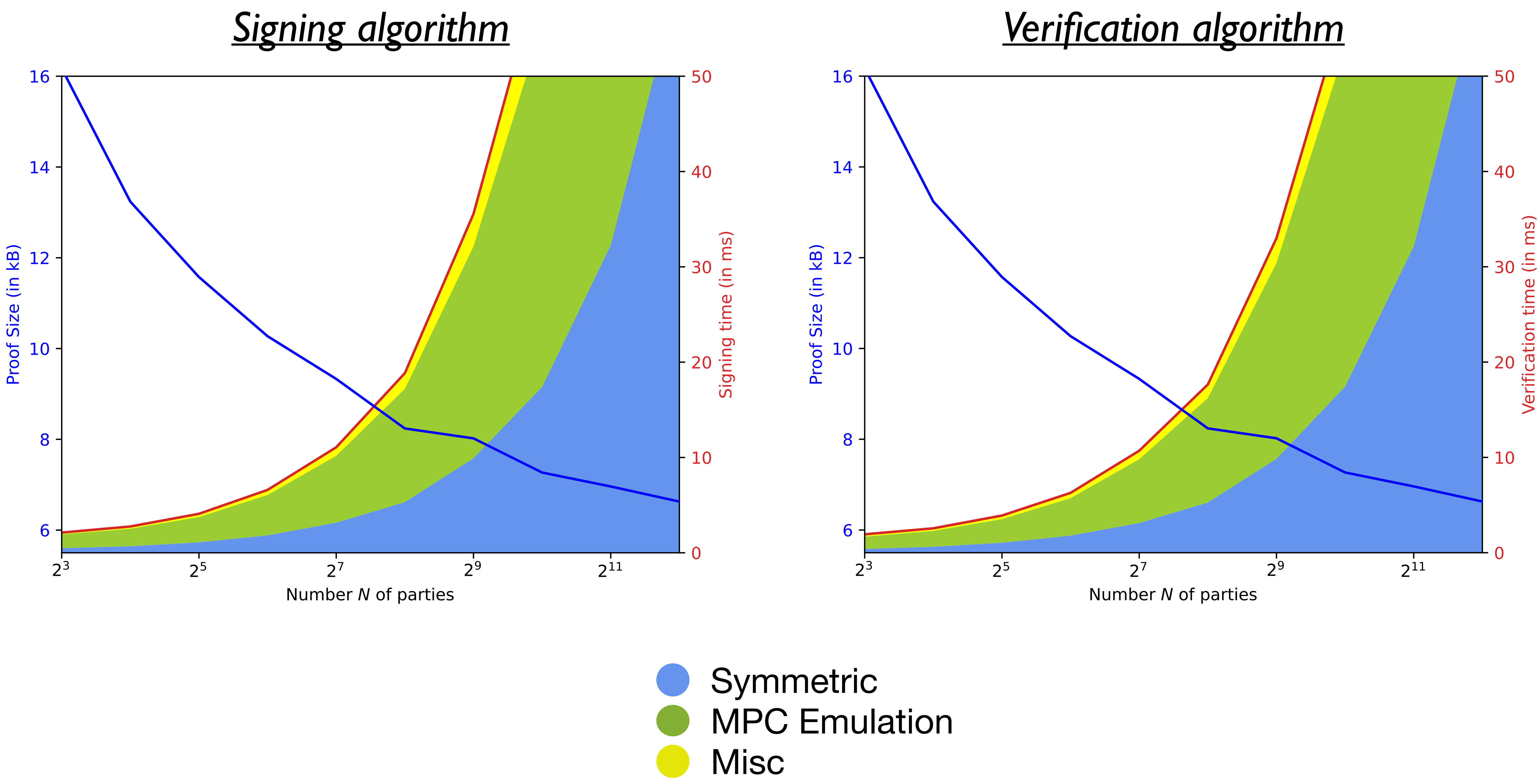


Verification algorithm



Running times @3.80Ghz

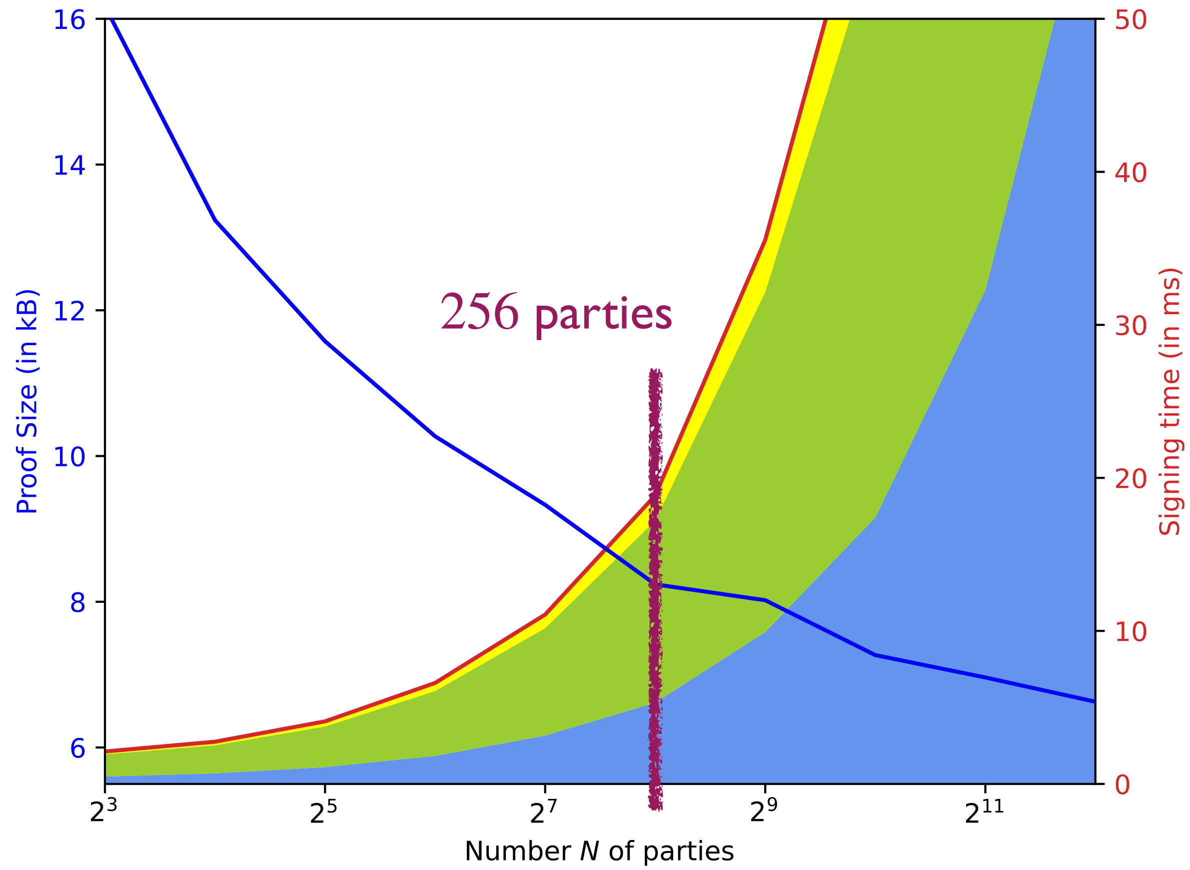
Traditional MPCitH transformation

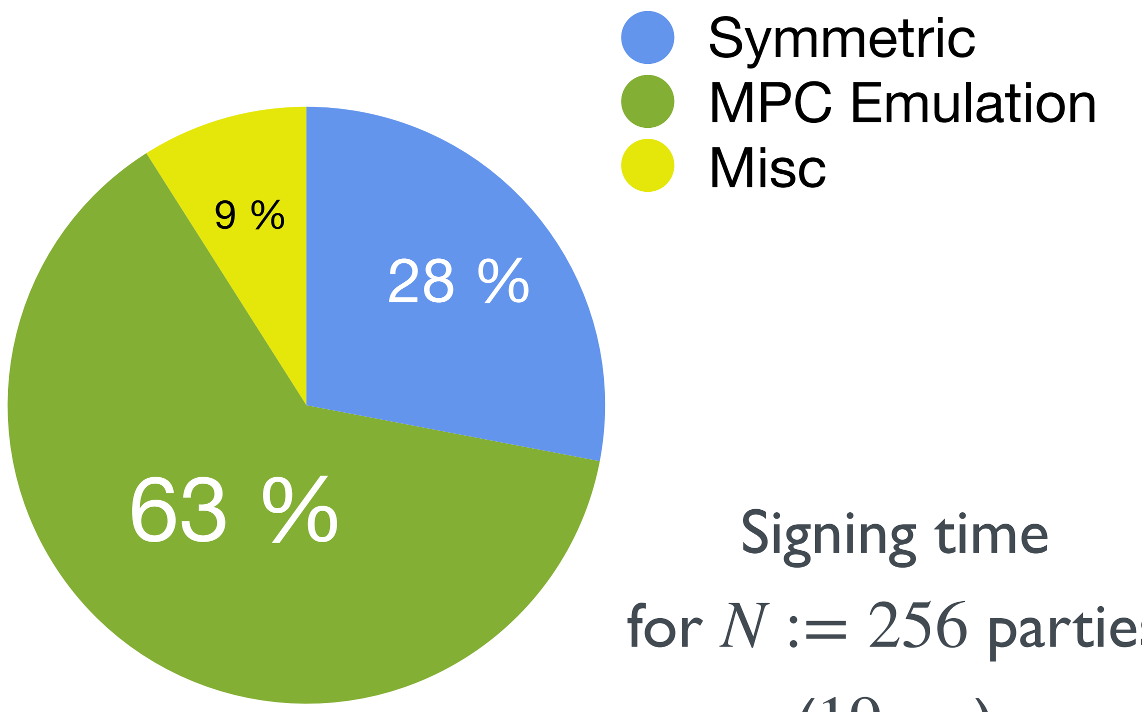


Running times @3.80Ghz

Traditional MPCitH transformation

Signing algorithm





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: 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

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

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?

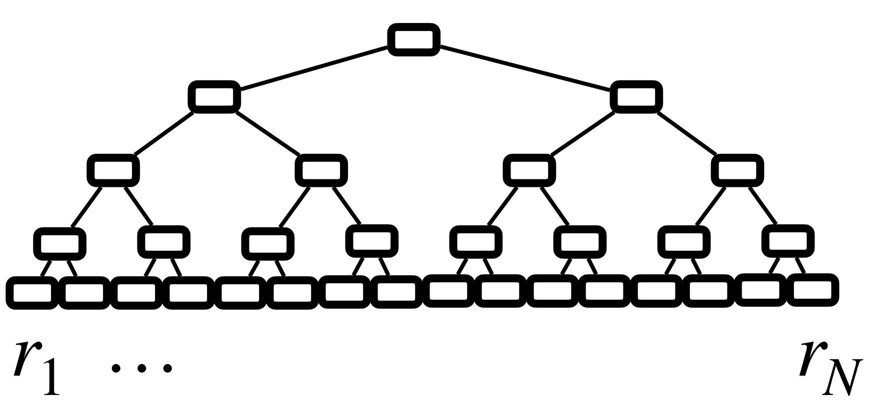
The Hypercube Technique

[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?



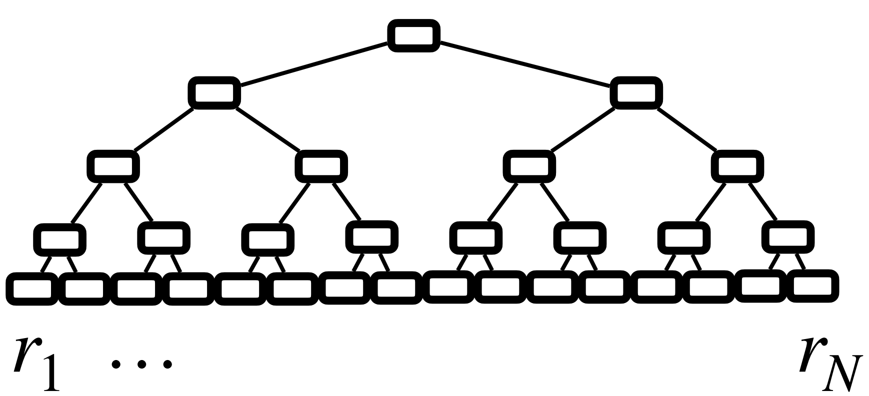
The Hypercube Technique

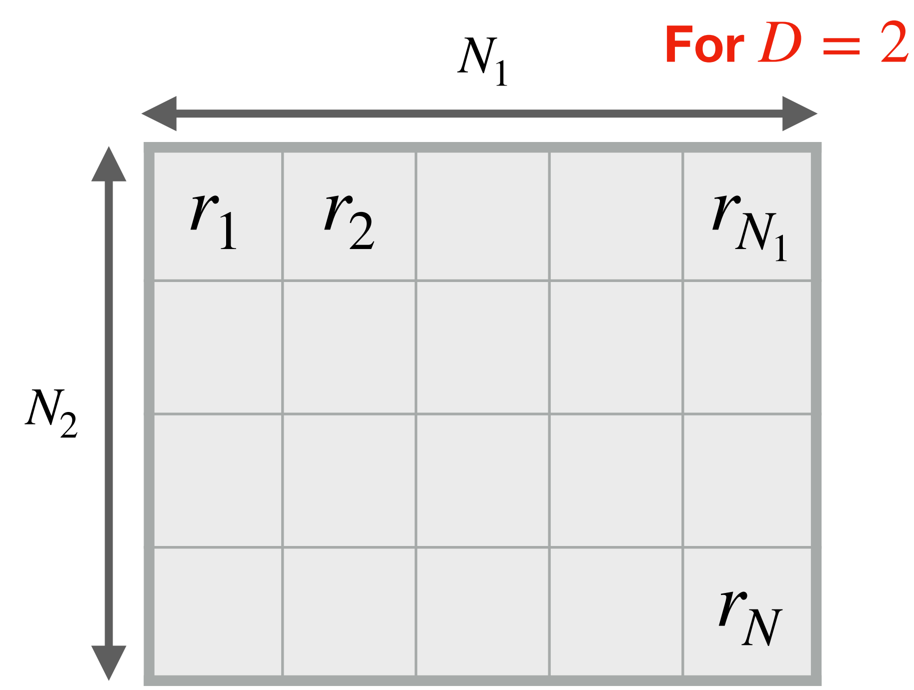
[AGHHJY23] Aguilar-Melchor, Gama, Howe, Hülsing, Joseph, Yue: "The Return of the SDitH" (Eurocrypt 2023)

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

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

How to build these D sharings?





The Hypercube Technique

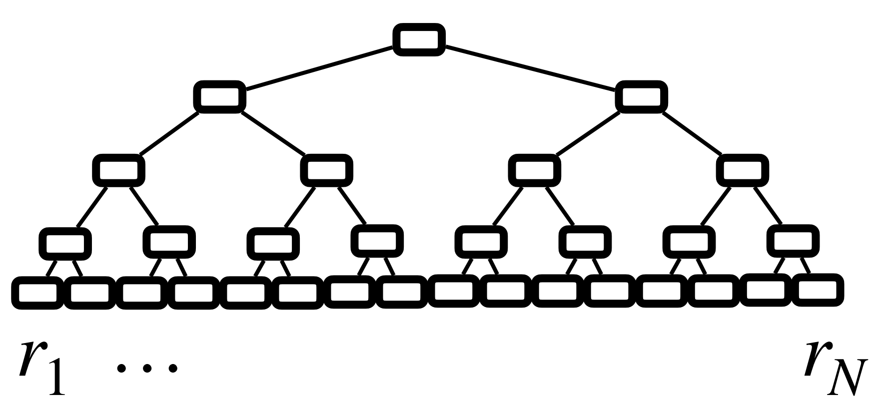
[AGHHJY23] Aguilar-Melchor, Gama, Howe, Hülsing, Joseph, Yue: "The Return of the SDitH" (Eurocrypt 2023)

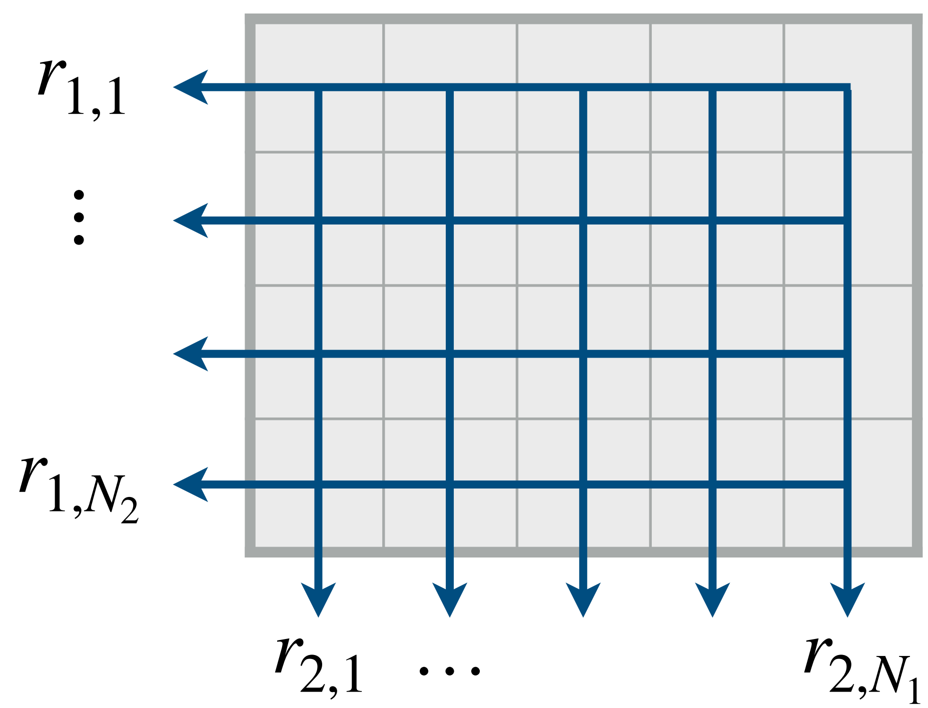
$$x = \left\{ \begin{array}{c} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \ldots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{array} \right\} + \Delta x$$

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

How to build these D sharings?

For $D = 2$





The Hypercube Technique

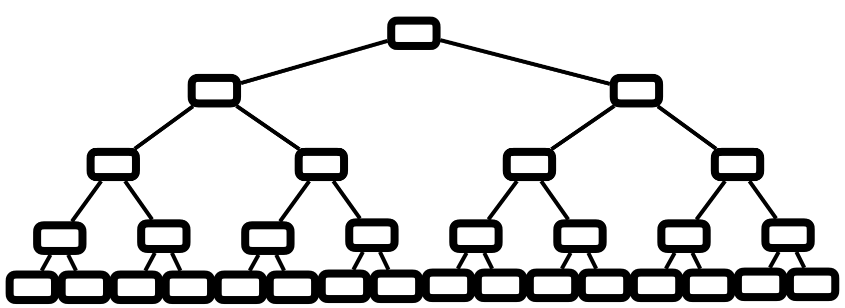
[AGHHJY23] Aguilar-Melchor, Gama, Howe, Hülsing, Joseph, Yue: "The Return of the SDitH" (Eurocrypt 2023)

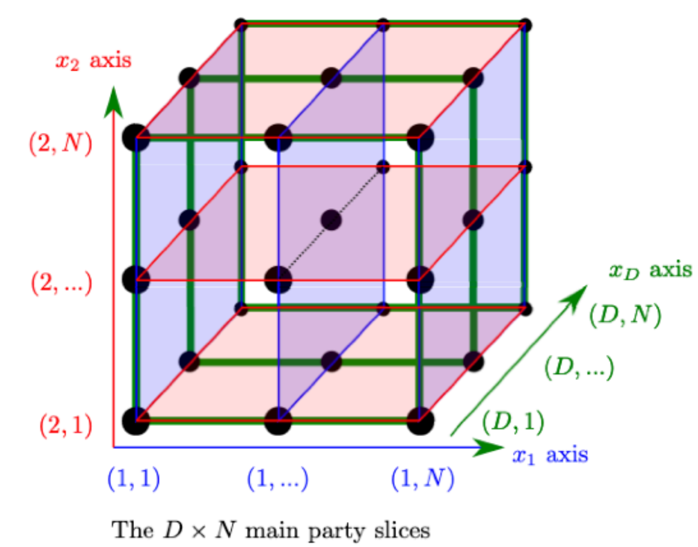
$$x = \begin{Bmatrix} r_{1,1} + r_{1,2} + \ldots + r_{1,N_1} \\ r_{2,1} + r_{2,2} + \ldots + r_{2,N_2} \\ \cdots \\ r_{D,1} + r_{D,2} + \ldots + r_{D,N_D} \end{Bmatrix} + \Delta x$$

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

How to build these D sharings?

For $D \geq 2$


$r_1 \quad \cdots \qquad r_N$



Source: Figure from [AGHHJY23]

The Hypercube Technique

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

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

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

Performance

- Same soundness error as before: $1/N$
- Same signature size as before: 1 auxiliary value + 1 seed tree of N leaves

The Hypercube Technique

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

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

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

Performance

- Same soundness error as before: $1/N$
- Same signature size as before: 1 auxiliary value + 1 seed tree of N leaves
- Emulation cost: one needs to emulate

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

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

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

$1 + \log_2 N$

The Hypercube Technique

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

Traditional: N party emulations per repetition

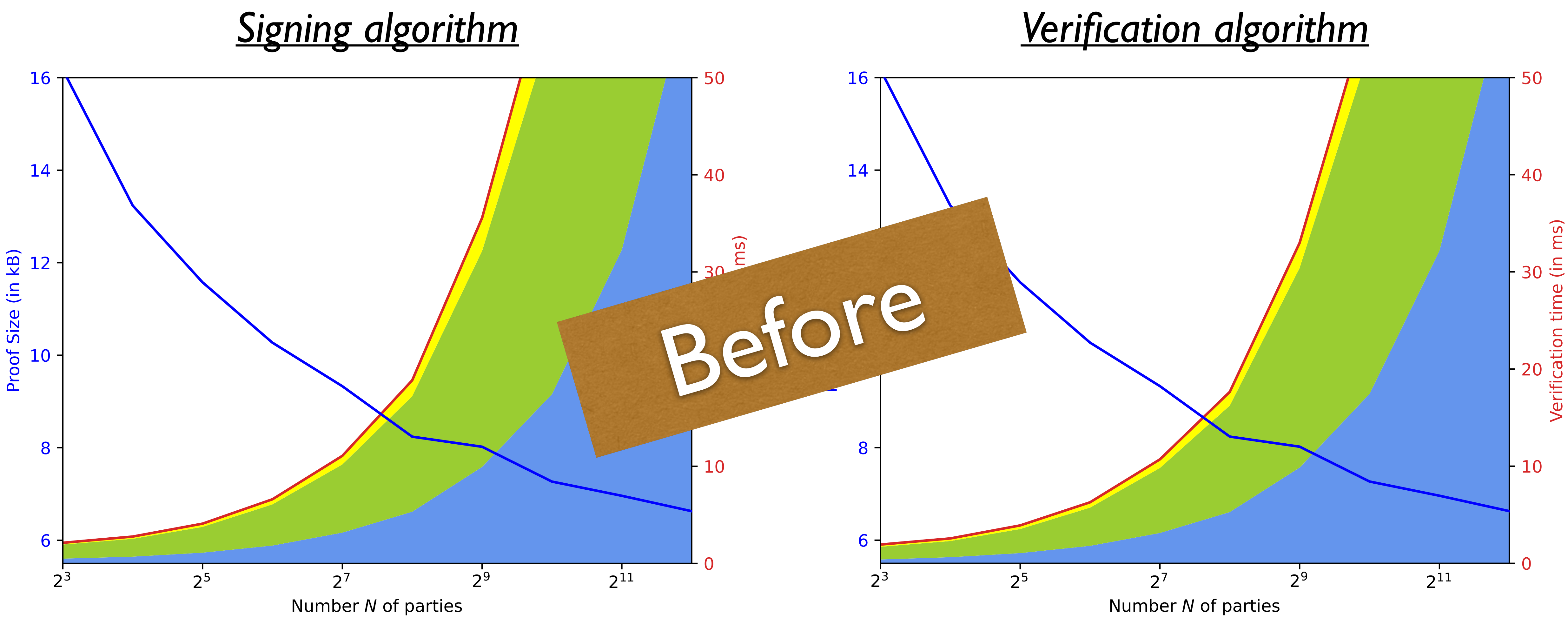
$$D = \log_2 N$$
$$N_1 = \ldots = N_D = 2$$

$$N = 256$$

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

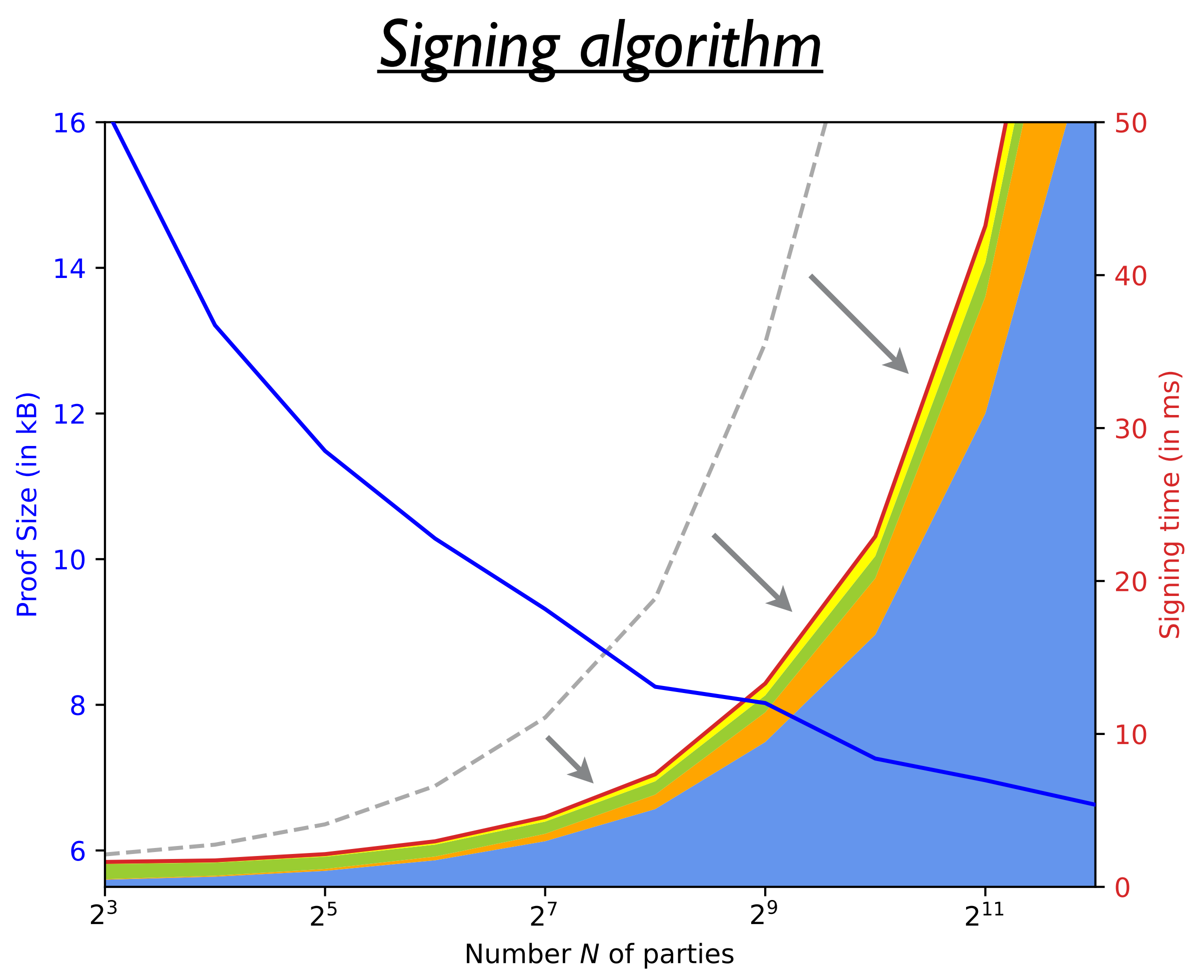
$$1 + \log_2 N = 9$$

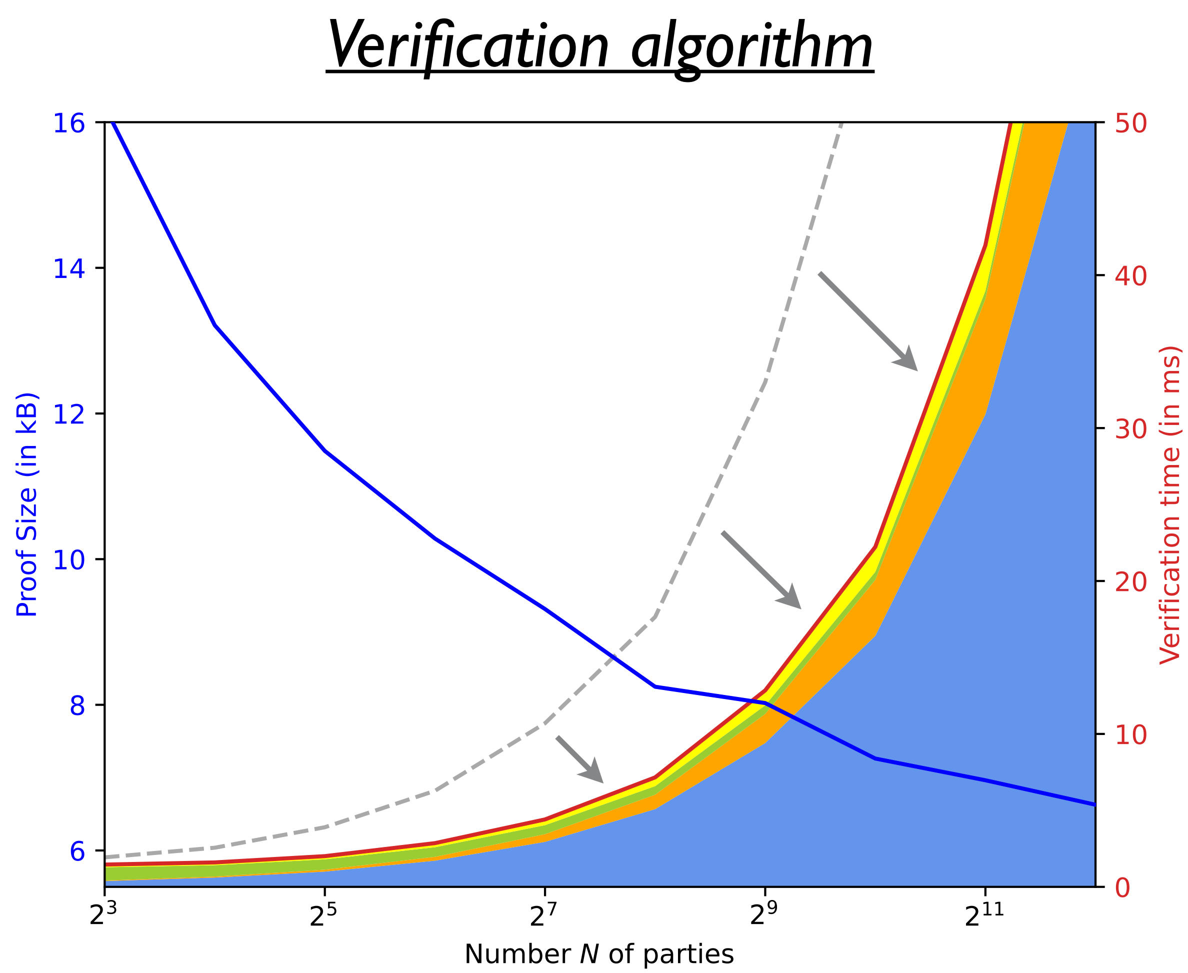
The Hypercube Technique



Running times @3.80Ghz

The Hypercube Technique



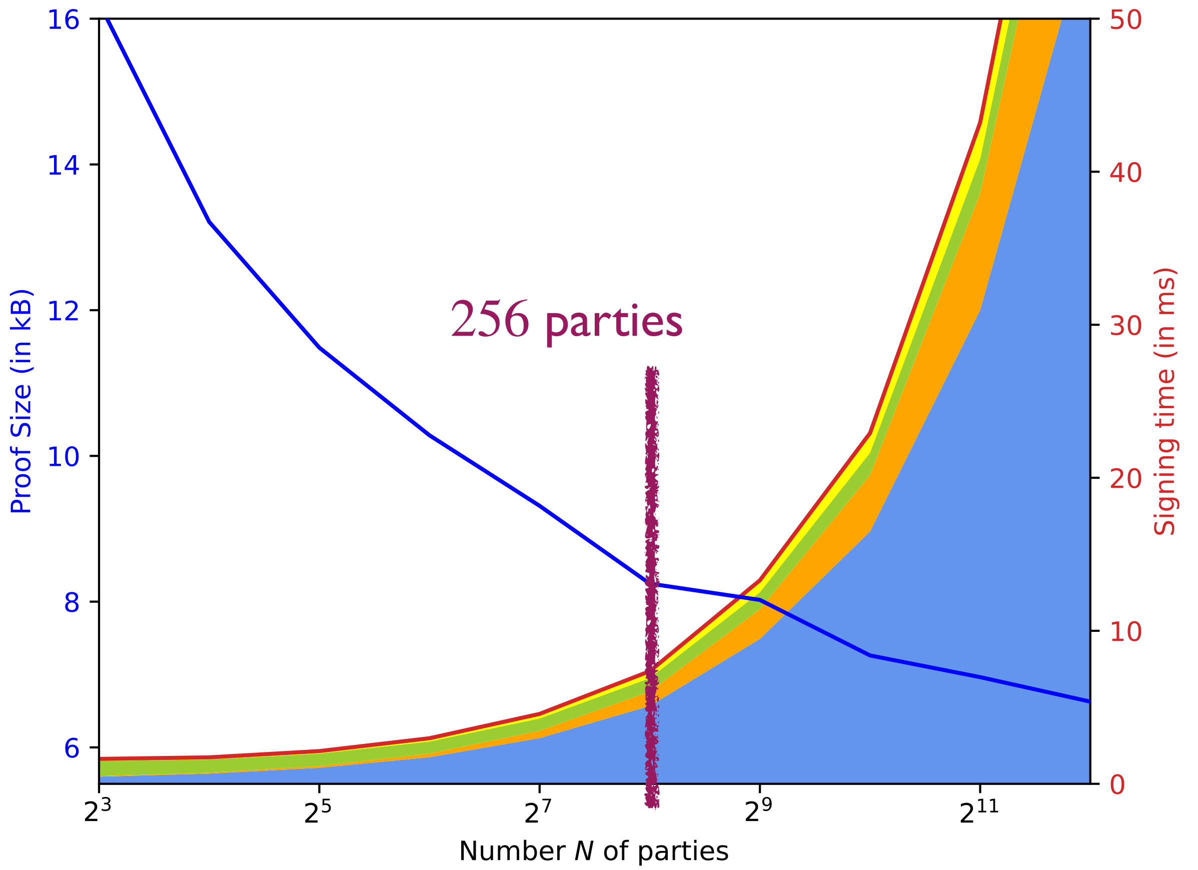


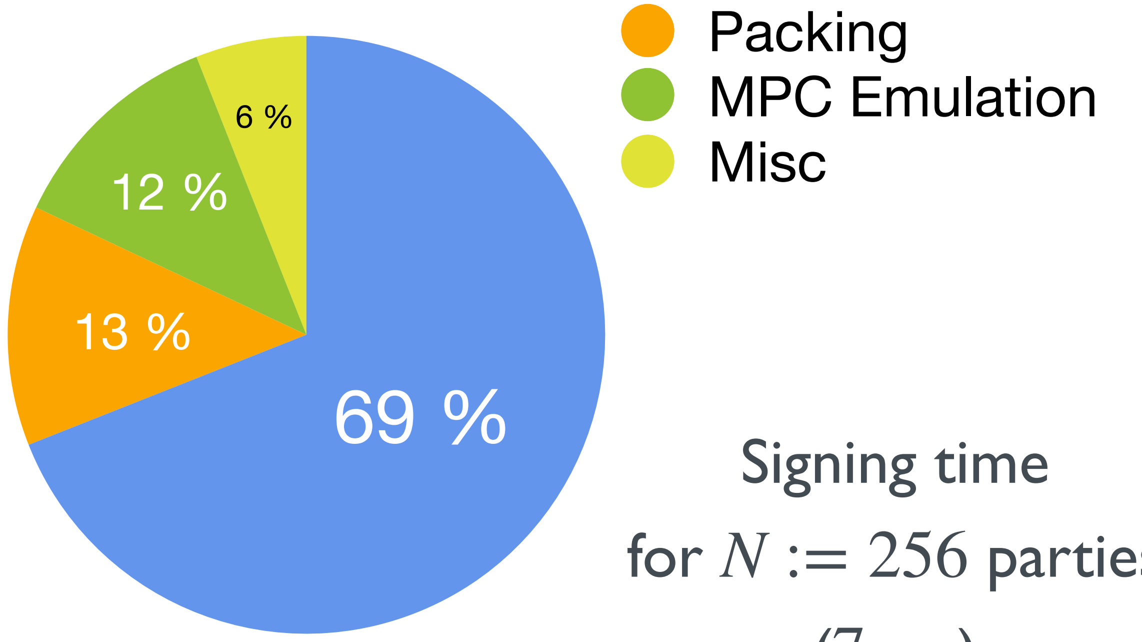
- Symmetric
- Packing
- MPC Emulation
- Misc

Running times @3.80Ghz

The Hypercube Technique

Signing algorithm





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

Running times @3.80Ghz

The (original) Threshold Approach

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

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

To share a value x,

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

The (original) Threshold Approach

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

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

The prover reveals only ℓ shares to the verifier (instead of $N - 1$).

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

The (original) Threshold Approach

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

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

The prover reveals only ℓ shares to the verifier (instead of $N - 1$).

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

Construction:

- The verifier just needs to re-emulate ℓ **parties** (per repetition);

The (original) Threshold Approach

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

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

The prover reveals only ℓ shares to the verifier (instead of $N - 1$).

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

Construction:

- The verifier just needs to re-emulate ℓ **parties** (per repetition);
- The prover just needs to emulate $1 + \ell$ **parties** (per repetition);

The (original) Threshold Approach

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

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

The prover reveals only ℓ shares to the verifier (instead of $N - 1$).

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

Construction:

- The verifier just needs to re-emulate ℓ **parties** (per repetition);
- The prover just needs to emulate $1 + \ell$ **parties** (per repetition);
- The prover uses a Merkle tree to commit the shares;

The (original) Threshold Approach

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

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

The prover reveals only ℓ shares to the verifier (instead of $N - 1$).

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

Construction:

- The verifier just needs to re-emulate ℓ **parties** (per repetition);
- The prover just needs to emulate $1 + \ell$ **parties** (per repetition);
- The prover uses a Merkle tree to commit the shares;
- The obtained signature size is **larger**;

The (original) Threshold Approach

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

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

The prover reveals only ℓ shares to the verifier (instead of $N - 1$).

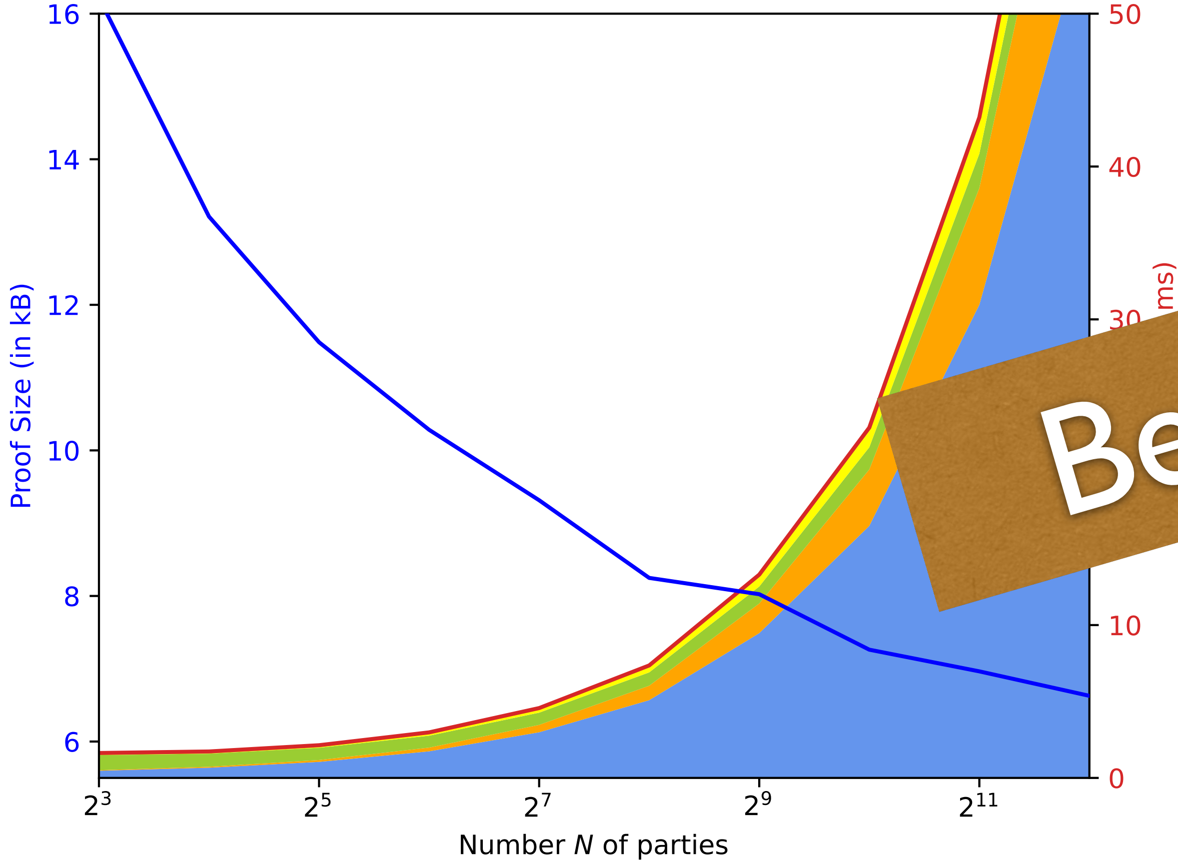
In practice, $\ell \in \{1,2,3\}$.

Construction:

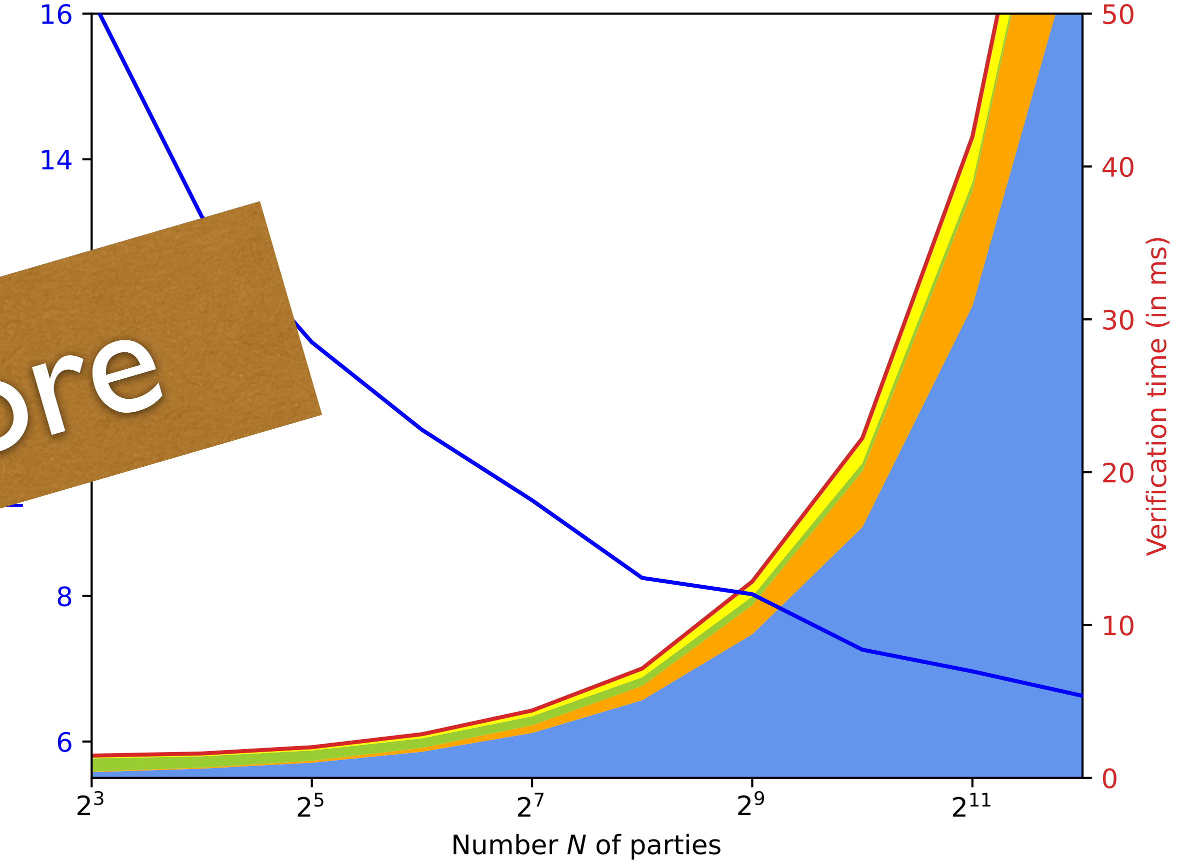
- The verifier just needs to re-emulate ℓ **parties** (per repetition);
- The prover just needs to emulate $1 + \ell$ **parties** (per repetition);
- The prover uses a Merkle tree to commit the shares;
- The obtained signature size is **larger**;
- We have the constraint: $N \leq |\mathbb{F}|$.

The (original) Threshold Approach

Signing algorithm



Verification algorithm

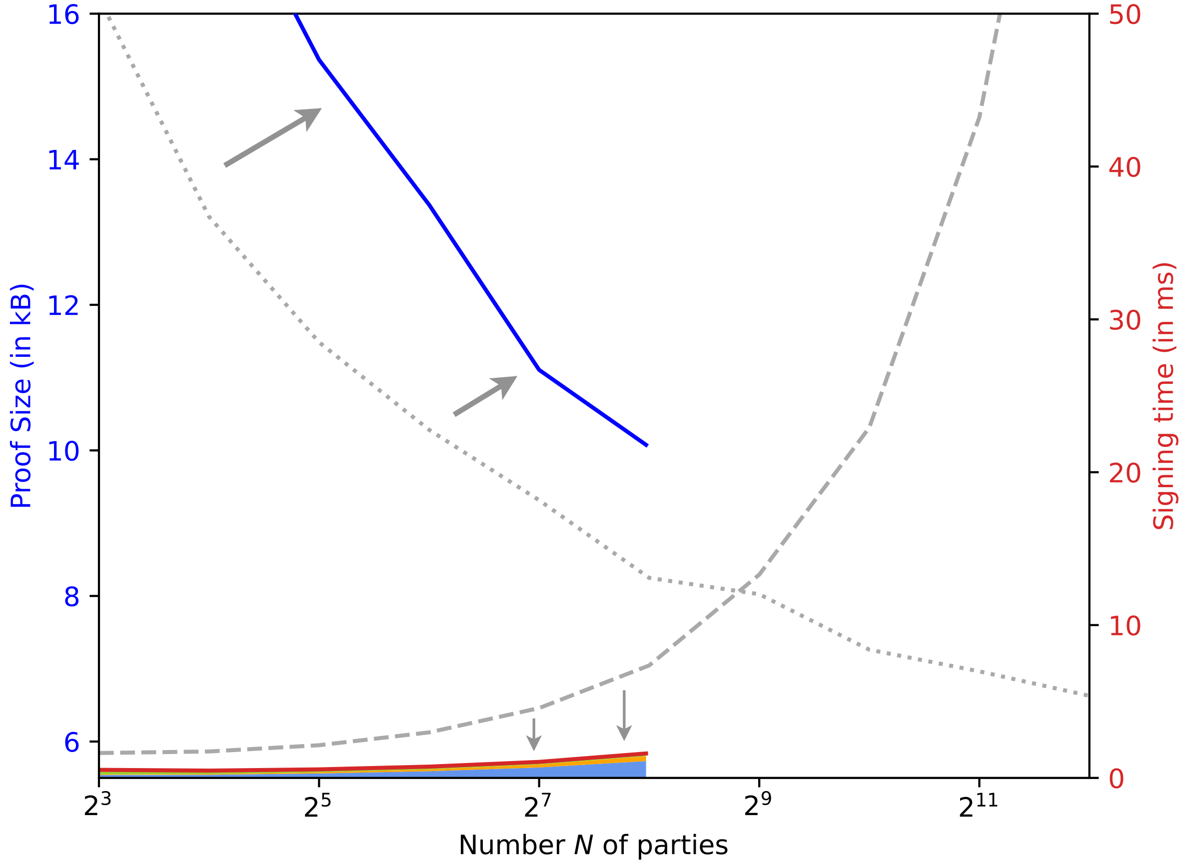


Before

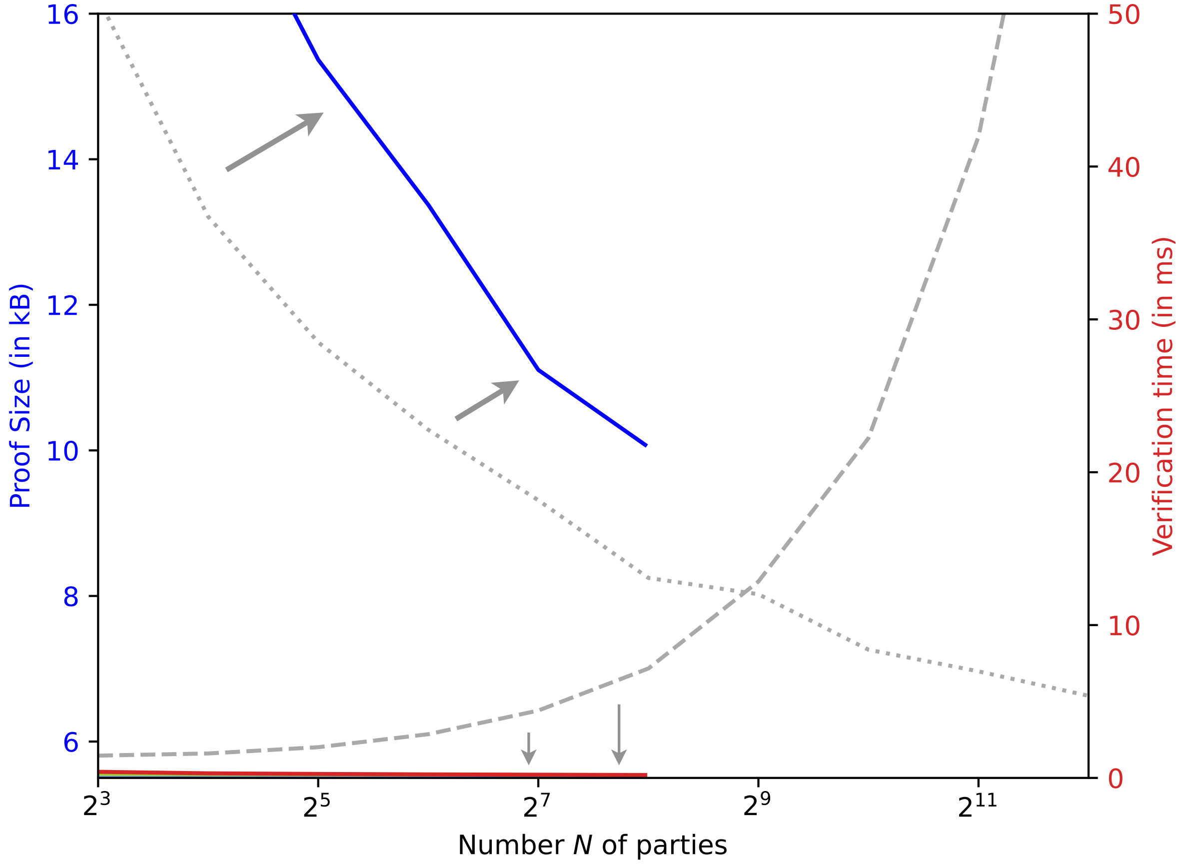
Running times @3.80Ghz

The (original) Threshold Approach

Signing algorithm



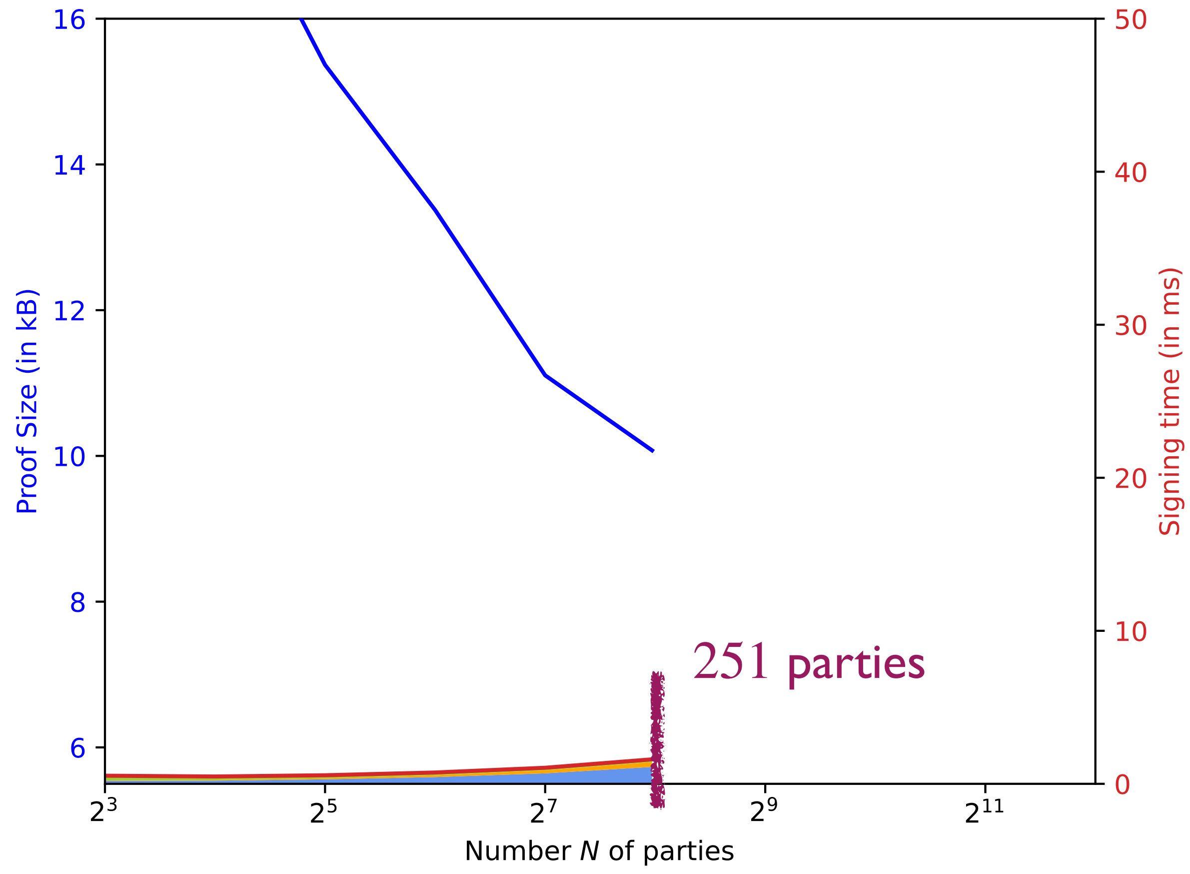
Verification algorithm

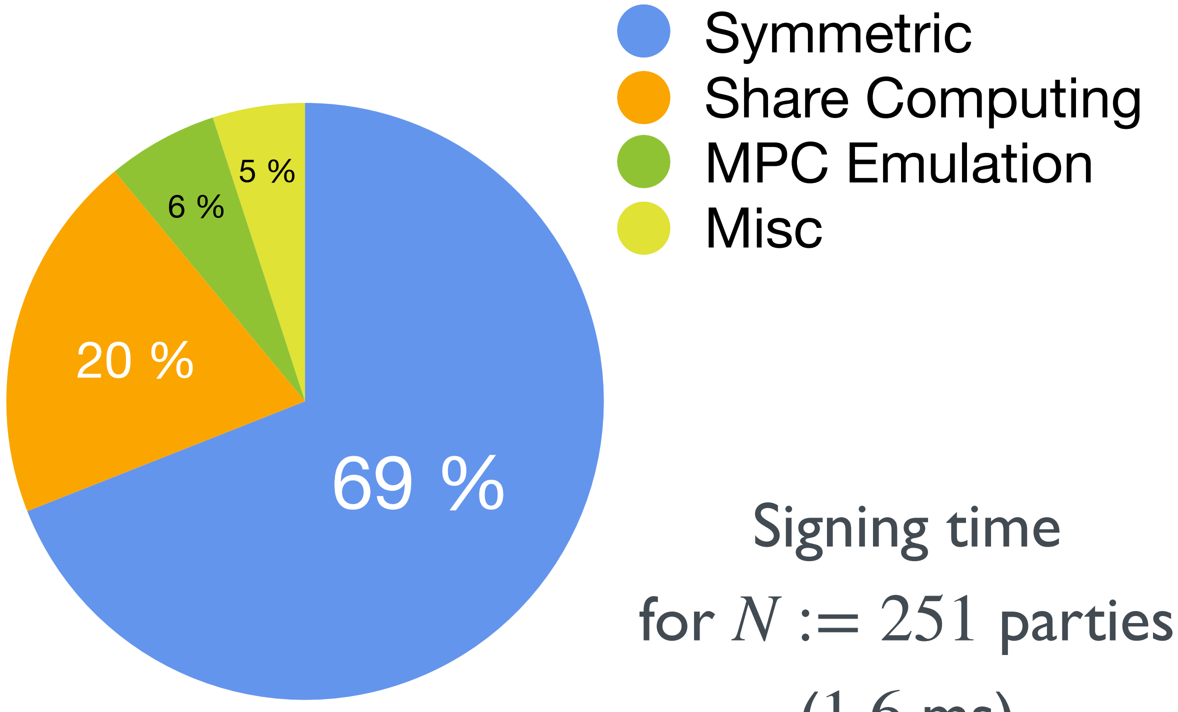


Running times @3.80Ghz

The (original) Threshold Approach

Signing algorithm

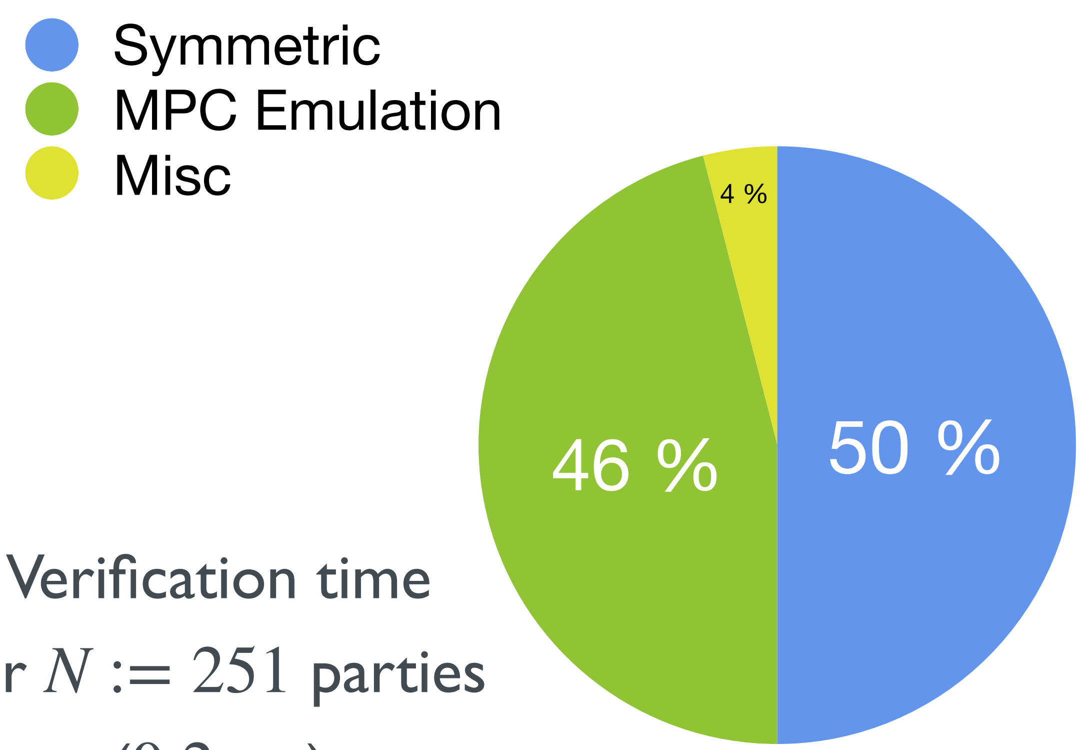




Signing time
for $N := 251$ parties
(1.6 ms)

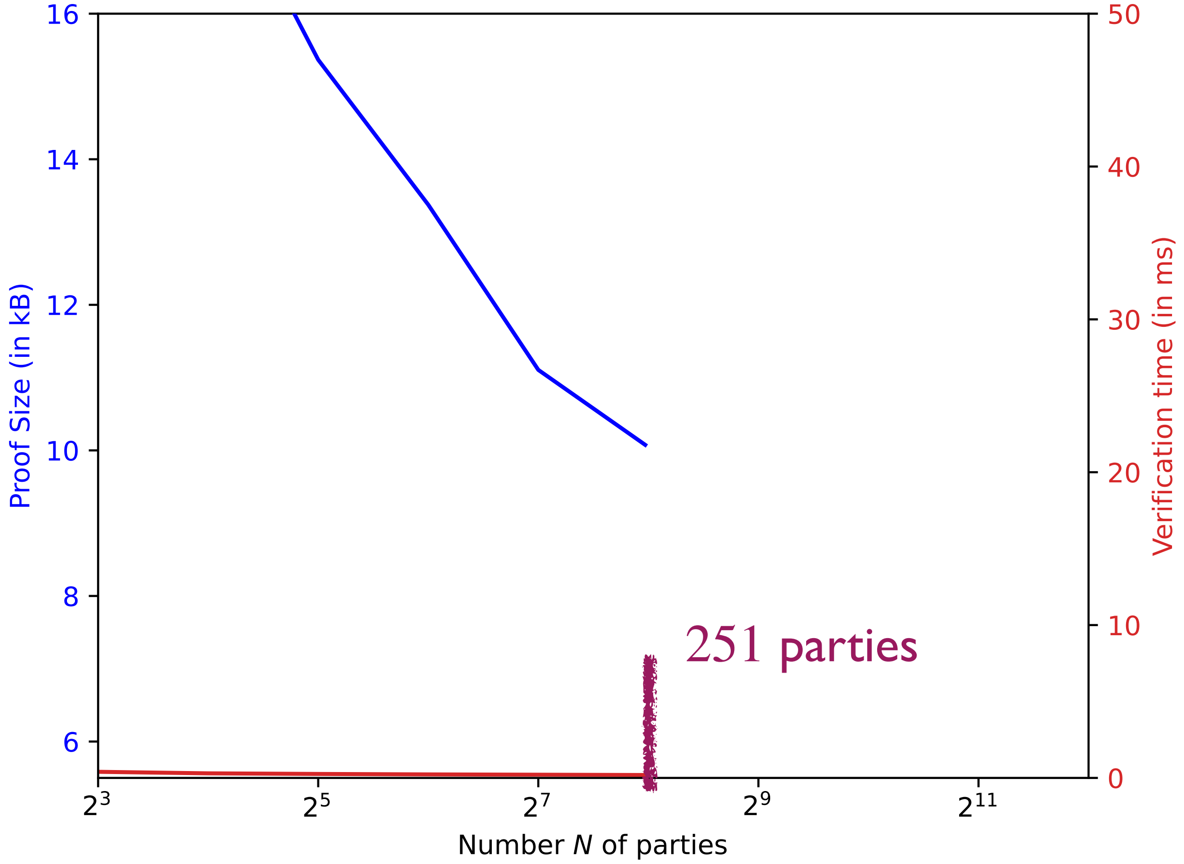
Running times @3.80Ghz

The (original) Threshold Approach



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

Verification algorithm



Running times @3.80Ghz

The existing MPCitH transforms

Traditional

Hypercube

Shorter signature sizes
Highly parallelizable
Slower signing time
Signing time ≈ Verification time
Computational cost is mainly due to symmetric primitives

Threshold

Faster signing time
Highly parallelizable
Very fast verification
Larger signature size
Restriction # of parties
Computational cost is mainly due to arithmetics

MPCitH-based NIST candidates

	Short Instance	Fast Instance
AIMer	Traditional (256-1615)	Traditional (16-57)
Biscuit	Traditional (256)	Traditional (16)
MIRA	Hypercube (256)	Hypercube (32)
MiRitH	Traditional (256)	Traditional (16)
	Hypercube (256)	Hypercube (16)
MQOM	Hypercube (256)	Hypercube (32)
RYDE	Hypercube (256)	Hypercube (32)
SDitH	Hypercube (256)	Threshold (251-256)

FAEST and PERK rely on other MPCitH techniques.

Conclusion

Advantages and limitations

- Limitations
  - Relatively ***slow*** *(few milliseconds)*
    - Greedy use of symmetric cryptography
  - Relatively ***large*** signatures (*3-10 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 (*3-10 KB for L1*)
  - Signature size: ***quadratic*** growth in the security level

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

Conclusion

- MPC-in-the-Head
  - Very versatile and tunable
  - Can be applied on any one-way function
  - A practical tool to build *conservative* signature schemes

Conclusion

- MPC-in-the-Head
  - Very versatile and tunable
  - Can be applied on any one-way function
  - A practical tool to build *conservative* signature schemes

- Recent MPCitH techniques

VOLE-in-the-Head	TC-in-the-Head
Vector-Oblivious-Linear-Evaluation-in-the-Head	Threshold-Computation-in-the-Head
presented by *Carsten Baum*	presented by *Matthieu Rivain*
June 18, 2024	July 2, 2024

Conclusion

- MPC-in-the-Head
  - Very versatile and tunable
  - Can be applied on any one-way function
  - A practical tool to build *conservative* signature schemes

- Recent MPCitH techniques

VOLE-in-the-Head	TC-in-the-Head
Vector-Oblivious-Linear-Evaluation-in-the-Head	Threshold-Computation-in-the-Head
presented by *Carsten Baum*	presented by *Matthieu Rivain*
June 18, 2024	July 2, 2024

Thank you for your attention.

References

[FMRV22] Feneuil, Maire, Rivain, Vergnaud. *Zero-Knowledge Protocols for the Subset Sum Problem from MPC-in-the-Head with Rejection*. Asiacrypt 2022.

[Fen23] Feneuil. *Post-Quantum Signatures from Secure Multiparty Computation*. PhD thesis 2023.

[Fen22] Feneuil. *Building MPCitH-based Signatures from MQ, MinRank, and Rank SD*. ACNS 2024.

[BFR23] Benadjila, Feneuil, Rivain. *MQ on my Mind: Post-Quantum Signatures from the Non-Structured Multivariate Quadratic Problem*. EuroS&P 2024.

[ARV22] Adj, Rivera-Zamarripa, Verbel. *MinRank in the Head: Short Signatures from Zero-Knowledge Proofs*. AfricaCrypt 2023

[ABB+23] Adj, Barbero, Bellini, Esser, Rivera-Zamarripa, Sanna, Verbel, Zweydinger. *MiRitH: Efficient Post-Quantum Signatures from MinRank in the Head*. TCHES 2024.

[BG22] Bidoux, Gaborit. *Compact Post-Quantum Signatures from Proofs of Knowledge leveraging Structure for the PKP, SD and RSD Problems*. C2SI 2023.

[BBD+24] Bettaieb, Bidoux, Dyseryn, Esser, Gaborit, Kulkarni, Palumbi. PERK: Compact Signature Scheme Based on a New Variant of the Permuted Kernel Problem. Journal DCC (2024).