Post-Quantum Signatures from Secure Multiparty Computation

Thibauld Feneuil

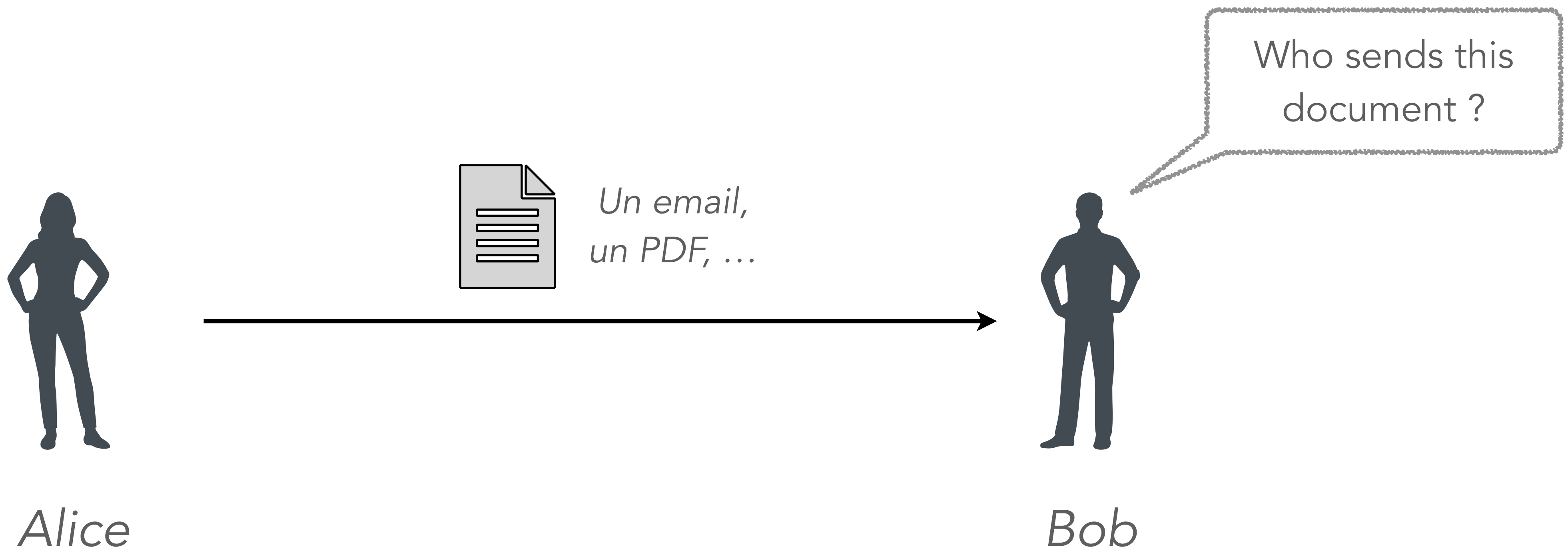
PhD Defense

October 23, 2023 — Paris (France)

Table of Contents

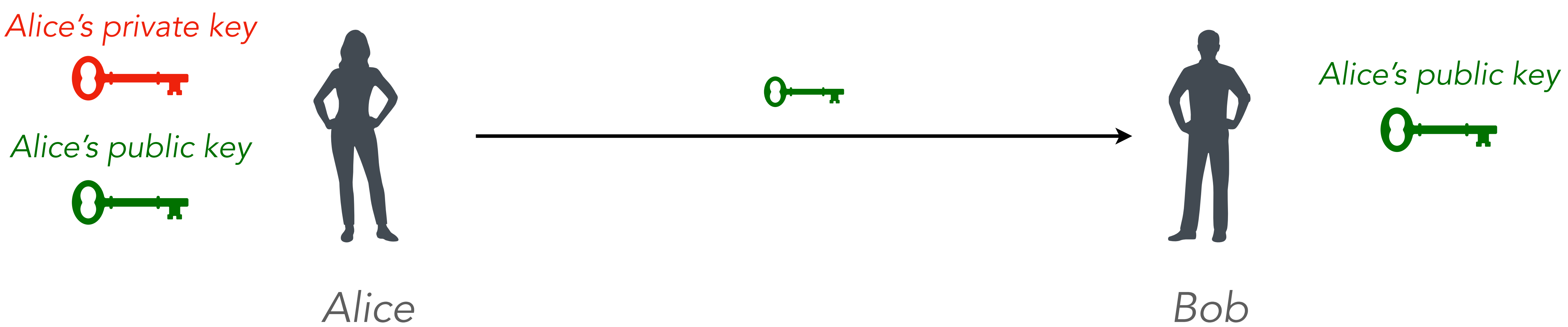
Introduction

Digital signatures



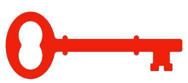
Digital signatures

Digital signatures

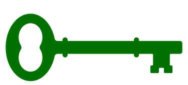


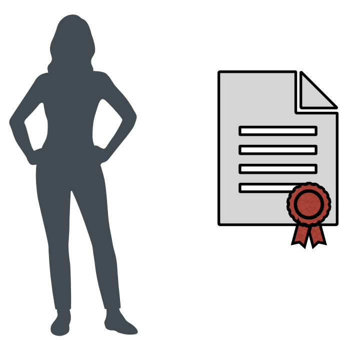
Digital signatures

Alice's private key



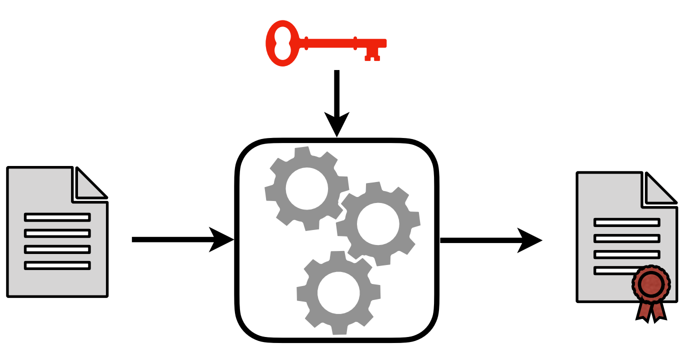
Alice's public key





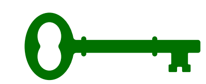
Alice

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

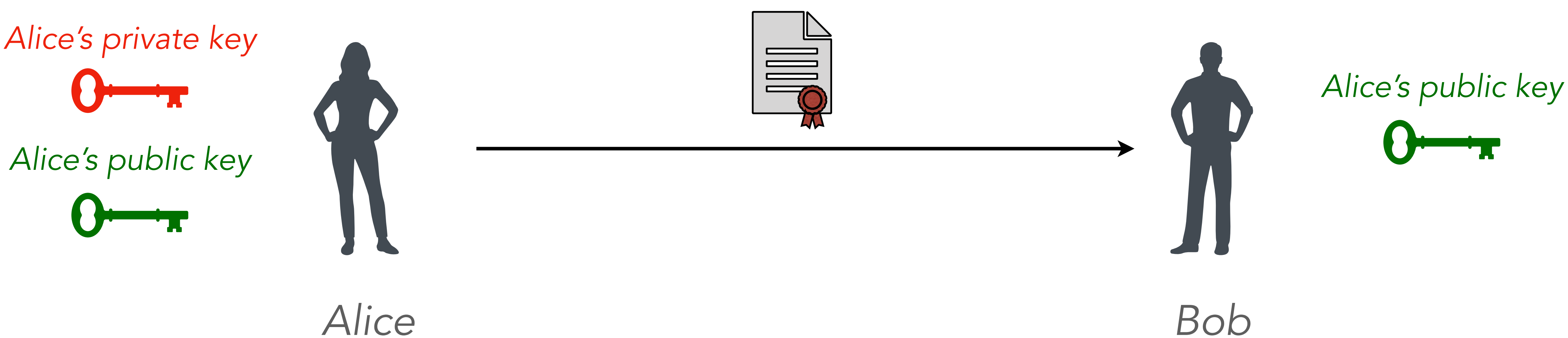


Bob

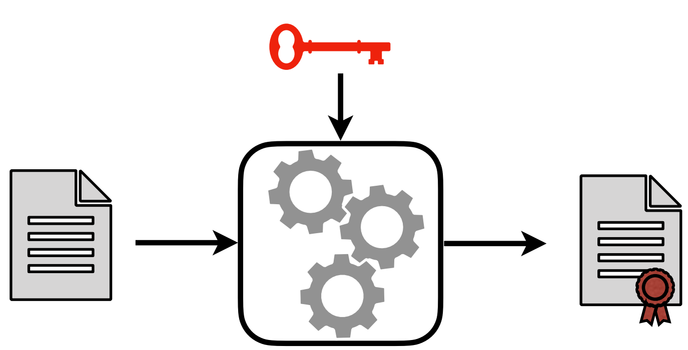
Alice's public key



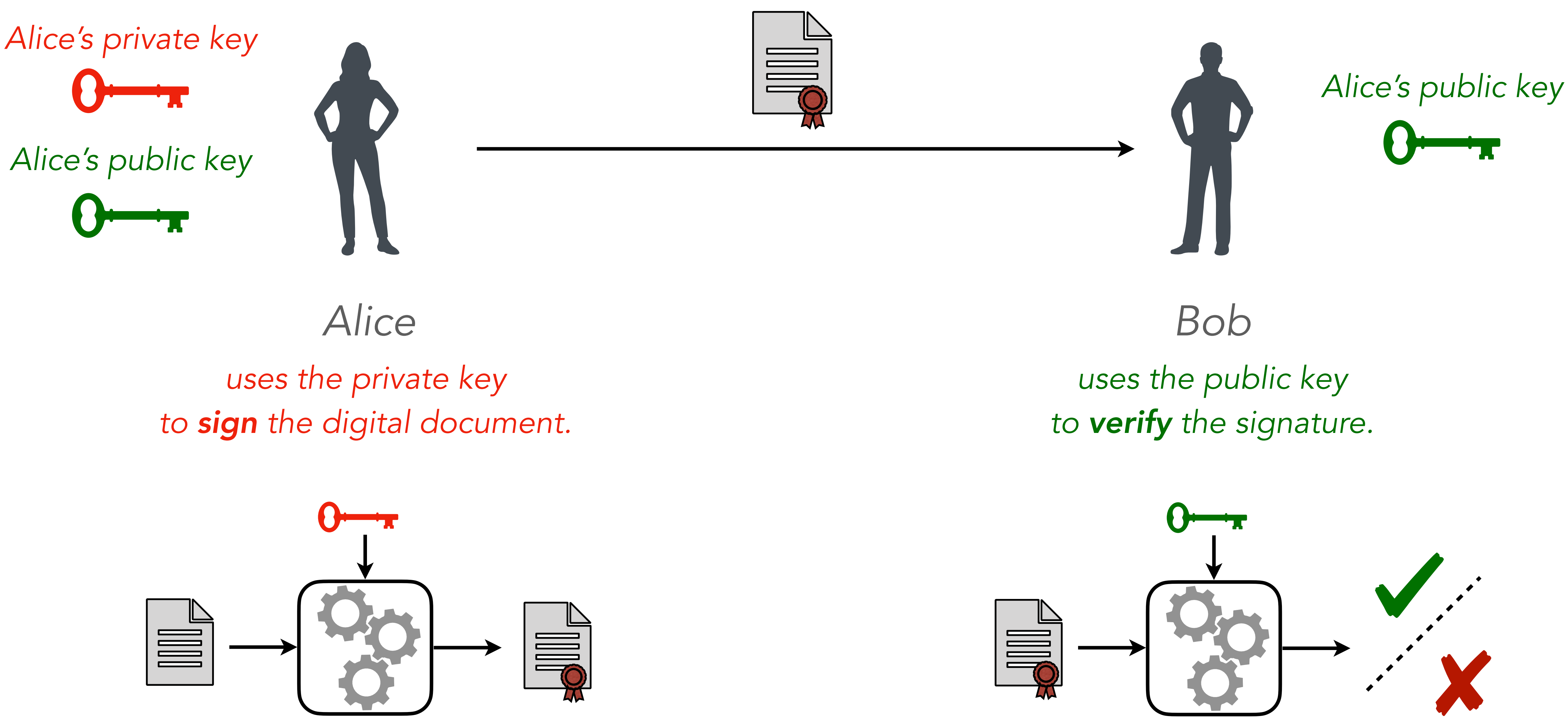
Digital signatures



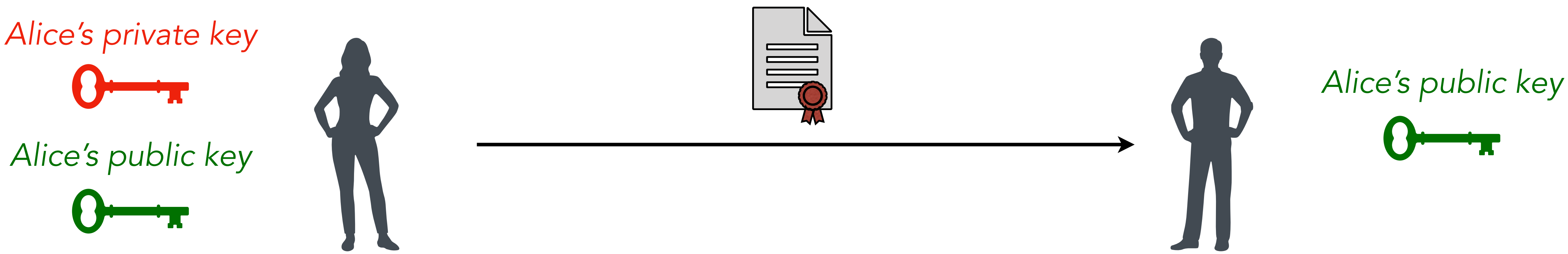
uses the private key
*to **sign** the digital document.*



Digital signatures



Digital signatures



Alice

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

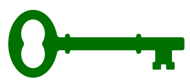
Bob

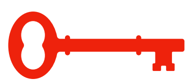
uses the public key
*to **verify** the signature.*

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

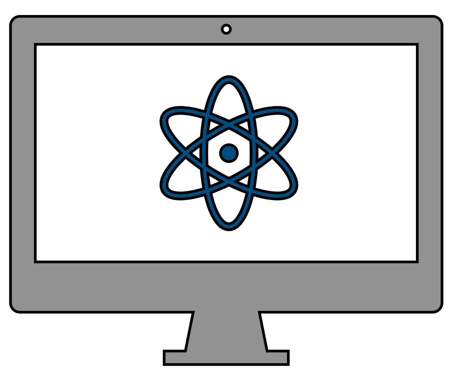
Digital signatures

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

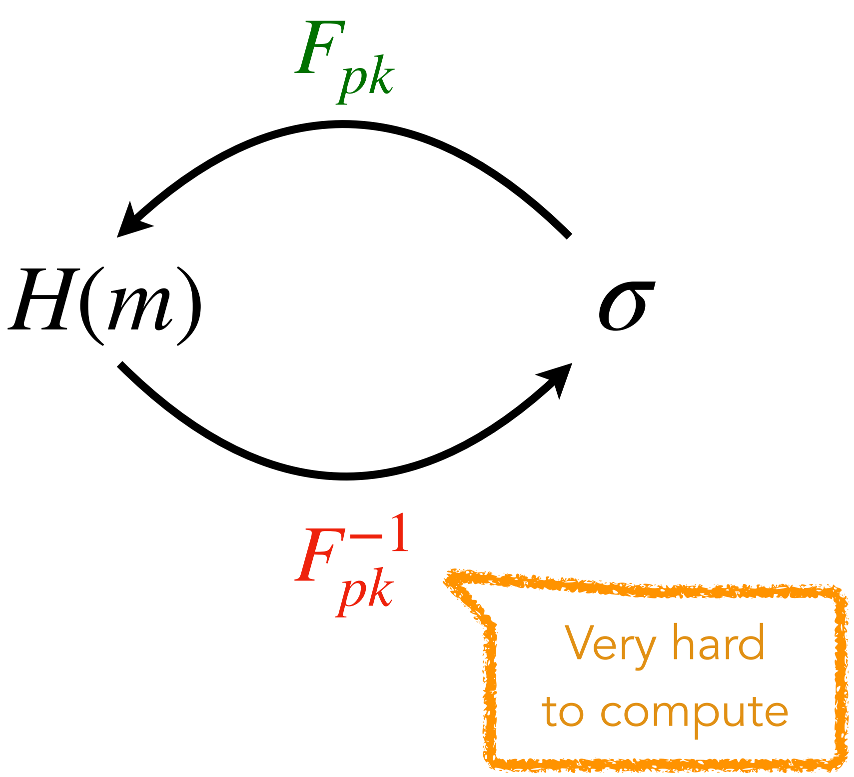
$$\begin{tikzcd} H(m) \arrow[rr, bend right, "F_{pk}^{-1}"'] & & \sigma \arrow[ll, bend right, "F_{pk}"'] \end{tikzcd}$$

Very hard
to compute

- Short signatures
- “Trapdoor” in the public key

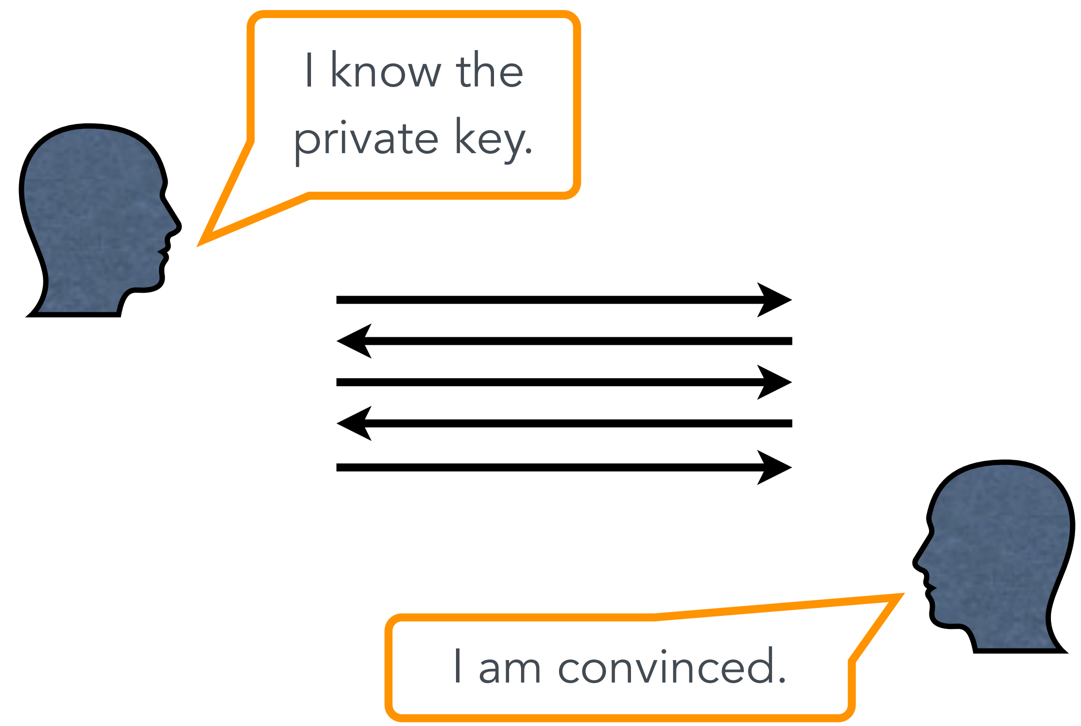
How to build signature schemes?

Hash & Sign



- Short signatures
- "Trapdoor" in the public key

From an identification scheme



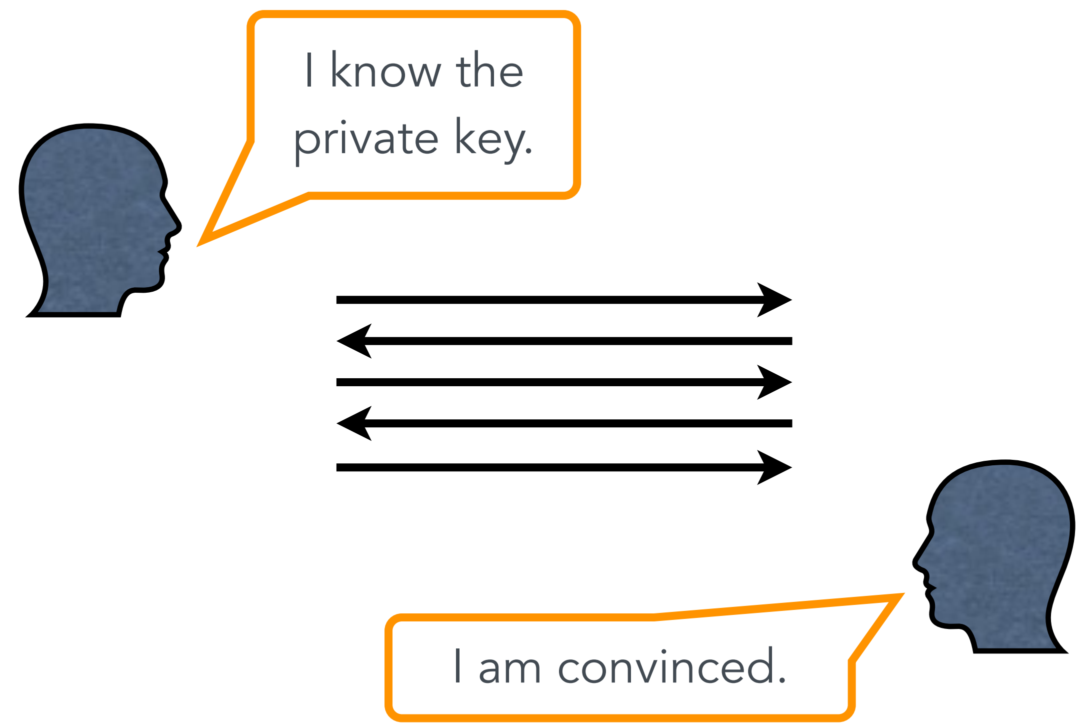
- Large(r) signatures
- Short public key

How to build signature schemes?

Hash & Sign

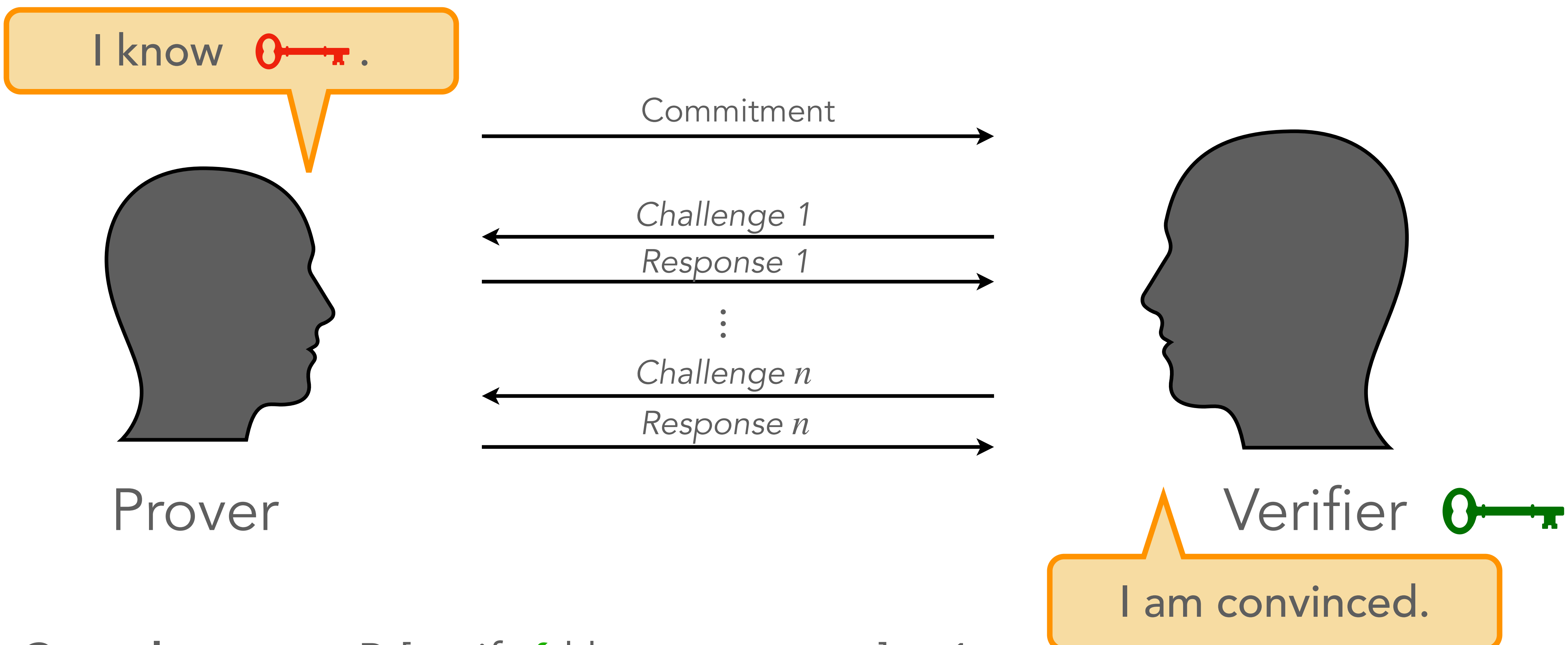
- Short signatures
- "Trapdoor" in the public key

From an identification scheme



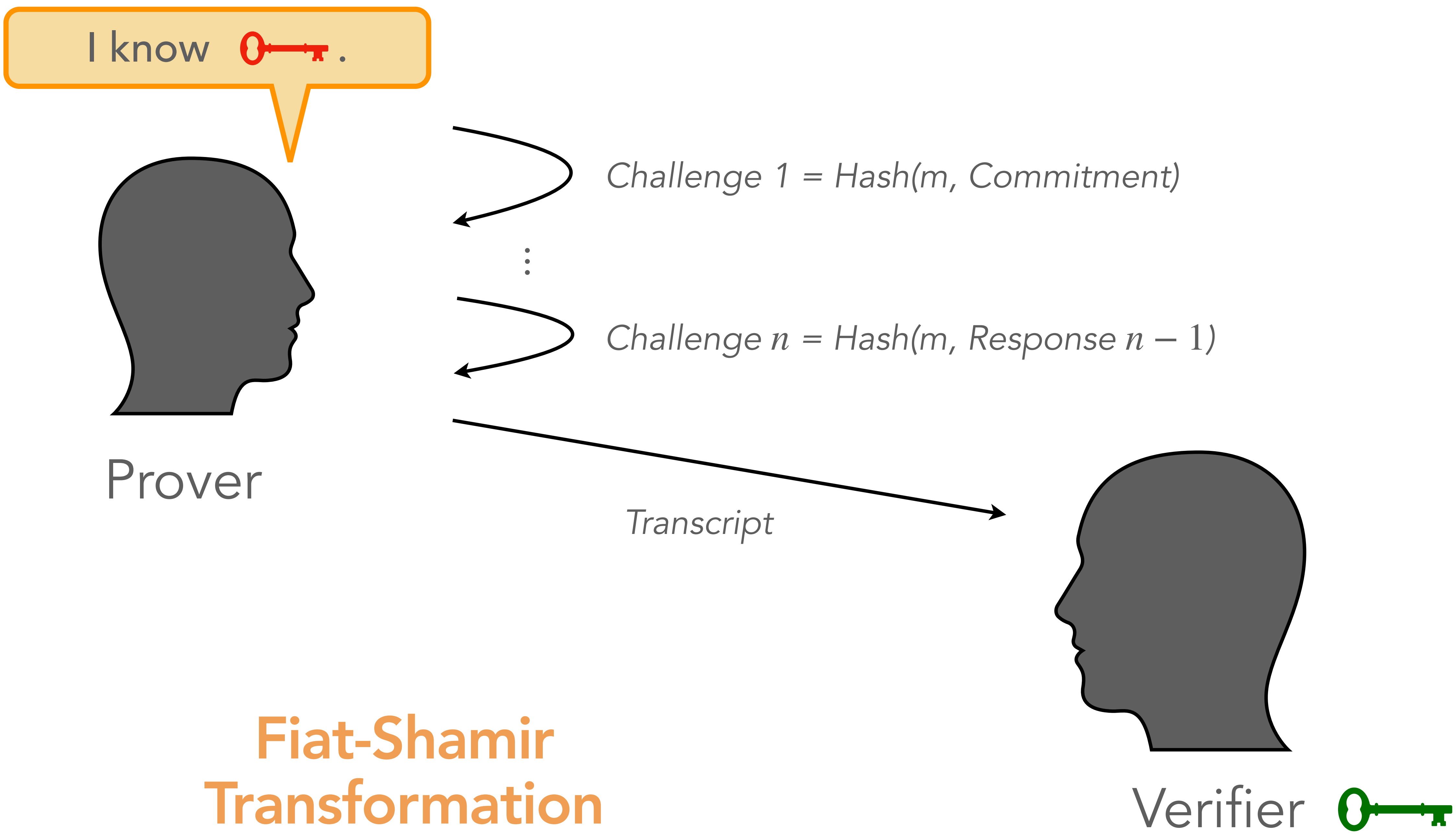
- Large(r) signatures
- Short public key

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



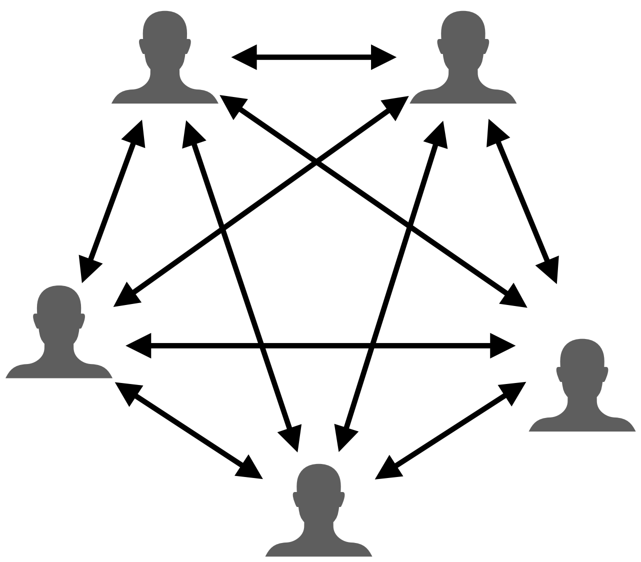
Fiat-Shamir Transformation

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



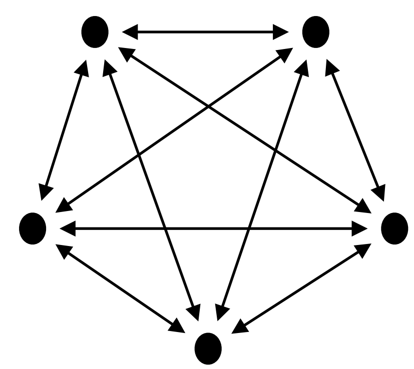
- **Generic**: can be apply to any cryptographic problem

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

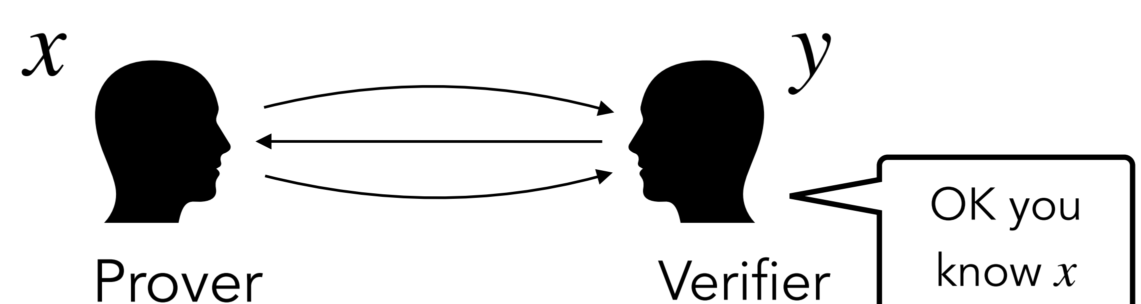


Input sharing $[\![x]\!]$

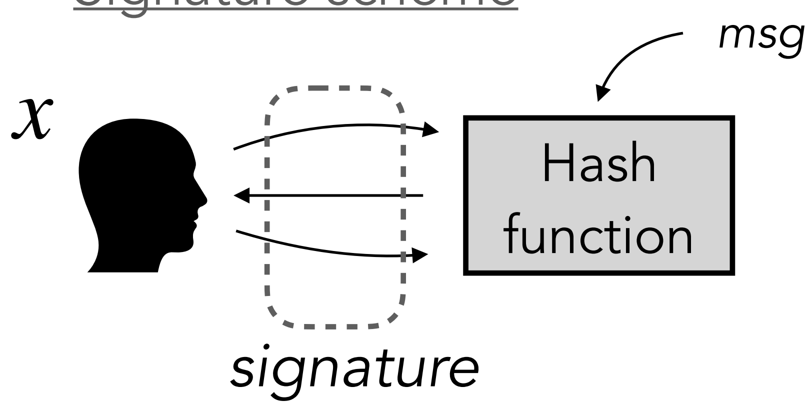
Joint evaluation of:

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

Zero-knowledge proof



Signature scheme

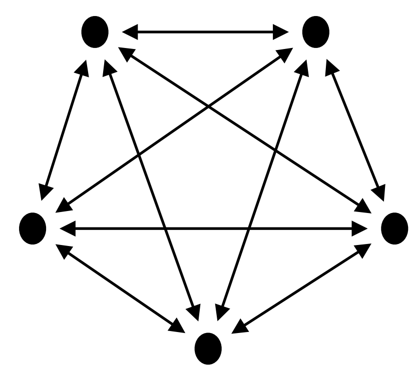


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

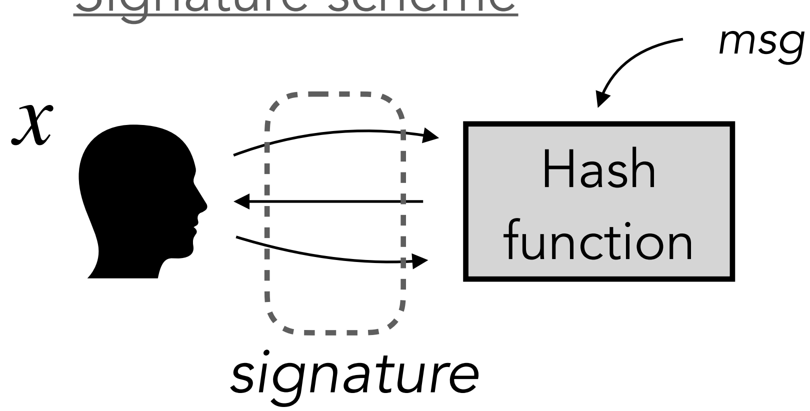


Input sharing $[\![x]\!]$

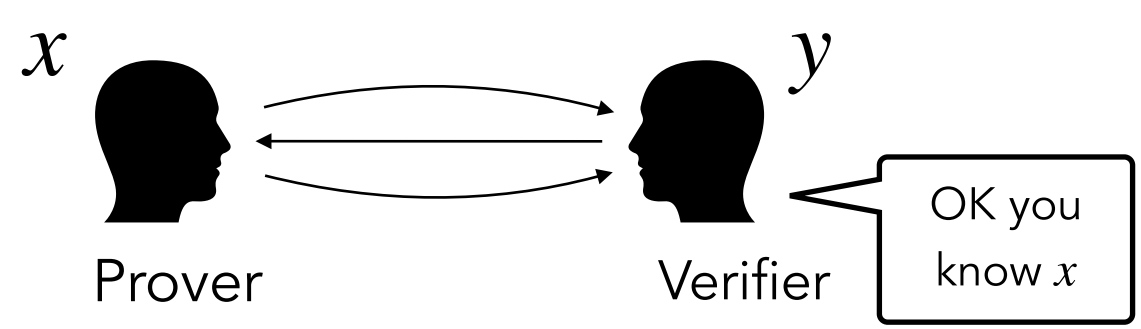
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof



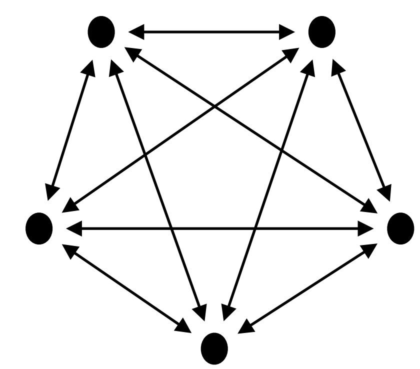
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ s.t. $x = [\![x]\!]_1 + \ldots + [\![x]\!]_N$

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

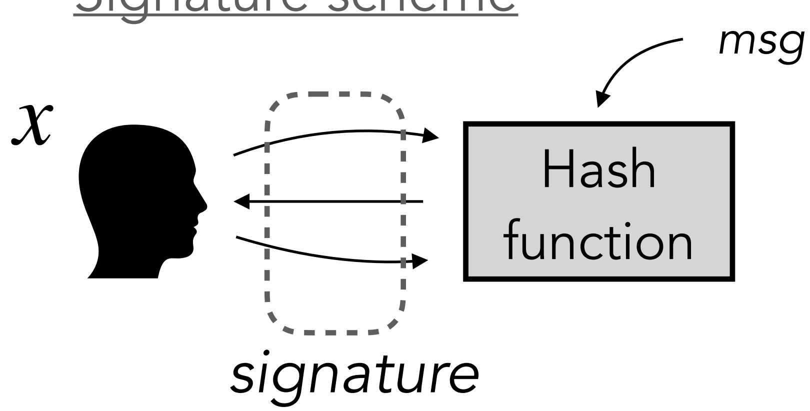


Input sharing $[\![x]\!]$

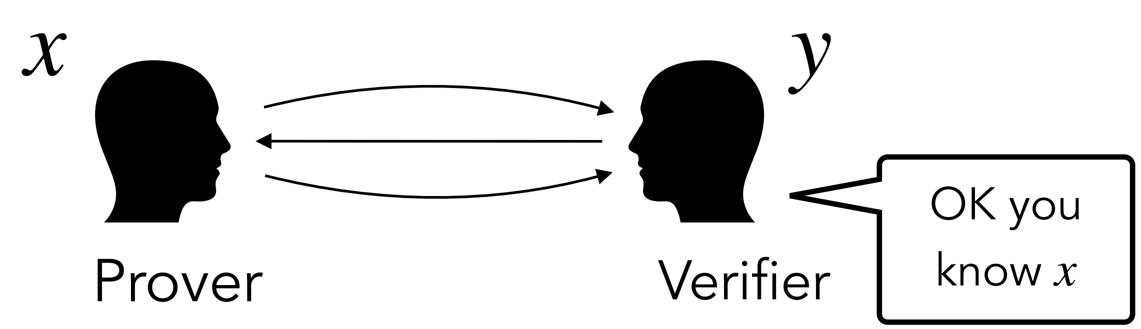
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

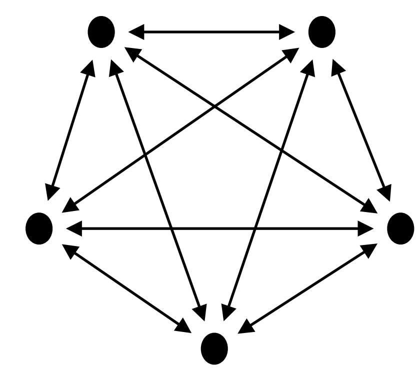


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

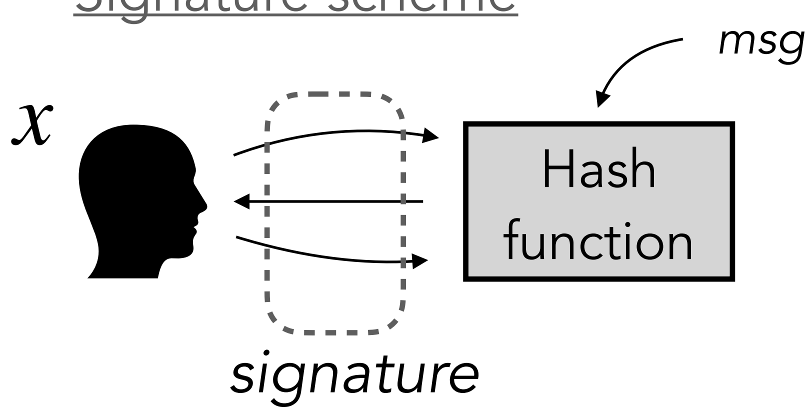


Input sharing $[\![x]\!]$

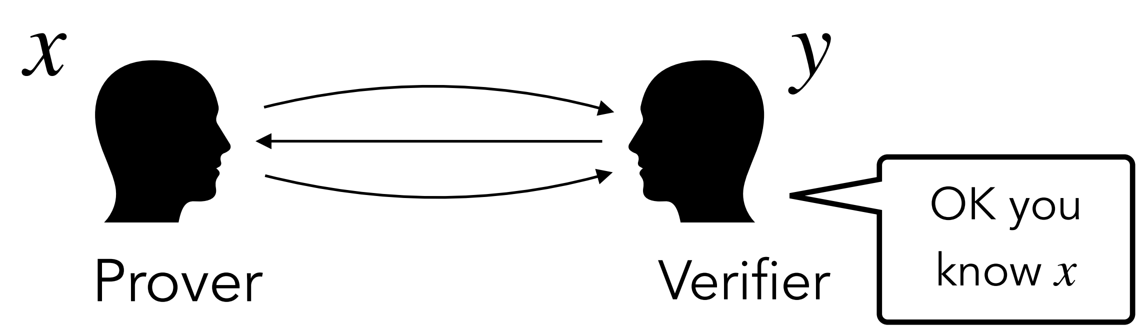
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

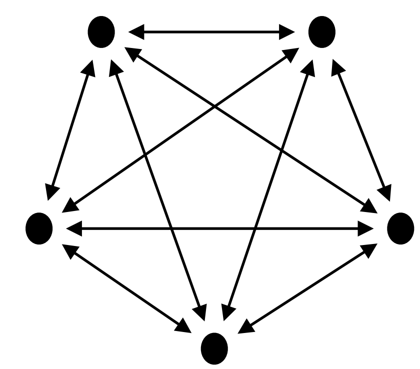


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

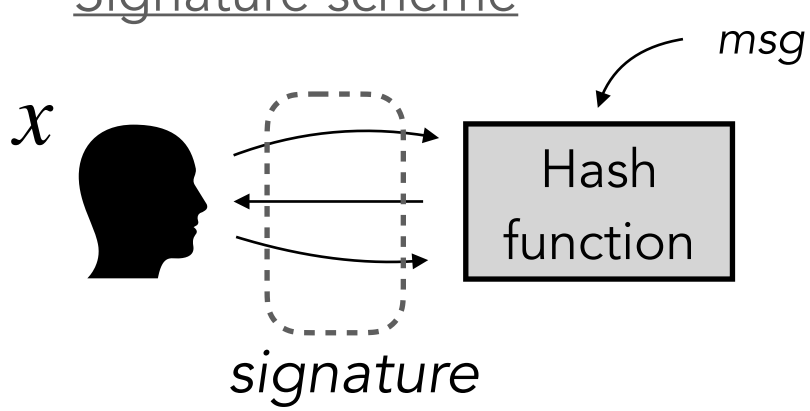


Input sharing $[\![x]\!]$

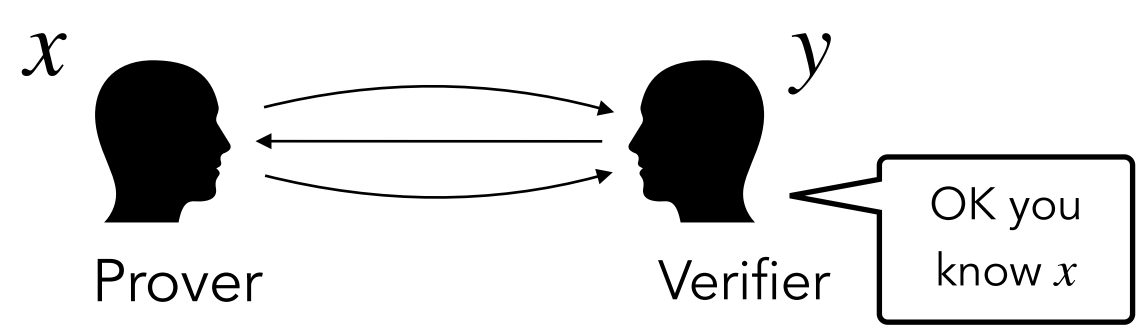
Joint evaluation of:

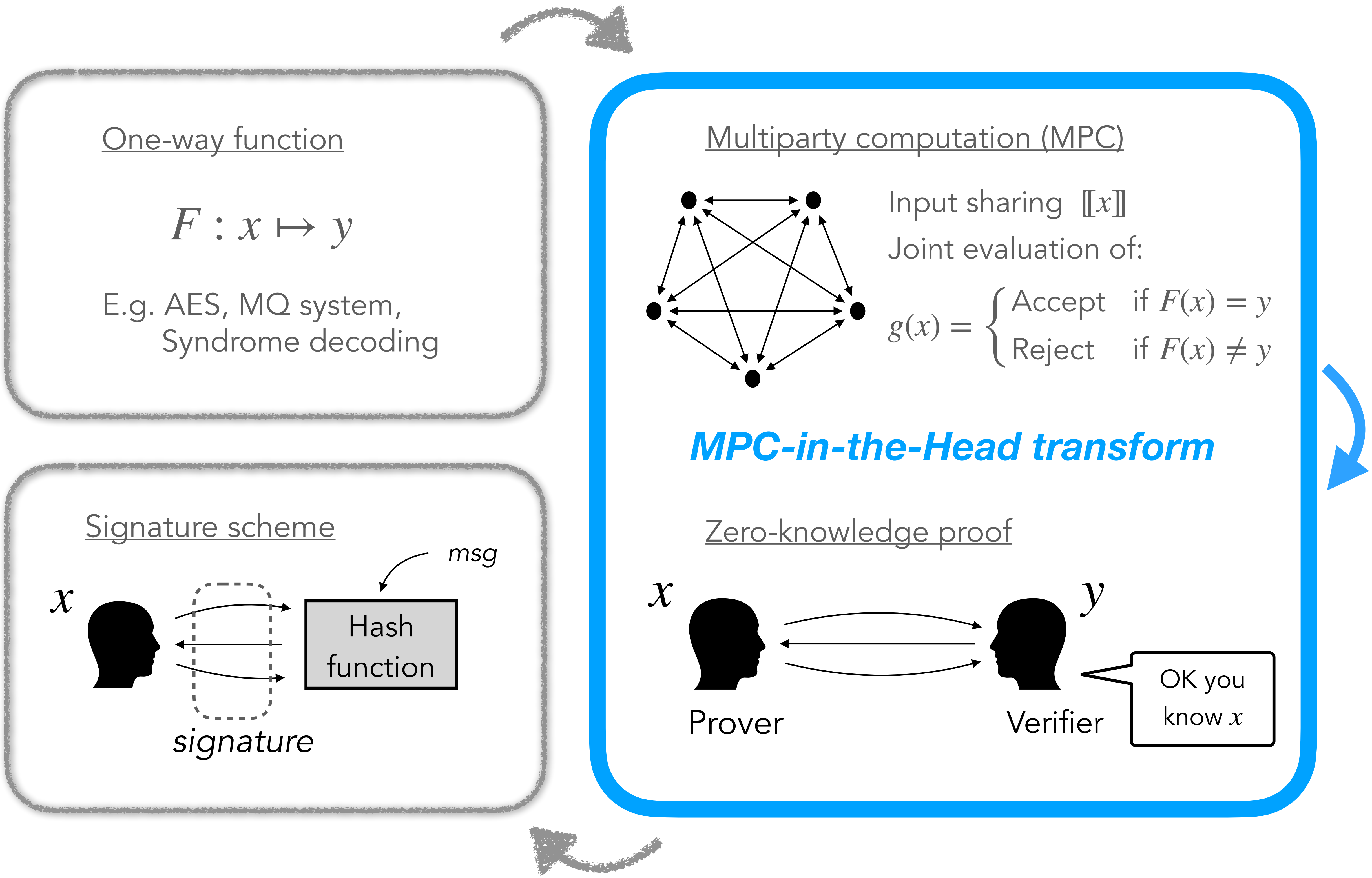
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Signature scheme



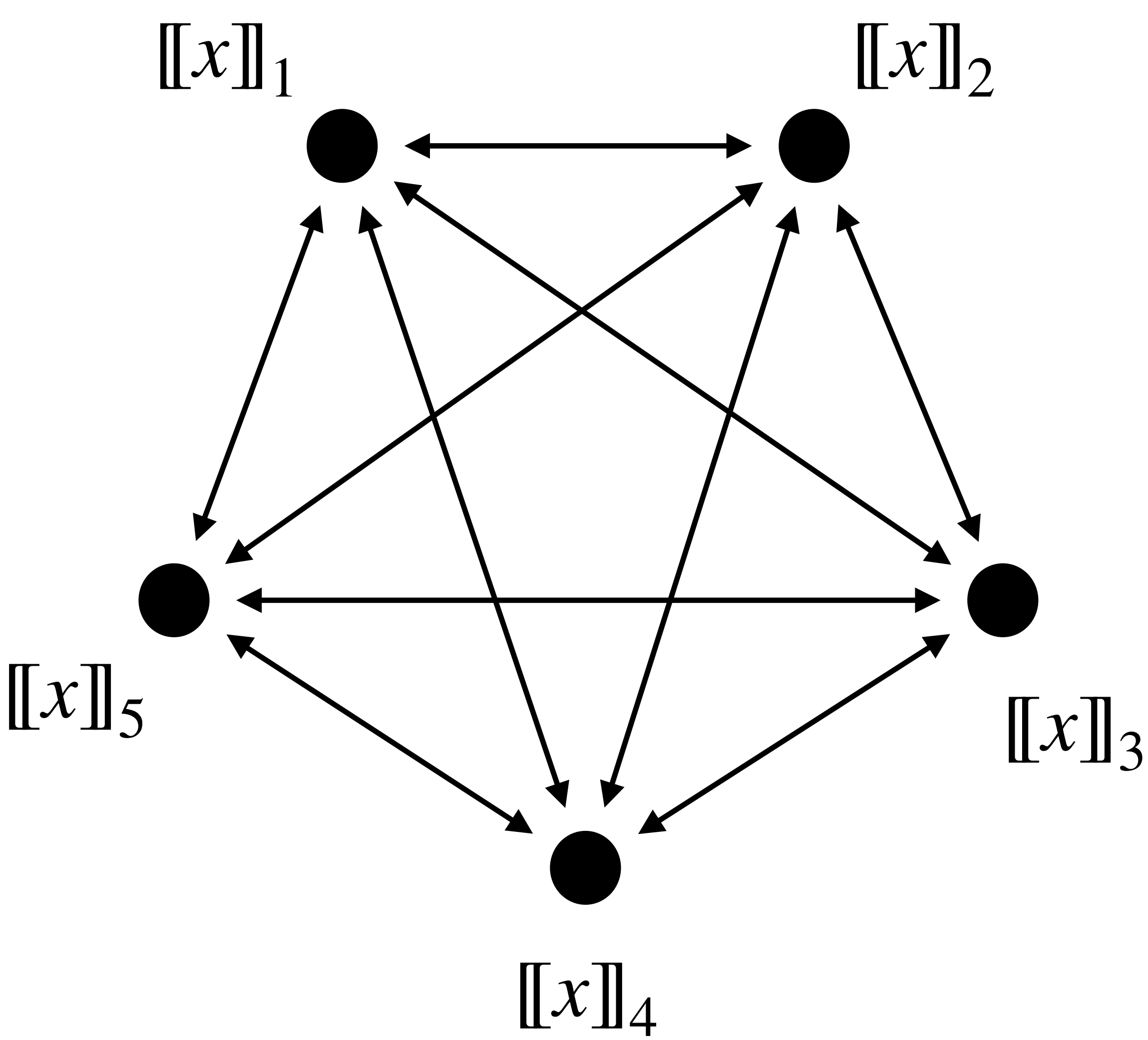
Zero-knowledge proof




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



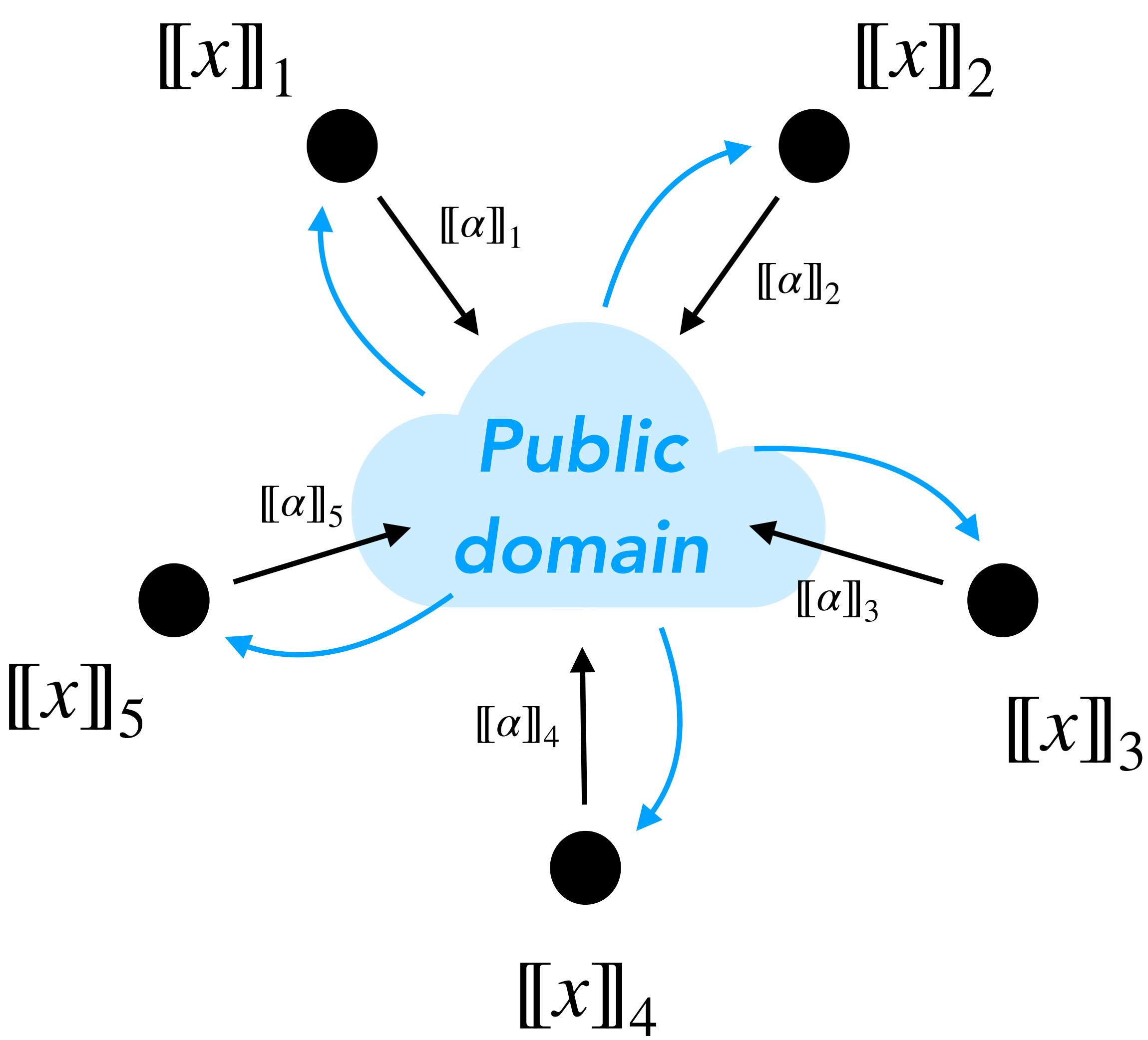
$$x = [\![x]\!]_1 + [\![x]\!]_2 + \ldots + [\![x]\!]_N$$

- **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

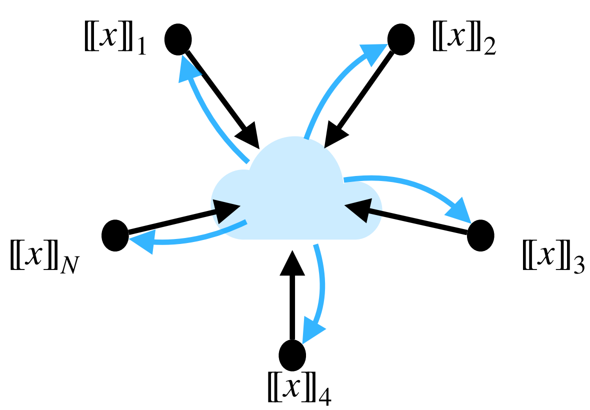
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\cdots$

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

send broadcast

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

Prover

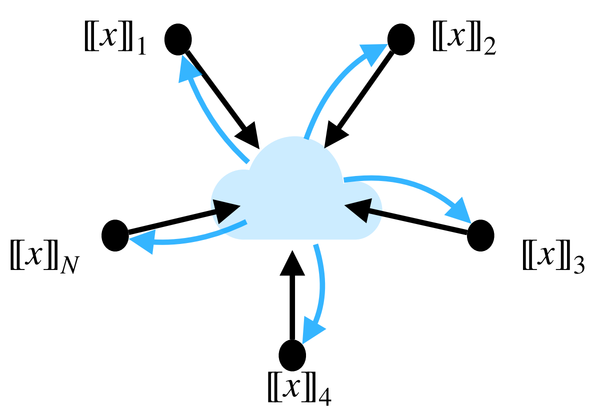
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

$\cdots$

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

send broadcast

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

i^*

③ Choose a random party

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

Prover

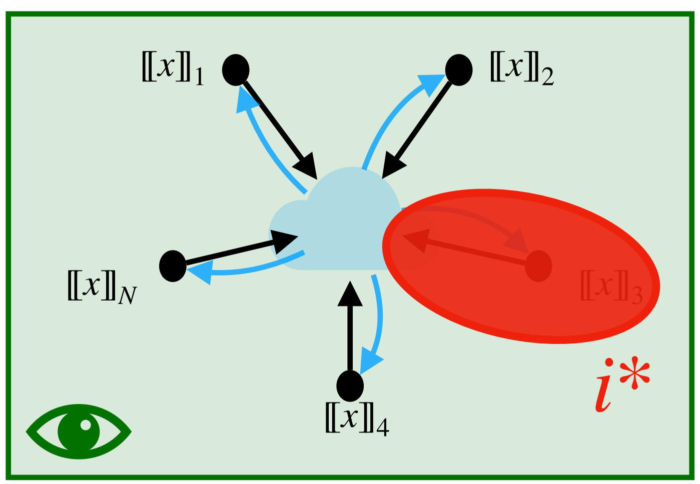
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

$\ldots$

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

$\longrightarrow$

send broadcast

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

$\longrightarrow$

i^*

$\longleftarrow$

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

$\longrightarrow$

③ Choose a random party

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

Prover

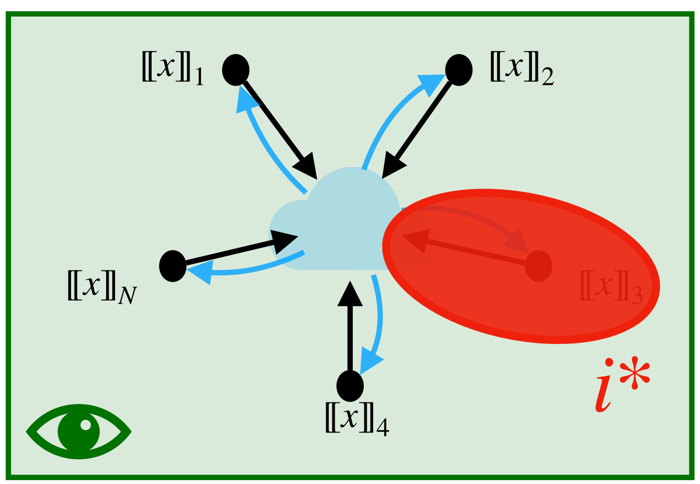
Verifier

MPCitH transform

① Generate and commit shares

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

② Run MPC in their head



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

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

…

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

send broadcast

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

← i^*

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

③ Choose a random party

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

⑤ Check $\forall i \neq i^*$

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

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

Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

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

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

$\dots$

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

$\longrightarrow$

Malicious Prover

Verifier

MPCitH transform

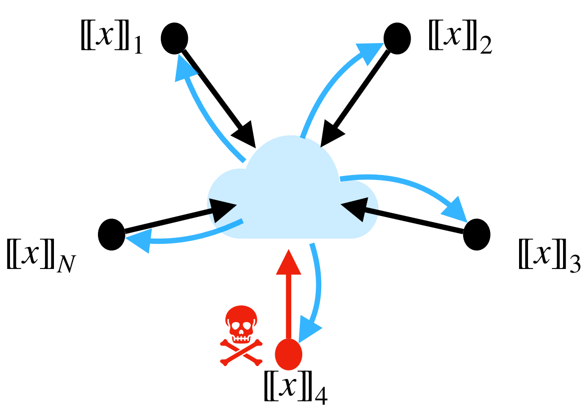
① Generate and commit shares

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

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

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

② Run MPC in their head



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

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

Malicious Prover

Verifier

MPCitH transform

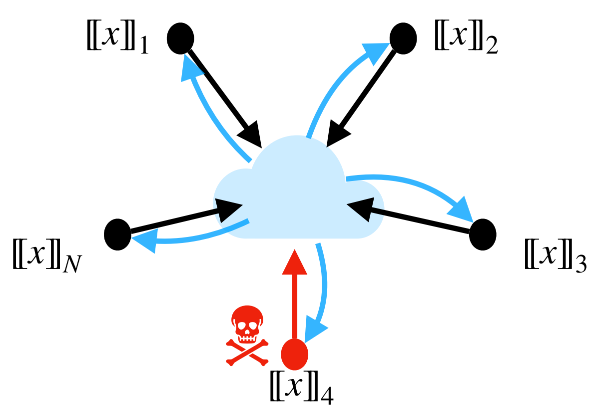
① Generate and commit shares

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

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

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

② Run MPC in their head



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

$\cdots$

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

send broadcast

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

③ Choose a random party

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

← i^*

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

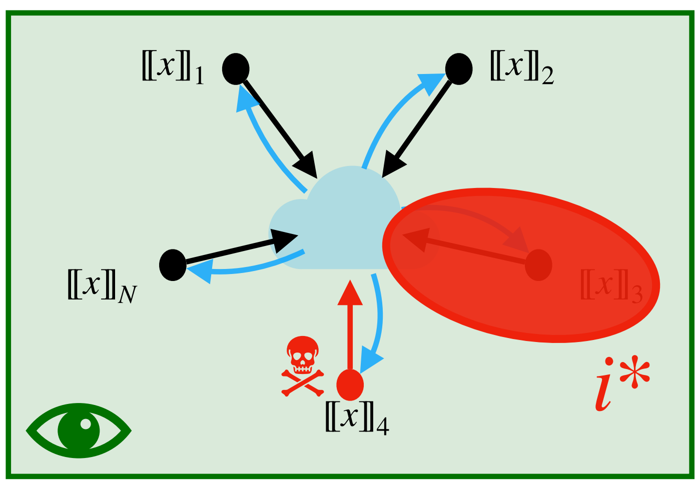
$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

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

$\cdots$

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

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

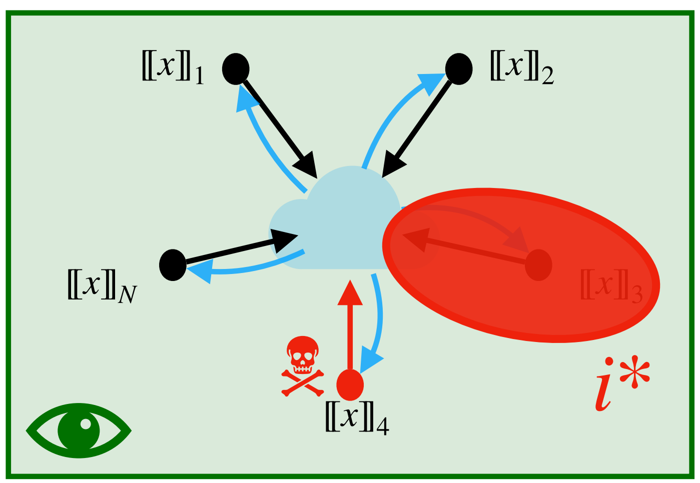
① Generate and commit shares

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

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

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

② Run MPC in their head



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

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

$\cdots$

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

send broadcast

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

i^* ←

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

③ Choose a random party

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

⑤ Check $\forall i \neq i^*$

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

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

Malicious Prover

Verifier

Cheating detected!

MPCitH transform

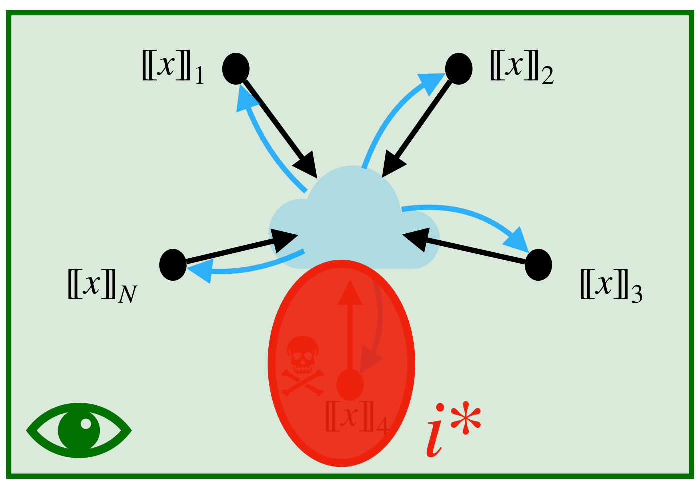
① Generate and commit shares

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

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

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

② Run MPC in their head



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

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

$\cdots$

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

send broadcast

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

i^*

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

③ Choose a random party

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

⑤ Check $\forall i \neq i^*$

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

Check $g(y, \alpha) =$ 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:**

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

MPCitH transform

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

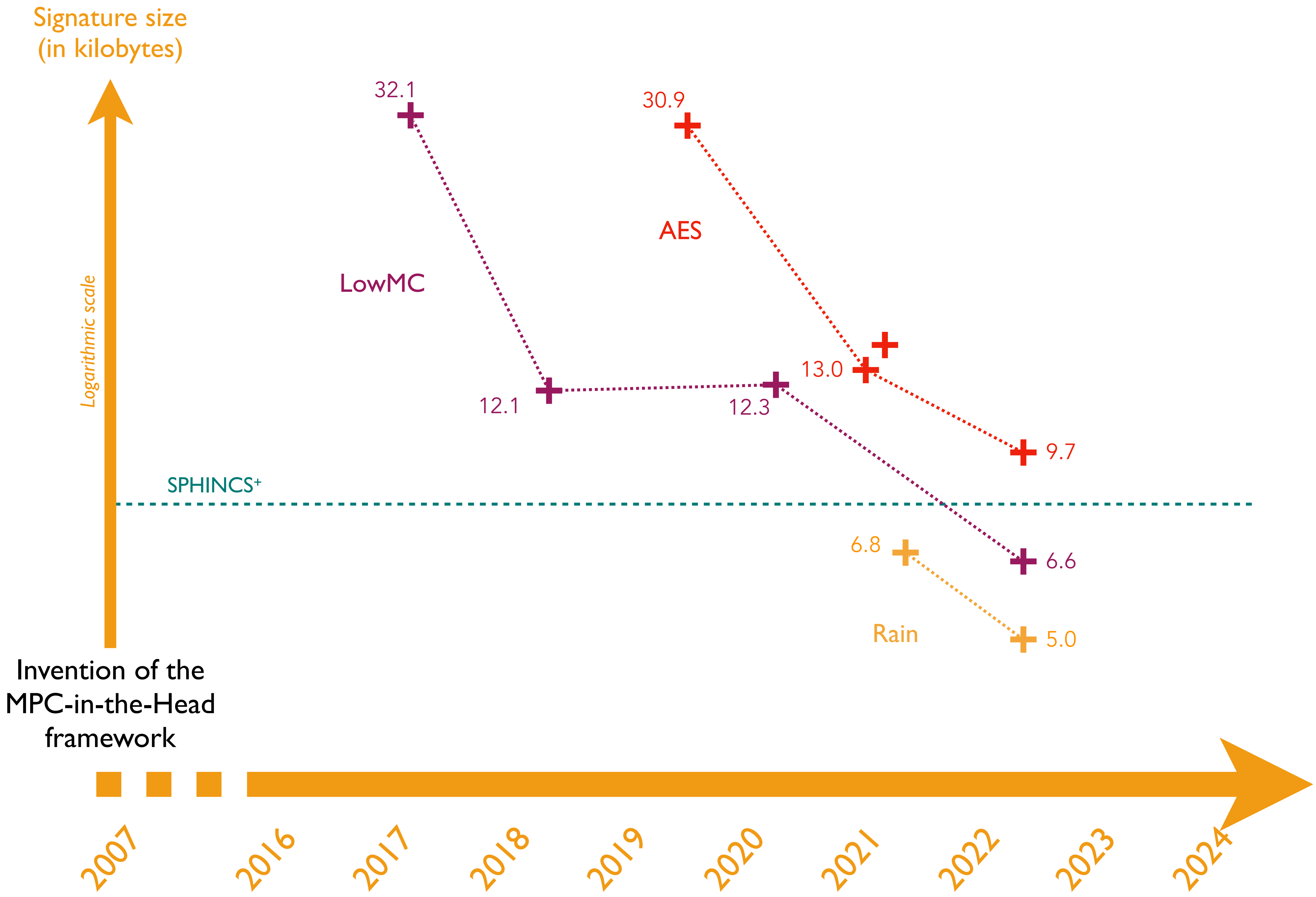
- **Soundness:**

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

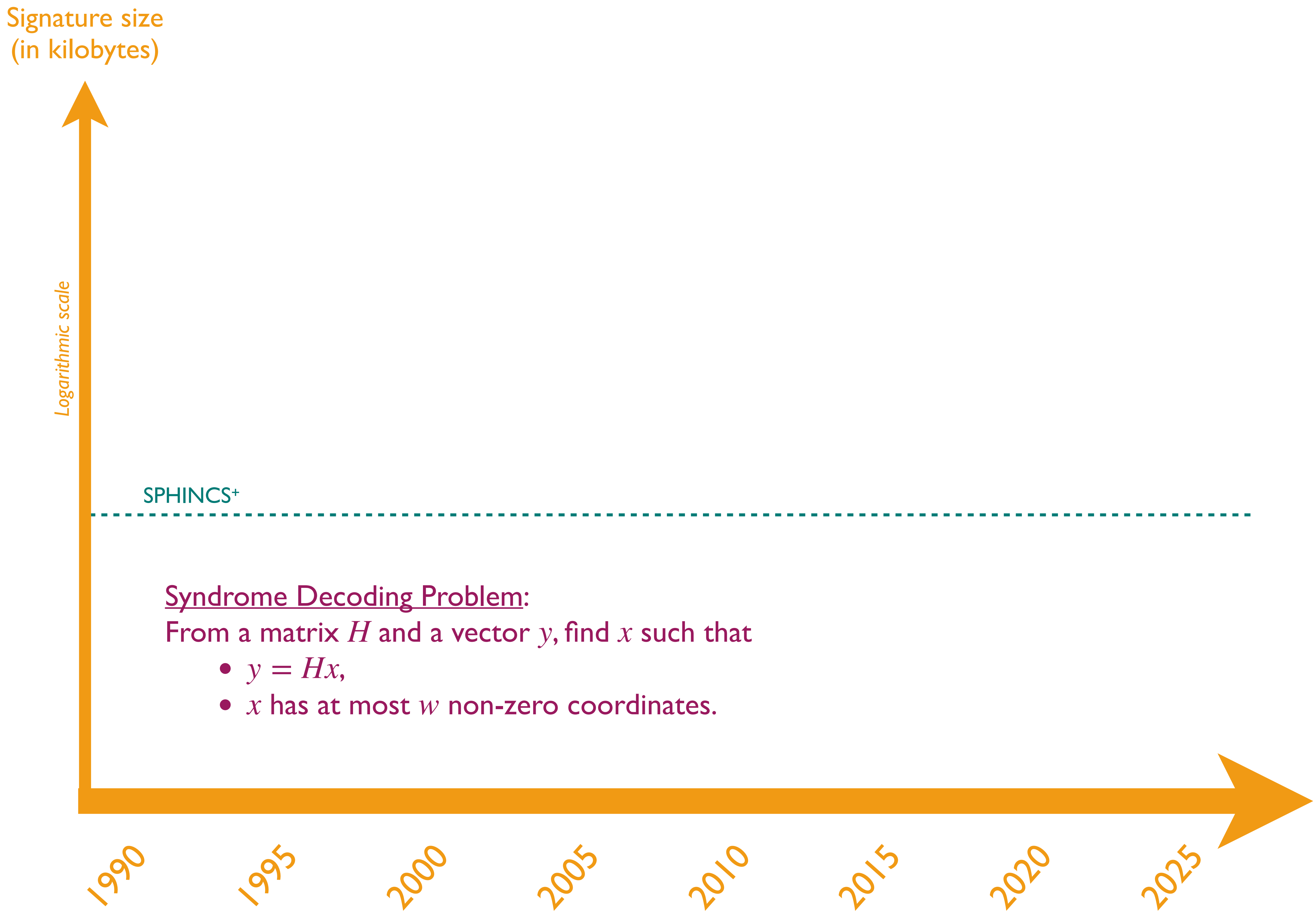
- **Parallel repetition**

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

From MPC-in-the-Head to signatures

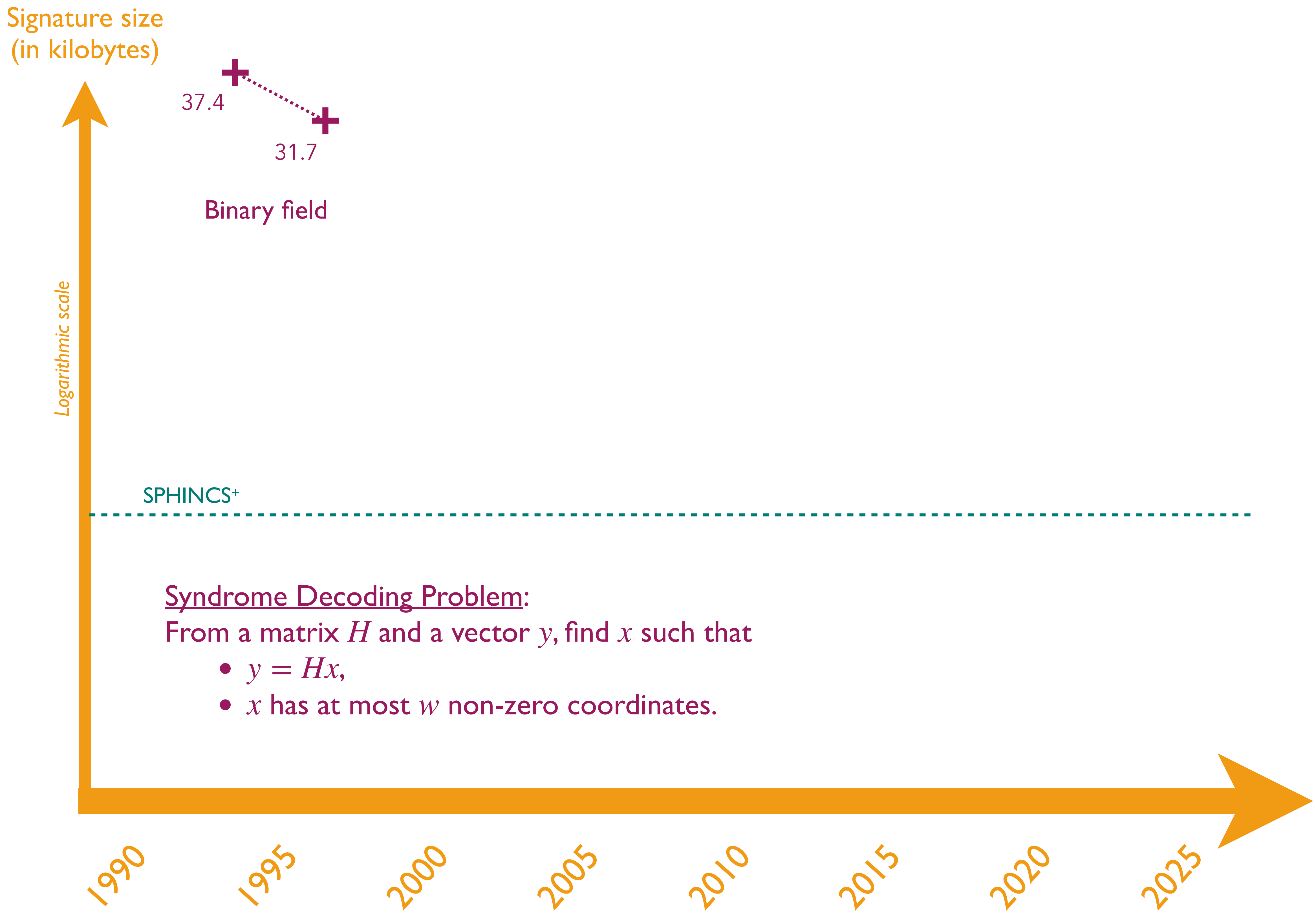


Signature size
(in kilobytes)
Logarithmic scale
32.1
30.9
AES
LowMC
13.0
12.1
12.3
9.7
SPHINCS+
6.8
6.6
Rain
5.0
Invention of the
MPC-in-the-Head
framework
2007
2016
2017
2018
2019
2020
2021
2022
2023
2024



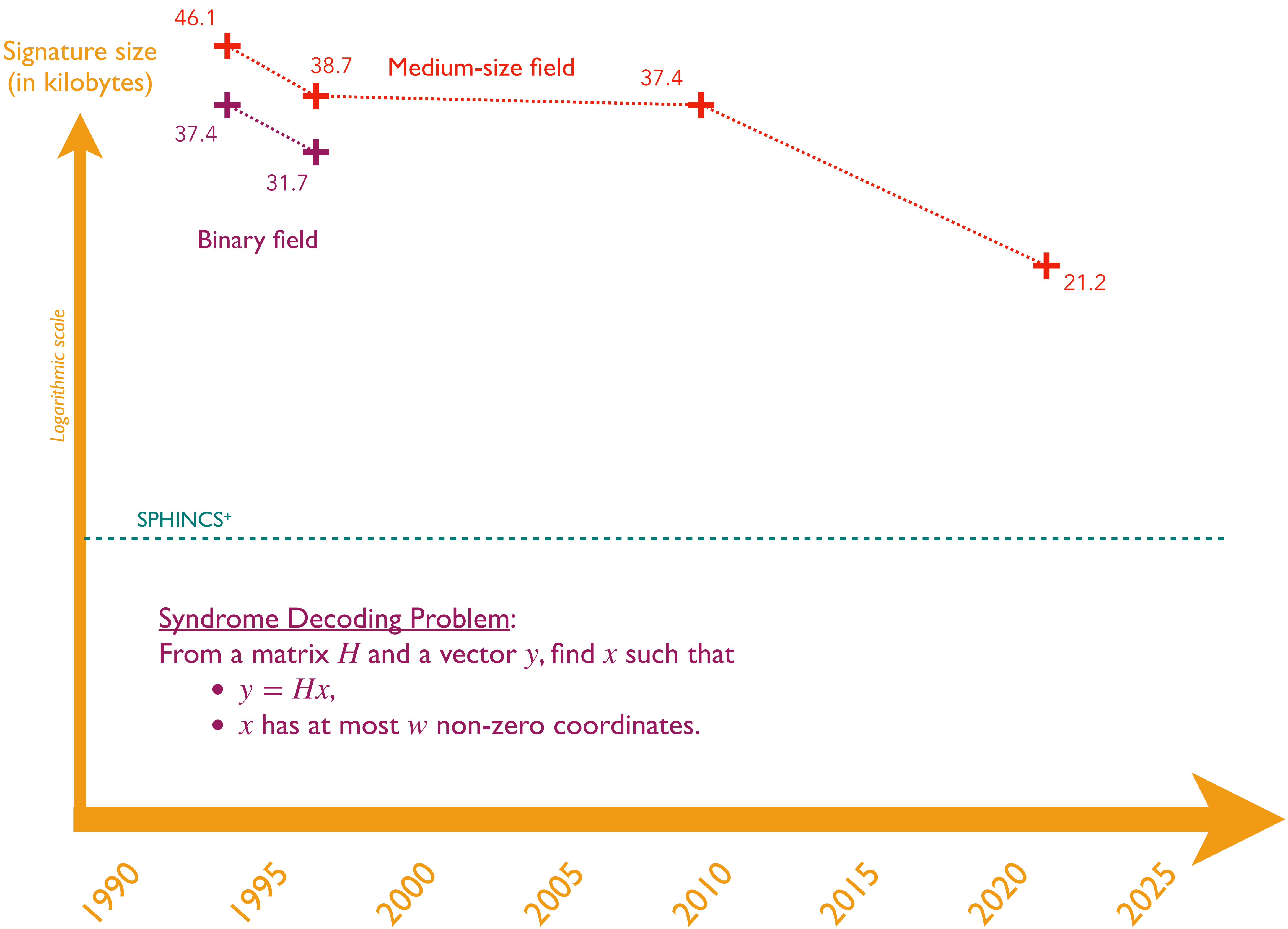
Syndrome Decoding Problem:
From a matrix H and a vector y, find x such that

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



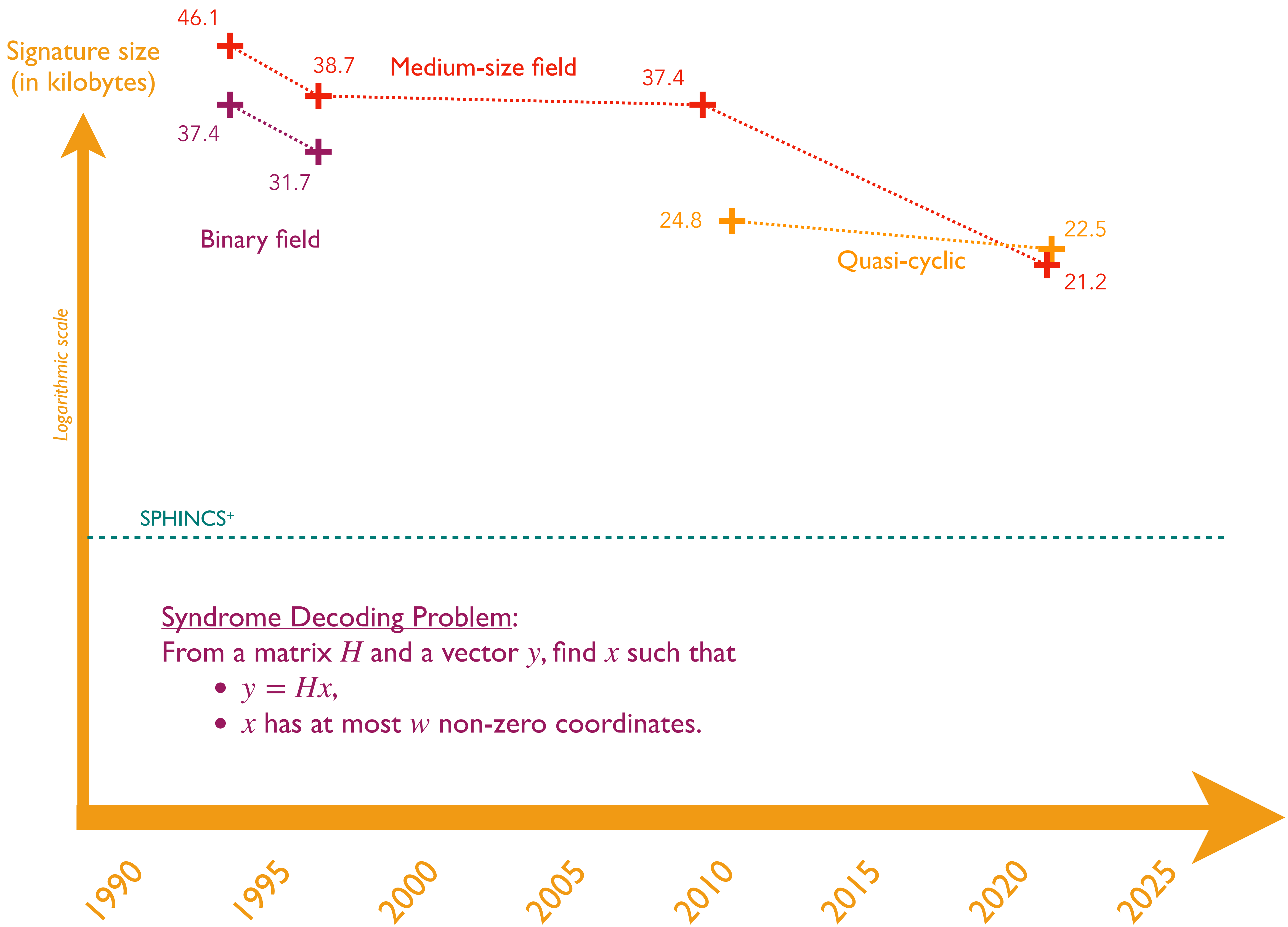
Syndrome Decoding Problem:
From a matrix H and a vector y, find x such that

- $y = Hx$,
- 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.



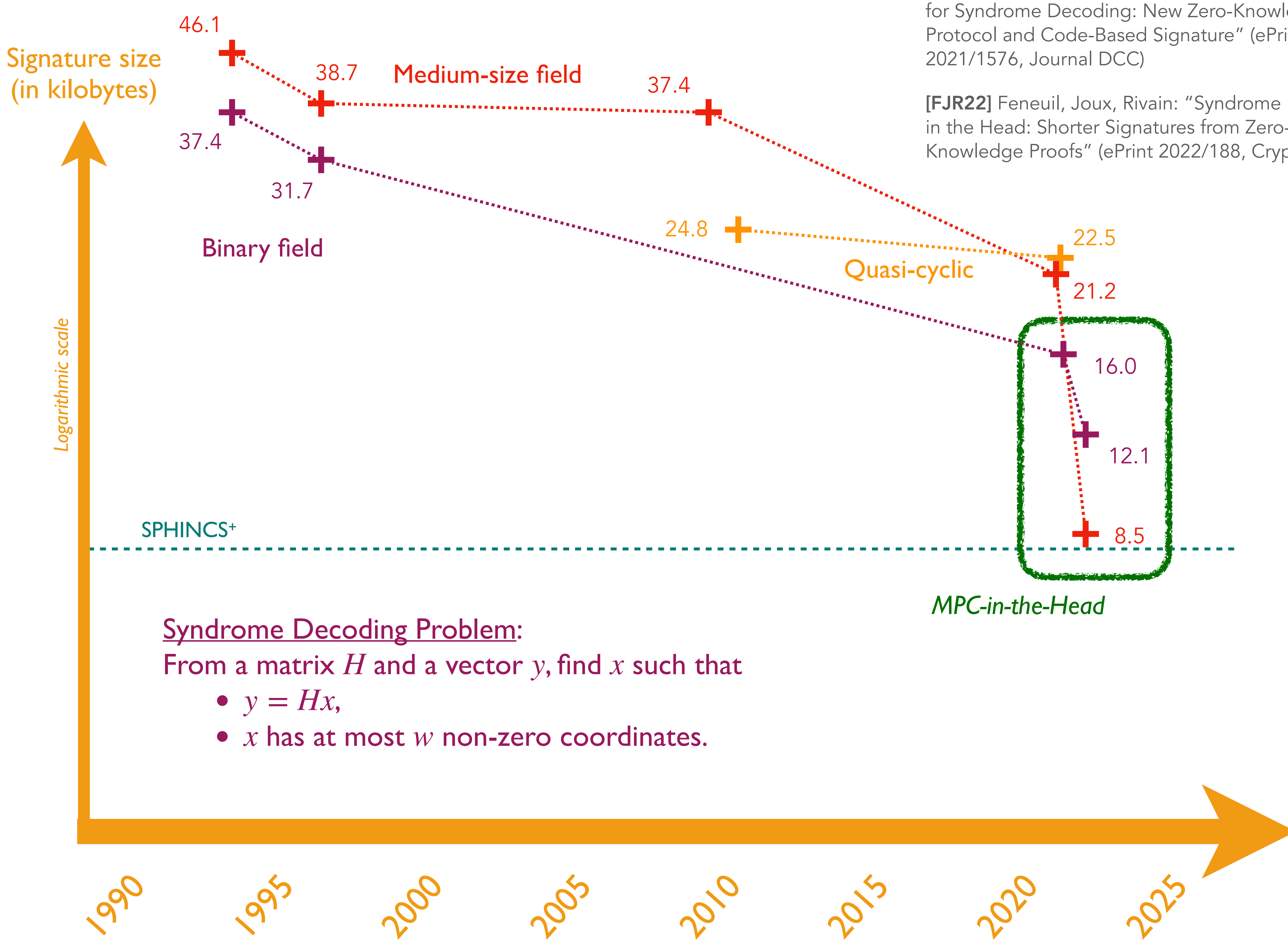
Syndrome Decoding Problem:

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

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

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

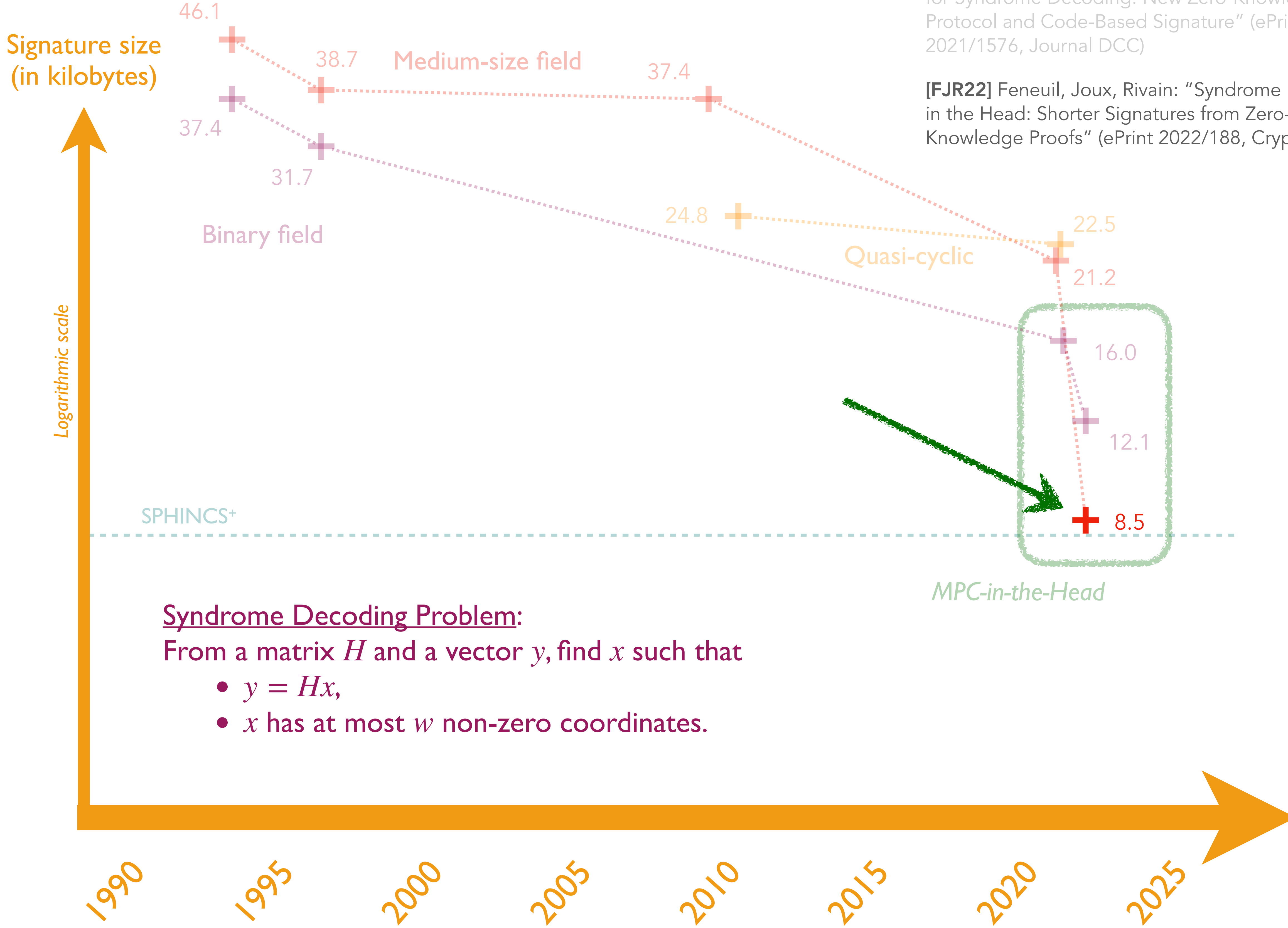
[FJR22] Feneuil, Joux, Rivain: "Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs" (ePrint 2022/188, Crypto 2022)



<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.



[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)

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.

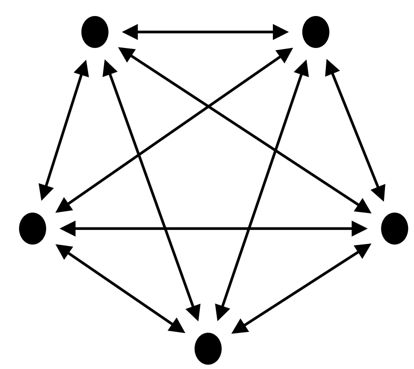
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ s.t. $x = [\![x]\!]_1 + \ldots + [\![x]\!]_N$

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

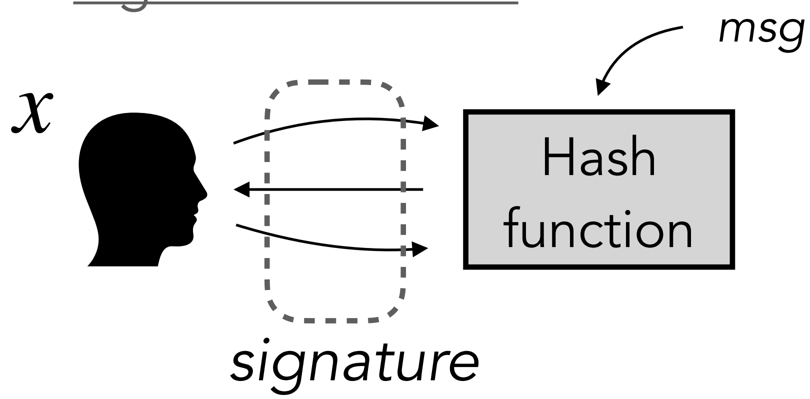


Input sharing $[\![x]\!]$

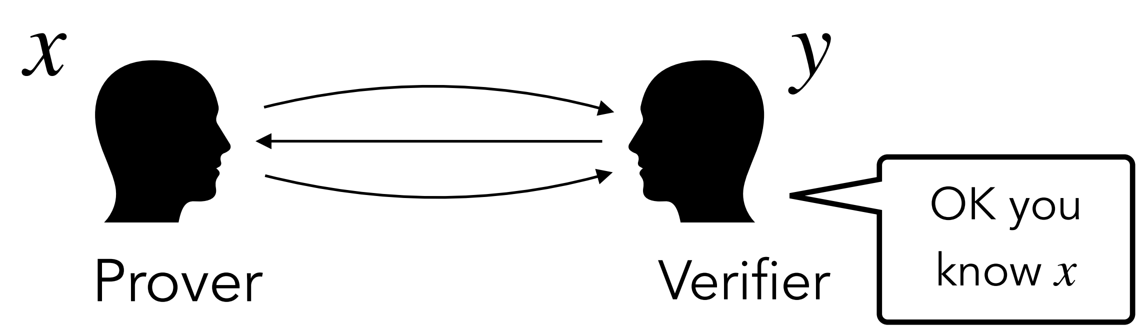
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof



A MPC protocol for the SD problem

The multiparty computation must check that the vector x satisfies

$$\underbrace{y = Hx}_{\text{linear, easy to check}} \qquad \text{and} \qquad \underbrace{\mathrm{w}_H(x) \leq w.}_{\text{non-linear, hard to check}}$$

A MPC protocol for the SD problem

The multiparty computation must check that the vector x satisfies

$$y = Hx$$

and

$$\exists Q, P \text{ two polynomials : } SQ = PF \text{ and } \deg Q = w$$

where

S is defined by interpolation such that $\forall i, S(\gamma_i) = x_i$,

$$F := \prod_{i=1}^{m} (X - \gamma_i).$$

Let us assume that there exists $Q, P \in \mathbb{F}[X]$ such that

$$S \cdot Q = P \cdot F \quad \text{and} \quad \deg Q = w$$

where

S is defined by interpolation such that $\forall i, S(\gamma_i) = x_i$

$F := \prod_{i=1}^{m} (X - \gamma_i)$.

Let us assume that there exists $Q, P \in \mathbb{F}[X]$ such that

$$S \cdot Q = P \cdot F \quad \text{and} \quad \deg Q = w$$

where

S is defined by interpolation such that $\forall i, S(\gamma_i) = x_i$

$$F := \prod_{i=1}^{m} (X - \gamma_i).$$

Then, one can deduce that

$$\forall i \leq m, (Q \cdot S)(\gamma_i) = P(\gamma_i) \cdot F(\gamma_i) = 0$$

Let us assume that there exists $Q, P \in \mathbb{F}[X]$ such that

$$S \cdot Q = P \cdot F \quad \text{and} \quad \deg Q = w$$

where

S is defined by interpolation such that $\forall i, S(\gamma_i) = x_i$

$$F := \prod_{i=1}^{m} (X - \gamma_i).$$

Then, one can deduce that

$$\forall i \leq m, (Q \cdot S)(\gamma_i) = P(\gamma_i) \cdot F(\gamma_i) = 0$$
$$\Rightarrow \forall i \leq m,\ Q(\gamma_i) = 0 \quad \text{or} \quad S(\gamma_i) = x_i = 0$$

Let us assume that there exists $Q, P \in \mathbb{F}[X]$ such that

$$S \cdot Q = P \cdot F \quad \text{and} \quad \deg Q = w$$

where

S is defined by interpolation such that $\forall i, S(\gamma_i) = x_i$

$$F := \prod_{i=1}^{m} (X - \gamma_i).$$

Then, one can deduce that

$$\forall i \leq m, (Q \cdot S)(\gamma_i) = P(\gamma_i) \cdot F(\gamma_i) = 0$$

$$\Rightarrow \forall i \leq m, \ Q(\gamma_i) = 0 \quad \text{or} \quad S(\gamma_i) = x_i = 0$$

i.e.,

$$\mathrm{wt}_H(x) = \#\{i : x_i \neq 0\} \leq w$$

Such polynomial Q can be built as

$$Q := Q' \cdot \prod_{i : x_i \neq 0} (X - \gamma_i)$$

The non-zero positions of x
are encoding as roots.

And $P := \dfrac{S \cdot Q}{F}$ since F divides $S \cdot Q$.

$$(\forall i, S(\gamma_i) = x_i)$$

A MPC protocol for the SD problem

We want to build a *MPC protocol* which checks if some vector is a syndrome decoding solution.

Let us assume that $H = (H' \mid I)$. We split x as $\begin{pmatrix} x_A \\ x_B \end{pmatrix}$.

We have $y = Hx$, so

$$x_B = y - H'x_A.$$

A MPC protocol for the SD problem

We want to build a *MPC protocol* which checks if some vector is a syndrome decoding solution.

Let us assume that $H = (H' \mid I)$. We split x as $\begin{pmatrix} x_A \\ x_B \end{pmatrix}$.

We have $y = Hx$, so

$$x_B = y - H'x_A.$$

Inputs of the MPC protocol: x_A, Q, P

Aim of the MPC protocol:

Check that x_A corresponds to a syndrome decoding solution.

A MPC protocol for the SD problem

Inputs of the MPC protocol: x_A, Q, P

1. Build $x_B := y - H' x_A$ and deduce $x := \begin{pmatrix} x_A \\ x_B \end{pmatrix}$.

   We have

$$y = Hx.$$

A MPC protocol for the SD problem

Inputs of the MPC protocol: x_A, Q, P

1. Build $x_B := y - H'x_A$ and deduce $x := \begin{pmatrix} x_A \\ x_B \end{pmatrix}$.

2. Build the polynomial S by interpolation such that

$$\forall i,\ S(\gamma_i) = x_i.$$

Interpolation Formula:

$$S(X) = \sum_i x_i \cdot \prod_{\ell \neq i} \frac{X - \gamma_\ell}{\gamma_i - \gamma_\ell}.$$

A MPC protocol for the SD problem

Inputs of the MPC protocol: x_A, Q, P

1. Build $x_B := y - H'x_A$ and deduce $x := \begin{pmatrix} x_A \\ x_B \end{pmatrix}$.

2. Build the polynomial S by interpolation such that

$$\forall i,\ S(\gamma_i) = x_i.$$

3. Check that $S \cdot Q = P \cdot F$.

A MPC protocol for the SD problem

Inputs of the MPC protocol: x_A, Q, P

1. Build $x_B := y - H'x_A$ and deduce $x := \begin{pmatrix} x_A \\ x_B \end{pmatrix}$.
2. Build the polynomial S by interpolation such that
$$\forall i,\ S(\gamma_i) = x_i.$$
3. Get a random point r from a field extension $\mathbb{F}_{points}$.
4. Compute $S(r)$, $Q(r)$ and $P(r)$.
5. Using [BN20], check that $S(r) \cdot Q(r) = P(r) \cdot F(r)$.

[BN20] Carsten Baum and Ariel Nof. *Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography.* PKC 2020.

A MPC protocol for the SD problem

Even if x_A does not describe a SD solution, implying that $S \cdot Q \neq P \cdot F$, the MPC protocol can output ***Accept*** if

Case 1:

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

which occurs with probability (Schwartz-Zippel Lemma)

$$\Pr_{r \leftarrow \mathbb{F}_{points}} [S(r) \cdot Q(r) = P(r) \cdot F(r)] \leq \frac{m + w - 1}{|\mathbb{F}_{points}|}$$

A MPC protocol for the SD problem

Even if x_A does not describe a SD solution, implying that $S \cdot Q \neq P \cdot F$, the MPC protocol can output ***Accept*** if

Case 1:

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

which occurs with probability (Schwartz-Zippel Lemma)

$$\Pr_{r \leftarrow \mathbb{F}_{points}} [S(r) \cdot Q(r) = P(r) \cdot F(r)] \leq \frac{m + w - 1}{|\mathbb{F}_{points}|}$$

Case 2: the [BN20] protocol failed, which occurs with probability

$$\frac{1}{|\mathbb{F}_{points}|}.$$

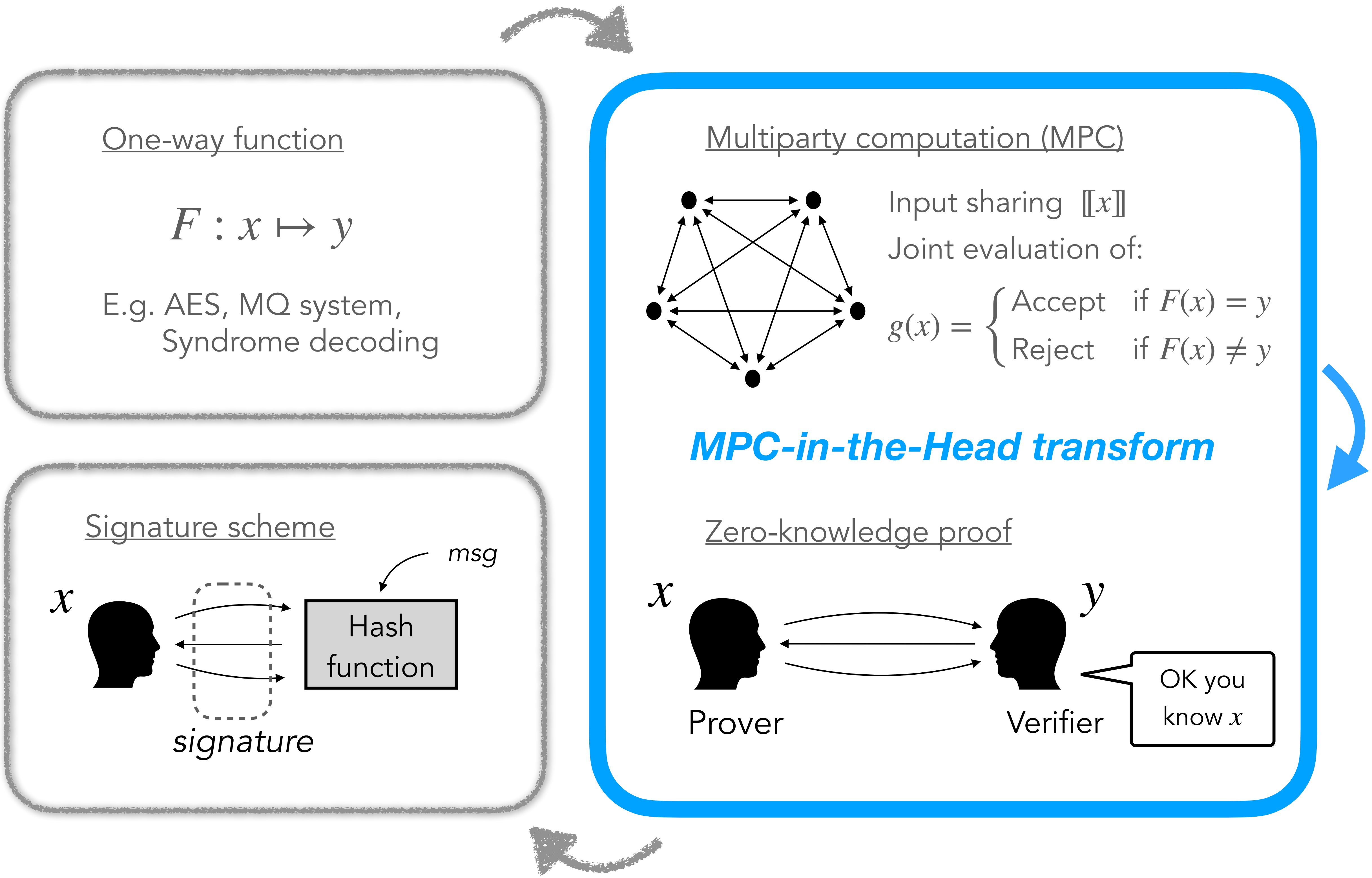
A MPC protocol for the SD problem

The MPC protocol checks that (x_A, Q, P) describes a solution of the SD instance (H, y).

	Protocol Output	
	`Accept`	`Reject`
A good witness	1	0
Not a good witness	p	$1-p$

where

$$p = \underbrace{\frac{m+w-1}{|\mathbb{F}_{points}|}}_{\text{False positive from Schwartz-Zippel}} + \left(1 - \frac{m+w-1}{|\mathbb{F}_{points}|}\right) \cdot \underbrace{\frac{1}{|\mathbb{F}_{points}|}}_{\text{False positive from [BN20]}}$$


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
Signature scheme
x
msg
Hash function
signature
Zero-knowledge proof
x
y
Prover
Verifier
OK you know x

Resulting Zero-Knowledge Proof for SD

Soundness error:

$$\epsilon := \frac{1}{N} + \left(1 - \frac{1}{N}\right) \cdot p$$

To achieve *negligible soundness error*, we repeat the zero-knowledge proof τ times such that $\epsilon^\tau < 2^{-\lambda}$.

Resulting Zero-Knowledge Proof for SD

Soundness error:

$$\epsilon := \frac{1}{N} + \left(1 - \frac{1}{N}\right) \cdot p$$

To achieve *negligible soundness error*, we repeat the zero-knowledge proof τ times such that $\epsilon^\tau < 2^{-\lambda}$.

Signature scheme: to obtain the signature scheme, we just need to apply the

Fiat-Shamir transform.

Signature Scheme

Parameter Selection (128-bit security):

- Syndrome Decoding problem over $\mathbb{F}_{256}$
- The MPCitH parameters: $N = 256,\ \tau = 17$

Resulting size (short variant):

Using few optimisations (Seed trees, …)

$\approx 8{,}5$ kilobytes

Signature Scheme

Parameter Selection (128-bit security):

- Syndrome Decoding problem over $\mathbb{F}_{256}$
- The MPCitH parameters: $N = 256,\ \tau = 17$

Resulting size (short variant):

Using few optimisations (Seed trees, …)

$\approx 8{,}5$ kilobytes

Notes:

- We can apply to *binary syndrome decoding problem*, but it requires a field lifting for the polynomials S, Q, P, F.
- In the thesis, we propose also another approach, namely

  the *shared-permutation* framework,

  but it leads to larger sizes for the SD problem.

Exploring other assumptions

- **Subset Sum Problem:** ≥ 100 KB $\Rightarrow$ 19.1 KB
  - Problem over a very large modulo $q \approx 2^{256}$
  - *Key Idea*: Sharing over integers, signature with aborts

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

Exploring other assumptions

- **Subset Sum Problem:** ≥ 100 KB $\Rightarrow$ 19.1 KB
  - Problem over a very large modulo $q \approx 2^{256}$
  - *Key Idea*: Sharing over integers, signature with aborts

- **Multivariate Quadratic Problem:** $6.3 - 7.3$ KB
  - Problem with a cubic number of multiplications
  - Key Idea: Batching over all the quadratic equations

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

Exploring other assumptions

- **Subset Sum Problem:** ≥ 100 KB $\Rightarrow$ 19.1 KB
  - Problem over a very large modulo $q \approx 2^{256}$
  - *Key Idea*: Sharing over integers, signature with aborts
- **Multivariate Quadratic Problem:** $6.3 - 7.3$ KB
  - Problem with a cubic number of multiplications
  - Key Idea: Batching over all the quadratic equations
- **MinRank Problem / Rank Syndrome Decoding Problem:** ≈ 5.5 KB
  - Problems relying on the rank metric
  - Key Idea: Usage of q-polynomials

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

Exploring other assumptions

- **Subset Sum Problem:** ≥ 100 KB $\Rightarrow$ 19.1 KB
  - Problem over a very large modulo $q \approx 2^{256}$
  - *Key Idea*: Sharing over integers, signature with aborts

- **Multivariate Quadratic Problem:** $6.3 - 7.3$ KB
  - Problem with a cubic number of multiplications
  - Key Idea: Batching over all the quadratic equations

- **MinRank Problem / Rank Syndrome Decoding Problem:** ≈ 5.5 KB
  - Problems relying on the rank metric
  - Key Idea: Usage of q-polynomials

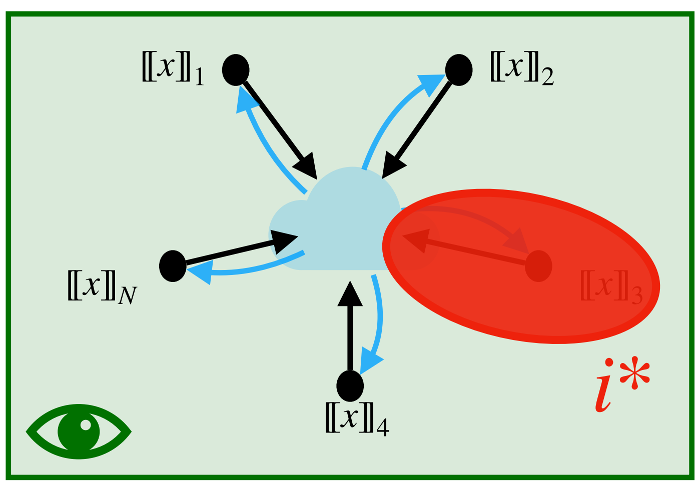
What about the computational cost ?

Computational Cost

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

Computational Cost

- **Syndrome-Decoding-in-the-Head:**

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

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

***Signing Time*:** 78 ms, with emulation phase of around 75 ms

Computational Cost

- **Syndrome-Decoding-in-the-Head**:

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

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

***Signing Time*:** 78 ms, with emulation phase of around 75 ms

- To deal with this issue, we propose the threshold approach:

[FR22] Feneuil, Rivain. *Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head.* To appear at Asiacrypt 2023.

The Threshold Approach

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

To share a value x,

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

The Threshold Approach

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

Properties:

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

The Threshold Approach

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

Properties:

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

Zero-Knowledge:

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

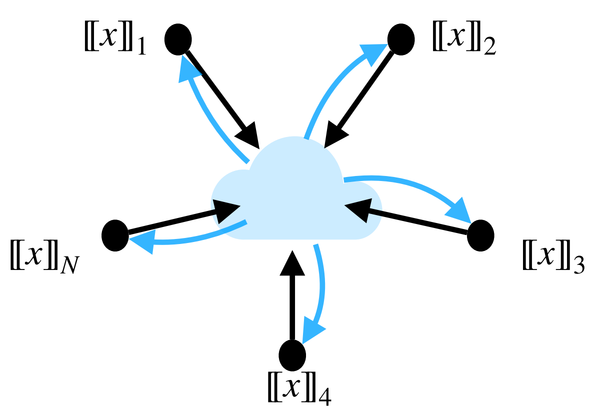
In practice, $\ell \in \{1,2,3\}$

MPCitH Transform with Threshold LSSS

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

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

$\ldots$

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

send broadcast

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

I $\longleftarrow$

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

③ Choose a random set of parties

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

⑤ Check $\forall i \in I$

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

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

Prover

Verifier

MPCitH Transform with Threshold LSSS

① Generate and commit shares

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

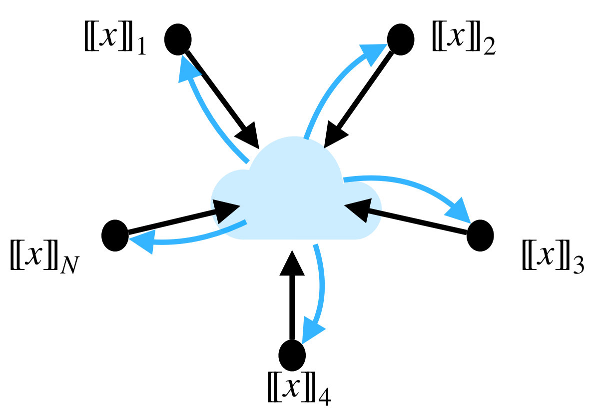
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

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

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

Prover

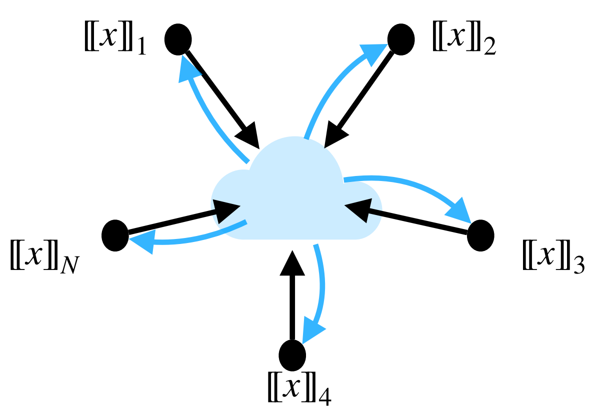
Verifier

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

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

authentication path

③ Choose a random set of parties

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

⑤ Check $\forall i \in I$

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

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

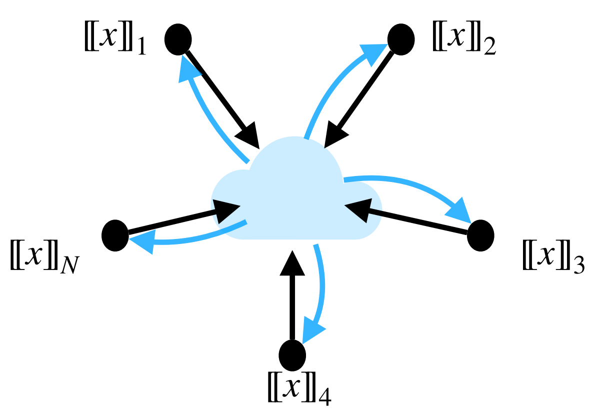
Prover

Verifier

MPCitH Transform with Threshold LSSS

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

② Run MPC in their head



④ Open parties in I

Merkle root

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

I

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

authentication path

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

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

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

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

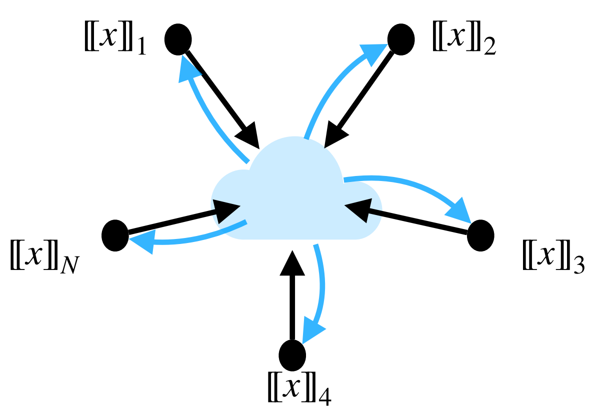
Prover

Verifier

MPCitH Transform with Threshold LSSS

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

② Run MPC in their head



④ Open parties in I

Merkle root

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

I

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

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

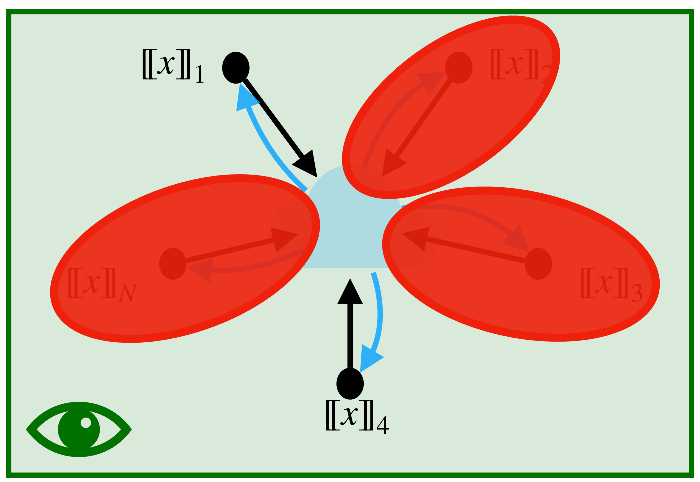
Prover

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

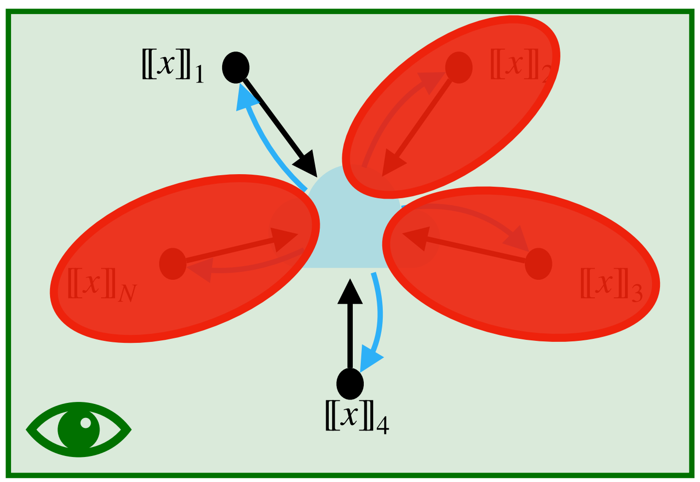
Verifier

MPCitH Transform with Threshold LSSS

① Generate and commit shares

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

② Run MPC in their head



④ Open parties in I

Prover

Merkle root

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

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

⑤ Check $\forall i \in I$

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

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

ℓ parties opened instead of $N - 1$

only ℓ party computations required

The Threshold Approach - Soundness

- ***Soundness error (for any ℓ):***

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

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

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

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

The Threshold Approach

	Additive sharing + seed trees	**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	N	2
Verifier # party computations	$N-1$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

The Threshold Approach

	Additive sharing + seed trees	**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	N	2
Verifier # party computations	$N - 1$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

Much cheaper emulation

The Threshold Approach

	Additive sharing + seed trees	**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	N	2
Verifier # party computations	$N-1$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	$\boxed{\text{Merkle tree } 2\lambda \cdot \log N}$

Fast verification algorithm

The Threshold Approach

	Additive sharing + seed trees	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	N	2
Verifier # party computations	$N - 1$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

Larger signature sizes

The Threshold Approach

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

	Additive sharing + seed trees	**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	N	2
Verifier # party computations	$N-1$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

The Threshold Approach

	Additive sharing + seed trees	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	$\cancel{N}$ $\quad 1 + \log_2 N$	2
Verifier # party computations	$\cancel{N-1}$ $\quad \log_2 N$	1
Sharing Generation and Commitment	Seed tree $\lambda \cdot \log N$	Merkle tree $2\lambda \cdot \log N$

The Hypercube technique

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

The Threshold Approach

Additive sharing
(with hypercube optimisation)

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

Threshold LSSS

Benchmark of the SDitH submission package of the NIST call

Conclusion

Conclusion

- Many signature schemes using MPC-in-the-Head, for which the security relies on the hardness of

Syndrome decoding problem

Subset sum problem

Multivariate quadratic problem

MinRank problem

Rank syndrome decoding problem

Conclusion

- Many signature schemes using MPC-in-the-Head, for which the security relies on the hardness of

  Syndrome decoding problem

  Subset sum problem

  Multivariate quadratic problem

  MinRank problem

  Rank syndrome decoding problem

- A new technique to deal small secrets with large modulus in MPCitH:

  MPCitH with rejection, with sharings over integers

- A new MPCitH transformation targeting fast running times:

  the Threshold approach

Conclusion

- NIST call for additional post-quantum signatures
  - 4 schemes directly rely on this thesis: MIRA, MQOM, RYDE, SDitH
  - 2 schemes partially use ideas of this thesis: MiRitH, PERK

Conclusion

- NIST call for additional post-quantum signatures
  - 4 schemes directly rely on this thesis: MIRA, MQOM, RYDE, SDitH
  - 2 schemes partially use ideas of this thesis: MiRitH, PERK
- Popularising the MPCitH paradigm to other cryptography communities
  - For example, the code-based community

Conclusion

- NIST call for additional post-quantum signatures
  - 4 schemes directly rely on this thesis: MIRA, MQOM, RYDE, SDitH
  - 2 schemes partially use ideas of this thesis: MiRitH, PERK
- Popularising the MPCitH paradigm to other cryptography communities
  - For example, the code-based community
- A low-level library dedicated to the MPC-in-the-Head paradigm
  - Available at `https://github.com/CryptoExperts/libmpcith`

Conclusion

- NIST call for additional post-quantum signatures
  - 4 schemes directly rely on this thesis: MIRA, MQOM, RYDE, SDitH
  - 2 schemes partially use ideas of this thesis: MiRitH, PERK
- Popularising the MPCitH paradigm to other cryptography communities
  - For example, the code-based community
- A low-level library dedicated to the MPC-in-the-Head paradigm
  - Available at `https://github.com/CryptoExperts/libmpcith`
- Some of the thesis results are not limited to the context of signatures
  - Can be applied to zero-knowledge proofs/arguments

Perspectives

- More efficient MPCitH transformations / more efficient MPC protocols
  - already some new works
    - Crypto 2023: the *VOLE-in-the-Head* construction
    - ePrint 2023/1573: *improved* Threshold approach

Perspectives

- More efficient MPCitH transformations / more efficient MPC protocols
  - already some new works
    - Crypto 2023: the *VOLE-in-the-Head* construction
    - ePrint 2023/1573: *improved* Threshold approach

- Signature schemes with advanced functionalities

  ring signatures, threshold signatures, multi-signatures,

  blind signatures, …

Perspectives

- More efficient MPCitH transformations / more efficient MPC protocols
  - already some new works
    - Crypto 2023: the *VOLE-in-the-Head* construction
    - ePrint 2023/1573: *improved* Threshold approach

- Signature schemes with advanced functionalities

ring signatures, threshold signatures, multi-signatures,

blind signatures, …

Thank you for your attention !