Cryptographic design: Trapdoor and MPCitH digital signatures

Monika Trimoska

PQSCA summer school
June 17, Albena, Bulgaria

TU/e

Trapdoor-based digital signature schemes

Multivariate signatures

Fiat-Shamir construction

trapdoor construction

Examples.

MQDSS

SOFIA

Examples.

HFEv-

UOV

The MQ problem

A quadratic system of m equations in n variables over a finite field $\mathbb{F}_q$:

$$f^{(k)}(x_1, \ldots, x_n) = \sum_{1 \le i \le j \le n} \gamma_{ij}^{(k)} x_i x_j + \sum_{1 \le i \le n} \beta_i^{(k)} x_i + \alpha^{(k)}$$

The MQ problem

Given m multivariate quadratic polynomials $f^{(1)}, \ldots, f^{(m)}$ of n variables over a finite field $\mathbb{F}_q$, find a tuple $\mathbf{x} = (x_1, \ldots, x_n)$ in $\mathbb{F}_q^n$, such that $f^{(1)}(\mathbf{x}) = \ldots = f^{(m)}(\mathbf{x}) = 0$.

The MQ problem

A quadratic system of m equations in n variables over a finite field $\mathbb{F}_q$:

$$f^{(k)}(x_1, \ldots, x_n) = \sum_{1 \le i \le j \le n} \gamma_{ij}^{(k)} x_i x_j + \sum_{1 \le i \le n} \beta_i^{(k)} x_i + \alpha^{(k)}$$

The MQ problem

Given m multivariate quadratic polynomials $f^{(1)}, \ldots, f^{(m)}$ of n variables over a finite field $\mathbb{F}_q$, find a tuple $\mathbf{x} = (x_1, \ldots, x_n)$ in $\mathbb{F}_q^n$, such that $f^{(1)}(\mathbf{x}) = \ldots = f^{(m)}(\mathbf{x}) = 0$.

→ Hard in general (should be hard for randomly generated instances).

The MQ problem

A quadratic system of m equations in n variables over a finite field $\mathbb{F}_q$:

$$f^{(k)}(x_1, \ldots, x_n) = \sum_{1 \le i \le j \le n} \gamma_{ij}^{(k)} x_i x_j + \sum_{1 \le i \le n} \beta_i^{(k)} x_i + \alpha^{(k)}$$

The MQ problem

Given m multivariate quadratic polynomials $f^{(1)}, \ldots, f^{(m)}$ of n variables over a finite field $\mathbb{F}_q$, find a tuple $\mathbf{x} = (x_1, \ldots, x_n)$ in $\mathbb{F}_q^n$, such that $f^{(1)}(\mathbf{x}) = \ldots = f^{(m)}(\mathbf{x}) = 0$.

→ Hard in general (should be hard for randomly generated instances).

→ Can become easy if we have some structure (a trapdoor).

The trapdoor construction

The trapdoor construction

- Central map:
  $f : (x_1, \dots, x_n) \in \mathbb{F}_q^n \to \left(f^{(1)}(x_1, \dots, x_n), \dots, f^{(m)}(x_1, \dots, x_n)\right) \in \mathbb{F}_q^m$

- Two bijective linear (or affine) transformations:
  $\mathbf{S} \in \mathrm{GL}_n(\mathbb{F}_q)$ and $\mathbf{T} \in \mathrm{GL}_m(\mathbb{F}_q)$

- Public map:
  $p = \mathbf{T} \circ f \circ \mathbf{S}$

The trapdoor construction

- Central map:
  $f : (x_1, \ldots, x_n) \in \mathbb{F}_q^n \to \left(f^{(1)}(x_1, \ldots, x_n), \ldots, f^{(m)}(x_1, \ldots, x_n)\right) \in \mathbb{F}_q^m$

- Two bijective linear (or affine) transformations:
  $\mathbf{S} \in \mathrm{GL}_n(\mathbb{F}_q)$ and $\mathbf{T} \in \mathrm{GL}_m(\mathbb{F}_q)$

- Public map:
  $p = \mathbf{T} \circ f \circ \mathbf{S}$

Main idea:

- The central map has a structure such that it is easy to find preimages: it is easy (polynomial time) to compute $f^{-1}(\mathbf{x})$ for a target vector $\mathbf{x}$.

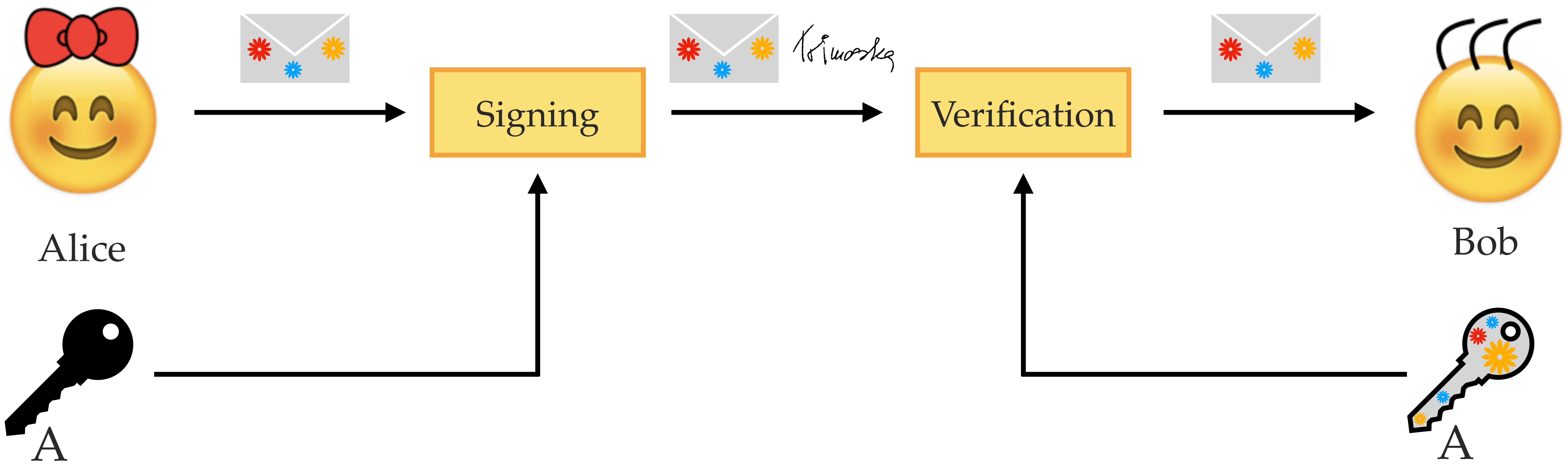
- The linear transformations hide the structure of the central map.

The trapdoor construction

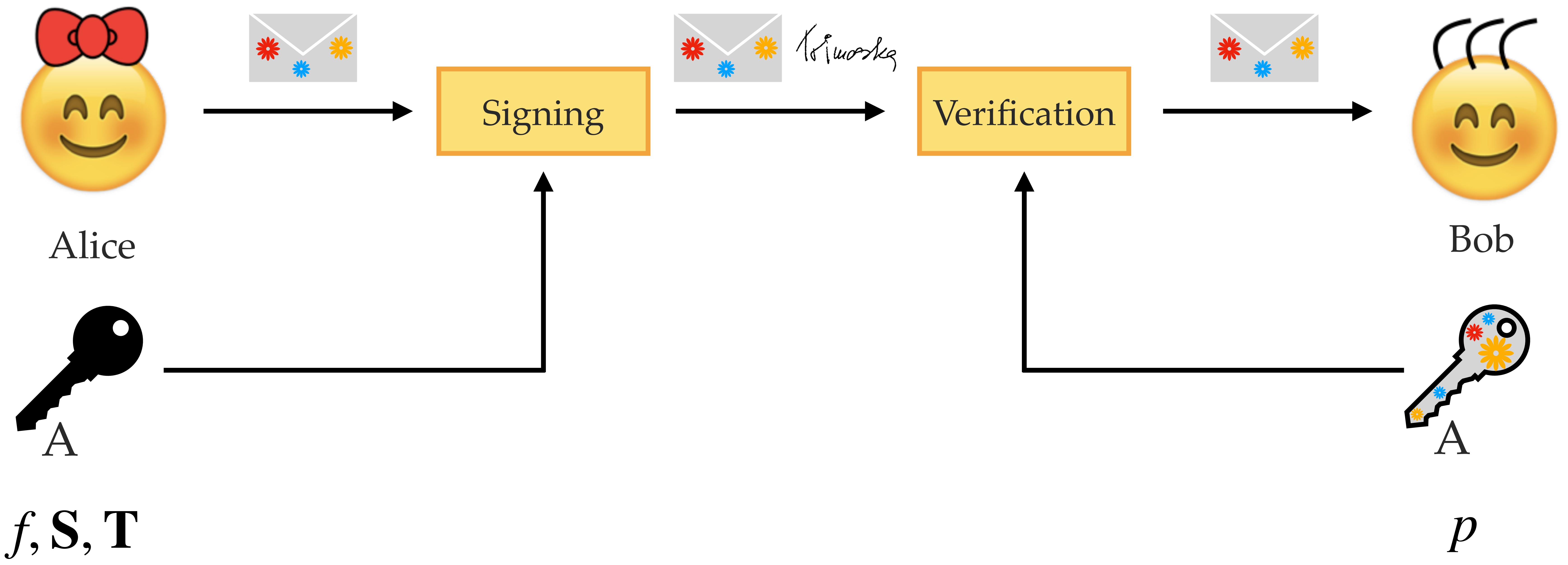
$$
\begin{tikzcd}
\mathbf{w} \in \mathbb{F}_q^m \arrow[r, "\mathbf{T}^{-1}"] & \mathbf{x} \in \mathbb{F}_q^m \arrow[r, "f^{-1}"] & \mathbf{y} \in \mathbb{F}_q^n \arrow[r, "\mathbf{S}^{-1}"] & \mathbf{z} \in \mathbb{F}_q^n \arrow[lll, bend left, "p"]
\end{tikzcd}
$$

General workflow

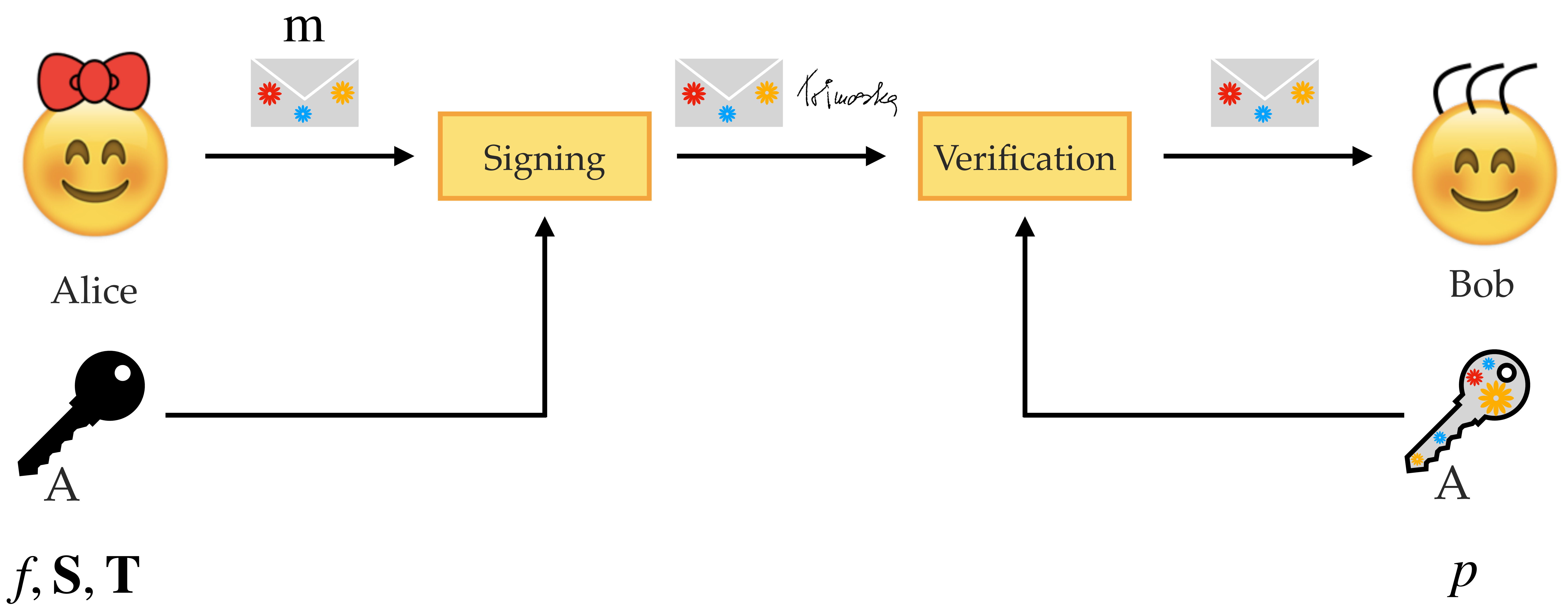
The trapdoor construction



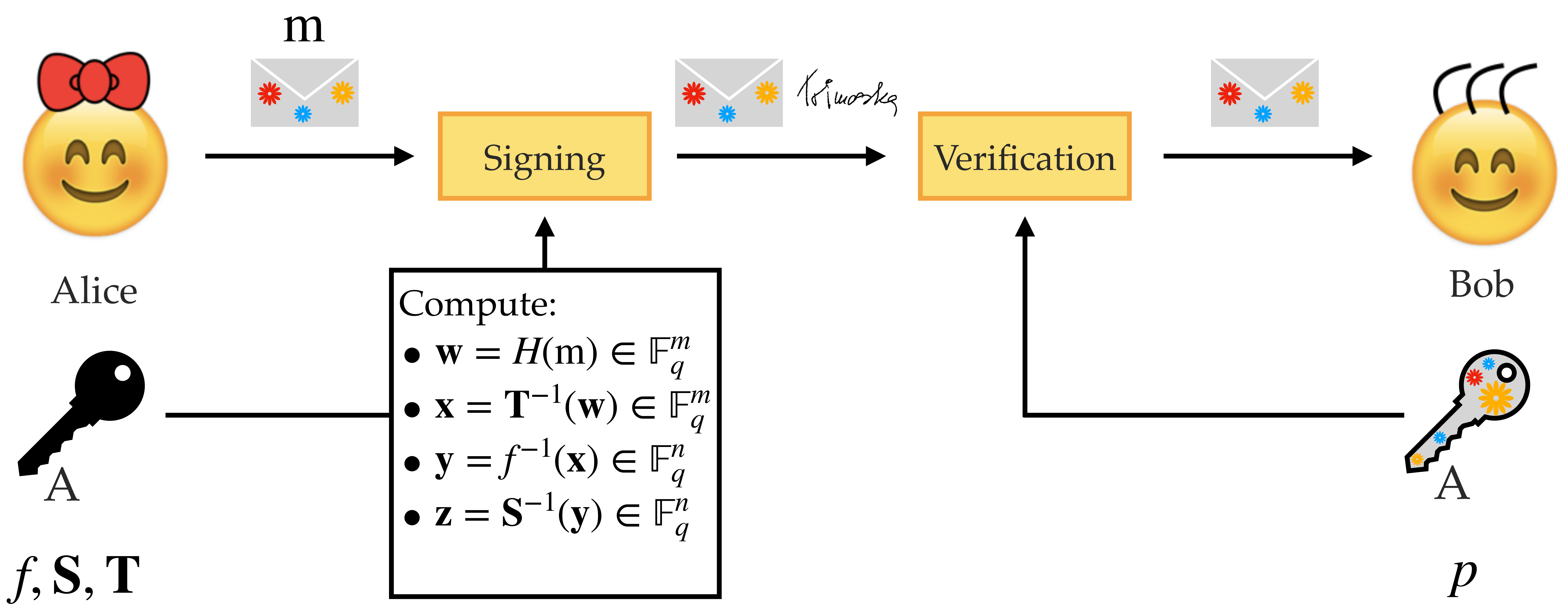
The trapdoor construction



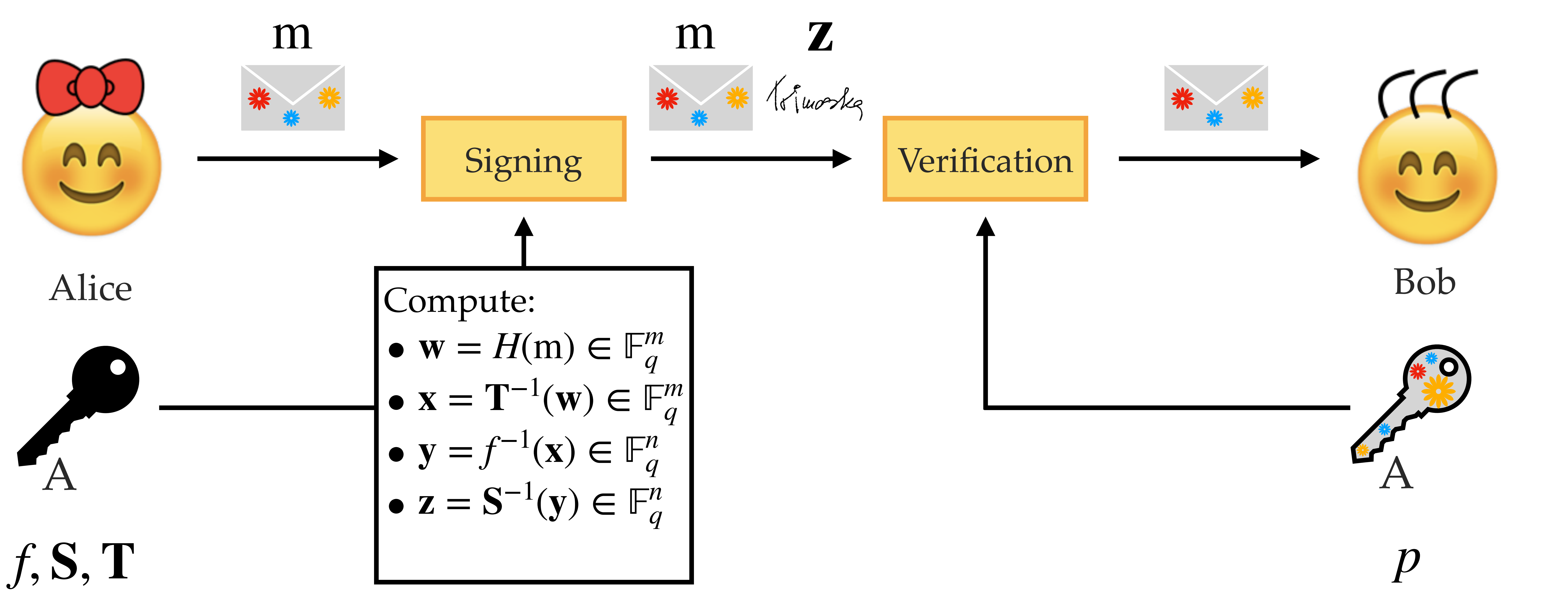
The trapdoor construction



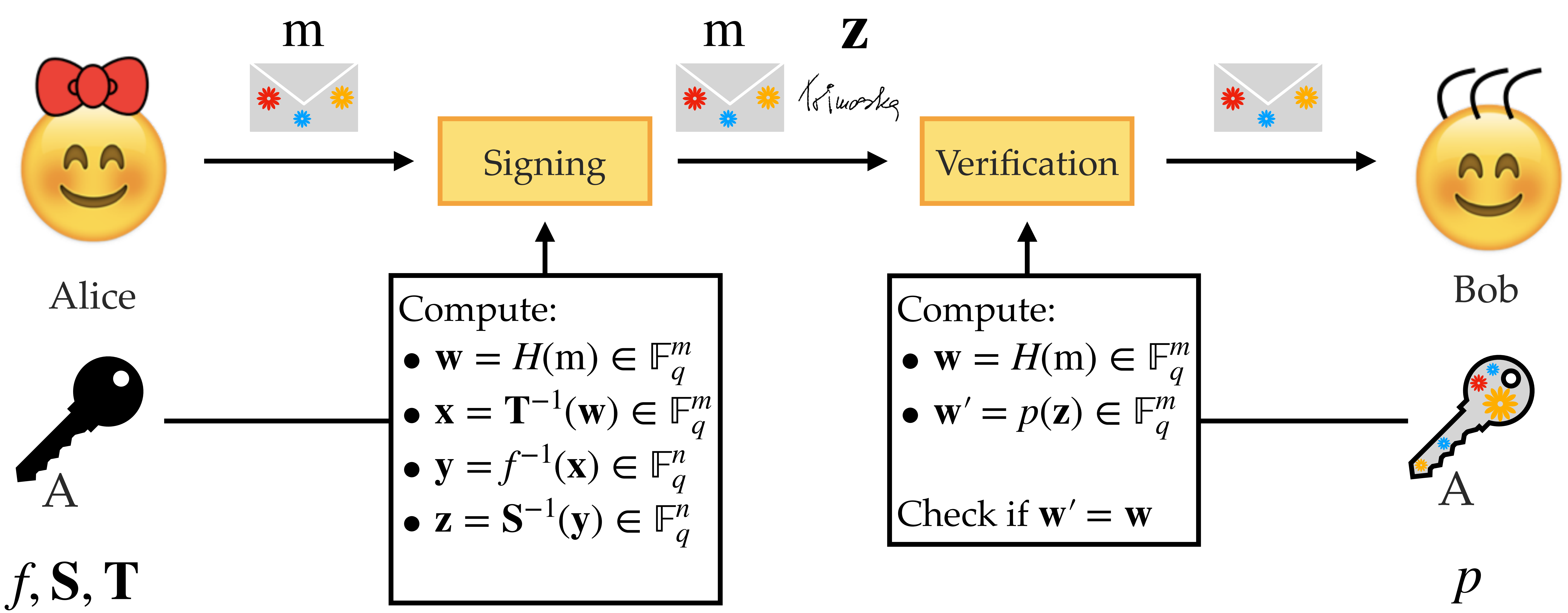
The trapdoor construction



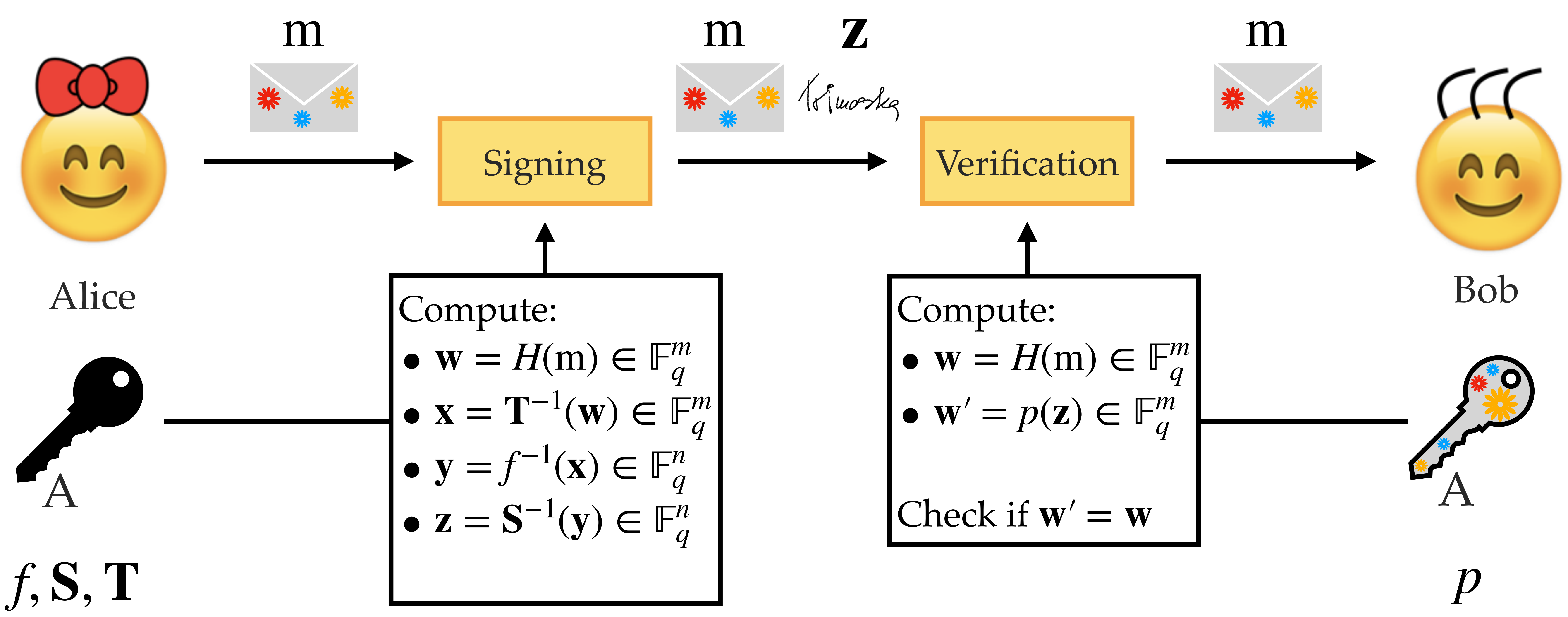
The trapdoor construction



The trapdoor construction



The trapdoor construction



Isomorphism of polynomials

Isomorphism of polynomials

The Isomorphism of Polynomials (IP) problem

Input: Two m-tuples of multivariate polynomials
$f = (f^{(1)}, \ldots, f^{(m)}), p = (p^{(1)}, \ldots, p^{(m)}) \in \mathbb{F}_q[x_1, \ldots, x_n]^m$.
Question: Find - if any - $\mathbf{S} \in \mathrm{GL}_n(\mathbb{F}_q)$ and $\mathbf{T} \in \mathrm{GL}_m(\mathbb{F}_q)$ such that $p = \mathbf{T} \circ f \circ \mathbf{S}$.

Isomorphism of polynomials

The Isomorphism of Polynomials (IP) problem

Input: Two m-tuples of multivariate polynomials $f = (f^{(1)}, \ldots, f^{(m)}), p = (p^{(1)}, \ldots, p^{(m)}) \in \mathbb{F}_q[x_1, \ldots, x_n]^m$.
Question: Find - if any - $\mathbf{S} \in \mathrm{GL}_n(\mathbb{F}_q)$ and $\mathbf{T} \in \mathrm{GL}_m(\mathbb{F}_q)$ such that $p = \mathbf{T} \circ f \circ \mathbf{S}$.

The Extended Isomorphism of Polynomials (EIP) problem

Input: An m-tuple of multivariate polynomials $p = (p^{(1)}, \ldots, p^{(m)}) \in \mathbb{F}_q[x_1, \ldots, x_n]^m$ and a special class of m-tuples of multivariate polynomials $\mathscr{C} \subseteq \mathbb{F}_q[x_1, \ldots, x_n]^m$.
Question: Find - if any - $\mathbf{S} \in \mathrm{GL}_n(\mathbb{F}_q)$, $\mathbf{T} \in \mathrm{GL}_m(\mathbb{F}_q)$ and $f = (f^{(1)}, \ldots, f^{(m)}) \in \mathscr{C}$ such that $p = \mathbf{T} \circ f \circ \mathbf{S}$.

Isomorphism of polynomials

The Isomorphism of Polynomials (IP) problem

Input: Two m-tuples of multivariate polynomials
$f = (f^{(1)}, \ldots, f^{(m)}), p = (p^{(1)}, \ldots, p^{(m)}) \in \mathbb{F}_q[x_1, \ldots, x_n]^m$.
Question: Find - if any - $\mathbf{S} \in \mathrm{GL}_n(\mathbb{F}_q)$ and $\mathbf{T} \in \mathrm{GL}_m(\mathbb{F}_q)$ such that $p = \mathbf{T} \circ f \circ \mathbf{S}$.

The Extended Isomorphism of Polynomials (EIP) problem

Input: An m-tuple of multivariate polynomials $p = (p^{(1)}, \ldots, p^{(m)}) \in \mathbb{F}_q[x_1, \ldots, x_n]^m$ and a special class of m-tuples of multivariate polynomials $\mathscr{C} \subseteq \mathbb{F}_q[x_1, \ldots, x_n]^m$.
Question: Find - if any - $\mathbf{S} \in \mathrm{GL}_n(\mathbb{F}_q)$, $\mathbf{T} \in \mathrm{GL}_m(\mathbb{F}_q)$ and $f = (f^{(1)}, \ldots, f^{(m)}) \in \mathscr{C}$ such that $p = \mathbf{T} \circ f \circ \mathbf{S}$.

Signature schemes with the trapdoor construction rely on EIP, because we do not have the central map f, but we know the special class to which it belongs (example - UOV - coming up).

Unbalanced Oil and Vinegar (UOV)

The UOV central map

Unbalanced Oil and Vinegar [Kipnis, Patarin, Goubin, '99]

$$f^{(k)}(x_1, \ldots, x_n) = \sum_{i \in V, j \in V} \gamma_{ij}^{(k)} x_i x_j + \sum_{i \in V, j \in O} \gamma_{ij}^{(k)} x_i x_j + \sum_{i=1}^{n} \beta_i^{(k)} x_i + \alpha^{(k)}$$

Index set of vinegar variables: $V = \{1, \ldots, v\}$

Index set of oil variables: $O = \{v+1, \ldots, n\}$

The UOV central map

Unbalanced Oil and Vinegar [Kipnis, Patarin, Goubin, '99]

$$f^{(k)}(x_1, \ldots, x_n) = \sum_{i \in V, j \in V} \gamma_{ij}^{(k)} x_i x_j + \sum_{i \in V, j \in O} \gamma_{ij}^{(k)} x_i x_j + \sum_{i=1}^{n} \beta_i^{(k)} x_i + \alpha^{(k)}$$

Index set of vinegar variables: $V = \{1, \ldots, v\}$

Index set of oil variables: $O = \{v+1, \ldots, n\}$

→ The central map is constructed in such a way that enumerating all of the vinegar variables leaves us with a linear system in the oil variables (oil does not mix with oil).

The UOV central map

Unbalanced Oil and Vinegar [Kipnis, Patarin, Goubin, '99]

$$f^{(k)}(x_1, \ldots, x_n) = \sum_{i \in V, j \in V} \gamma_{ij}^{(k)} x_i x_j + \sum_{i \in V, j \in O} \gamma_{ij}^{(k)} x_i x_j + \sum_{i=1}^{n} \beta_i^{(k)} x_i + \alpha^{(k)}$$

Index set of vinegar variables: $V = \{1, \ldots, v\}$

Index set of oil variables: $O = \{v+1, \ldots, n\}$

- The central map is constructed in such a way that enumerating all of the vinegar variables leaves us with a linear system in the oil variables (oil does not mix with oil).
- Everything is as described in the previous slides, except that we do not have a linear transformation on the output: $\mathbf{T} = \mathbf{I}$.

Matrix representation of quadratic forms

Quadratic form: $f(\mathbf{x}) = \sum \gamma_{ij} x_i x_j$

$$
\underbrace{\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}}_{\mathbf{x}^\top}
\quad
\underbrace{\begin{bmatrix}
\gamma_{1,1} & \frac{\gamma_{1,2}}{2} & \frac{\gamma_{1,3}}{2} & \frac{\gamma_{1,4}}{2} \\
\frac{\gamma_{2,1}}{2} & \gamma_{2,2} & \frac{\gamma_{2,3}}{2} & \frac{\gamma_{2,4}}{2} \\
\frac{\gamma_{3,1}}{2} & \frac{\gamma_{3,2}}{2} & \gamma_{3,3} & \frac{\gamma_{3,4}}{2} \\
\frac{\gamma_{4,1}}{2} & \frac{\gamma_{4,2}}{2} & \frac{\gamma_{4,3}}{2} & \gamma_{4,4}
\end{bmatrix}}_{\mathbf{F}}
\quad
\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{x}}
$$

so with $\mathbf{x} = (x_1, \ldots, x_n)$, we get $\mathbf{x}^\top \mathbf{F} \mathbf{x}$.

Matrix representation of bilinear forms

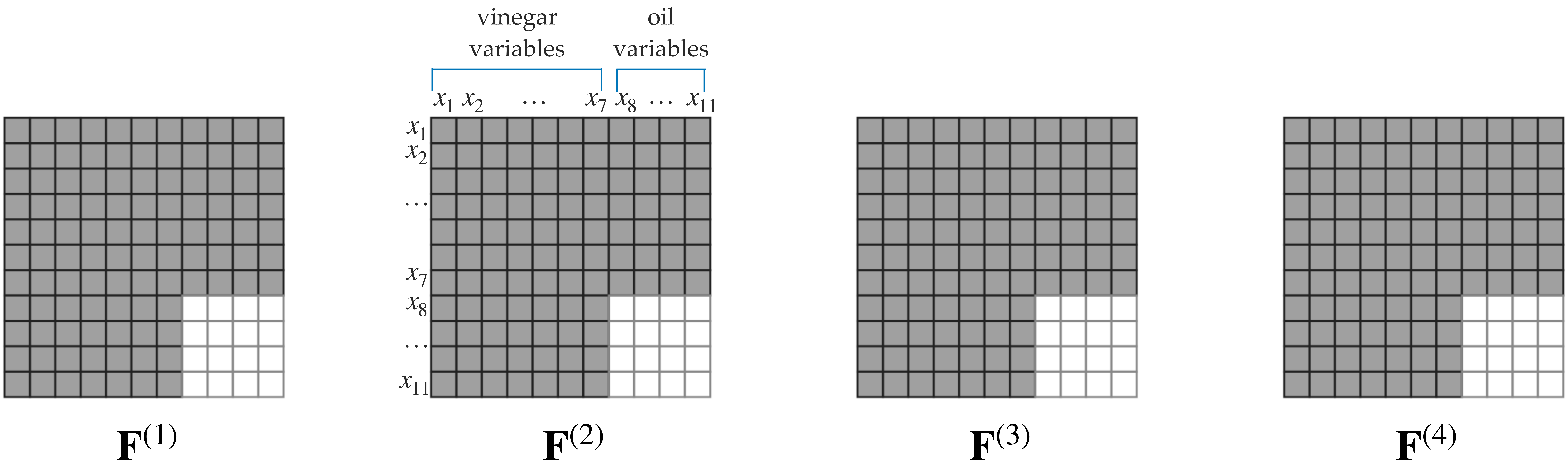
Bilinear form: $f(\mathbf{x}, \mathbf{y}) = \sum \gamma_{ij} x_i y_j$

$$
\overset{\mathbf{x}^\top}{\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}}
\qquad
\overset{\mathbf{B}}{\begin{bmatrix} \gamma_{1,1} & \gamma_{1,2} & \gamma_{1,3} & \gamma_{1,4} \\ \gamma_{2,1} & \gamma_{2,2} & \gamma_{2,3} & \gamma_{2,4} \\ \gamma_{3,1} & \gamma_{3,2} & \gamma_{3,3} & \gamma_{3,4} \\ \gamma_{4,1} & \gamma_{4,2} & \gamma_{4,3} & \gamma_{4,4} \end{bmatrix}}
\qquad
\overset{\mathbf{y}}{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}
$$

so with $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, we get $\mathbf{x}^\top \mathbf{B} \mathbf{y}$.

The UOV central map

Toy example: $v = 7, m = 4$



*Grayed areas represent the entries that are possibly nonzero; blank areas denote the zero entries;

UOV key generation

In matrix representation

$$\mathbf{P}^{(k)} = \mathbf{S}^\top \mathbf{F}^{(k)} \mathbf{S}, \text{ for all } k \in \{1, \ldots, m\}.$$

UOV key generation

In matrix representation

$$\mathbf{P}^{(k)} = \mathbf{S}^\top \mathbf{F}^{(k)} \mathbf{S}, \text{ for all } k \in \{1, \ldots, m\}.$$

Why ?

UOV key generation

In matrix representation

$\mathbf{P}^{(k)} = \mathbf{S}^{\top}\mathbf{F}^{(k)}\mathbf{S}$, for all $k \in \{1, \ldots, m\}$.

Why ?

By definition, $p = f \circ \mathbf{S}$.

UOV key generation

In matrix representation

$\mathbf{P}^{(k)} = \mathbf{S}^{\top}\mathbf{F}^{(k)}\mathbf{S}$, for all $k \in \{1, \ldots, m\}$.

Why ?

By definition, $p = f \circ \mathbf{S}$.

In matrix representation, we need:

$$\mathbf{x}^{\top}\mathbf{P}^{(k)}\mathbf{x} = (\mathbf{S}\mathbf{x})^{\top}\mathbf{F}^{(k)}(\mathbf{S}\mathbf{x})$$

UOV key generation

In matrix representation

$\mathbf{P}^{(k)} = \mathbf{S}^\top \mathbf{F}^{(k)} \mathbf{S}$, for all $k \in \{1, \ldots, m\}$.

Why ?

By definition, $p = f \circ \mathbf{S}$.

In matrix representation, we need:

$$\mathbf{x}^\top \mathbf{P}^{(k)} \mathbf{x} = (\mathbf{S}\mathbf{x})^\top \mathbf{F}^{(k)} (\mathbf{S}\mathbf{x})$$

$$\mathbf{x}^\top \mathbf{P}^{(k)} \mathbf{x} = \mathbf{x}^\top \mathbf{S}^\top \mathbf{F}^{(k)} \mathbf{S}\mathbf{x}$$

UOV key generation

In matrix representation

$$\mathbf{P}^{(k)} = \mathbf{S}^\top \mathbf{F}^{(k)} \mathbf{S}, \text{ for all } k \in \{1, \ldots, m\}.$$

Why ?

By definition, $p = f \circ \mathbf{S}$.

In matrix representation, we need:

$$\mathbf{x}^\top \mathbf{P}^{(k)} \mathbf{x} = (\mathbf{S}\mathbf{x})^\top \mathbf{F}^{(k)} (\mathbf{S}\mathbf{x})$$

$$\mathbf{x}^\top \mathbf{P}^{(k)} \mathbf{x} = \mathbf{x}^\top \mathbf{S}^\top \mathbf{F}^{(k)} \mathbf{S} \mathbf{x}$$

UOV in the NIST competition

UOV
TUOV
PROV
MAYO
VOX
QR-UOV
SNOVA

UOV in the NIST competition

UOV
TUOV
PROV
MAYO
VOX
QR-UOV
SNOVA

Example.

	NIST S.L.	n	m	q	$\lvert\mathsf{epk}\rvert$ (bytes)	$\lvert\mathsf{esk}\rvert$ (bytes)	$\lvert\mathsf{cpk}\rvert$ (bytes)	$\lvert\mathsf{csk}\rvert$ (bytes)	signatur (bytes)
`uov-Ip`	1	112	44	256	278 432	237 896	43 576	32	128
`uov-Is`	1	160	64	16	412 160	348 704	66 576	32	96
`uov-III`	3	184	72	256	1 225 440	1 044 320	189 232	32	200
`uov-V`	5	244	96	256	2 869 440	2 436 704	446 992	32	260

UOV in the NIST competition

- UOV
- TUOV
- PROV
- MAYO
- VOX
- QR-UOV
- SNOVA

Example.

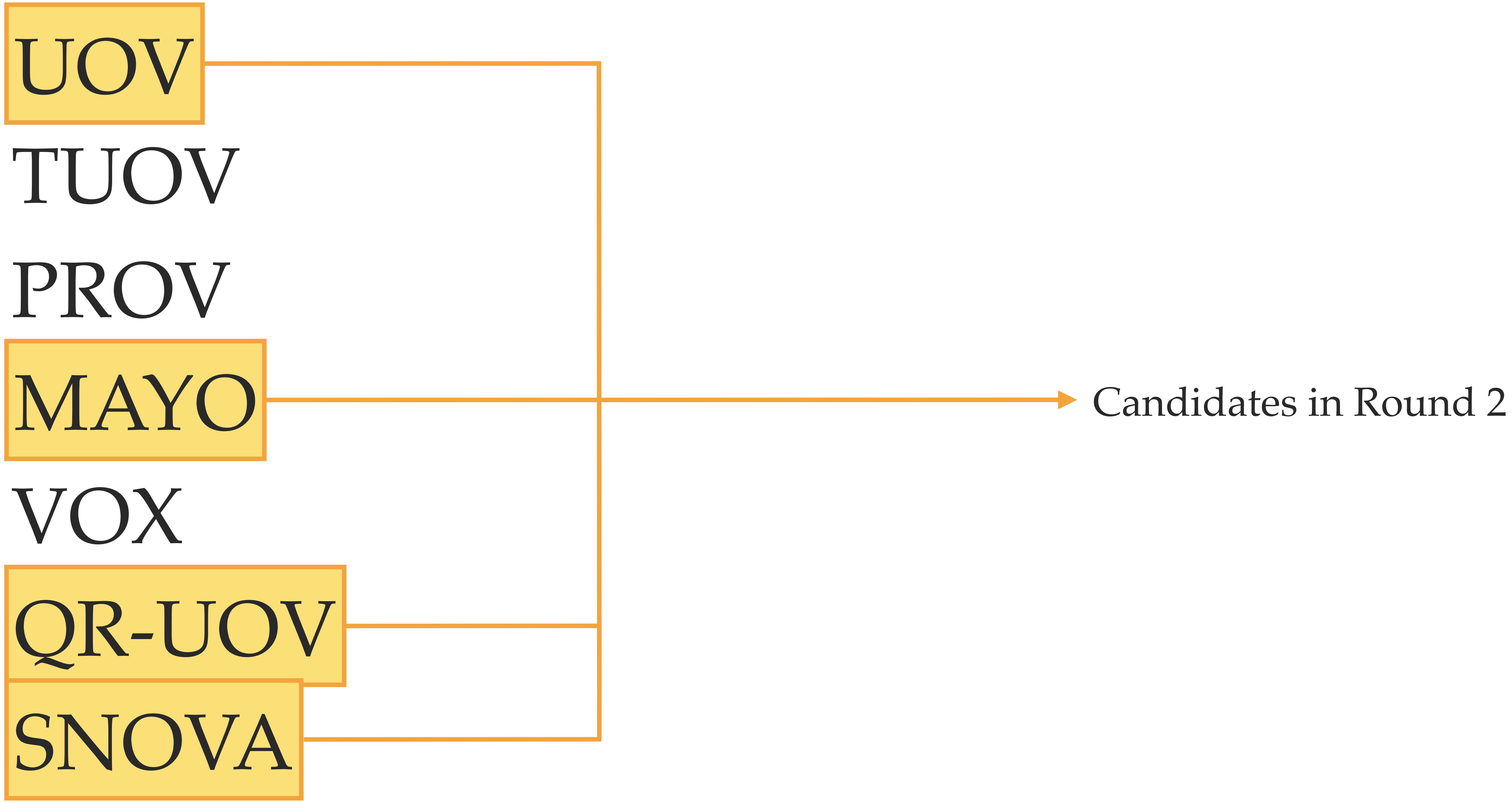
	NIST S.L.	n	m	q	|epk| (bytes)	|esk| (bytes)	|cpk| (bytes)	|csk| (bytes)	signatur (bytes)
uov-Ip	1	112	44	256	278 432	237 896	43 576	32	128
uov-Is	1	160	64	16	412 160	348 704	66 576	32	96
uov-III	3	184	72	256	1 225 440	1 044 320	189 232	32	200
uov-V	5	244	96	256	2 869 440	2 436 704	446 992	32	260

- We choose $n \sim 2.5m$ (slightly bigger than)

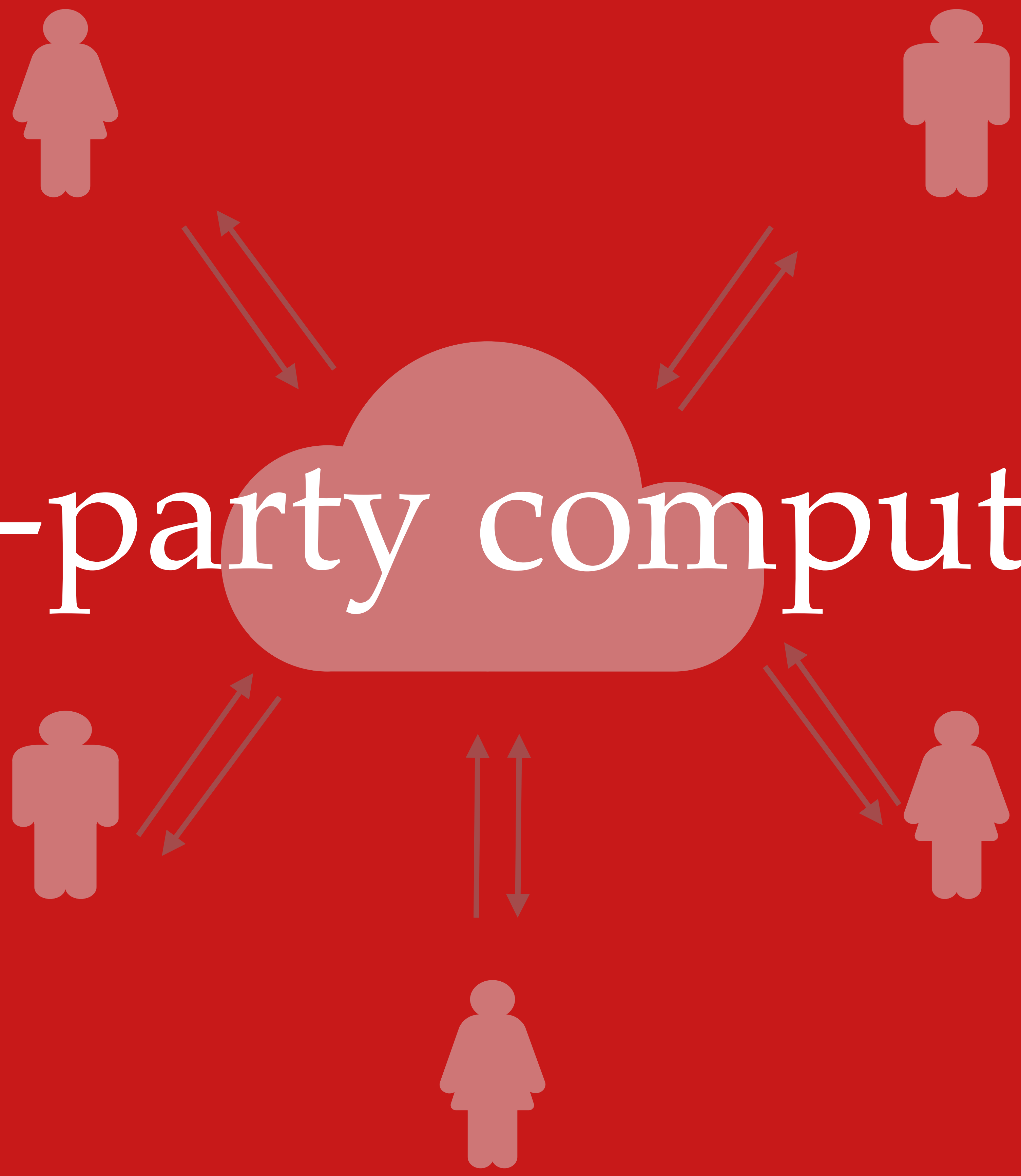
UOV-like schemes have:

- Big public keys
- Small signatures

UOV in the NIST competition



Multi-party computation



The MPCitH construction

One-way function
$f(x) = y$

MPC protocol

MPCitH
identification
scheme

Digital signature

The MPCitH construction

One-way function
$f(x) = y$

MPC protocol

MPCitH
identification
scheme

Digital signature

The MPCitH construction

One-way function
$f(x) = y$

Zero-knowledge proof of knowledge: the prover needs to prove that they know a secret x s.t. $f(x) = y$, for a public y, without revealing any information about x.

MPC protocol

MPCitH identification scheme

Digital signature

One-way function

→ Standard symmetric primitives

→ MPC-friendly symmetric primitives

→ Well-known hard problems

One-way function

→ Standard symmetric primitives

→ MPC-friendly symmetric primitives

→ Well-known hard problems

With the MPC framework, we can compute anything in a shared manner. For problems with additively homomorphic properties, this is straightforward. Otherwise, we need to find workarounds or reformulate the problem.

One-way function

→ Standard symmetric primitives

→ MPC-friendly symmetric primitives

→ Well-known hard problems

With the MPC framework, we can compute anything in a shared manner. For problems with additively homomorphic properties, this is straightforward. Otherwise, we need to find workarounds or reformulate the problem.

One-way function

- Standard symmetric primitives
- MPC-friendly symmetric primitives
- Well-known hard problems

With the MPC framework, we can compute anything in a shared manner. For problems with additively homomorphic properties, this is straightforward. Otherwise, we need to find workarounds or reformulate the problem.

Examples in this talk: Discrete log Syndrome decoding

The MPCitH construction

One-way function
$f(x) = y$

Zero-knowledge proof of knowledge: the prover needs to prove that they know a secret x s.t. $f(x) = y$, for a public y, without revealing any information about x.

MPC protocol

Multi-party computation

MPCitH identification scheme

Digital signature

MPC: Multi-party computation

$f(x) = y$

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

MPC: Multi-party computation

$f(x) = y$

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

Jointly compute:

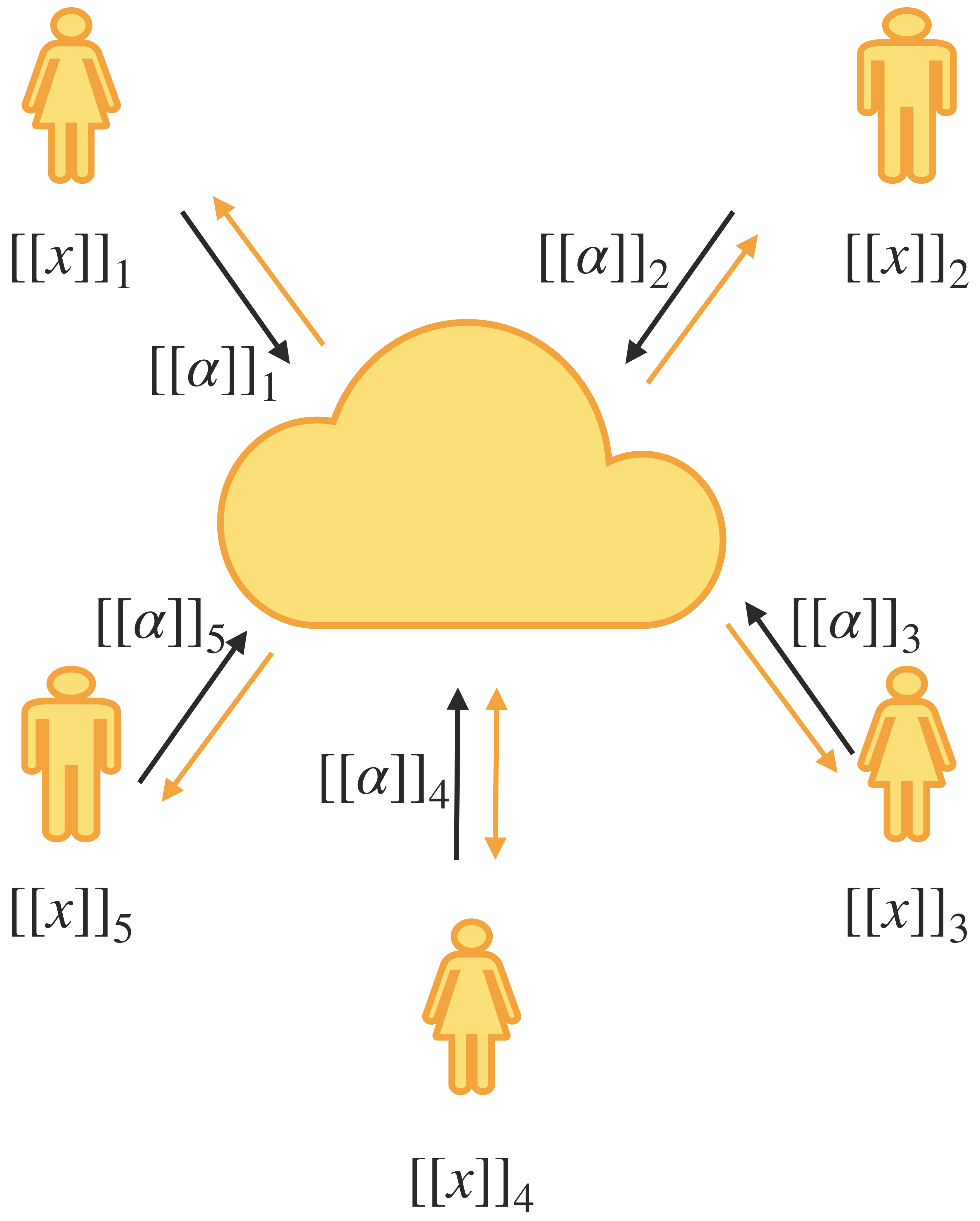
$$g(x) = \begin{cases} \text{Accept}, & \text{if } f(x) = y \\ \text{Reject}, & \text{if } f(x) \neq y \end{cases}$$

MPC: Multi-party computation

$f(x) = y$

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

Broadcast model



Jointly compute:

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

Toy example: discrete logarithm

The discrete log problem

Given elements g and h of a finite cyclic group, find x such that $g^x = h$.

Toy example: discrete logarithm

The discrete log problem

Given elements g and h of a finite cyclic group, find x such that $g^x = h$.

Easy example because it is additively homomorphic.

$$g^{[[x]]_1} \cdot g^{[[x]]_2} \cdot \ldots \cdot g^{[[x]]_N} = g^x,$$

$$\text{for } x = [[x]]_1 + [[x]]_2 + \ldots + [[x]]_N.$$

Toy example: discrete logarithm

The discrete log problem

Given elements g and h of a finite cyclic group, find x such that $g^x = h$.

Easy example because it is additively homomorphic.

$$g^{[[x]]_1} \cdot g^{[[x]]_2} \cdot \ldots \cdot g^{[[x]]_N} = g^x,$$

$$\text{for } x = [[x]]_1 + [[x]]_2 + \ldots + [[x]]_N.$$

Jointly check whether $g^x = h$:

- Each party computes $[[h]]_i = g^{[[x]]_i}$.
- (After broadcast) check whether $\prod_{i=1}^{N} [[h]]_i = h$.

Toy example: discrete logarithm

The discrete log problem

Given elements g and h of a finite cyclic group, find x such that $g^x = h$.

Easy example because it is additively homomorphic.

$$g^{[[x]]_1} \cdot g^{[[x]]_2} \cdot \ldots \cdot g^{[[x]]_N} = g^x,$$

$$\text{for } x = [[x]]_1 + [[x]]_2 + \ldots + [[x]]_N.$$

Jointly check whether $g^x = h$:

→ Each party computes $[[h]]_i = g^{[[x]]_i}$.

→ (After broadcast) check whether $\prod_{i=1}^{N} [[h]]_i = h$.

! Not post-quantum

Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k\times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

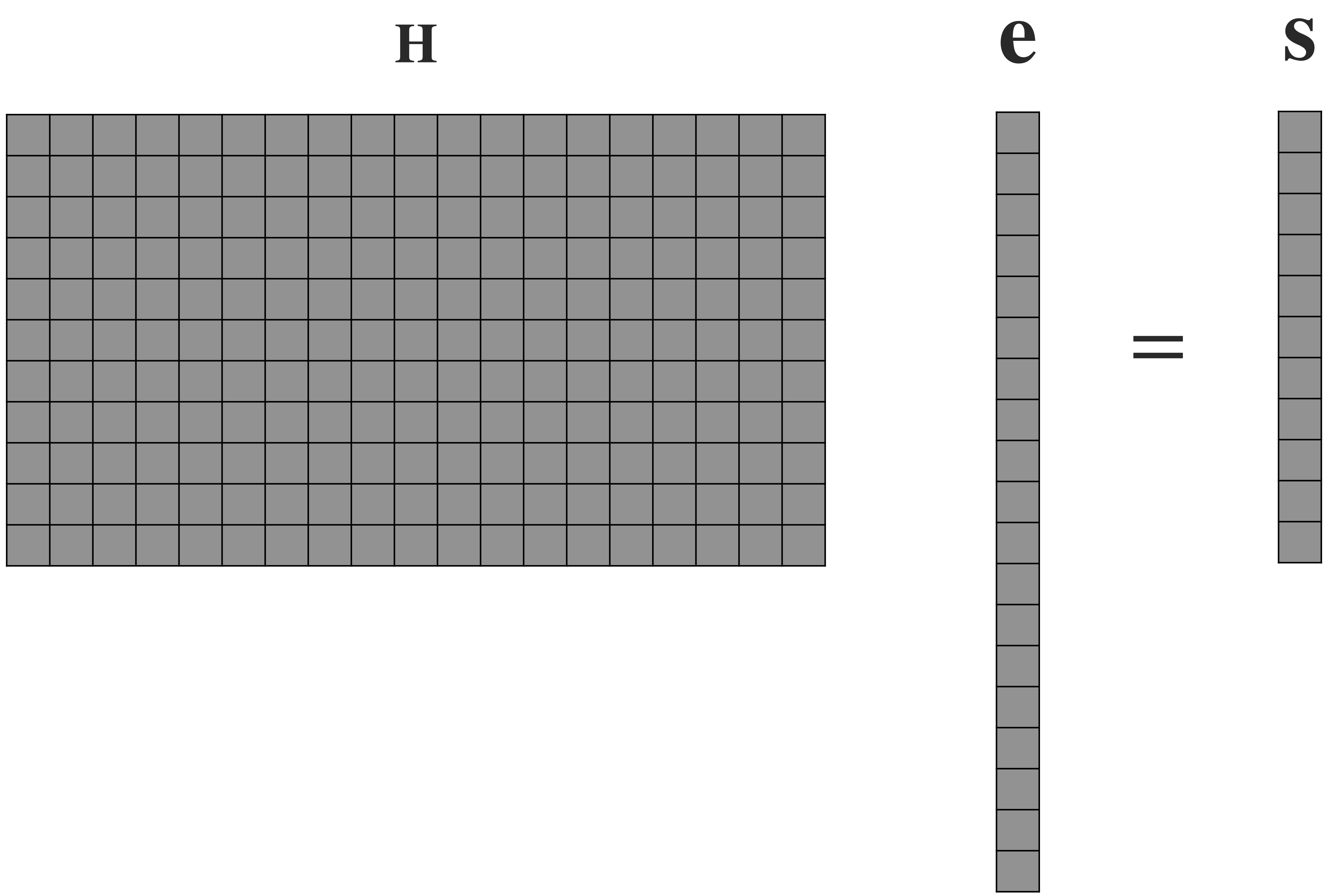
Hamming weight (number of nonzero entries)

Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k \times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

Hamming weight (number of nonzero entries)



Entry is 0

Entry is 1

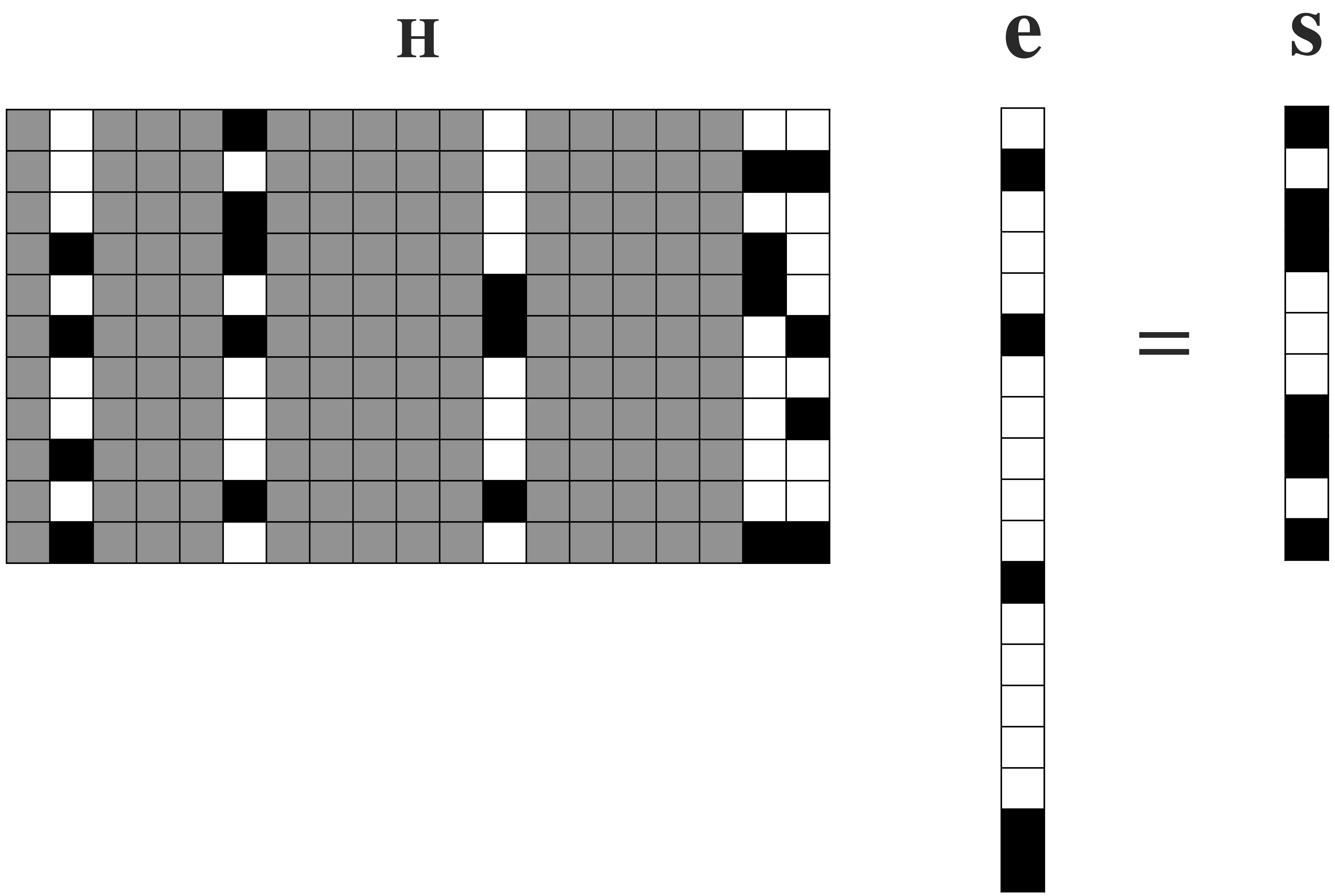
Entry is 0 or 1

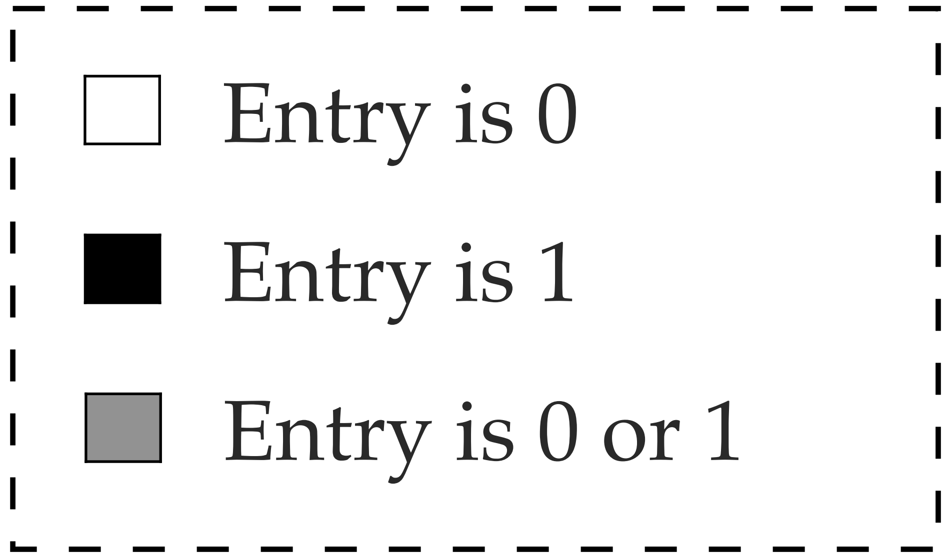
Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k \times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\underline{\mathrm{wt}(\mathbf{e})} = t$.

Hamming weight (number of nonzero entries)



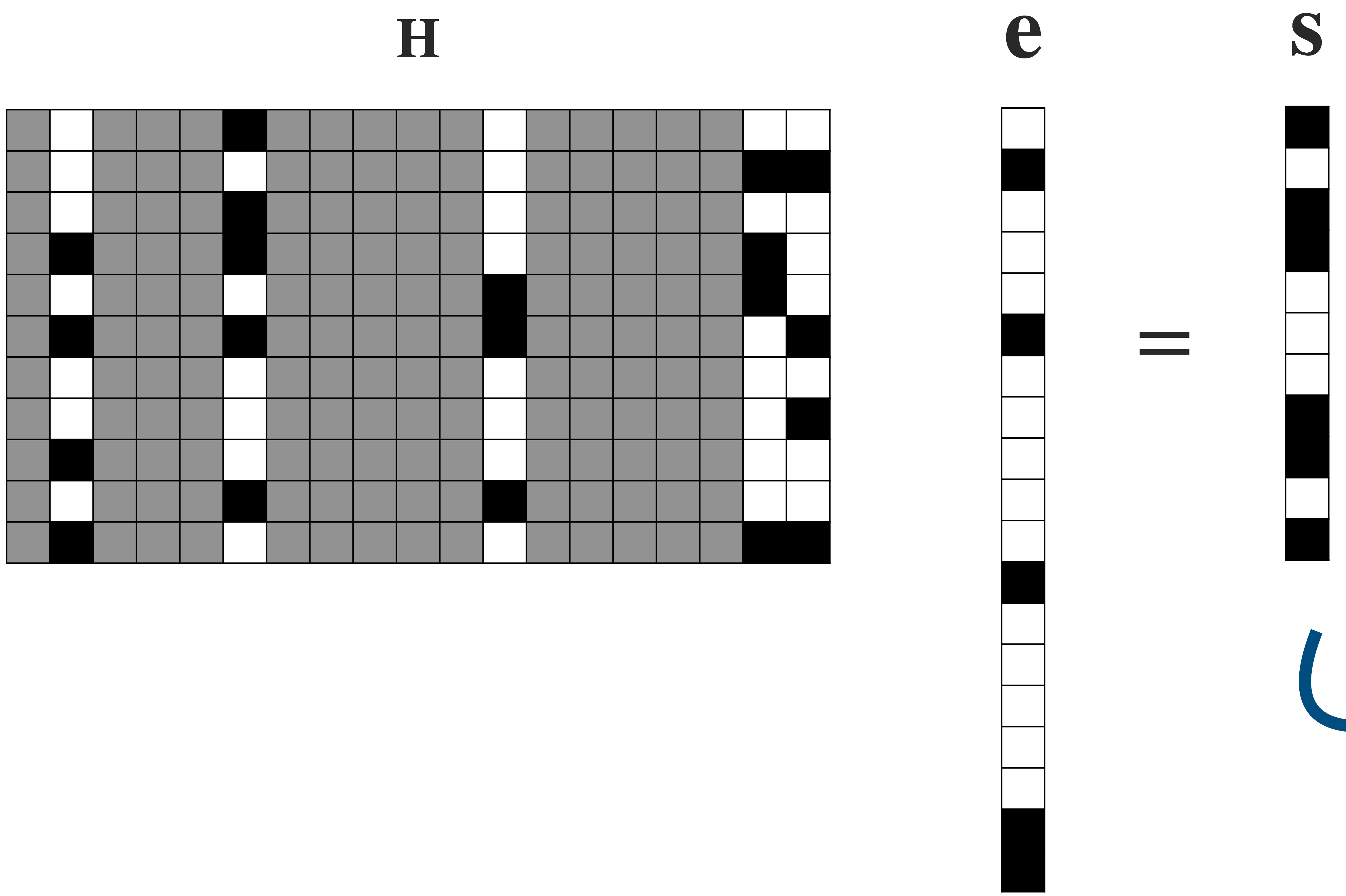


Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k\times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

Hamming weight (number of nonzero entries)



Entry is 0

Entry is 1

Entry is 0 or 1

$\mathbf{s}$ is equal to the sum of the columns where e_i is nonzero.

Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k\times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

Not a 'linear problem' (otherwise, it would be easy).

because of the weight constraint

Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k\times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

Not a 'linear problem' (otherwise, it would be easy).

because of the weight constraint

Finding $\mathbf{e}$ such that $\mathbf{s} = \mathbf{He}$ is easy.

Finding $\mathbf{e}$ such that $\mathrm{wt}(\mathbf{e}) = t$ is easy.

Finding $\mathbf{e}$ that satisfies both constraints is hard.

Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k\times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k\times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

Jointly check whether $\mathbf{s} = \mathbf{He}$:

$\longrightarrow$ Each party computes $[[\mathbf{s}_i]] = \mathbf{H}[[\mathbf{e}_i]]$.

$\longrightarrow$ (After broadcast) check whether $\displaystyle\sum_{i=1}^{N} [[\mathbf{s}_i]] = \mathbf{s}$.

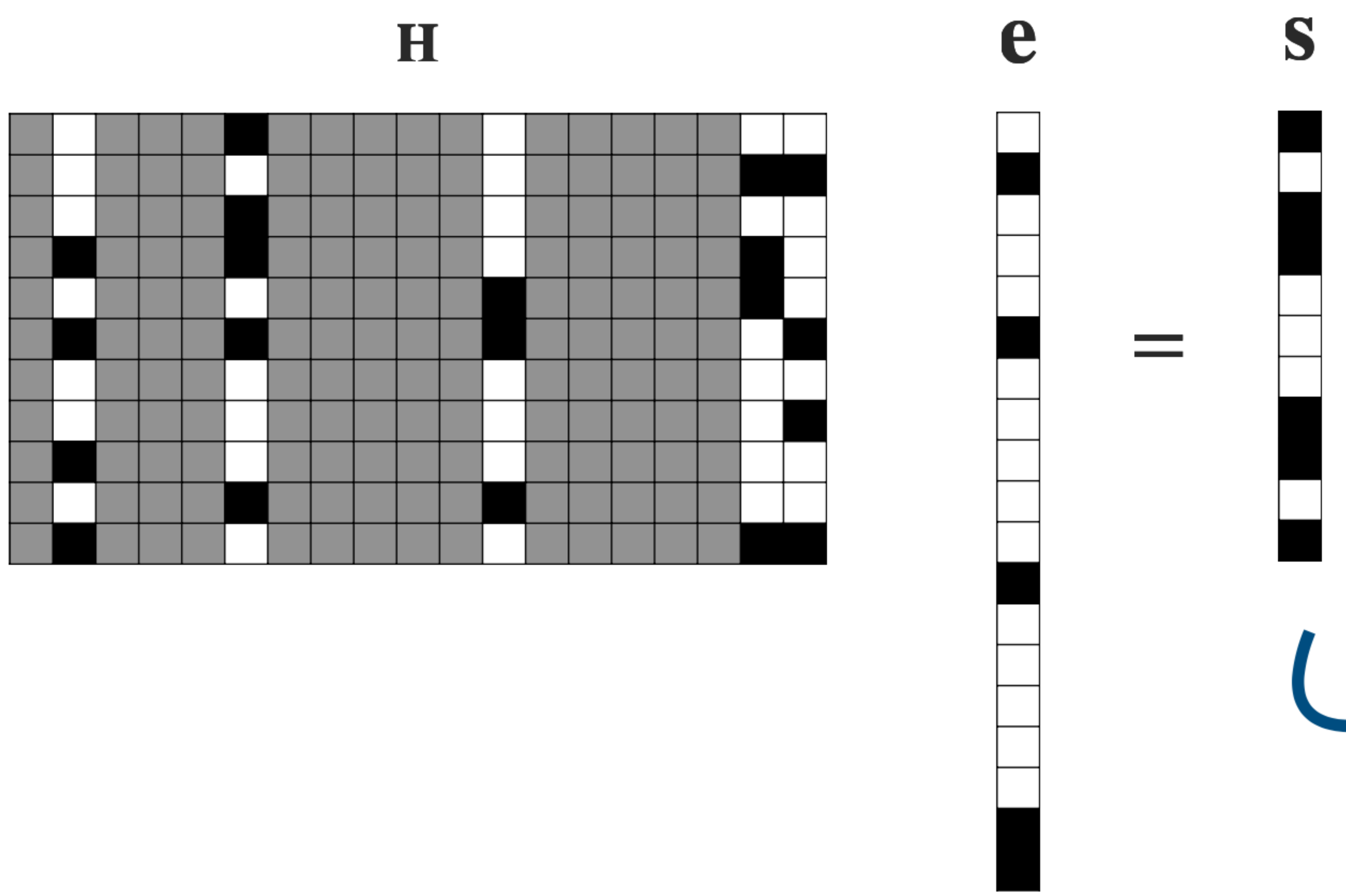
Relevant example: syndrome decoding

The syndrome decoding problem

Jointly check whethe

→ Each pa

→ (After b



s is equal to the sum of the columns where e_i is nonzero.

Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k\times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

Jointly check whether $\mathbf{s} = \mathbf{He}$:

→ Each party computes $[[\mathbf{s}_i]] = \mathbf{H}[[\mathbf{e}_i]]$.

→ (After broadcast) check whether $\sum_{i=1}^{N} [[\mathbf{s}_i]] = \mathbf{s}$.

Relevant example: syndrome decoding

The syndrome decoding problem

Given $\mathbf{s} \in \mathbb{F}_q^k$ and $\mathbf{H} \in \mathscr{M}_{k \times n}(\mathbb{F}_q)$ and an integer $t \leq n$, find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{s} = \mathbf{He}$ and $\mathrm{wt}(\mathbf{e}) = t$.

Jointly check whether $\mathbf{s} = \mathbf{He}$:

→ Each party computes $[[\mathbf{s}_i]] = \mathbf{H}[[\mathbf{e}_i]]$.

→ (After broadcast) check whether $\sum_{i=1}^{N} [[\mathbf{s}_i]] = \mathbf{s}$.

Jointly check whether $\mathrm{wt}(\mathbf{e}) = t$:

→ More complicated.

The MPCitH construction

One-way function
$f(x) = y$

Zero-knowledge proof of knowledge: the prover needs to prove that they know a secret x s.t. $f(x) = y$, for a public y, without revealing any information about x (as in Lecture 3).

MPC protocol

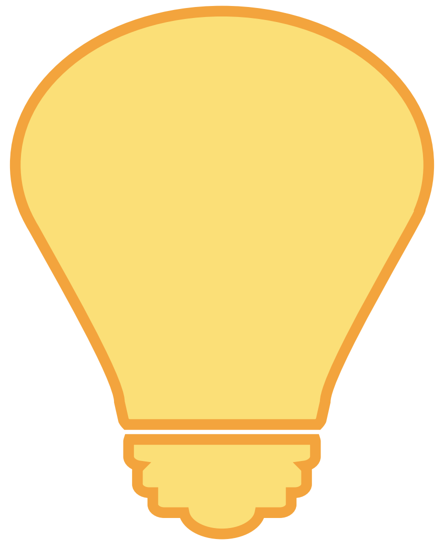
Multi-party computation

MPCitH identification scheme

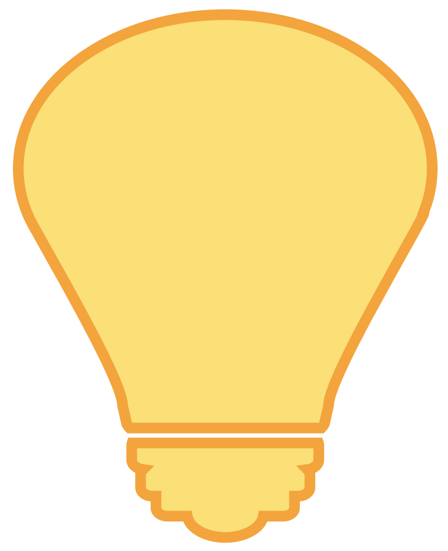
MPC-in-the-Head

Digital signature

MPCitH



MPCitH



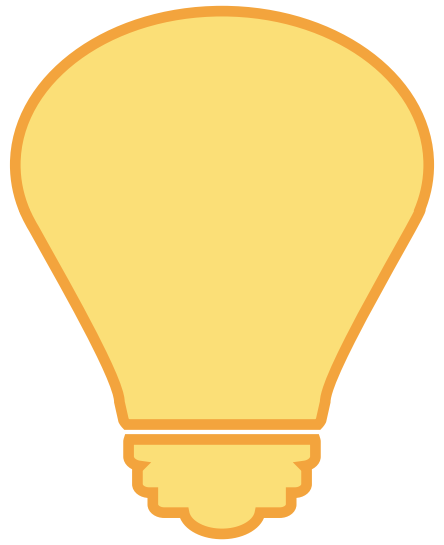
→ Properties that the underlying MPC model needs to have:

MPCitH

→ Properties that the underlying MPC model needs to have:

- $(N-1)$-private: the views of any $N-1$ parties do not reveal any information on x.
- Semi-honest: weak notion in MPC, but enough for the MPCitH application.

MPCitH

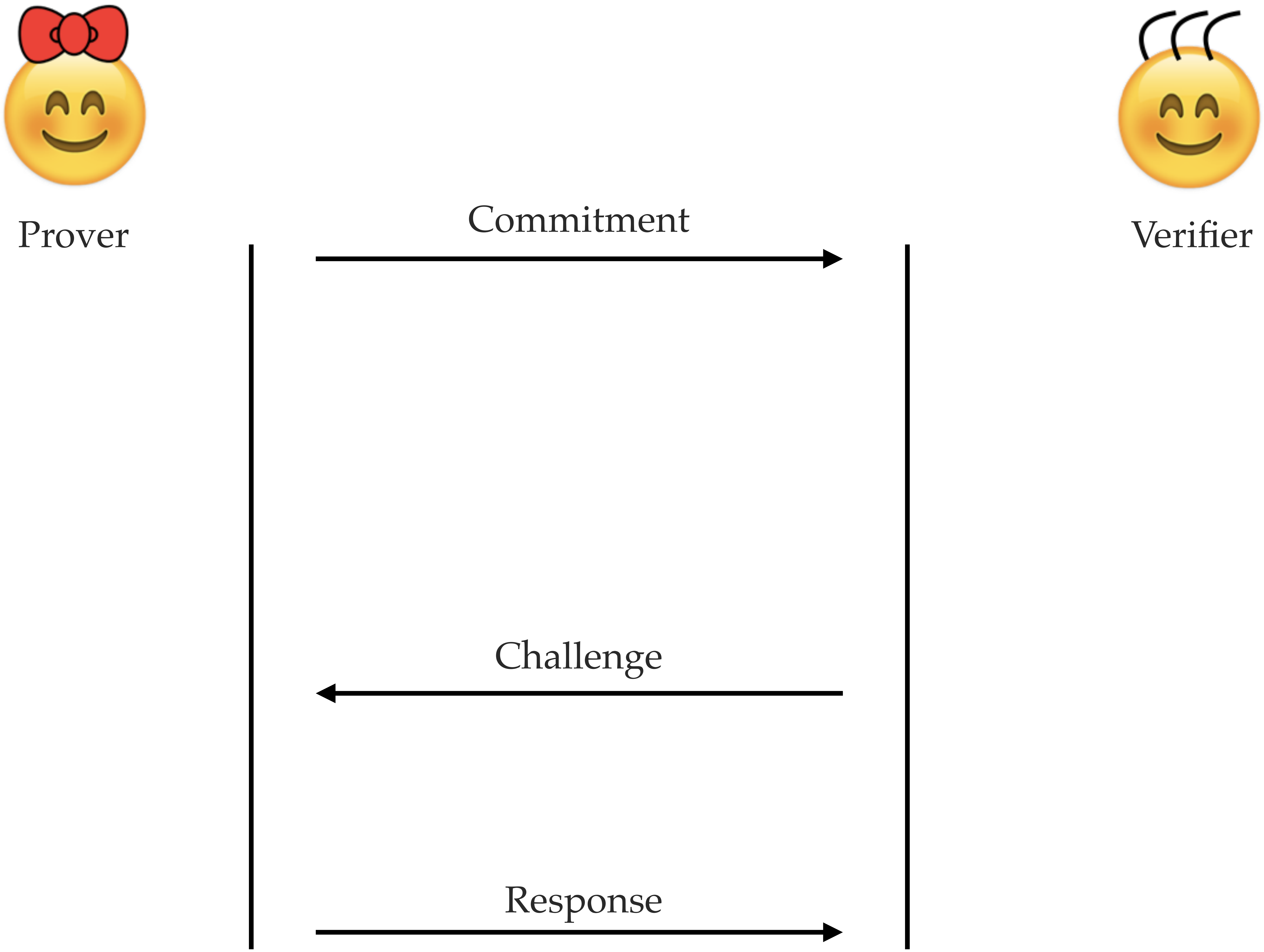


→ Properties that the underlying MPC model needs to have:

- $(N-1)$-private: the views of any $N-1$ parties do not reveal any information on x.
- Semi-honest: weak notion in MPC, but enough for the MPCitH application.

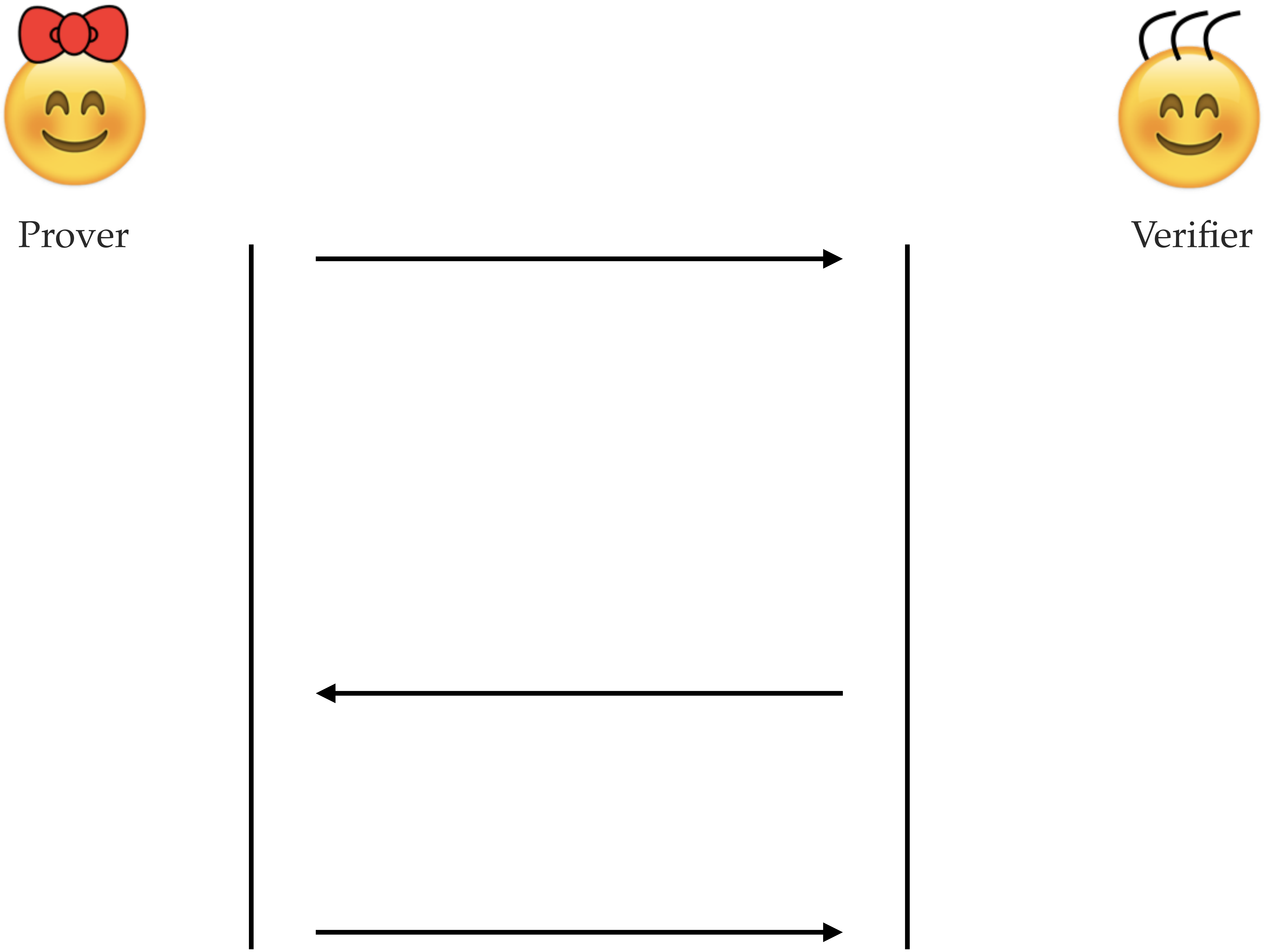
→ First instantiation: the PICNIC family (from symmetric primitives).

MPCitH identification scheme



MPCitH identification scheme

$\boxed{h = g^x}$



MPCitH identification scheme

$\boxed{h = g^x}$

Prover

Verifier

Computes $[[h]]_i = g^{[[x]]_i}$ ‘in his head’.

MPCitH identification scheme

$\boxed{h = g^x}$

Prover

Verifier

$\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$

Computes $[[h]]_i = g^{[[x]]_i}$ ‘in his head’.

MPCitH identification scheme

$\boxed{h = g^x}$

Prover

Verifier

$\text{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

Chooses $c \in [1; N]$.

$\xleftarrow{\;c\;}$

$\longrightarrow$

MPCitH identification scheme

$\boxed{h = g^x}$

Prover

Verifier

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

$$\xrightarrow{\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)}$$

Chooses $c \in [1; N]$.

$$\xleftarrow{c}$$

$$\xrightarrow{[[x]]_i \text{ for all } i \neq c}$$

MPCitH identification scheme

$\boxed{h = g^x}$

Prover

Verifier

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

$\xrightarrow{\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)}$

Chooses $c \in [1; N]$.

$\xleftarrow{c}$

$\xrightarrow{[[x]]_i \text{ for all } i \neq c}$

Checks that $[[h]]_i = g^{[[x]]_i}$ for all i.

MPCitH identification scheme

$\boxed{h = g^x}$

Prover

Verifier

$\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Computes $[[h]]_i = g^{[[x]]_i}$ ‘in his head’.

Chooses $c \in [1; N]$.

$\longleftarrow$ c

$[[x]]_i$ for all $i \neq c$ $\longrightarrow$

Checks that $[[h]]_i = g^{[[x]]_i}$ for all i.

Checks that $\prod_{i=1}^{N} [[h_i]] = h$.

MPCitH identification scheme

$\boxed{h = g^x}$

Prover

Verifier

$\xrightarrow{\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)}$

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

Chooses $c \in [1; N]$.

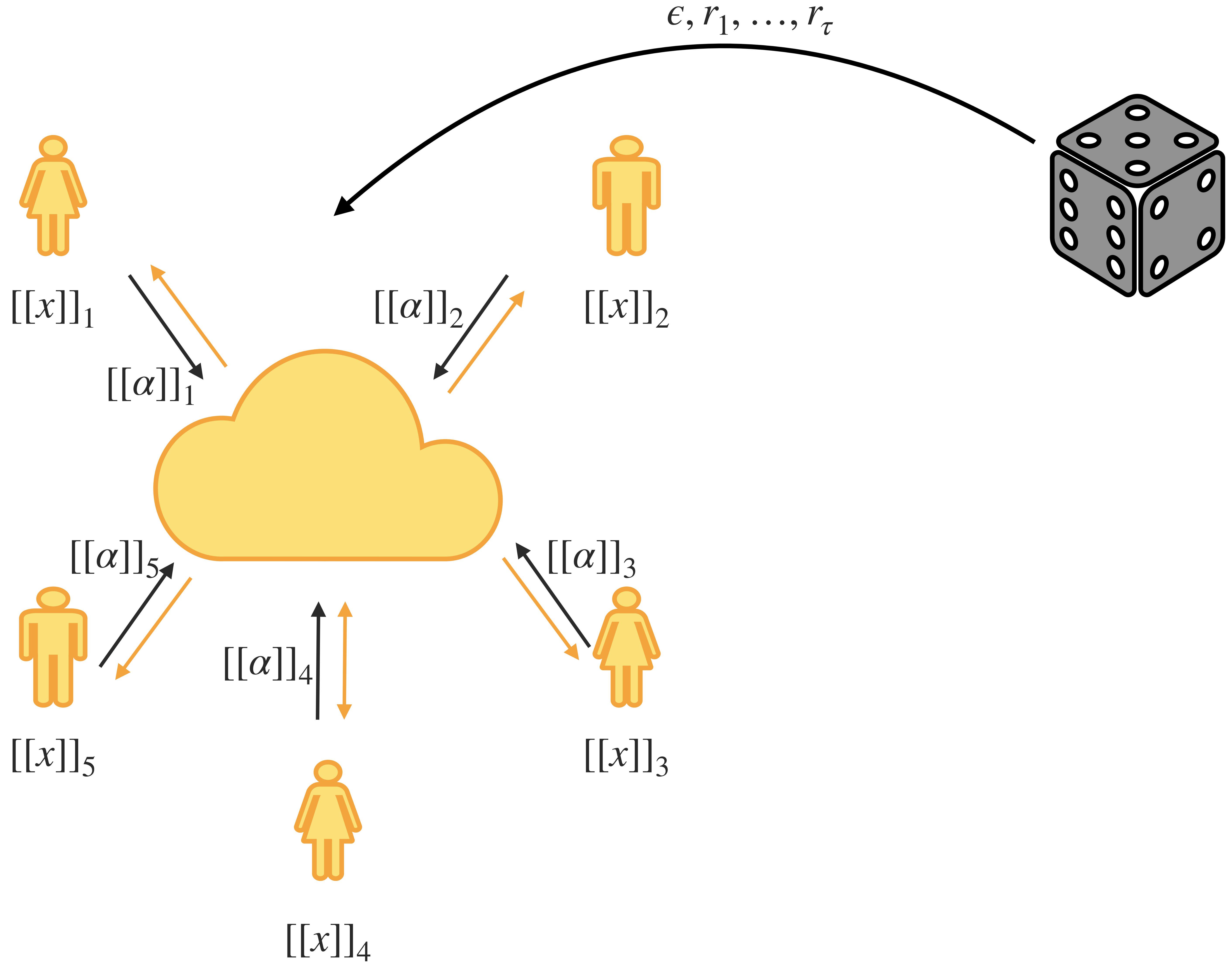
$\xleftarrow{c}$

$\xrightarrow{[[x]]_i \text{ for all } i \neq c}$

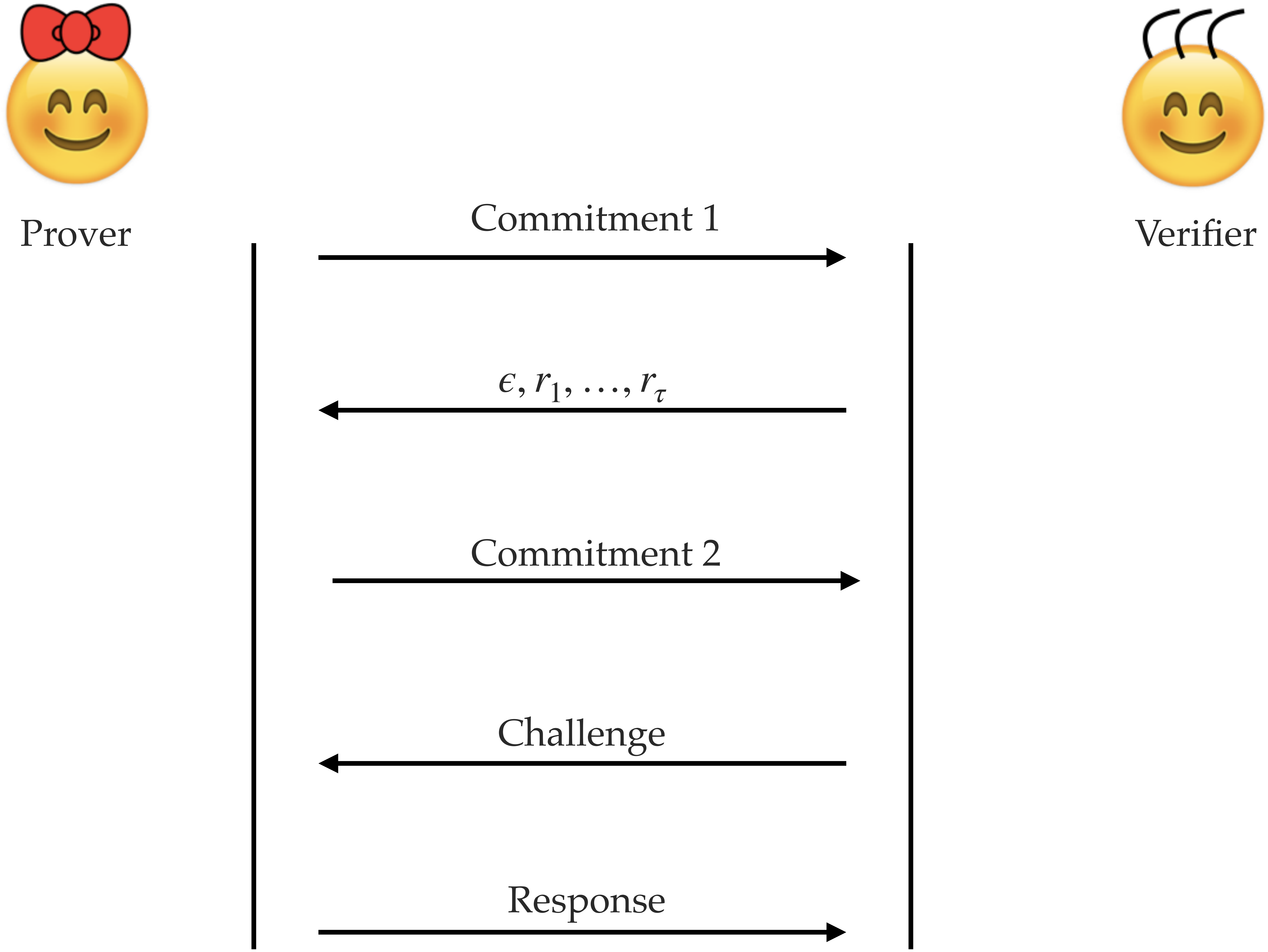
Checks that $[[h]]_i = g^{[[x]]_i}$ for all i.

Checks that $\prod_{i=1}^{N} [[h_i]] = h$. ✓

Broadcast model with oracle



A 5-pass protocol



Security properties

Completeness

If the statement is true, an honest prover is always able to convince an honest verifier.

Soundness

A dishonest prover cannot convince an honest verifier other than with a small probability.

Soundness

A dishonest prover cannot convince an honest verifier other than with a small probability.

2-Special soundness

Having obtained two valid transcripts with the same commitment and a different challenge, we can extract a solution for the underlying problem.

Zero-knowledge

Anyone observing the transcript (including the verifier) learns nothing other than the fact that the statement is true.

Security properties

→ Completeness

Security properties

→ Completeness ✓

Security properties

→ Completeness ✓

→ Soundness

Security properties

→ Completeness ✓

→ Soundness

Two ways of cheating:

→ Guessing an ϵ_i (input from oracle).

→ Guessing the second challenge: $\frac{1}{N}$.

Security properties

→ Completeness ✓

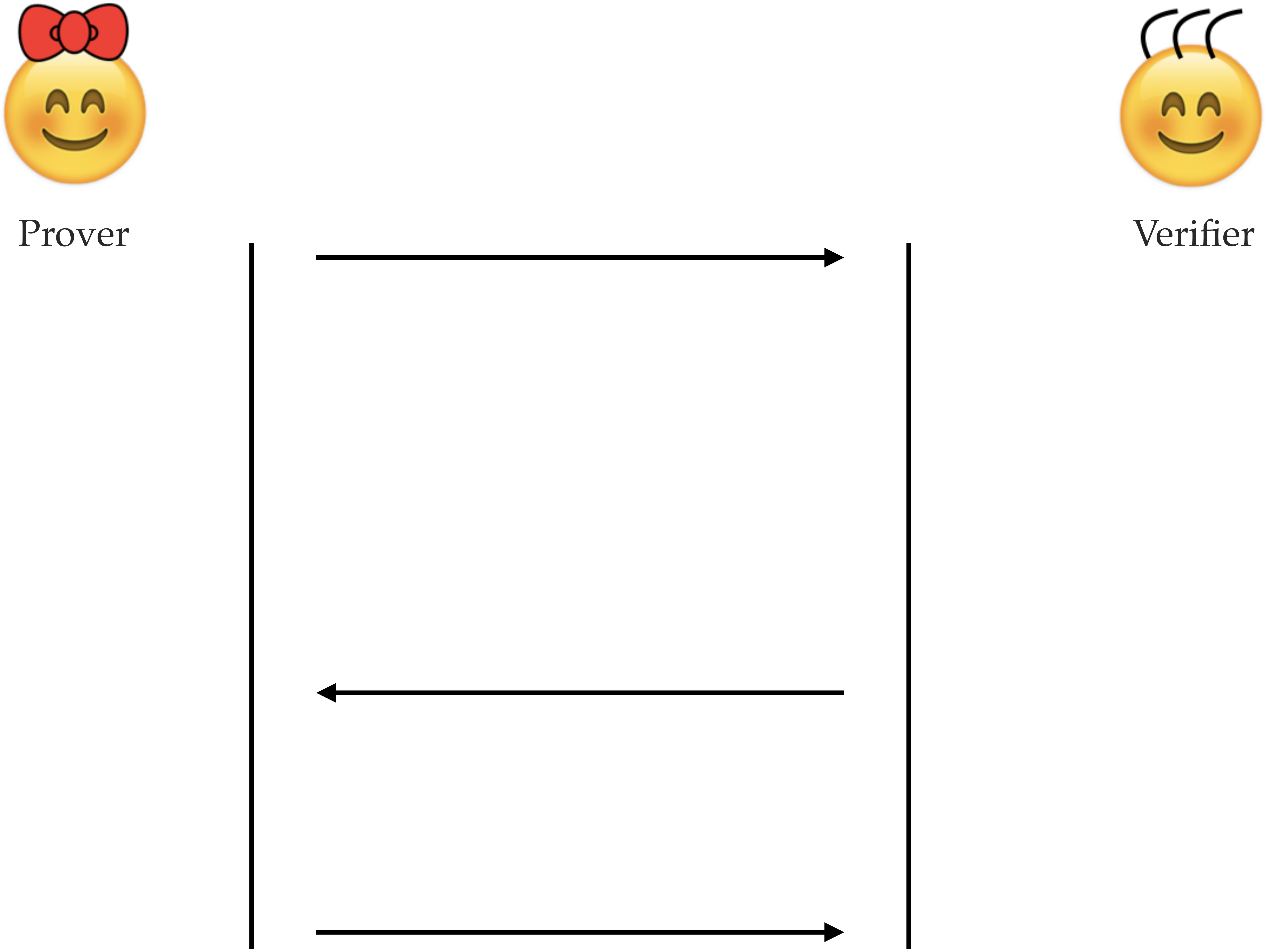
→ Soundness ✓

Two ways of cheating:

- Guessing an ϵ_i (input from oracle).
- Guessing the second challenge: $\frac{1}{N}$.

Malicious prover

$\boxed{h = g^x}$

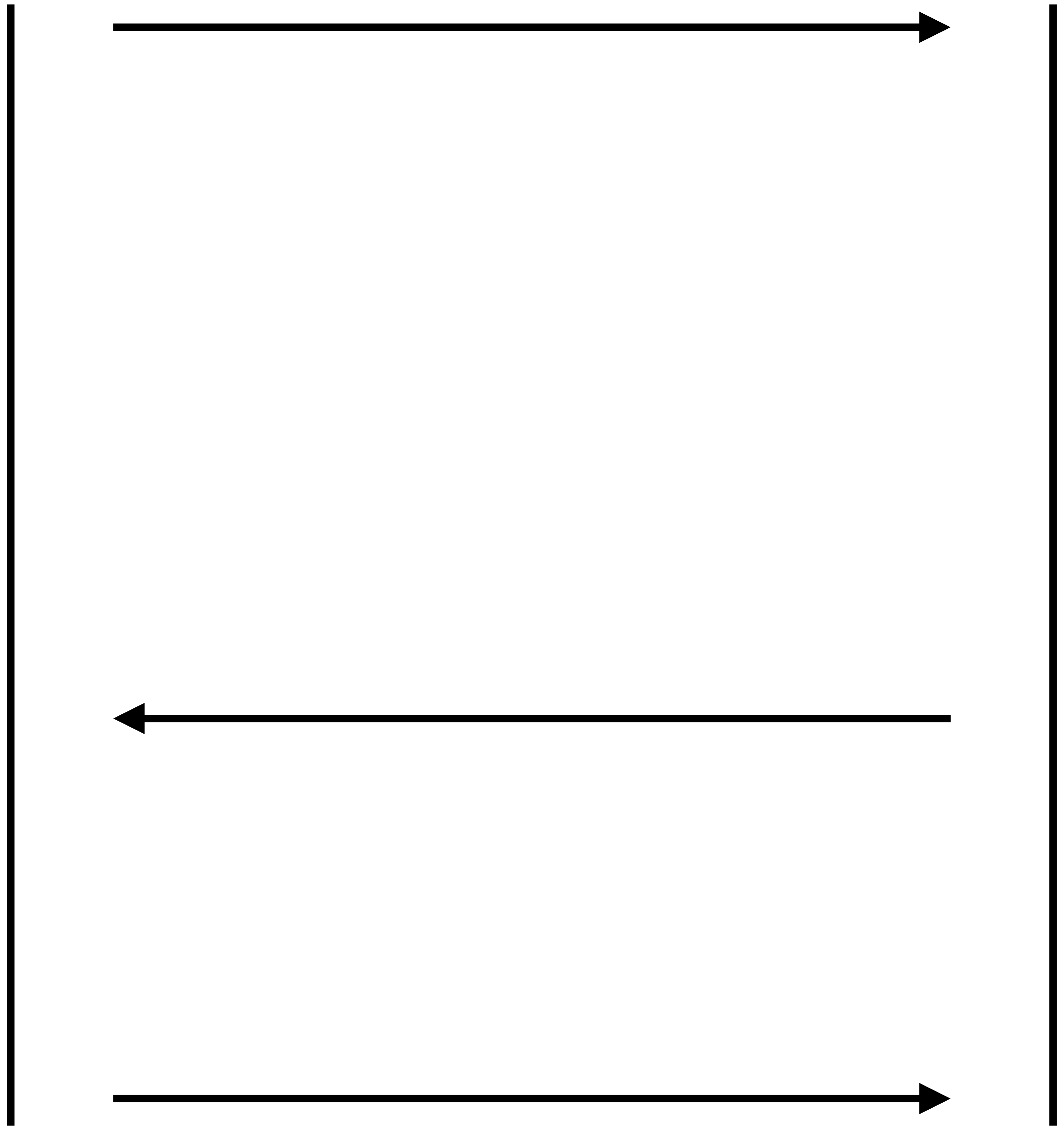


Malicious prover

$\boxed{h = g^x}$

Prover

Verifier



Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

! Not PQ (in this example)

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

$\xrightarrow{\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)}$

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

$\longleftarrow$

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

! Not PQ (in this example)

$\longrightarrow$

Malicious prover

$h = g^x$

Prover

Verifier

$\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Chooses $c \in [1; N]$.

$c \neq N$

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

! Not PQ (in this example)

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

$\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Chooses $c \in [1; N]$.

$\boxed{c \neq N}$ $\longleftarrow$

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

$[[x]]_i$ for all $i \neq c$ $\longrightarrow$

! Not PQ (in this example)

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

$\mathrm{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Chooses $c \in [1; N]$.

$\longleftarrow$ $\boxed{c \neq N}$

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

$[[x]]_i$ for all $i \neq c$ $\longrightarrow$

Checks that $[[h]]_i = g^{[[x]]_i}$ for all i.

! Not PQ (in this example)

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

$\text{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Chooses $c \in [1; N]$.

$\longleftarrow$ $\boxed{c \neq N}$

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

Checks that $[[h]]_i = g^{[[x]]_i}$ for all i. ✓

! Not PQ (in this example)

$[[x]]_i$ for all $i \neq c$ $\longrightarrow$

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

$\text{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Chooses $c \in [1; N]$.

$\longleftarrow$ $\boxed{c \neq N}$

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

! Not PQ (in this example)

$[[x]]_i$ for all $i \neq c$ $\longrightarrow$

Checks that $[[h]]_i = g^{[[x]]_i}$ for all i. ✓

Checks that $\prod_{i=1}^{N} [[h_i]] = h$.

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

$\text{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Chooses $c \in [1; N]$.

$\longleftarrow$ $\boxed{c \neq N}$

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

$[[x]]_i$ for all $i \neq c$ $\longrightarrow$

Checks that $[[h]]_i = g^{[[x]]_i}$ for all i. ✓

Checks that $\prod_{i=1}^{N} [[h_i]] = h$. ✓

! Not PQ (in this example)

Malicious prover

$\boxed{h = g^x}$

Prover

Verifier

$\text{com}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ →

Chooses $[[x]]_i$ for $i \in [1; N-1]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \in [1; N-1]$.

Computes $[[h]]_N = h / \prod_{i=1}^{N-1} [[h]]_i$.

Chooses $c \in [1; N]$.

← $\boxed{c = N}$

Cannot compute $[[x]]_N$ s.t. $[[h]]_N = g^{[[x]]_N}$ because the discrete log is hard.

! Not PQ (in this example)

$[[x]]_i$ for all $i \neq c$ →

Checks that $[[h]]_i = g^{[[x]]_i}$ for all i. ✗

Checks that $\prod_{i=1}^{N} [[h_i]] = h$. ✓

Security properties

→ Completeness ✓

→ Soundness ✓

Two ways of cheating:

- → Guessing an ϵ_i (input from oracle).
- → Guessing the second challenge: $\frac{1}{N}$.

Security properties

→ Completeness ✓

→ Soundness ✓

Two ways of cheating:

→ Guessing an ϵ_i (input from oracle).

→ Guessing the second challenge: $\frac{1}{N}$.

→ Zero-knowledge

Security properties

→ Completeness ✓

→ Soundness ✓

Two ways of cheating:

→ Guessing an ϵ_i (input from oracle).

→ Guessing the second challenge: $\frac{1}{N}$.

→ Zero-knowledge

As a result of the $(N-1)$-private property of the underlying MPC protocol.

Security properties

→ Completeness ✓

→ Soundness ✓

Two ways of cheating:

→ Guessing an ϵ_i (input from oracle).

→ Guessing the second challenge: $\dfrac{1}{N}$.

→ Zero-knowledge ✓

As a result of the $(N-1)$-private property of the underlying MPC protocol.

The MPCitH construction

One-way function
$f(x) = y$

Zero-knowledge proof of knowledge: the prover needs to prove that they know a secret x s.t. $f(x) = y$, for a public y, without revealing any information about x.

MPC protocol

Multi-party computation

MPCitH identification scheme

MPC-in-the-Head

via the Fiat-Shamir transform

Digital signature

The Fiat-Shamir transform

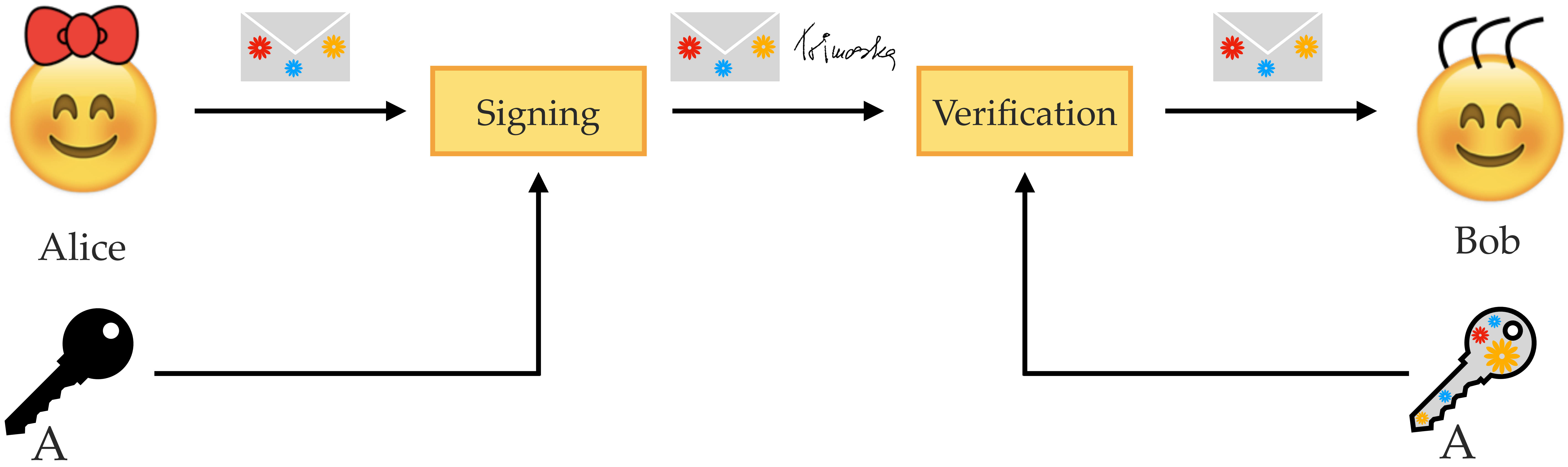
The goal is to transform an interactive identification scheme into a digital signature scheme.

Instead of the prover choosing a challenge, the challenge is determined by the hash of the message and commitments.

The Fiat-Shamir transform

The goal is to transform an interactive identification scheme into a digital signature scheme.

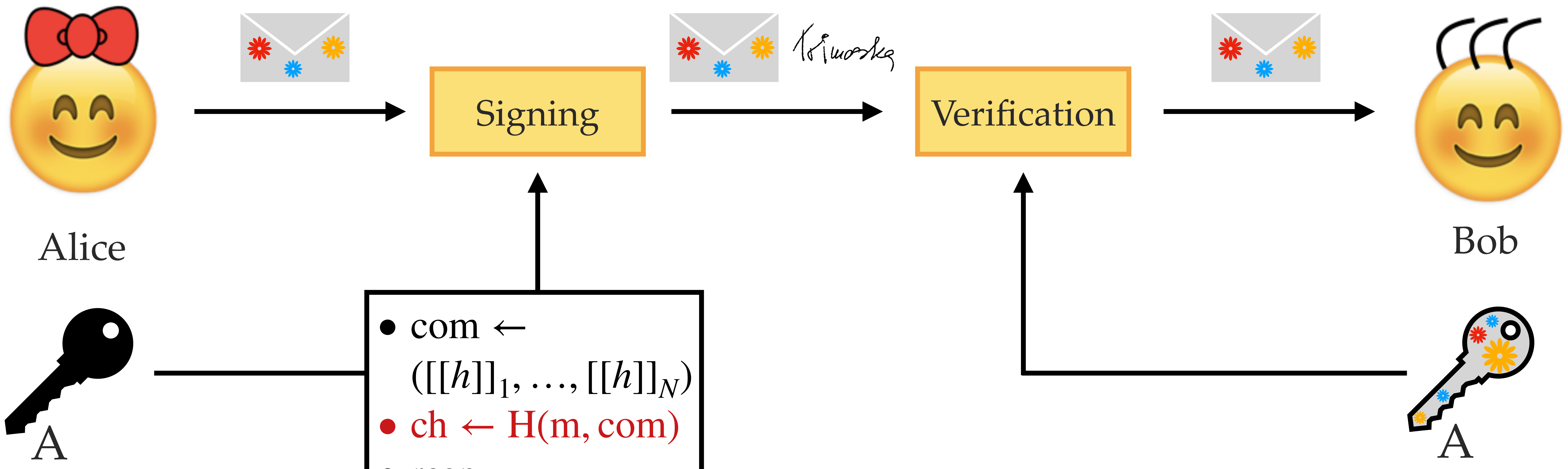
Instead of the prover choosing a challenge, the challenge is determined by the hash of the message and commitments.



The Fiat-Shamir transform

The goal is to transform an interactive identification scheme into a digital signature scheme.

Instead of the prover choosing a challenge, the challenge is determined by the hash of the message and commitments.

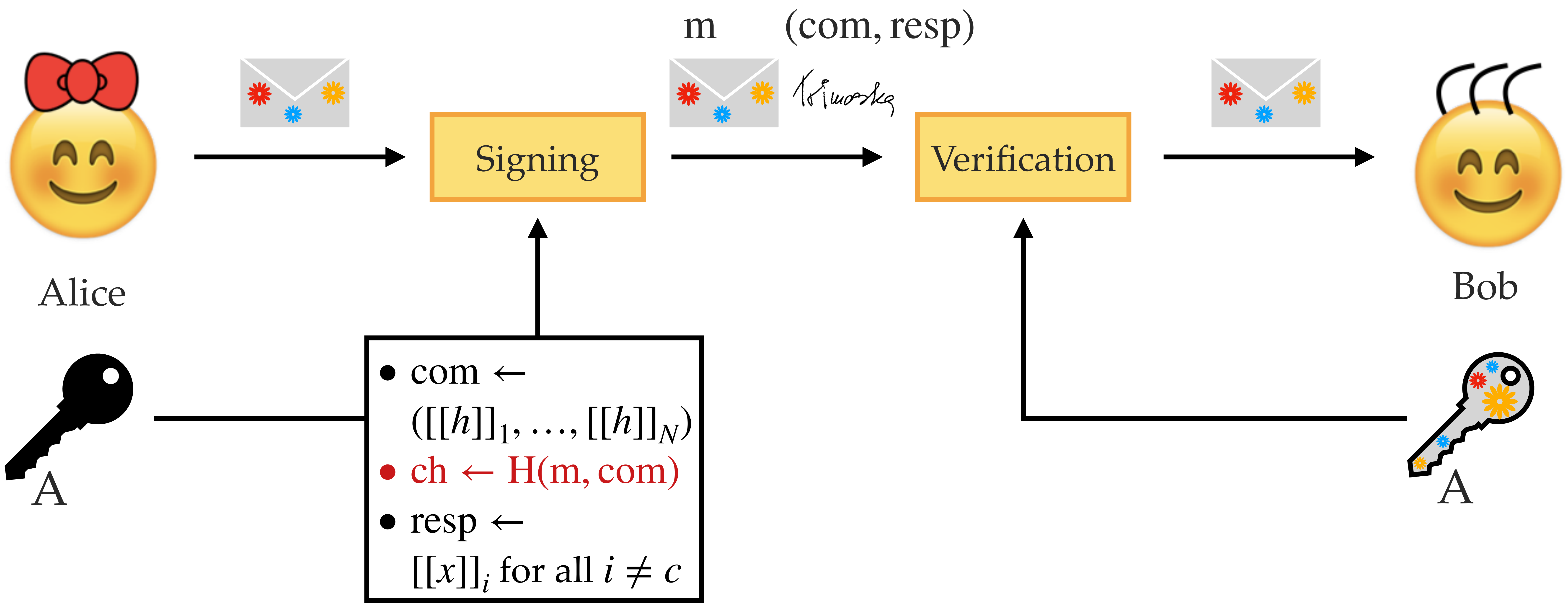


- com ← $([[h]]_1, \ldots, [[h]]_N)$
- ch ← H(m, com)
- resp ← $[[x]]_i$ for all $i \neq c$

The Fiat-Shamir transform

The goal is to transform an interactive identification scheme into a digital signature scheme.

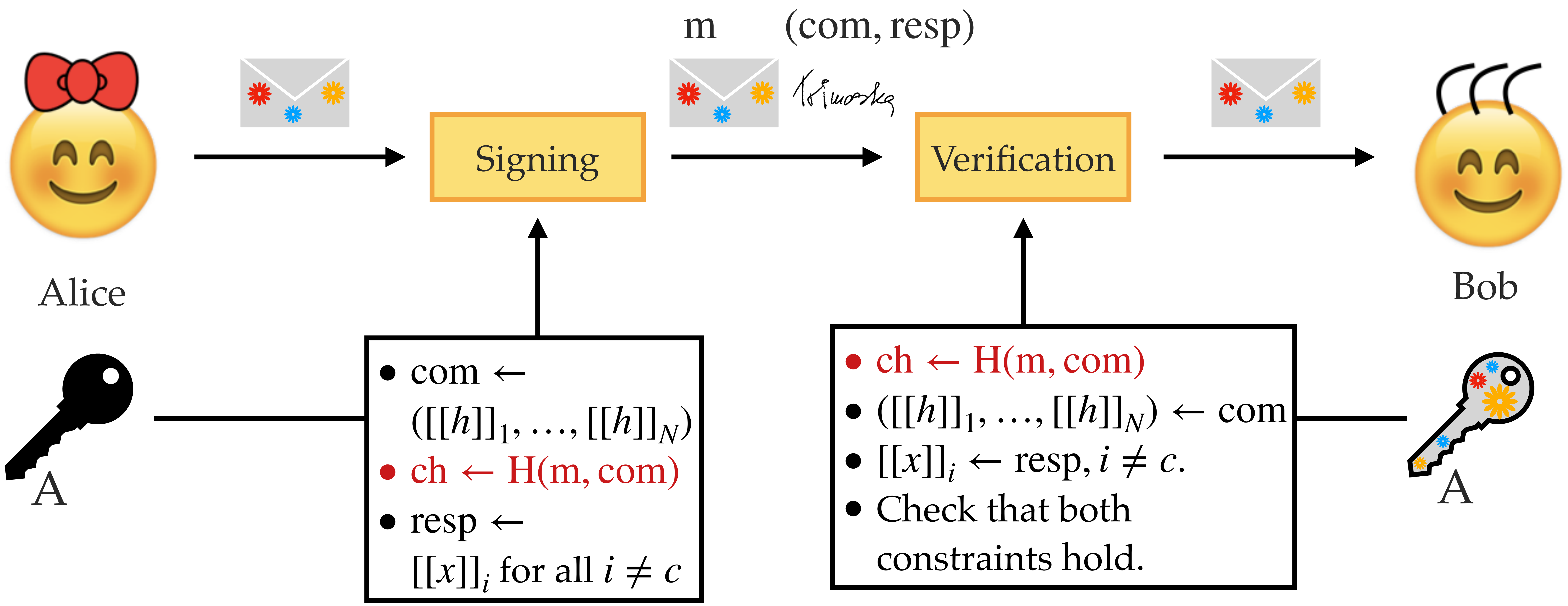
Instead of the prover choosing a challenge, the challenge is determined by the hash of the message and commitments.



The Fiat-Shamir transform

The goal is to transform an interactive identification scheme into a digital signature scheme.

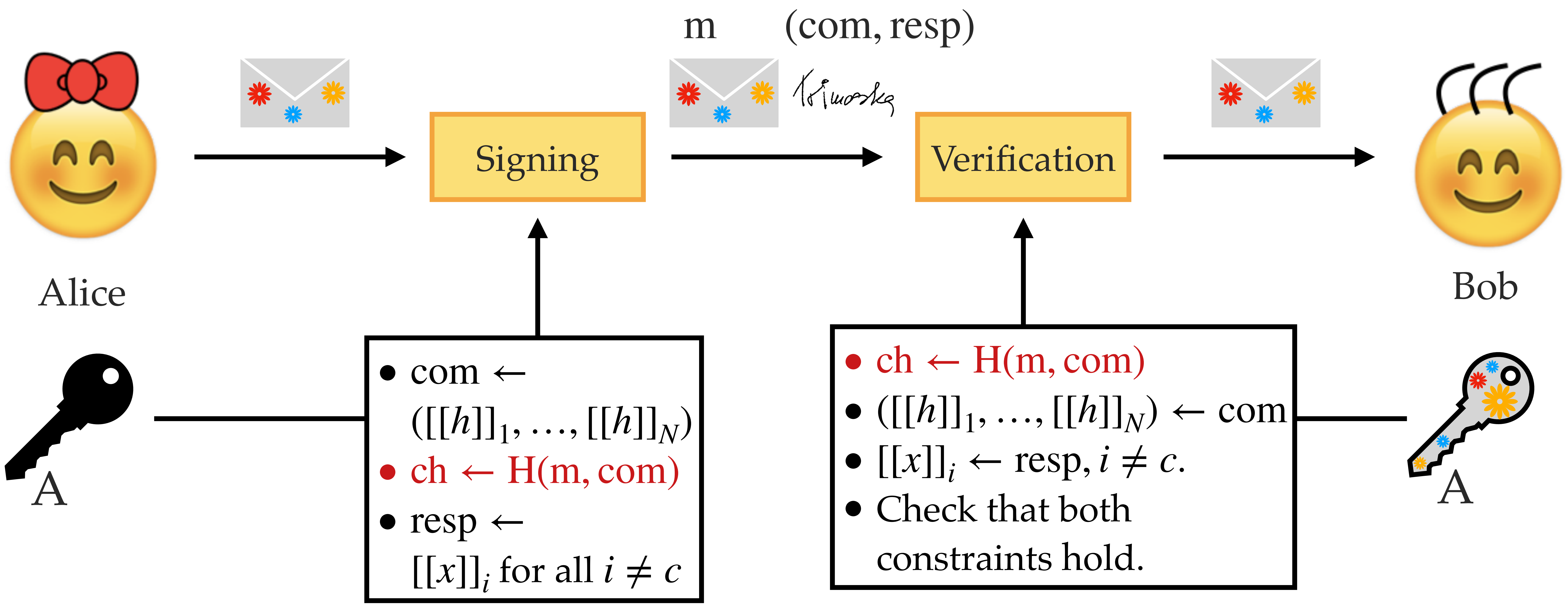
Instead of the prover choosing a challenge, the challenge is determined by the hash of the message and commitments.



The Fiat-Shamir transform

The goal is to transform an interactive identification scheme into a digital signature scheme.

Instead of the prover choosing a challenge, the challenge is determined by the hash of the message and commitments.



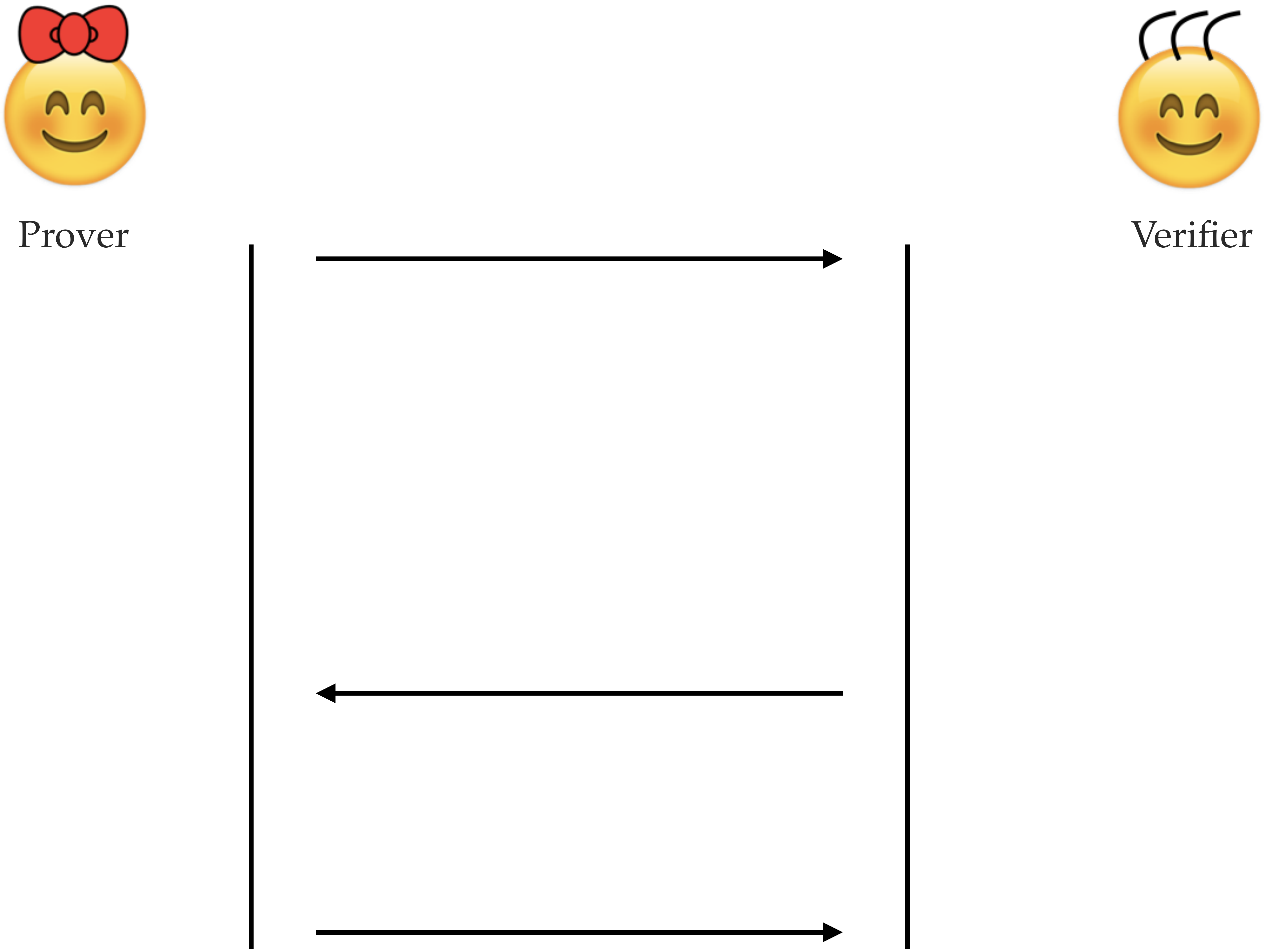
example: using the discrete log; showing a 3-pass (Sigma) protocol;

Optimizations

Optimizations

- Hashing the commitments
- Seed tree
- Hypercube
- Threshold

Hashing the commitments



Hashing the commitments

Prover

Verifier

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

Hashing the commitments

Prover

Verifier

$H = \mathrm{Hash}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

Hashing the commitments

Prover

Verifier

$H = \mathrm{Hash}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

Chooses $c \in [1; N]$.

$\xleftarrow{\;c\;}$

$\longrightarrow$

Hashing the commitments

Prover

Verifier

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

$H = \text{Hash}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ →

Chooses $c \in [1; N]$.

← c

$[[x]]_i$ for all $i \neq c$ →

Hashing the commitments

Prover

Verifier

$H = \mathrm{Hash}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ $\longrightarrow$

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

Chooses $c \in [1; N]$.

$\longleftarrow$ c

Computes $H' = \mathrm{Hash}([[h]]_1, \ldots, [[h]]_N)$.

$[[x]]_i$ for all $i \neq c$ $\longrightarrow$

Hashing the commitments

Prover

Verifier

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

$H = \text{Hash}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ →

Chooses $c \in [1; N]$.

← c

$[[x]]_i$ for all $i \neq c$ →

Computes $H' = \text{Hash}([[h]]_1, \ldots, [[h]]_N)$.

Checks that $H = H'$.

Hashing the commitments

Prover

Verifier

$H = \text{Hash}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ →

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

Chooses $c \in [1; N]$.

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \neq c$.

← c

Computes $H' = \text{Hash}([[h]]_1, \ldots, [[h]]_N)$.

Checks that $H = H'$.

$[[x]]_i$ for all $i \neq c$ →

Hashing the commitments

Prover

Verifier

Computes $[[h]]_i = g^{[[x]]_i}$ 'in his head'.

$H = \mathrm{Hash}([[h]]_1, [[h]]_2, \ldots, [[h]]_N)$ →

Chooses $c \in [1; N]$.

← c

$[[x]]_i$ for all $i \neq c$ →

Computes $[[h]]_i = g^{[[x]]_i}$ for $i \neq c$.

Computes $[[h]]_c = h / \prod_{i=1}^{N} [[h_i]]$, for $i \neq c$.

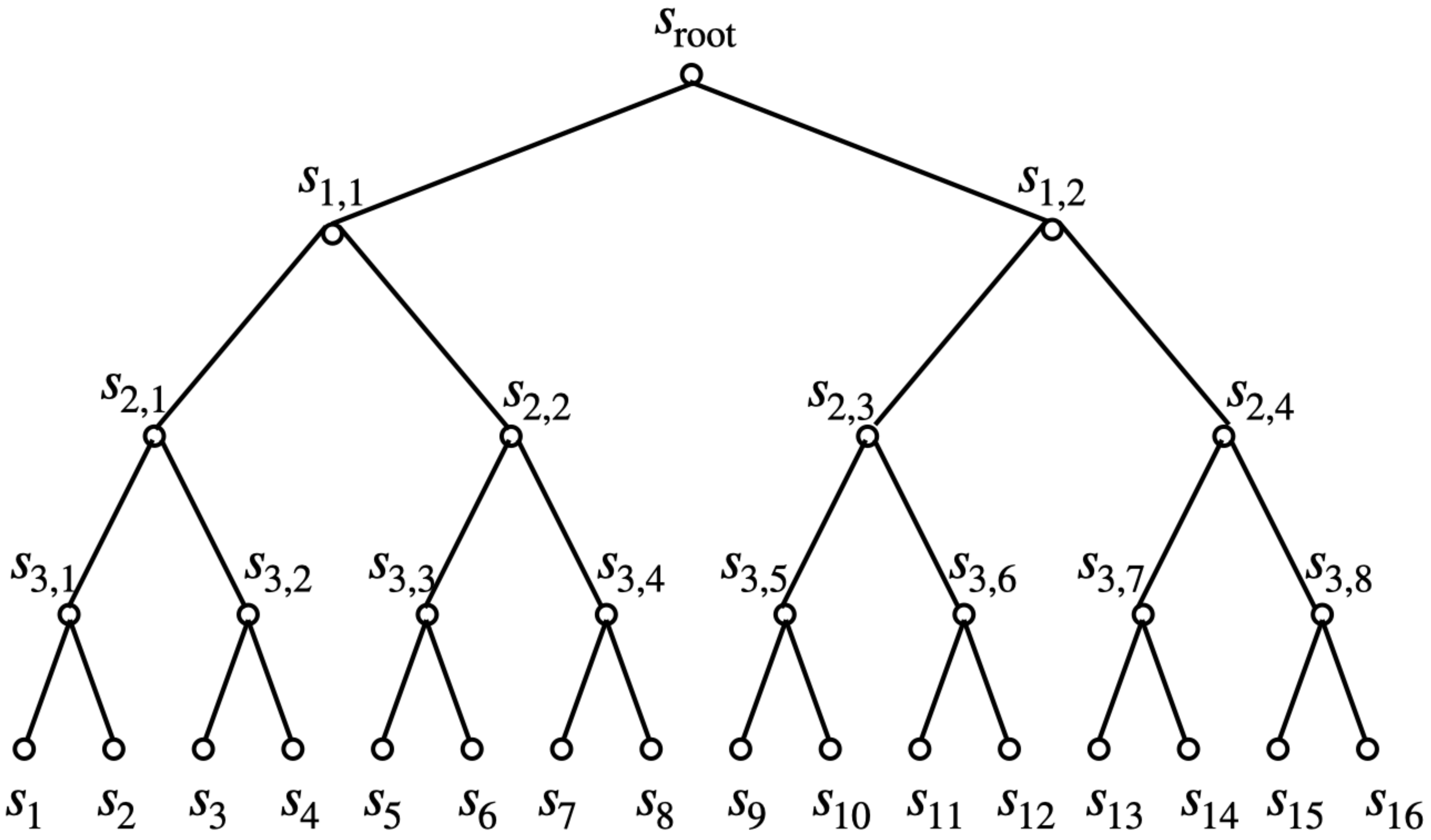
Computes $H' = \mathrm{Hash}([[h]]_1, \ldots, [[h]]_N)$.

Checks that $H = H'$.

Seed tree

Let $N = 2^t$.

We expand $[[x]]_1, \ldots, [[x]]_N$ from a root seed (adjust $[[x]]_N$ so that we get a sharing of x).

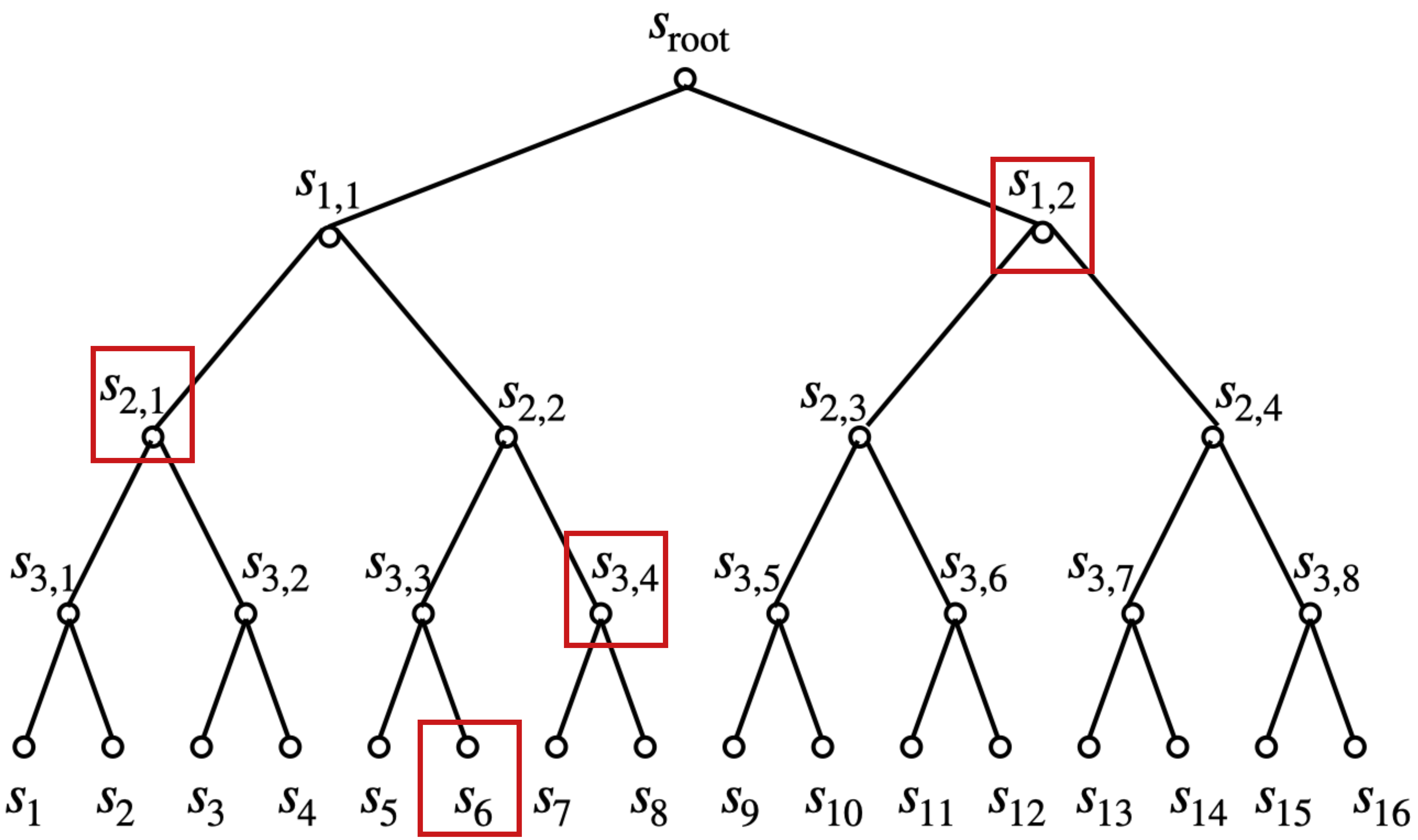


Seed tree

Let $N = 2^t$.

We expand $[[x]]_1, \ldots, [[x]]_N$ from a root seed (adjust $[[x]]_N$ so that we get a sharing of x).

$\longrightarrow c = 5$



Hypercube

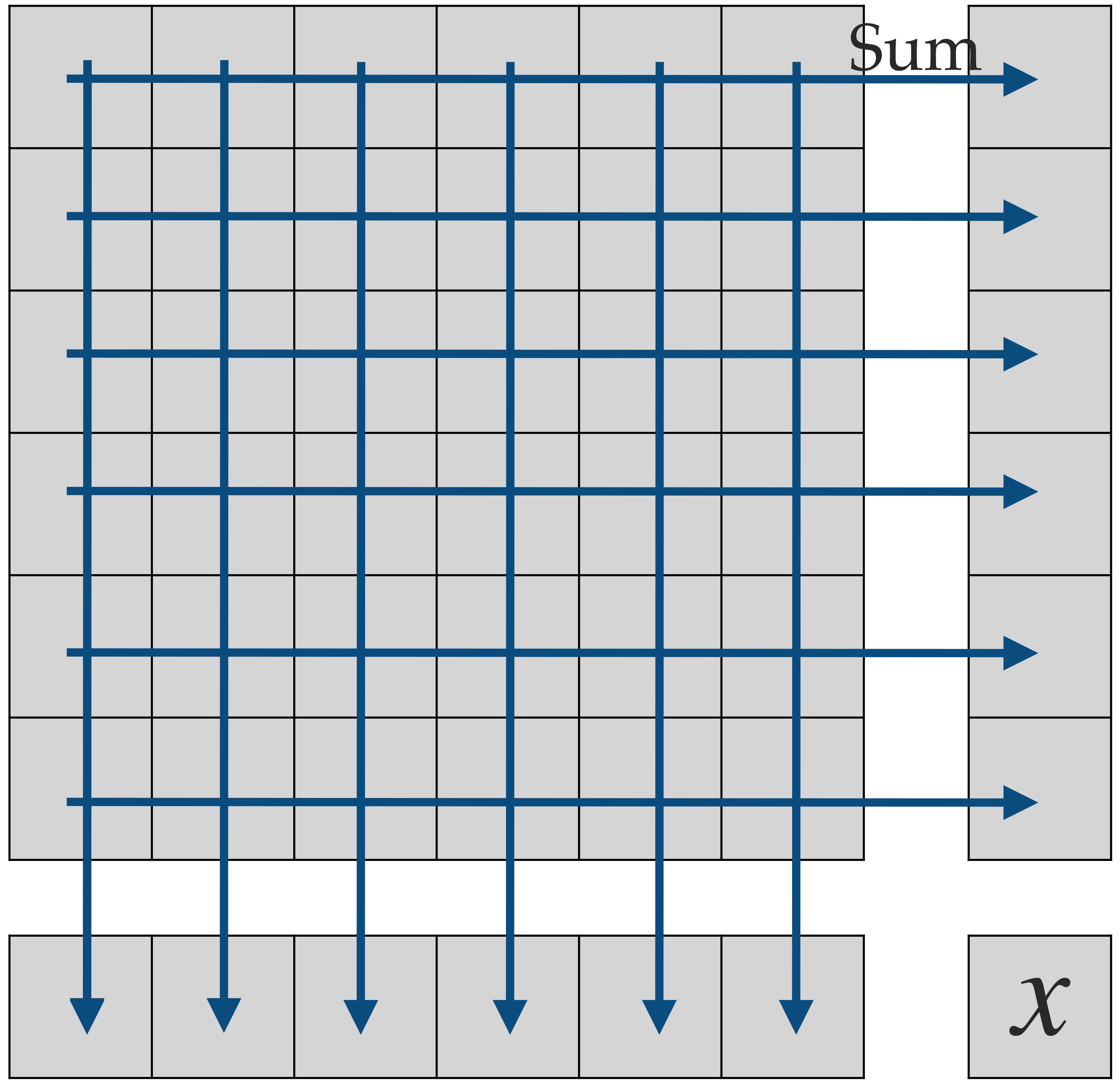
A technique to turn one MPC instance (simulation) of N^D into D instances of N parties.

Hypercube

A technique to turn one MPC instance (simulation) of N^D into D instances of N parties.

Hypercube

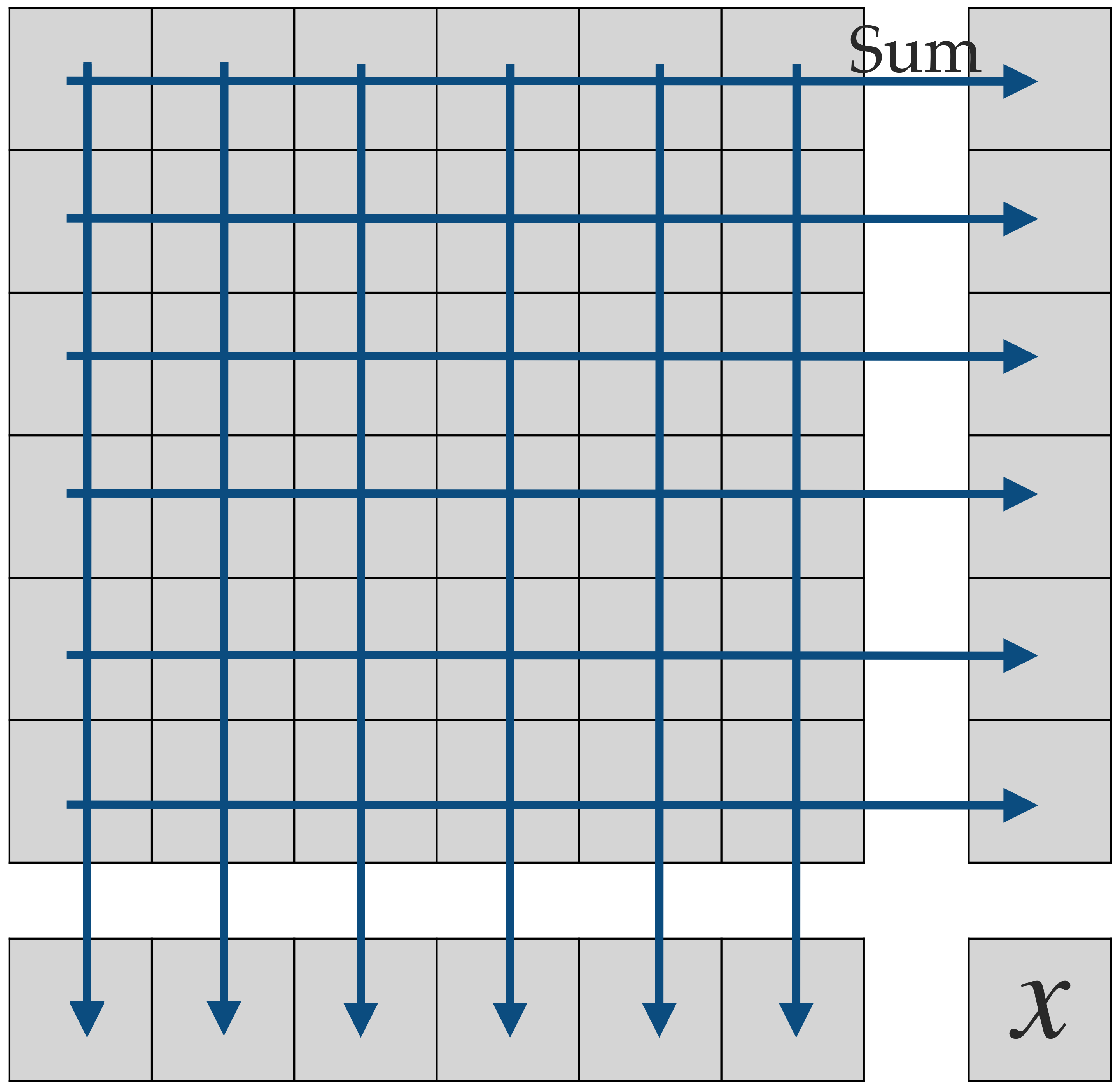
A technique to turn one MPC instance (simulation) of N^D into D instances of N parties.



Hypercube

A technique to turn one MPC instance (simulation) of N^D into D instances of N parties.

<u>MPC 1:</u>
A $\sqrt{N}$-party protocol with shares corresponding the sums of column entries.

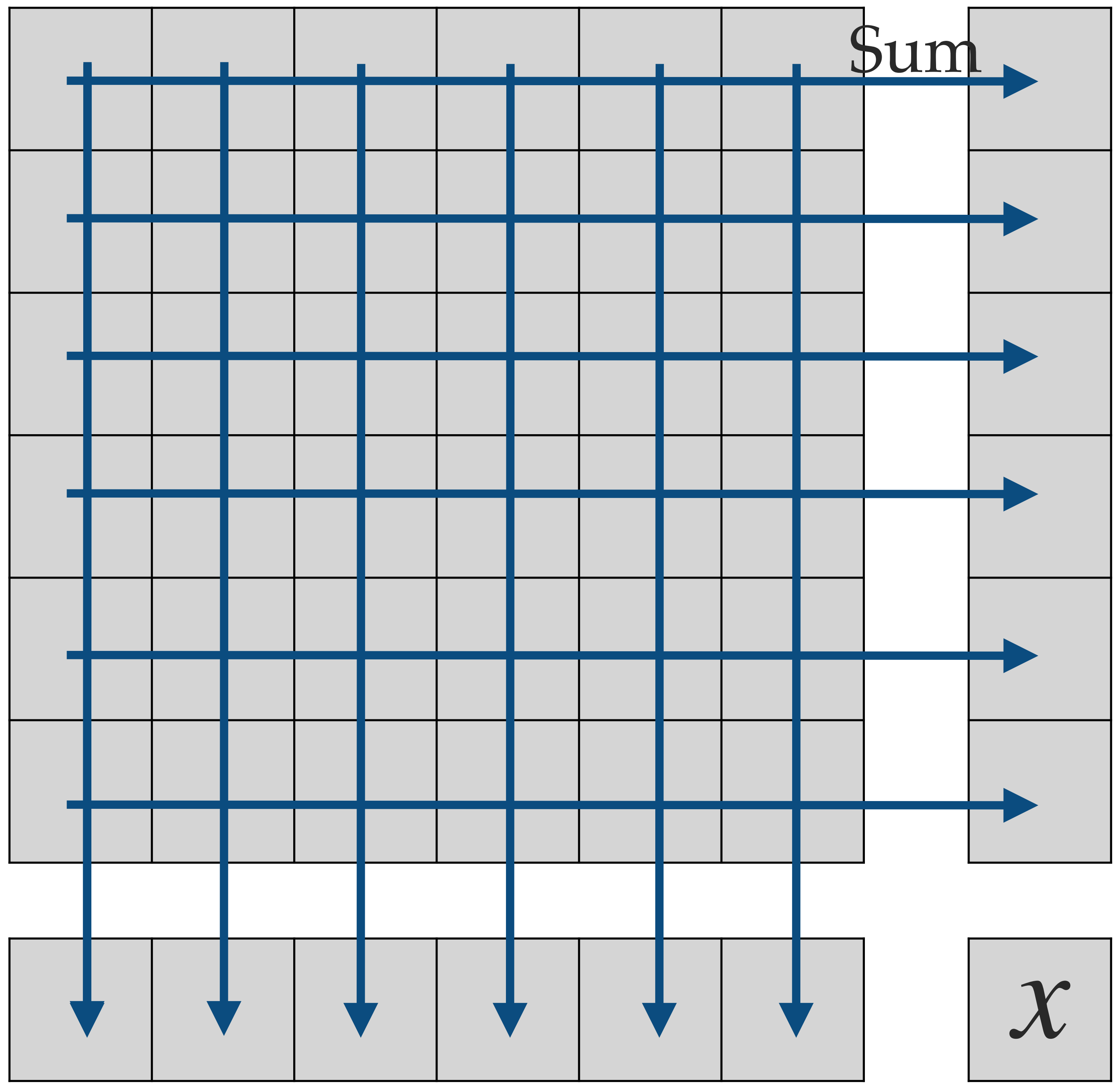


Hypercube

A technique to turn one MPC instance (simulation) of N^D into D instances of N parties.

<u>MPC 1:</u>
A $\sqrt{N}$-party protocol with shares corresponding the sums of column entries.

<u>MPC 2:</u>
A $\sqrt{N}$-party protocol with shares corresponding the sums of row entries.



Hypercube

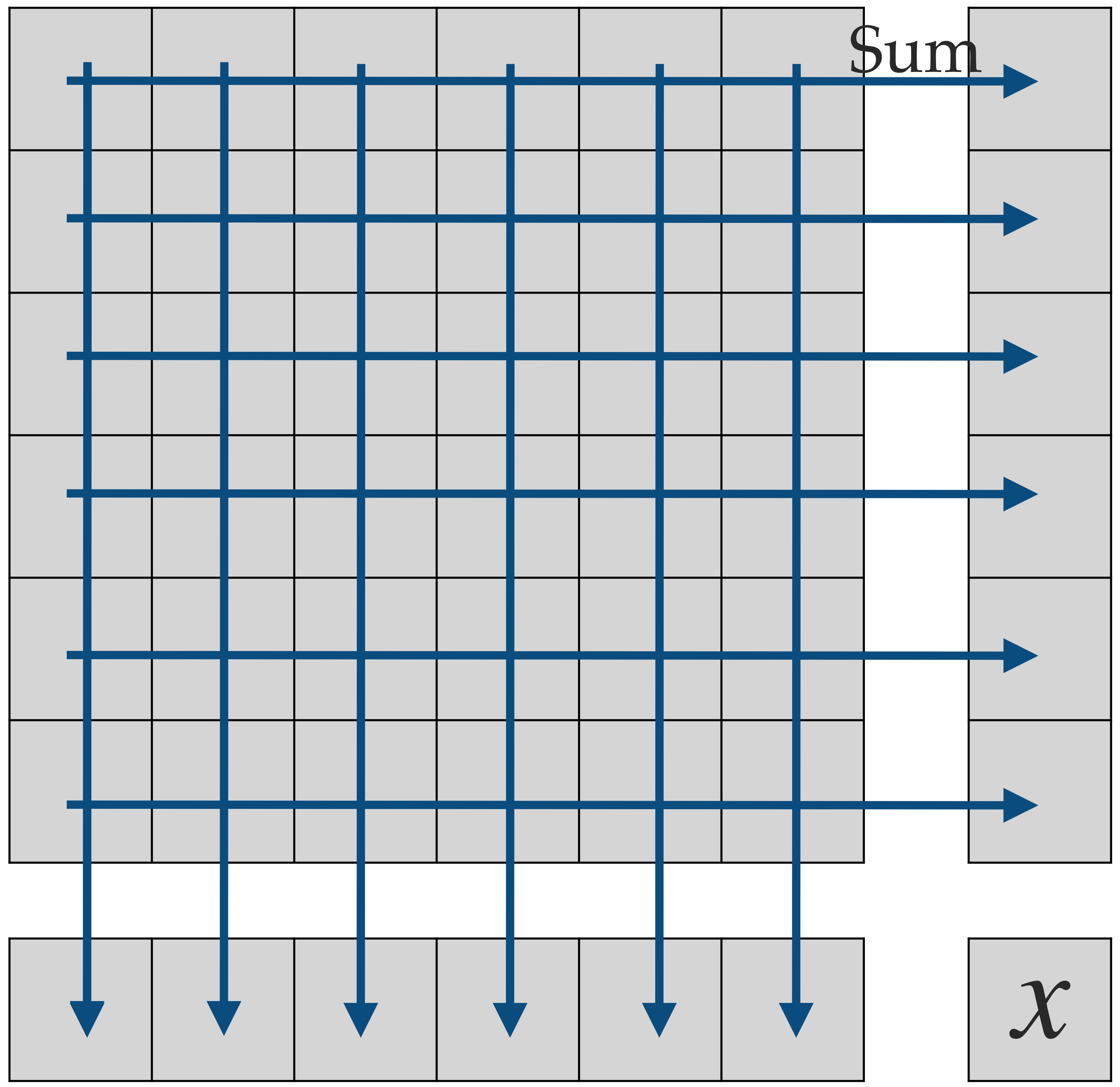
A technique to turn one MPC instance (simulation) of N^D into D instances of N parties.

MPC 1:
A $\sqrt{N}$-party protocol with shares corresponding the sums of column entries.

MPC 2:
A $\sqrt{N}$-party protocol with shares corresponding the sums of row entries.

Probability of cheating in both protocols:

$$\left(\frac{1}{\sqrt{N}}\right)^2$$

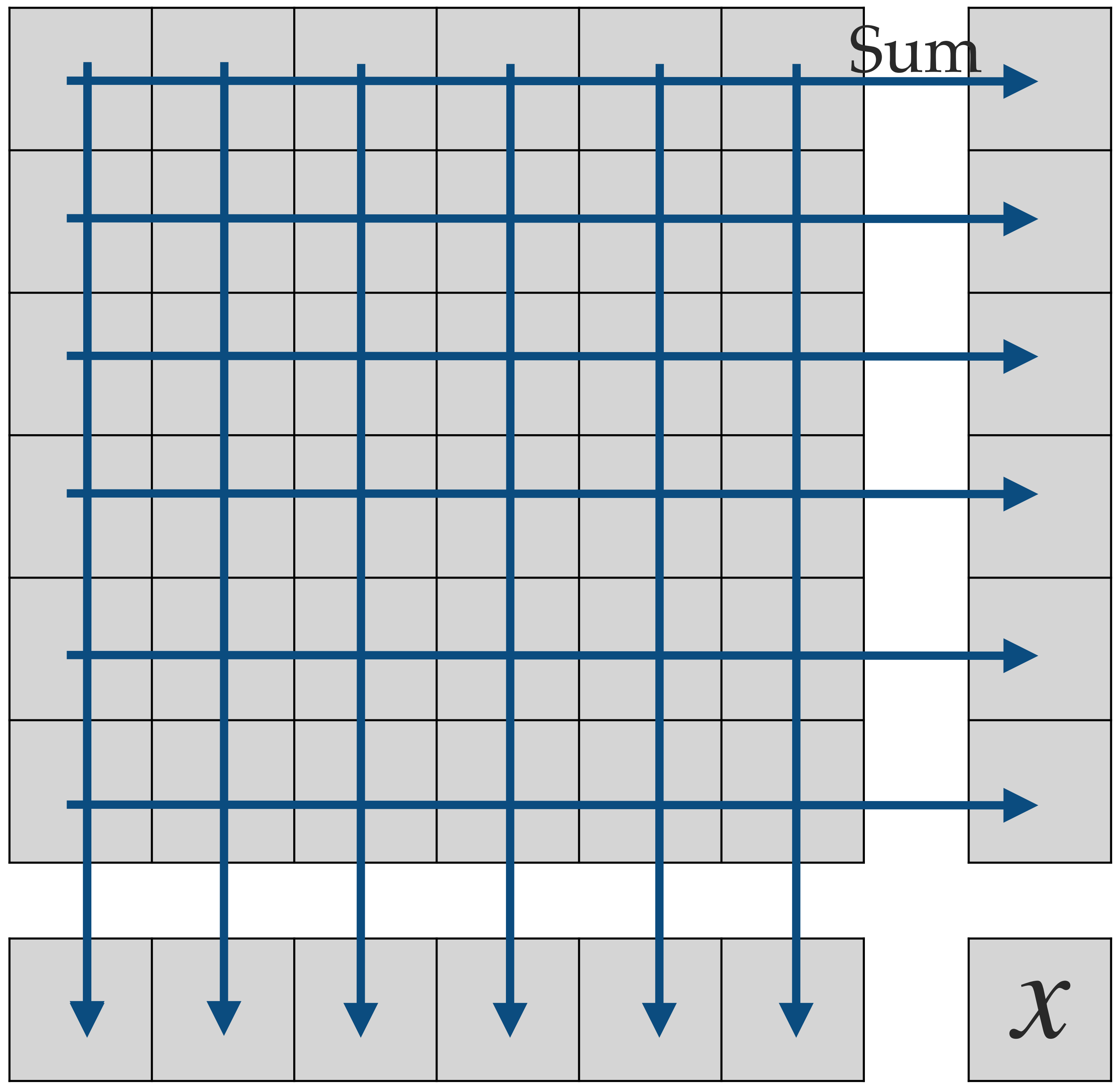


Hypercube

A technique to turn one MPC instance (simulation) of N^D into D instances of N parties.

MPC 1:
A $\sqrt{N}$-party protocol with shares corresponding the sums of column entries.

MPC 2:
A $\sqrt{N}$-party protocol with shares corresponding the sums of row entries.



→ Probability of cheating in both protocols:

$$\left(\frac{1}{\sqrt{N}}\right)^2$$

→ $\frac{1}{N}$ (same as before!)

Hypercube

Going to higher dimensions

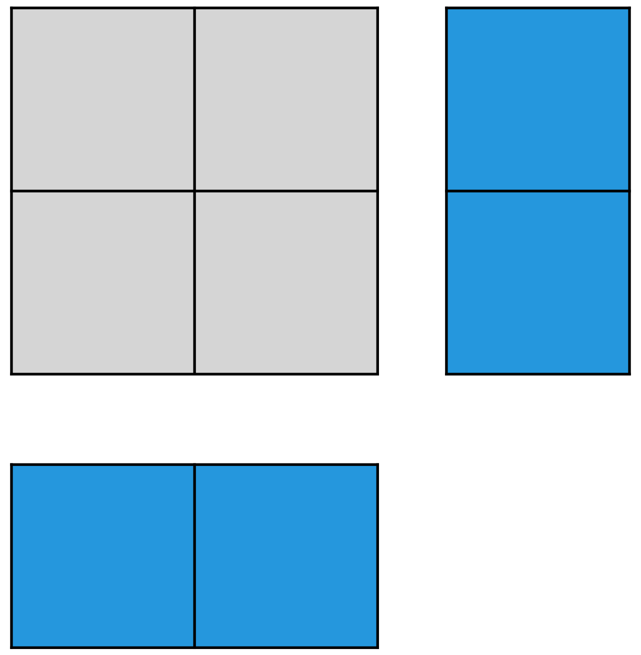
→ Best trade-off: take $N = 2$.

Hypercube

Going to higher dimensions

→ Best trade-off: take $N = 2$.

Example. $D = 2$

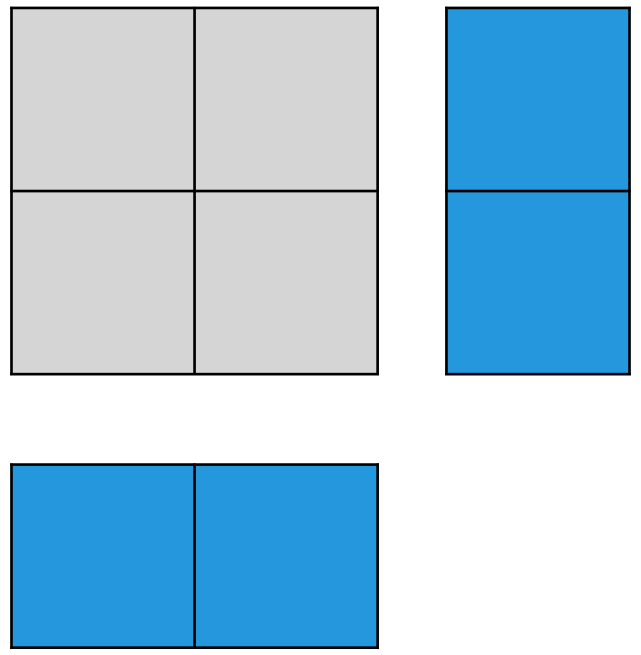


Hypercube

Going to higher dimensions

⟶ Best trade-off: take $N = 2$.

Example. $D = 2$



No difference between the two approaches.

Hypercube

Going to higher dimensions

Which shares participate in which MPC instance??

For k in $[1 \dots D]$
  MPC k:
  For j in [1,2]
    Take shares whose k-th coordinate is j.

Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 3$

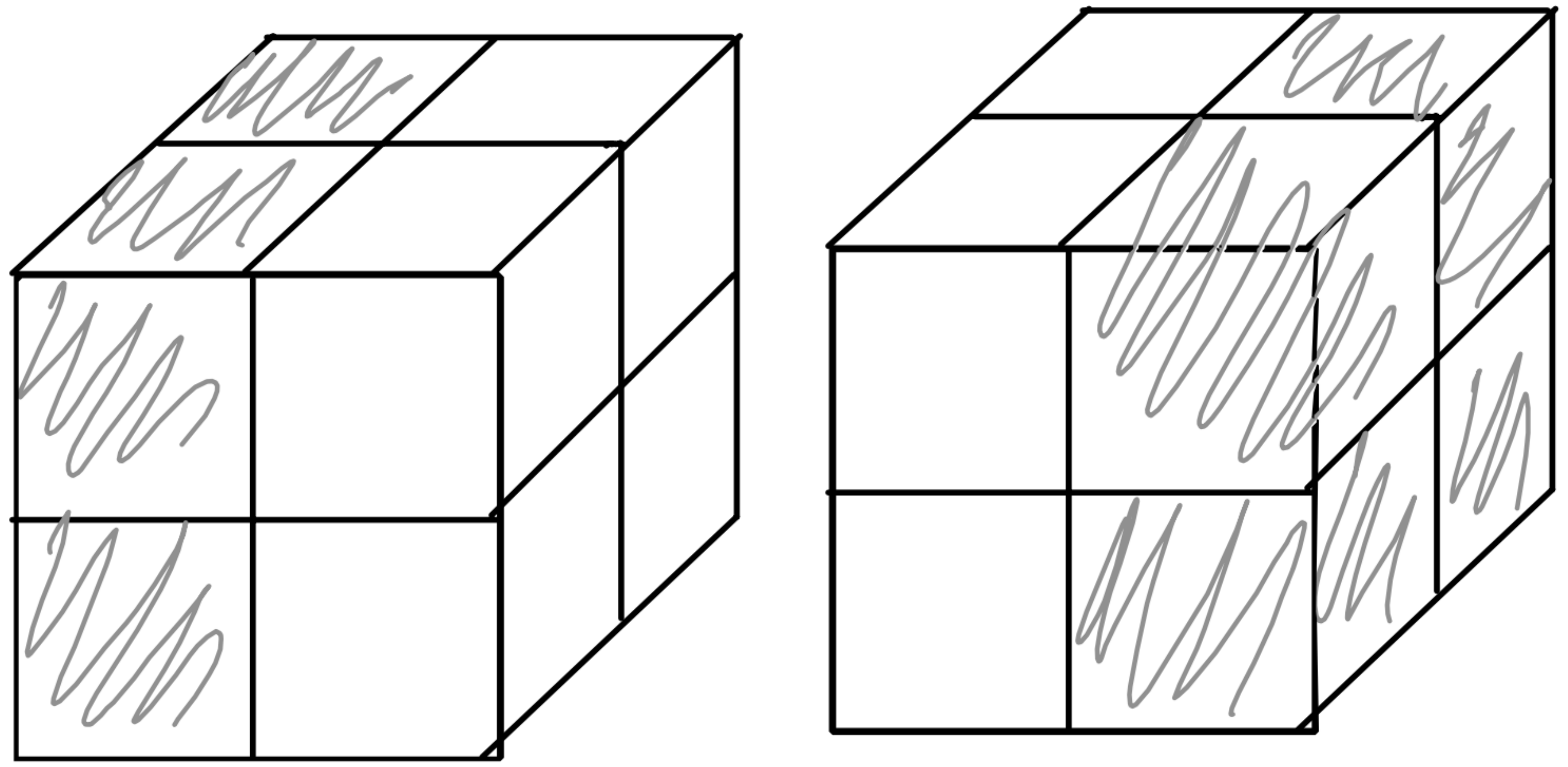
Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 3$

<u>MPC 1</u>



(1,1) (1,2)

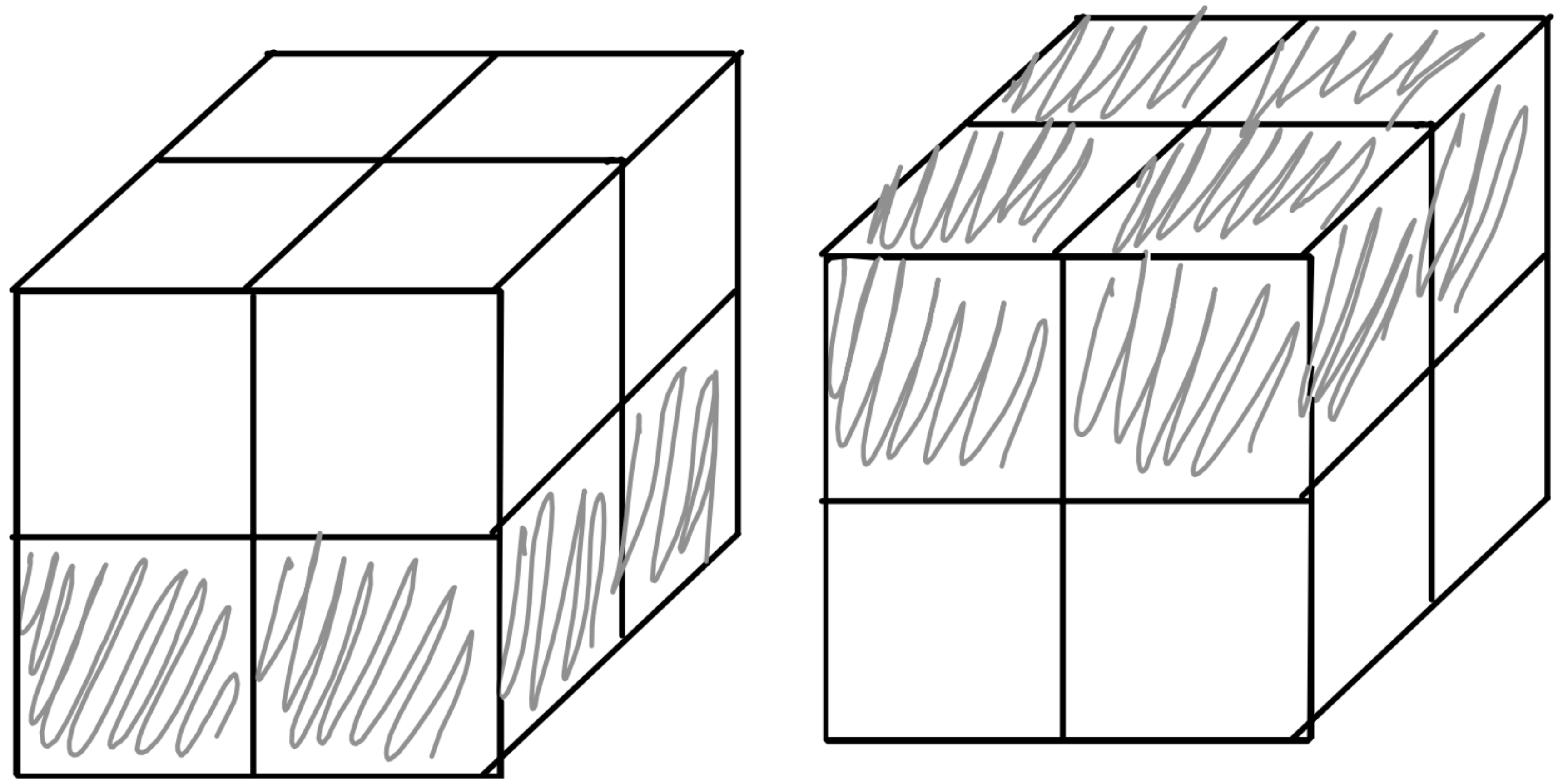
Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

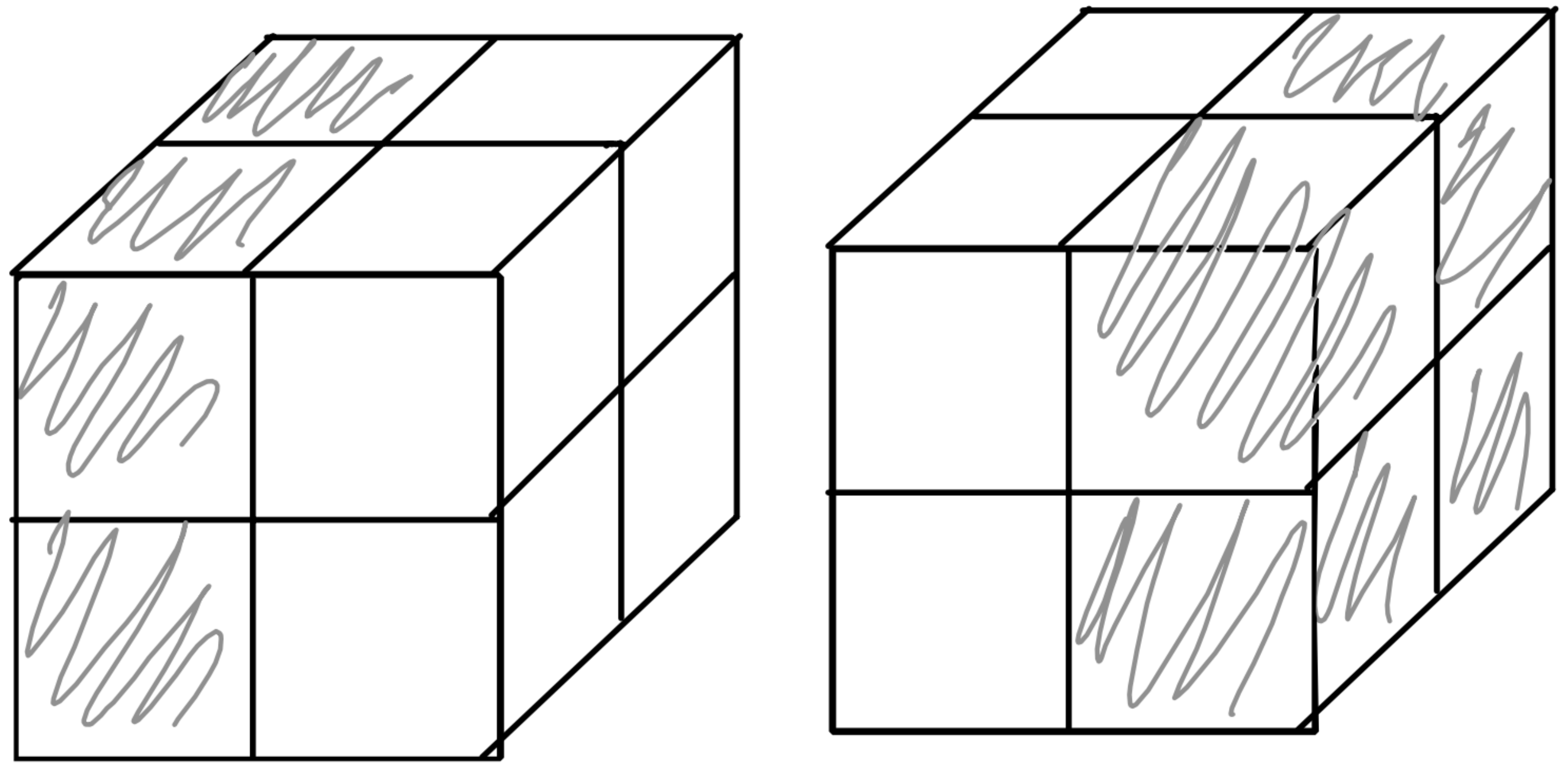
Example. $D = 3$

MPC 2



(2,1) (2,2)

MPC 1



(1,1) (1,2)

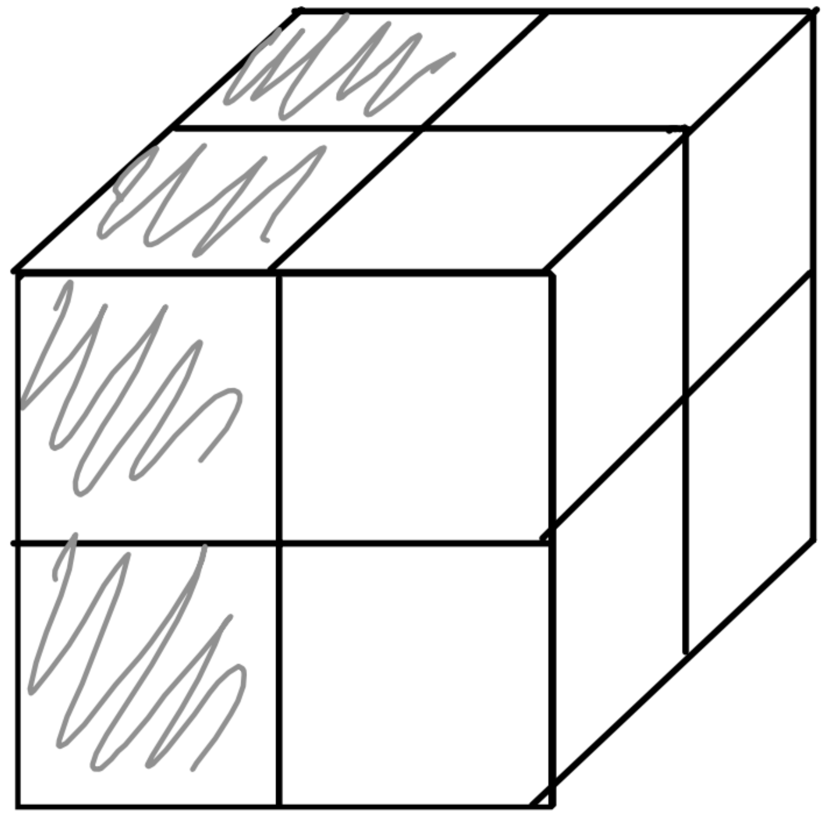
Hypercube

Take shares whose k-th coordinate is j.

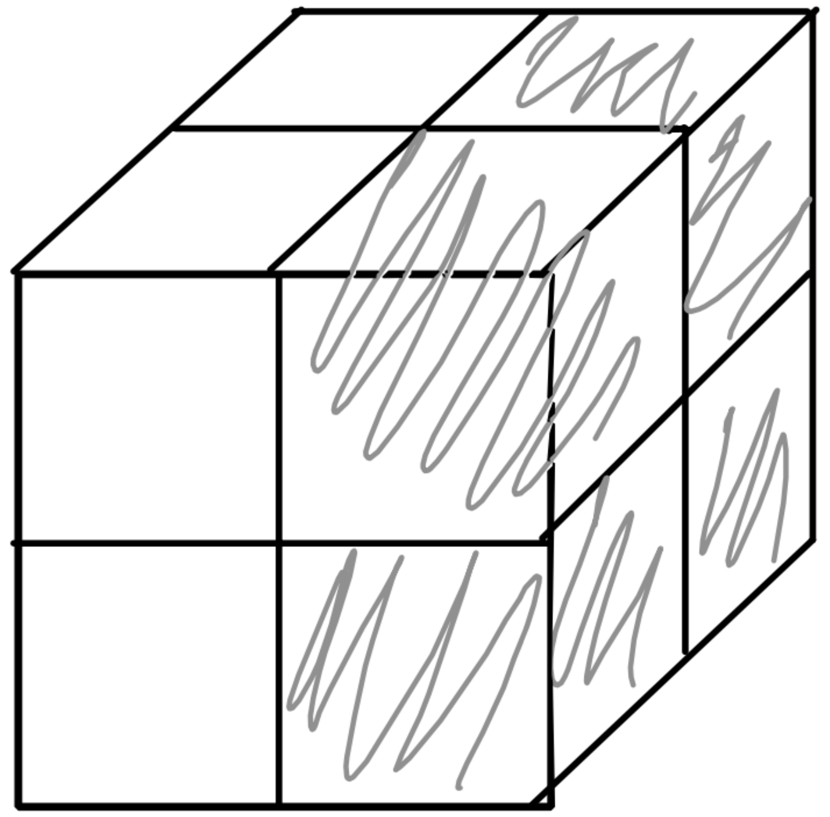
Going to higher dimensions

Example. $D = 3$

MPC 1

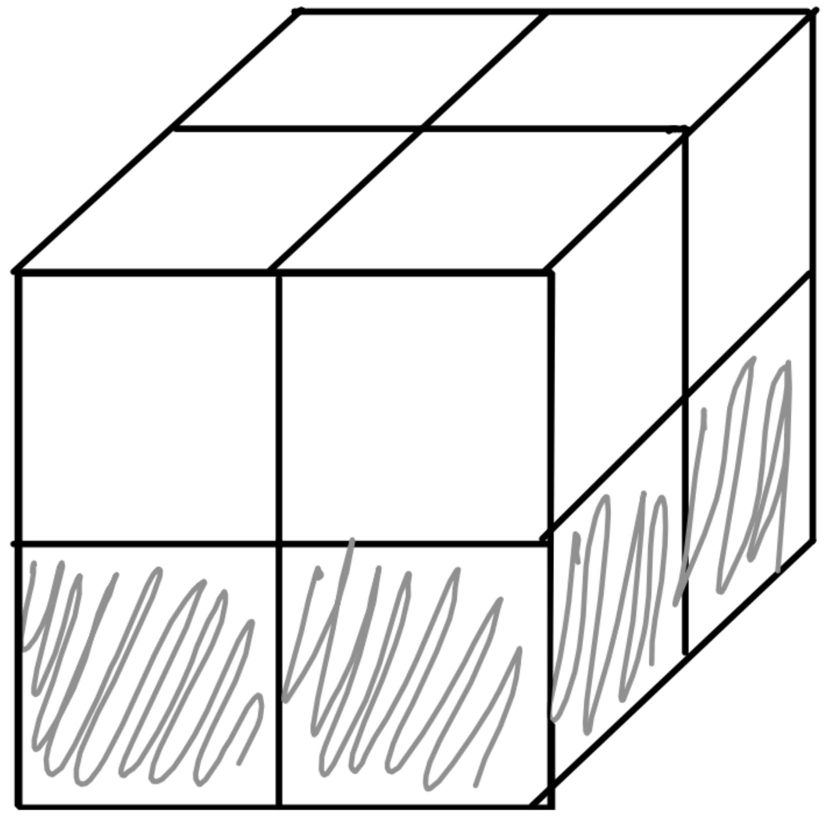


(1,1)

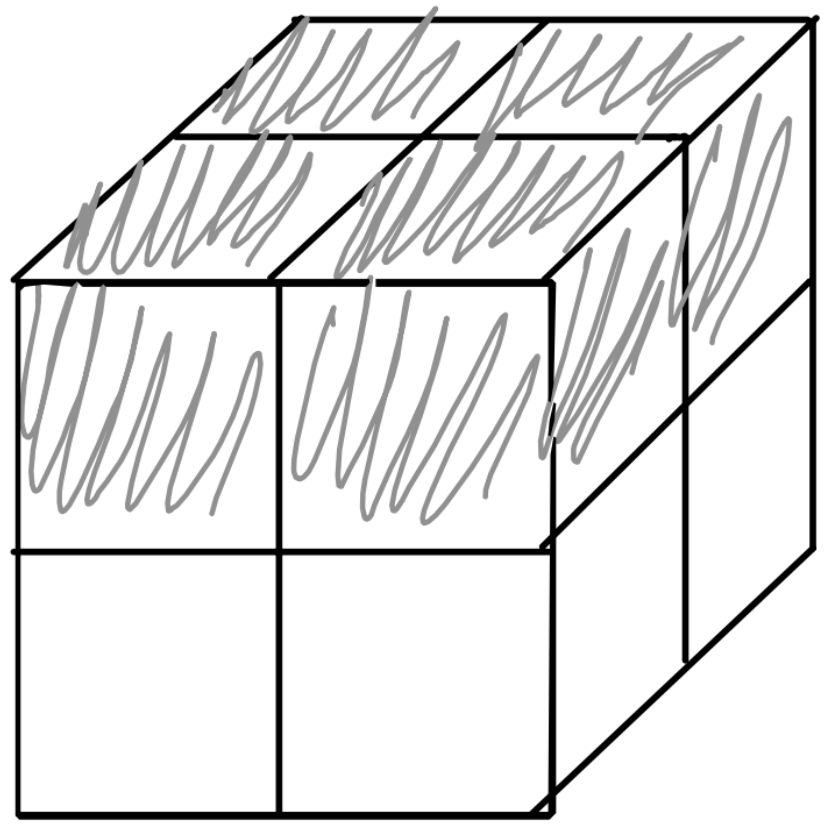


(1,2)

MPC 2

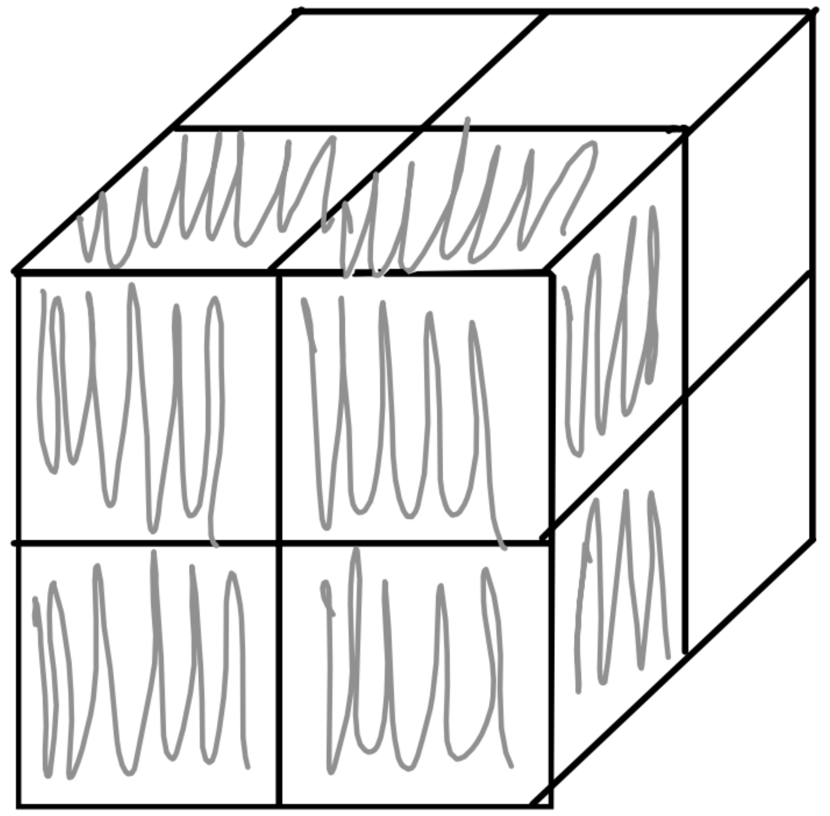


(2,1)

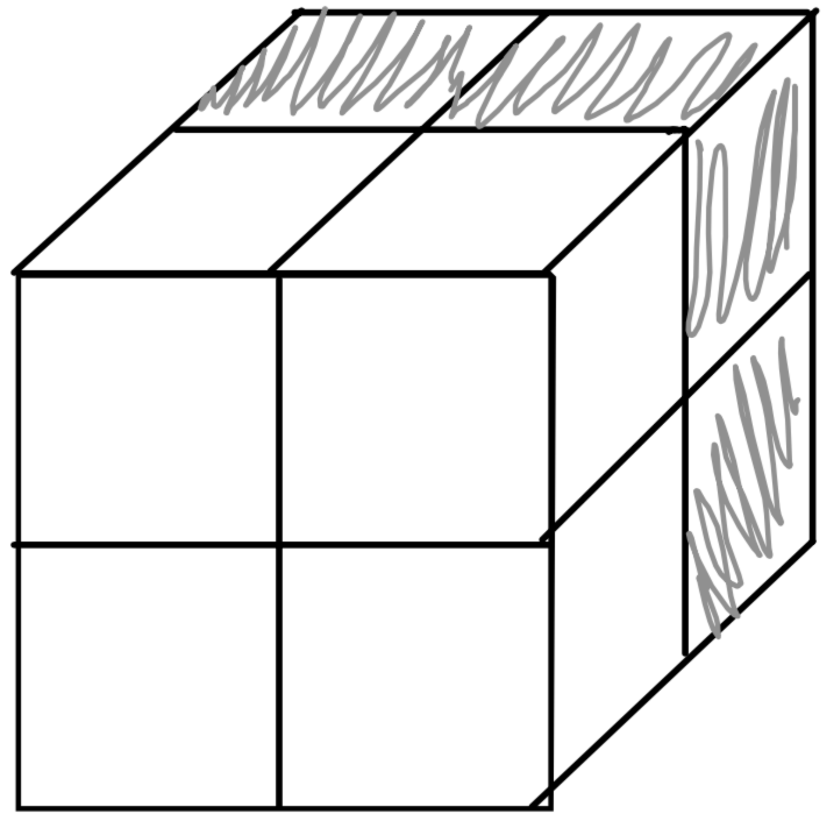


(2,2)

MPC 3



(3,1)



(3,2)

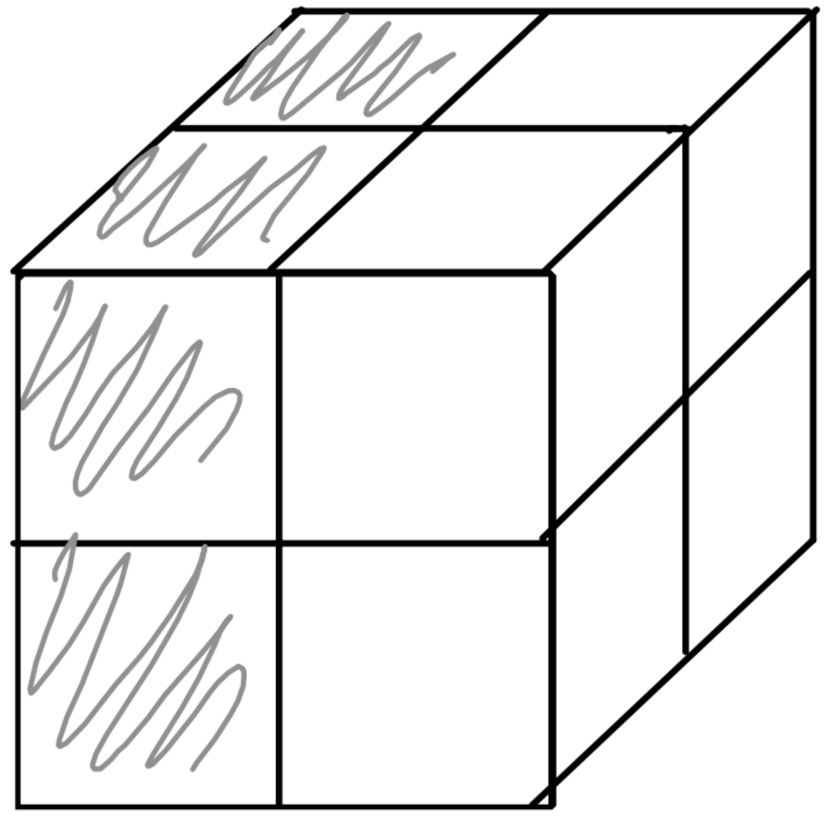
Hypercube

Take shares whose k-th coordinate is j.

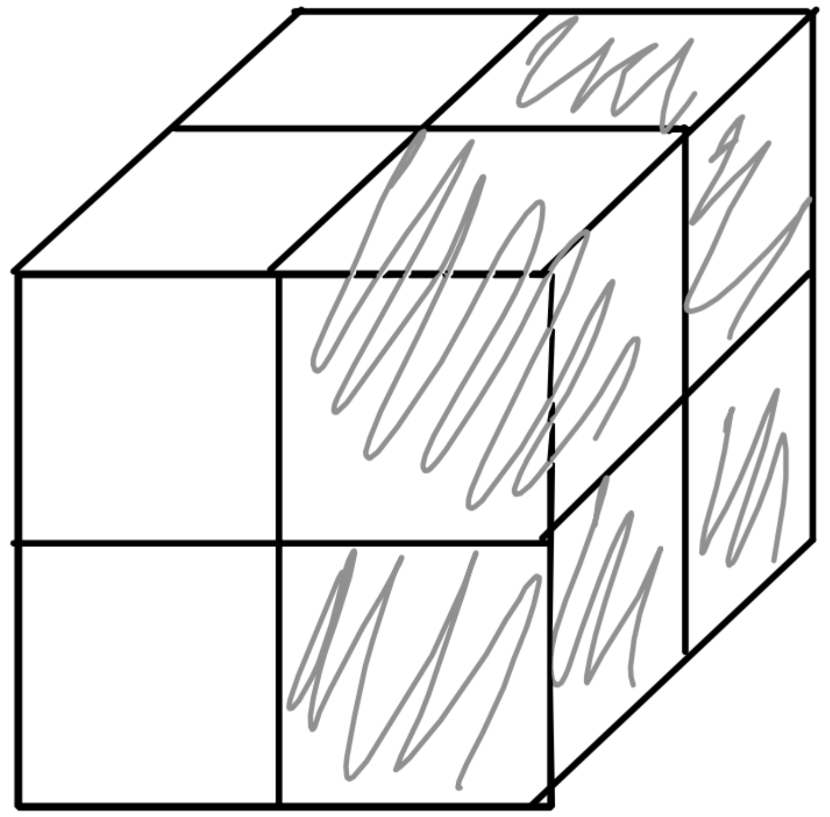
Going to higher dimensions

Example. $D = 3$

MPC 1



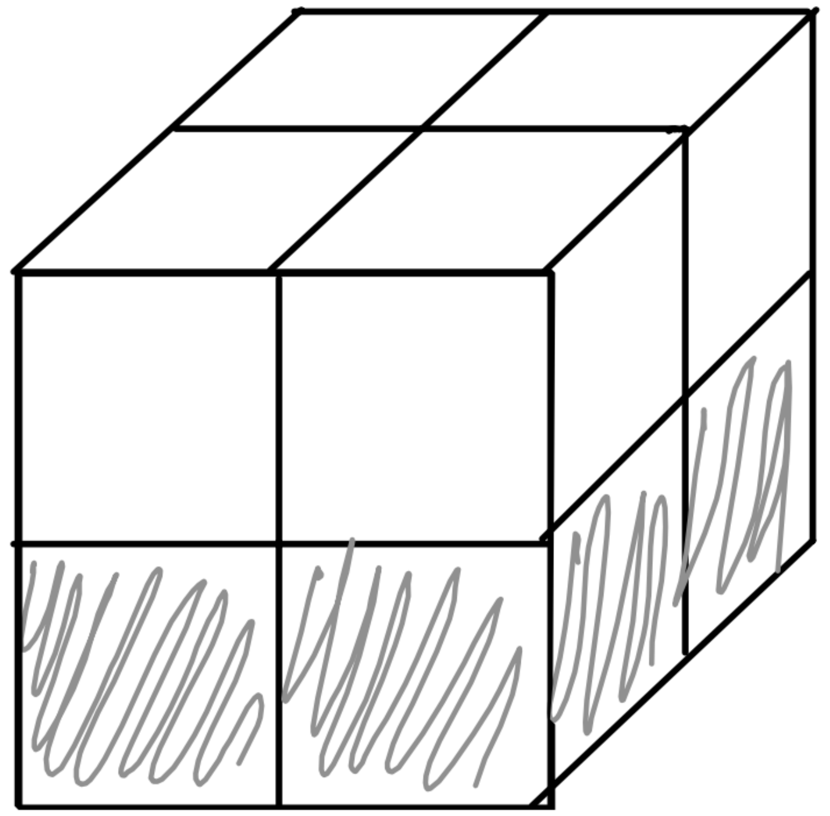
(1,1)



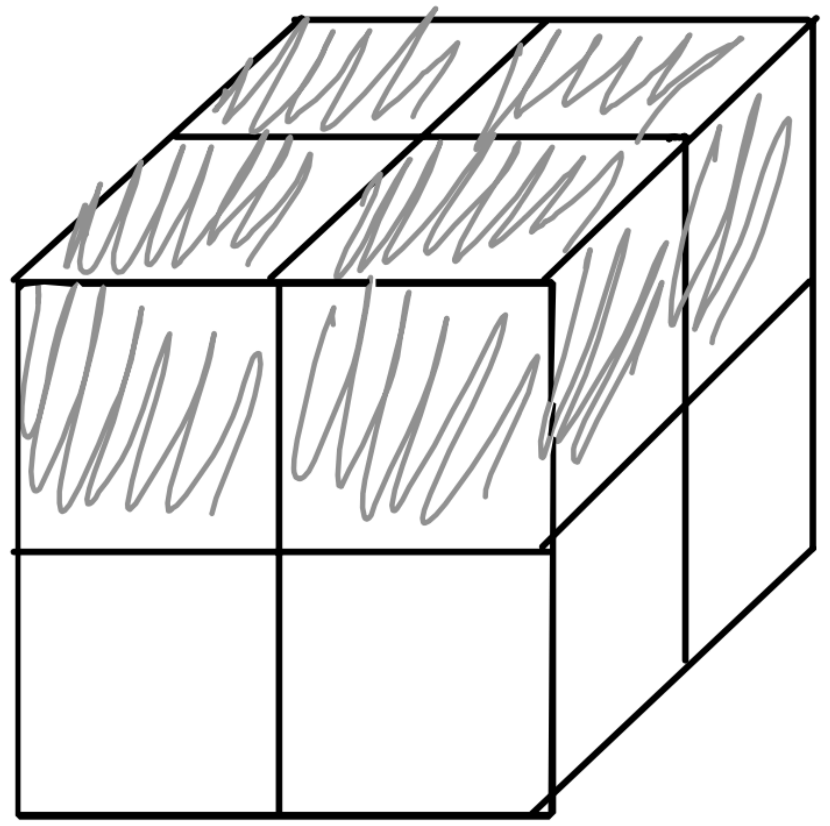
(1,2)

3 MPC instance of 2 main parties.

MPC 2

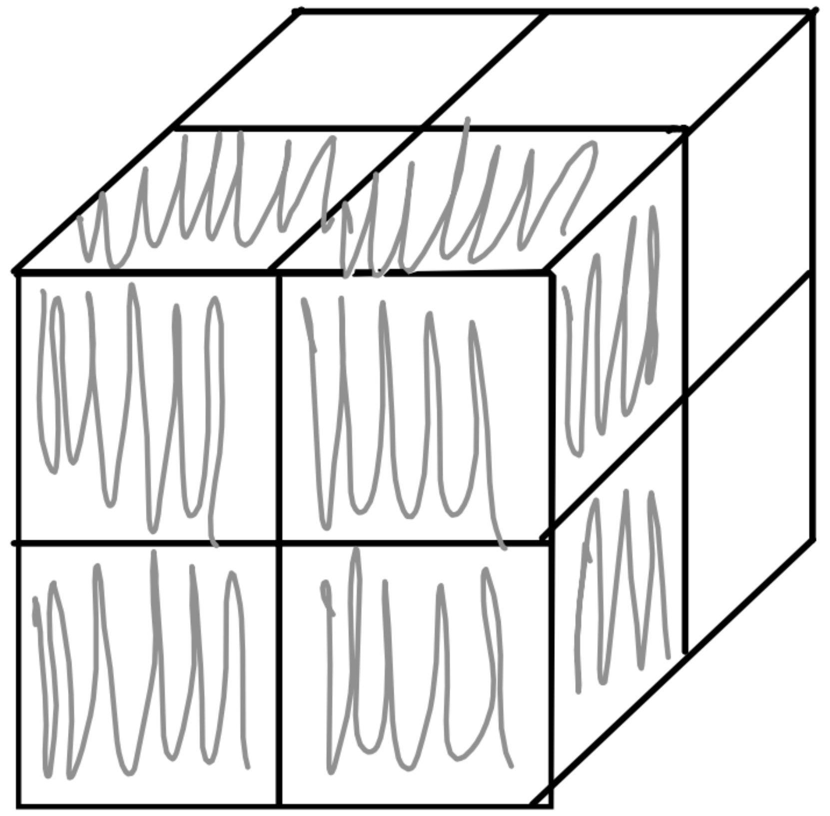


(2,1)

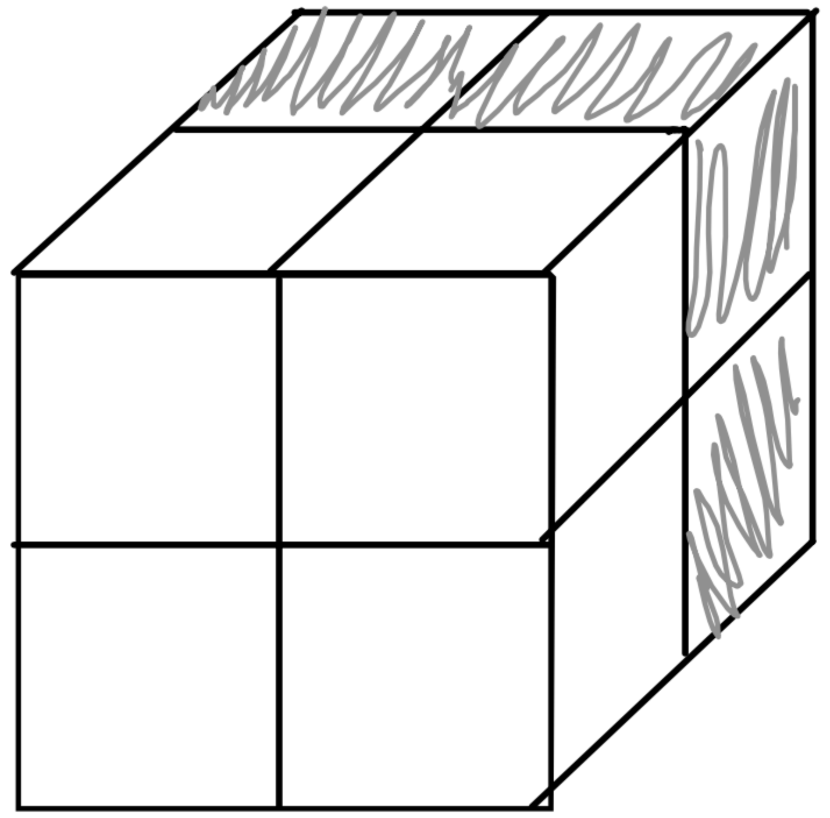


(2,2)

MPC 3



(3,1)



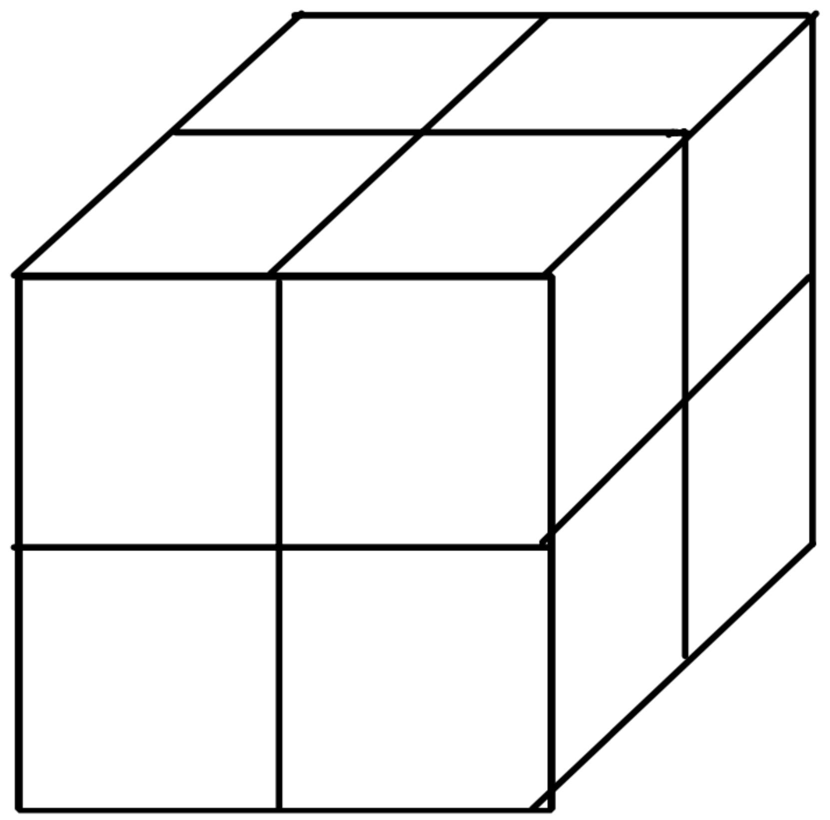
(3,2)

Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$

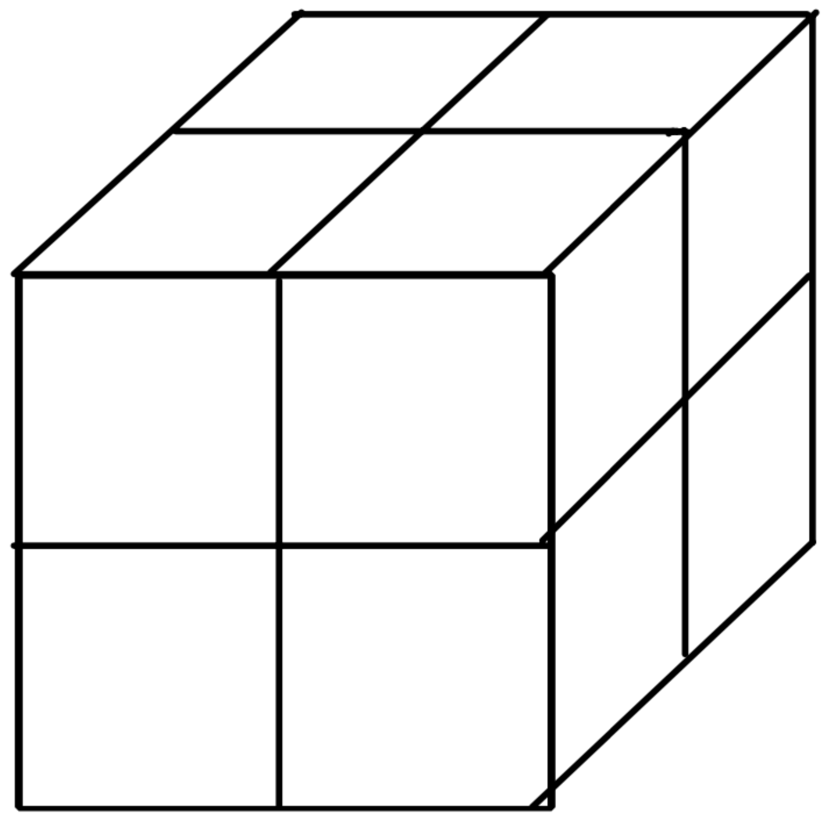


Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$

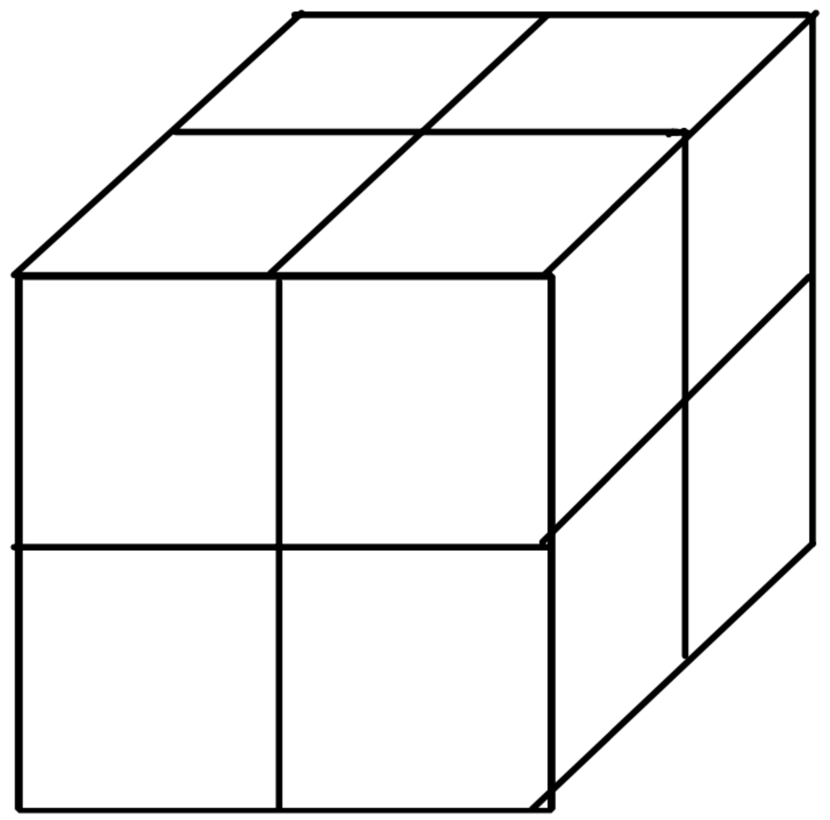


Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 4$

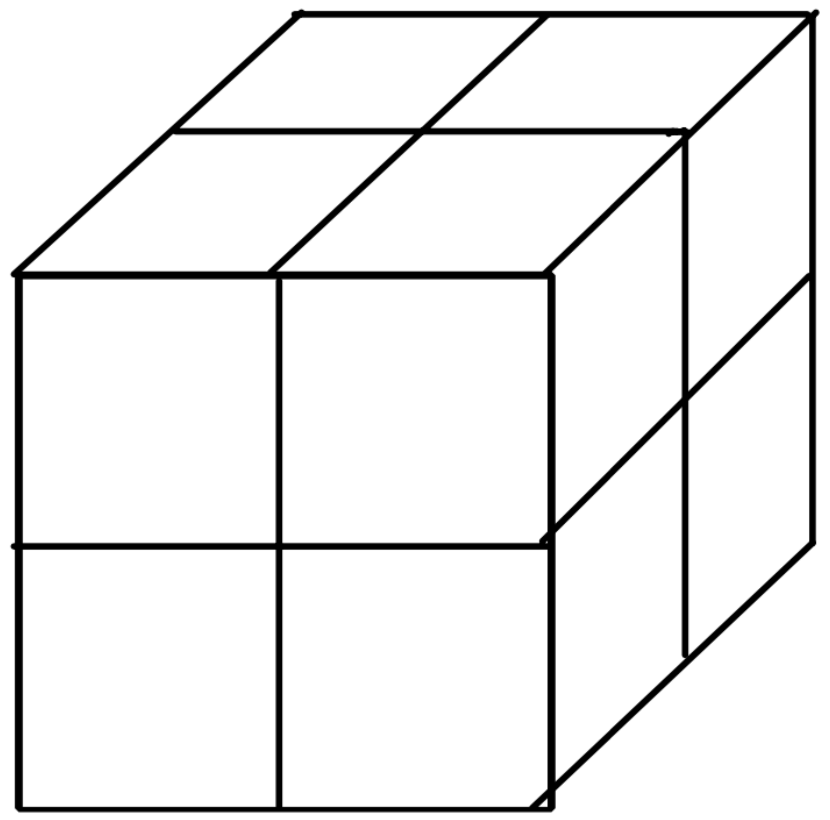


Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$

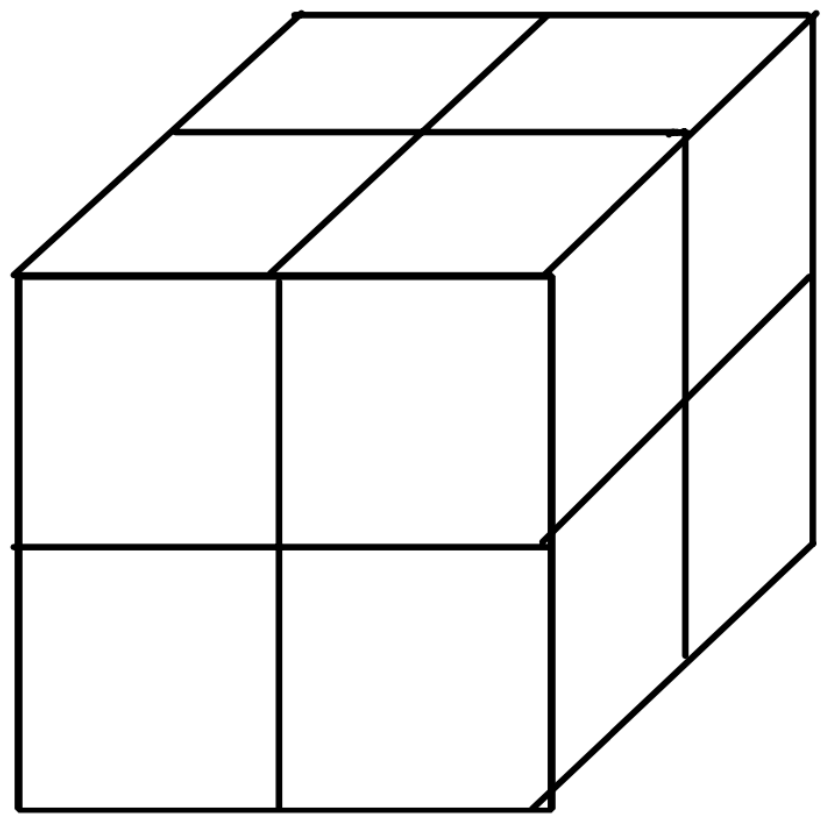


Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 4$

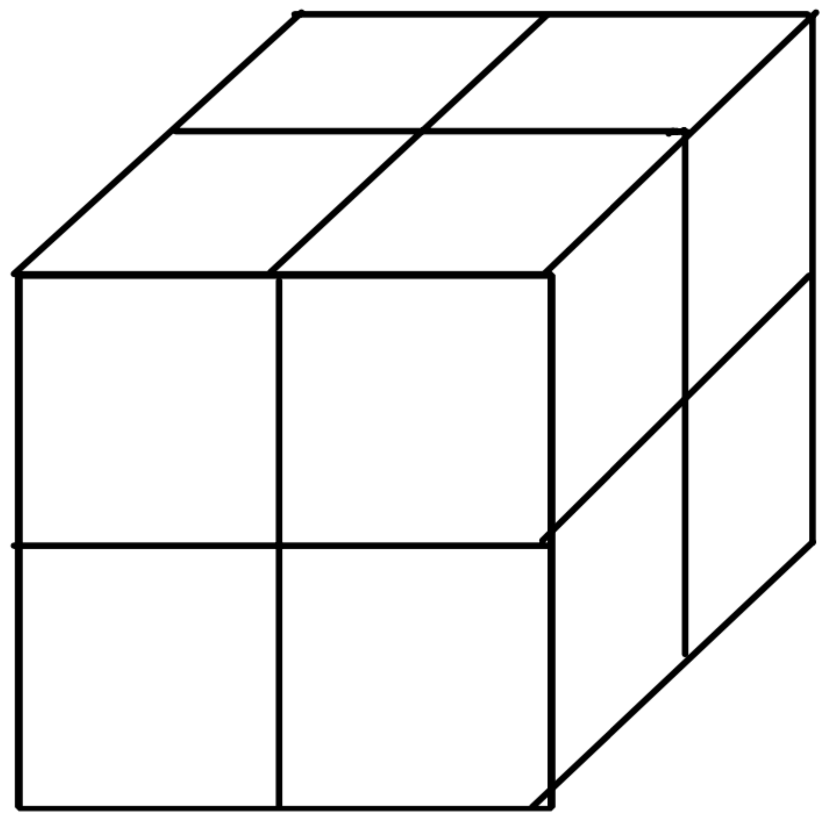


Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$

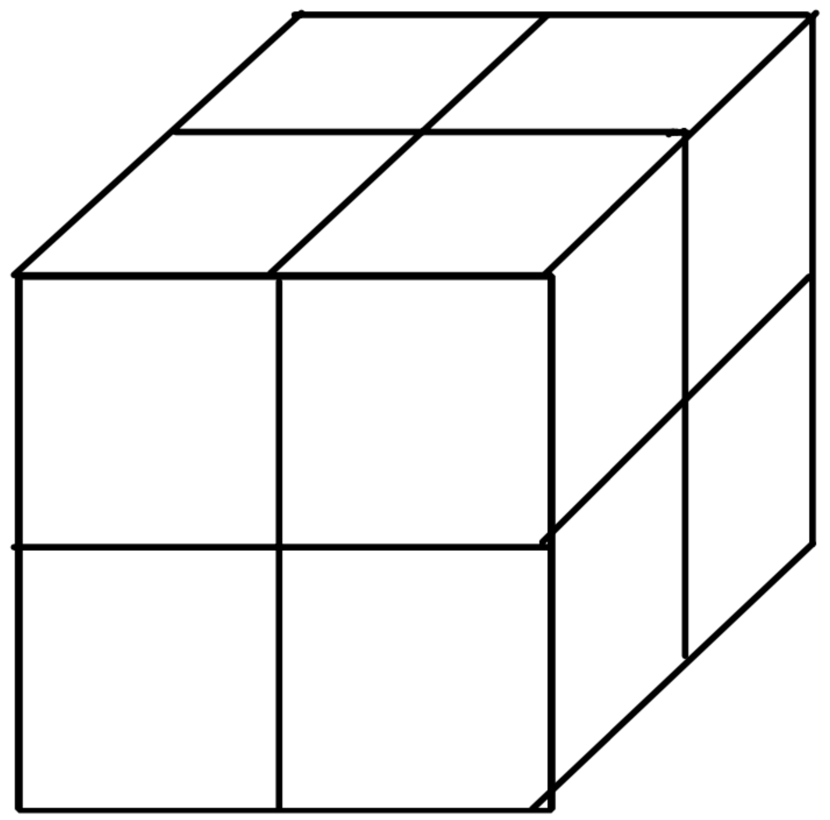


Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$

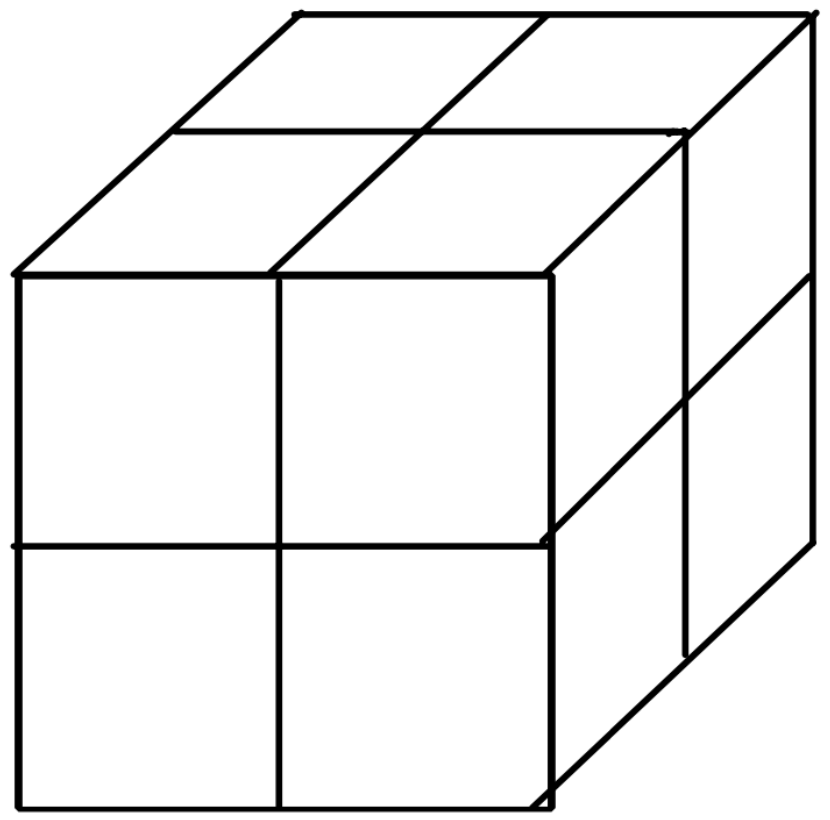


Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$



Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 4$

MPC 1

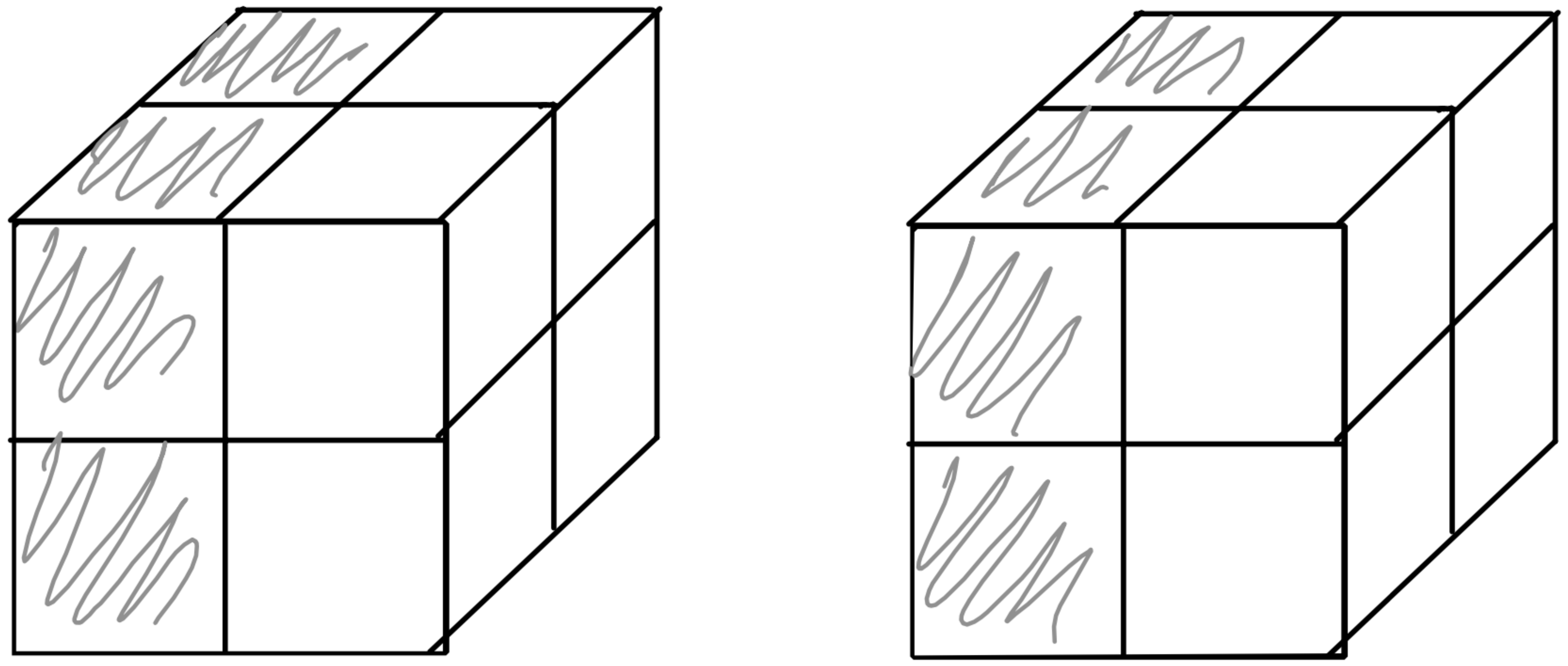
Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 4$

MPC 1



(1,1)

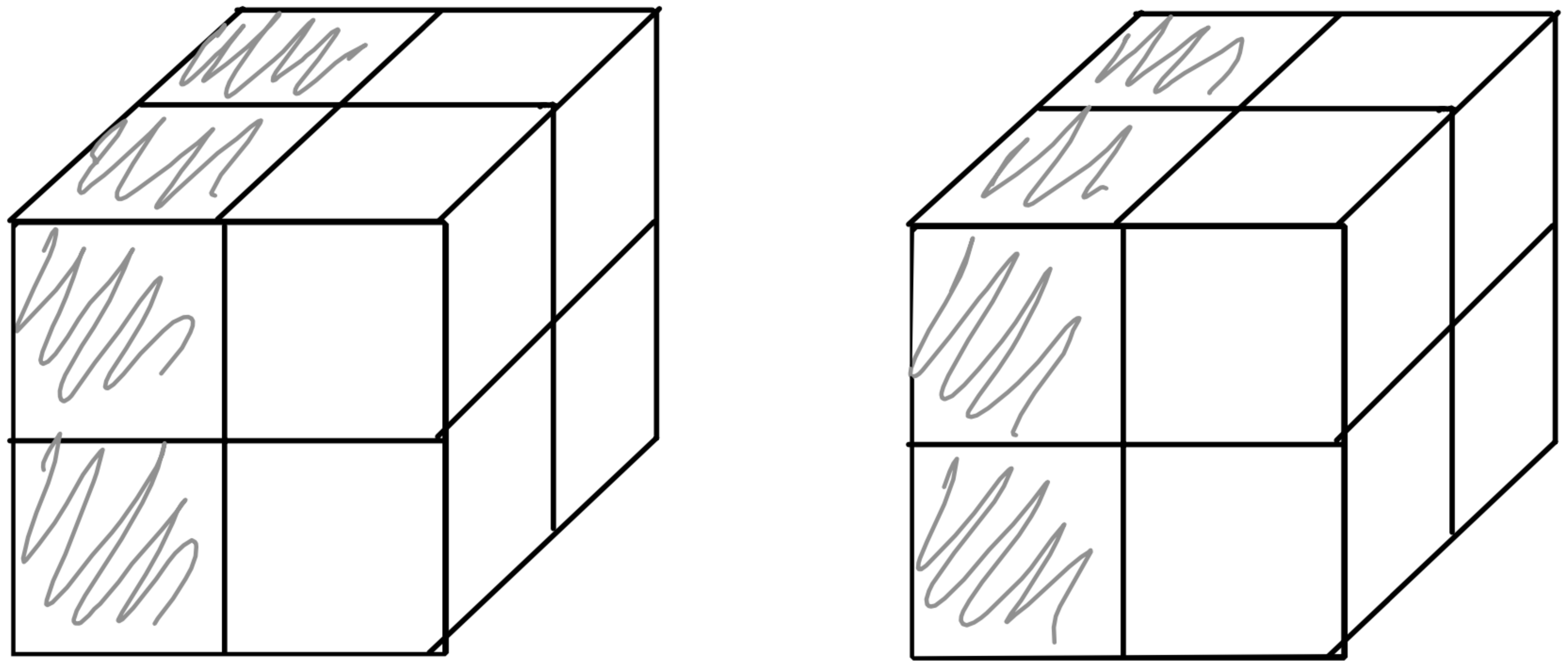
Hypercube

Take shares whose k-th coordinate is j.

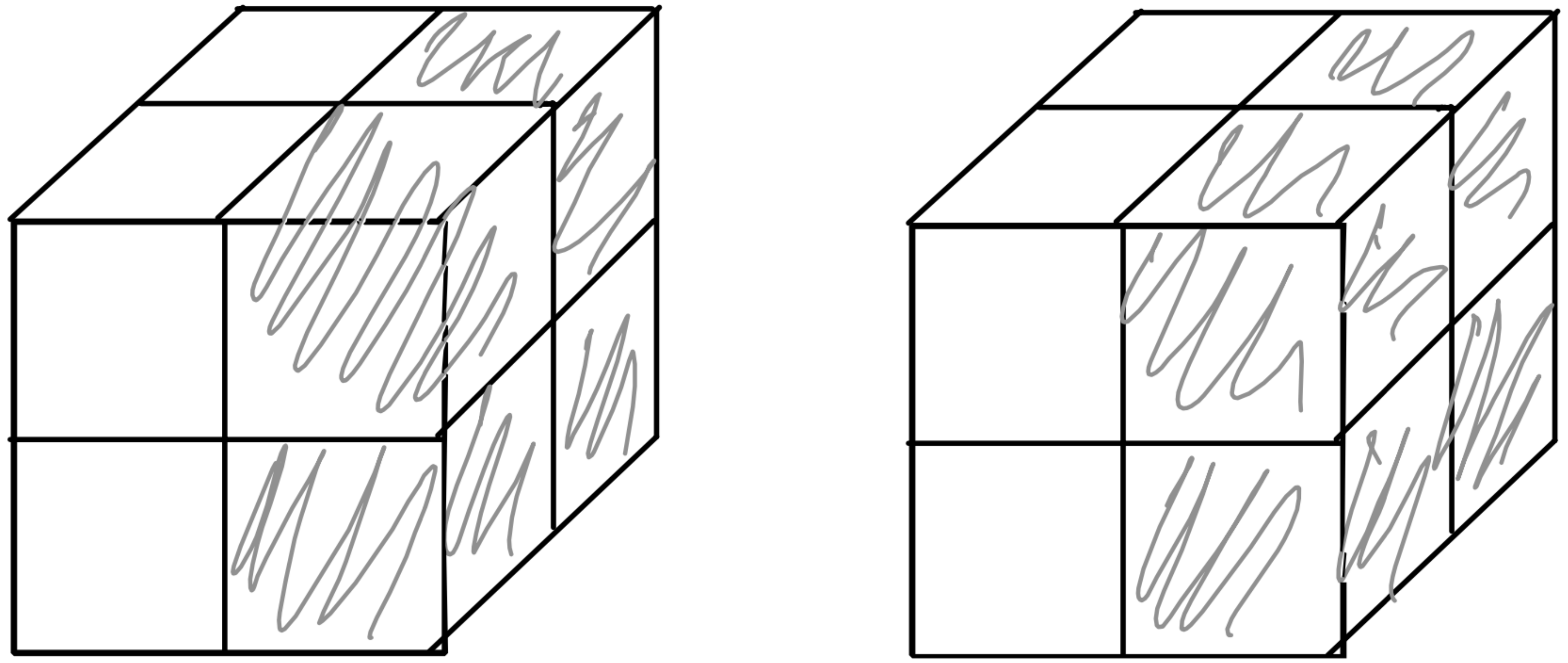
Going to higher dimensions

Example. $D = 4$

MPC 1



(1,1)



(1,2)

Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 4$

MPC 2

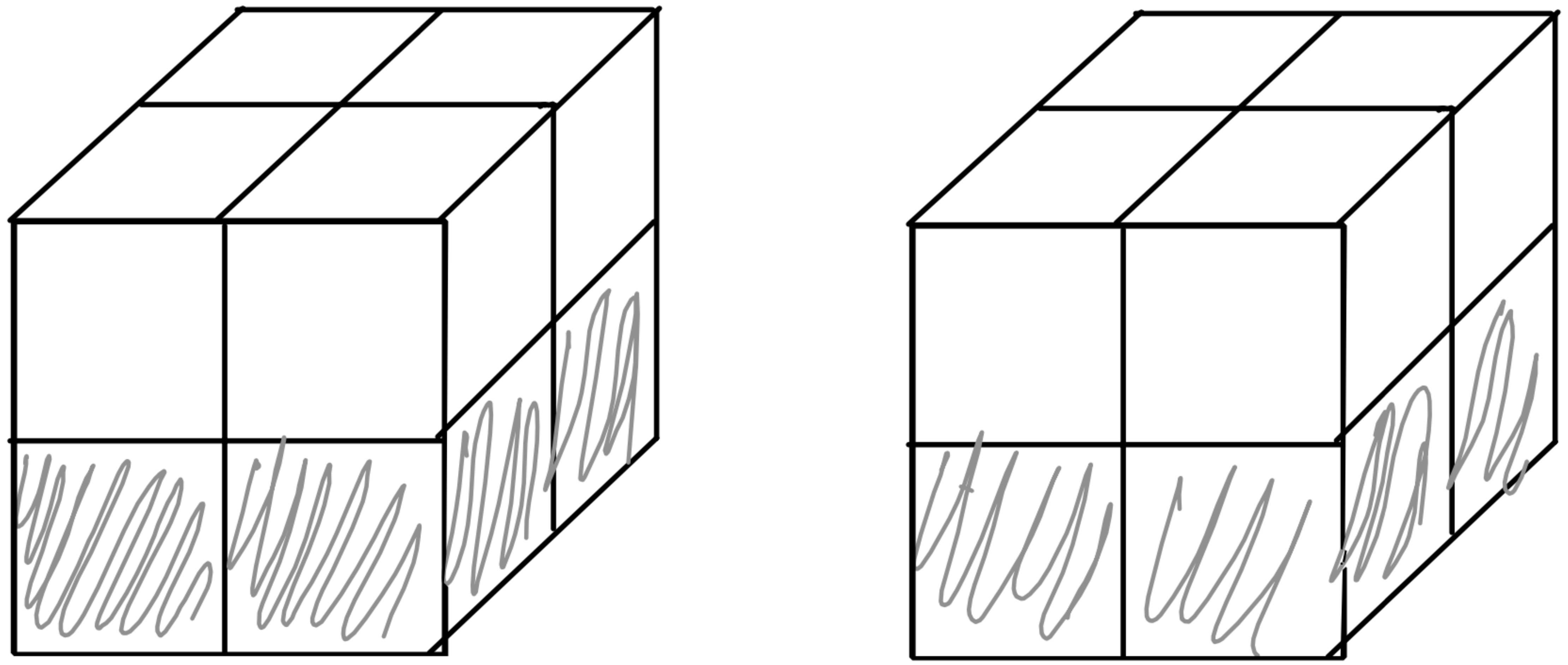
Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$

MPC 2



(2,1)

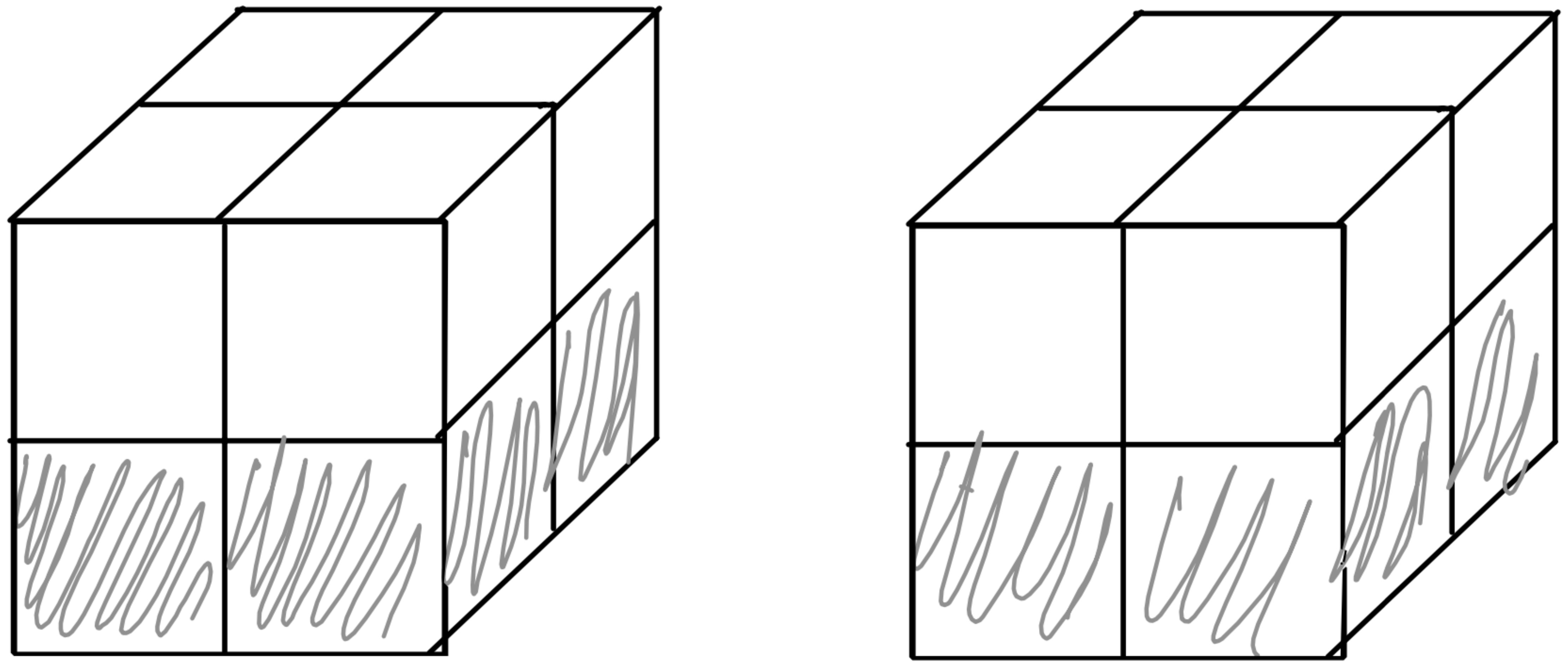
Hypercube

Take shares whose k-th coordinate is j.

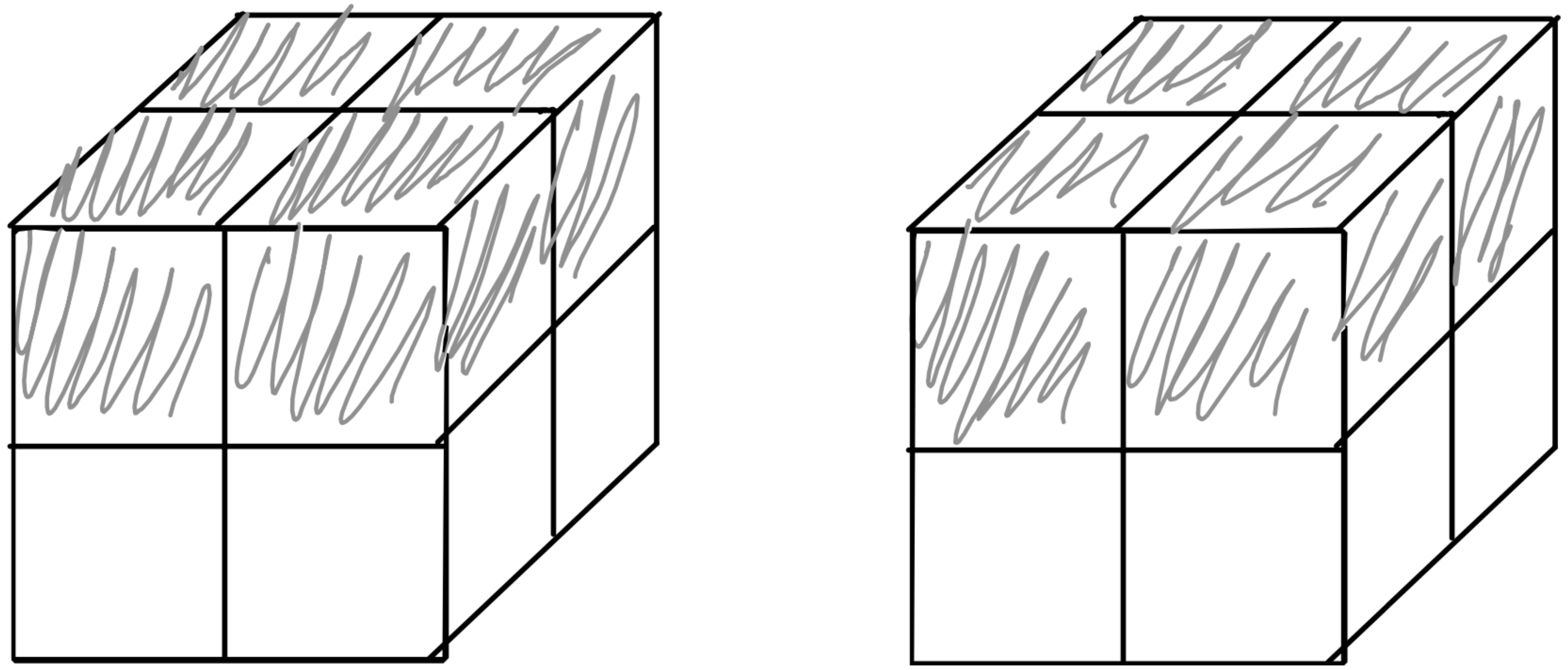
Going to higher dimensions

Example. $D = 4$

MPC 2



(2,1)



(2,2)

Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 4$

MPC 3

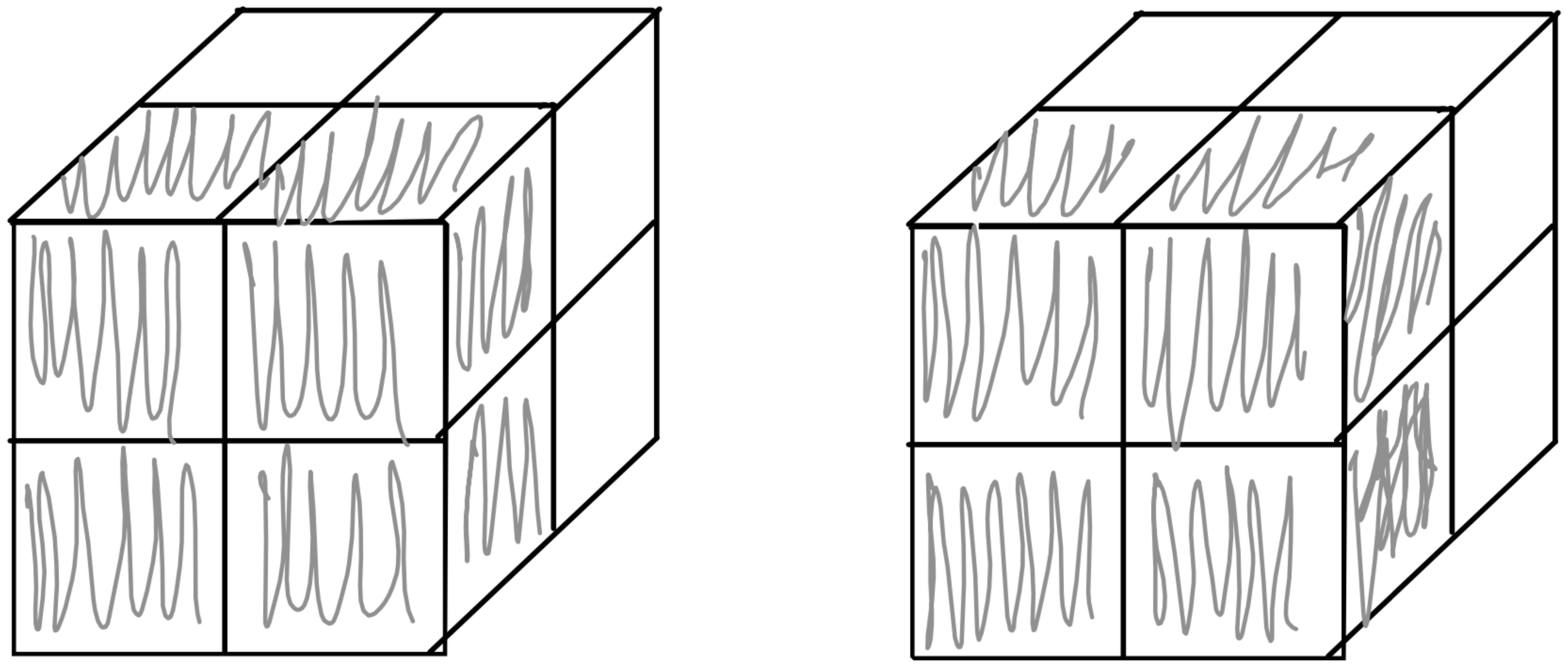
Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 4$

MPC 3



(3,1)

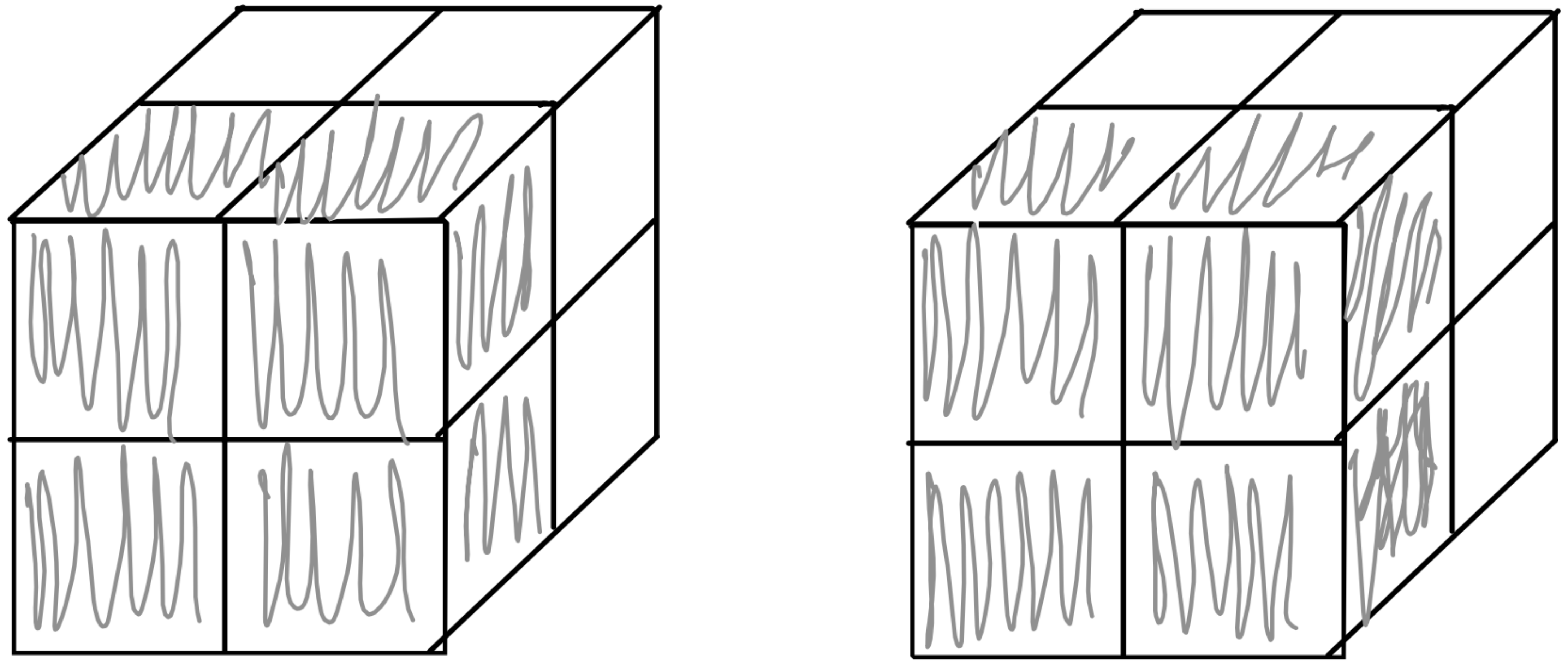
Take shares whose k-th coordinate is j.

Hypercube

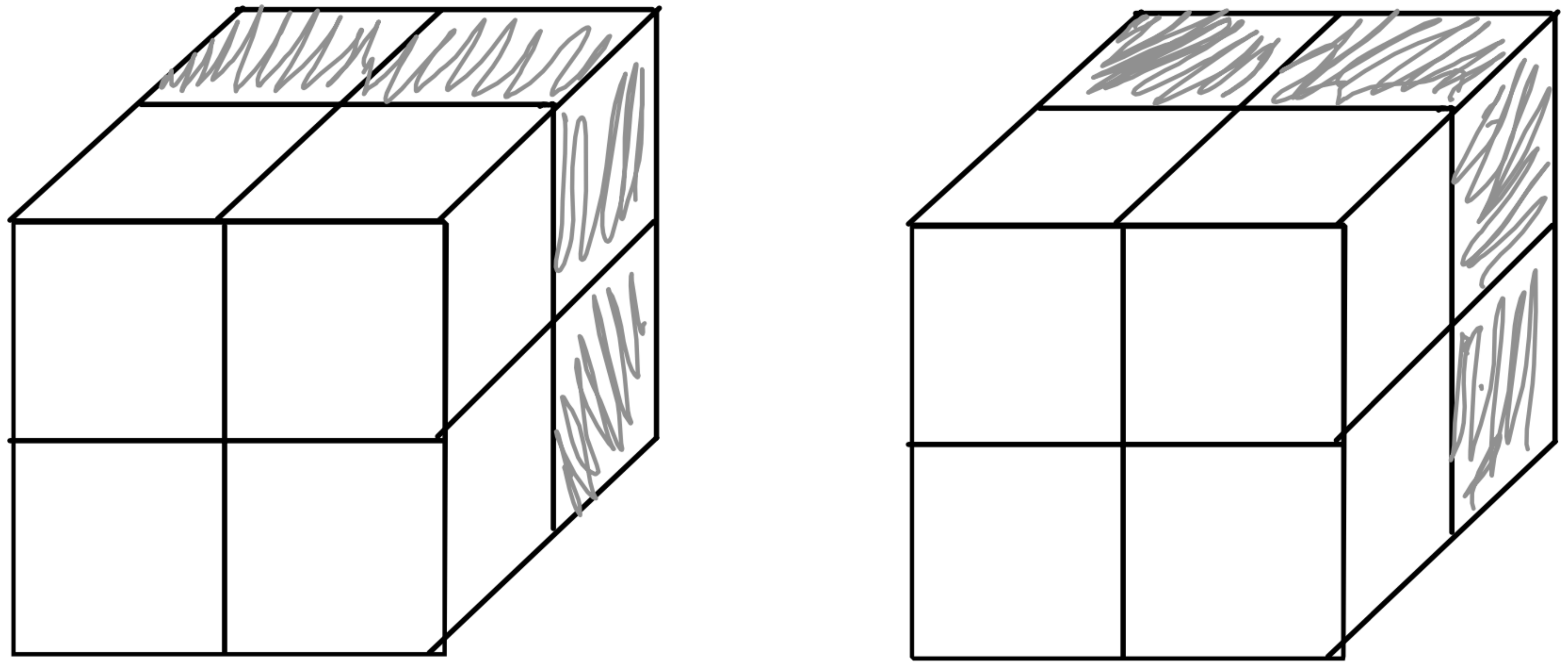
Going to higher dimensions

Example. $D = 4$

MPC 3



(3,1)



(3,2)

Take shares whose k-th coordinate is j.

Hypercube

Going to higher dimensions

Example. $D = 4$

MPC 4

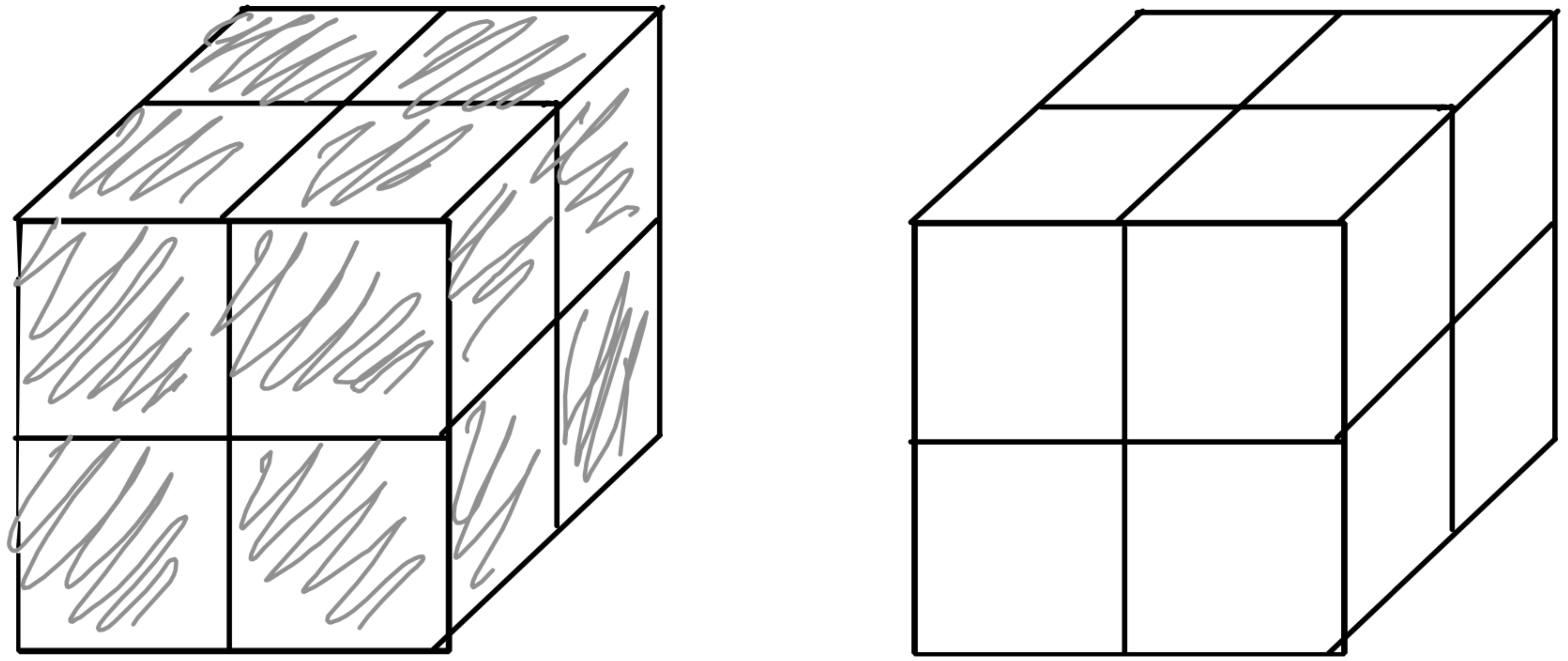
Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$

MPC 4



(4,1)

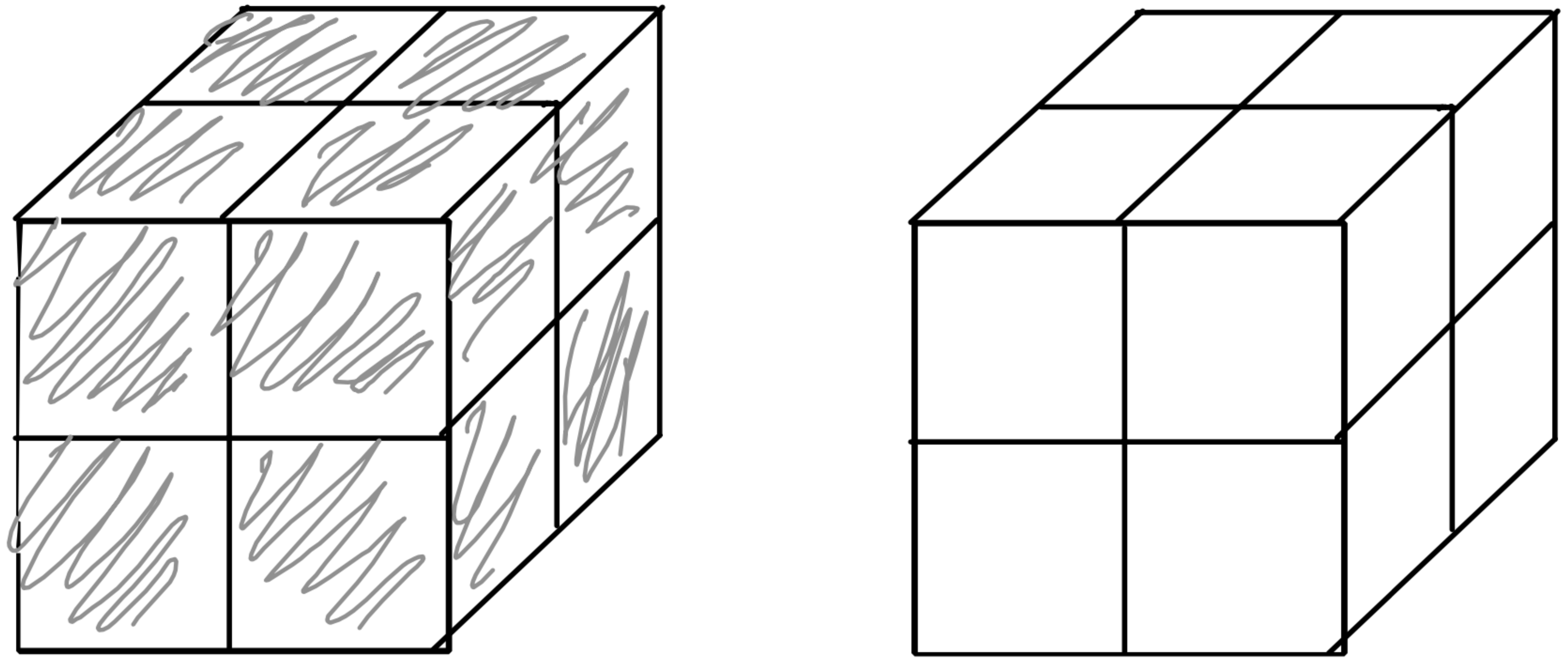
Take shares whose k-th coordinate is j.

Hypercube

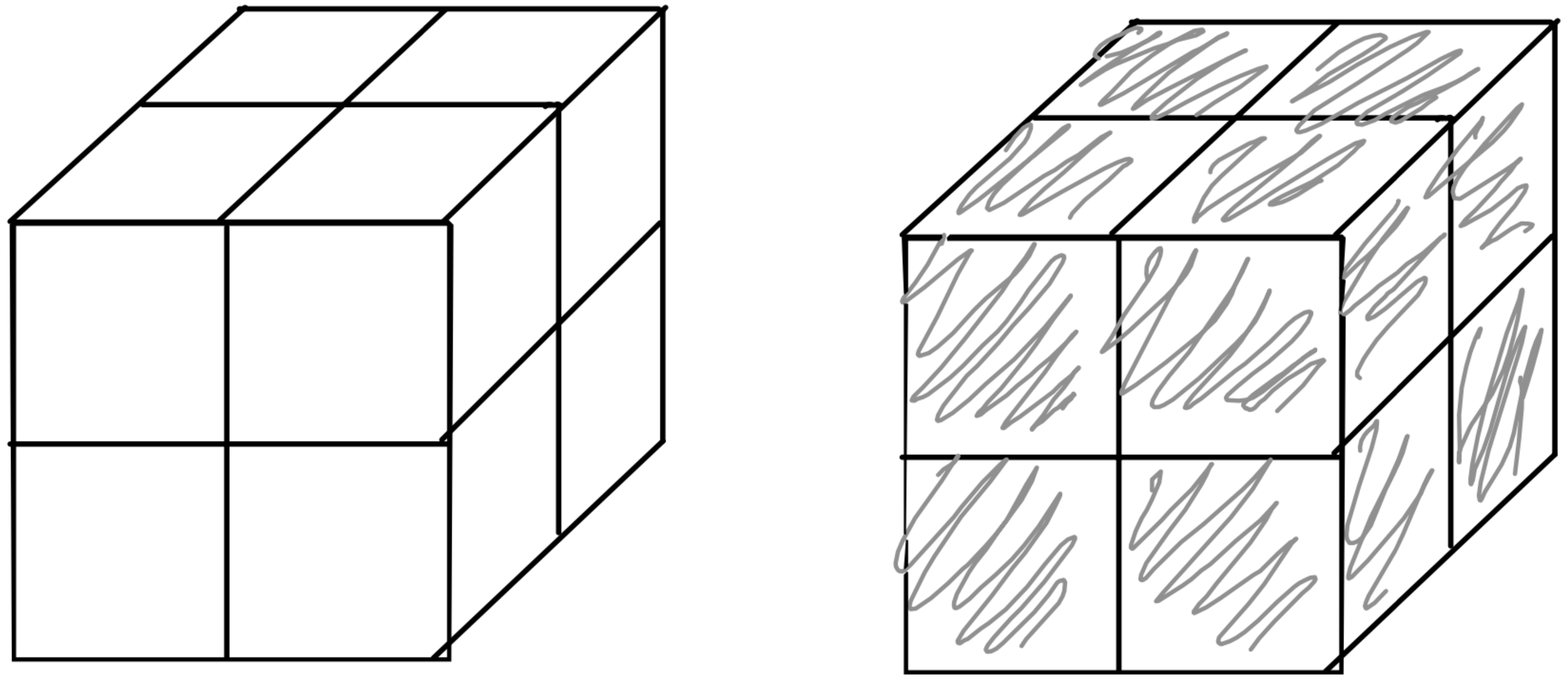
Going to higher dimensions

Example. $D = 4$

MPC 4



(4,1)



(4,2)

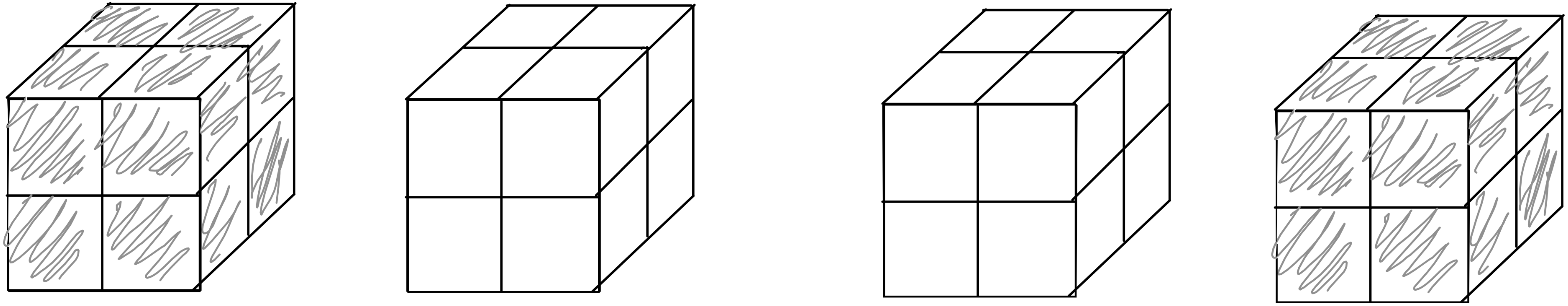
Hypercube

Take shares whose k-th coordinate is j.

Going to higher dimensions

Example. $D = 4$

<u>MPC 4</u>



(4,1)

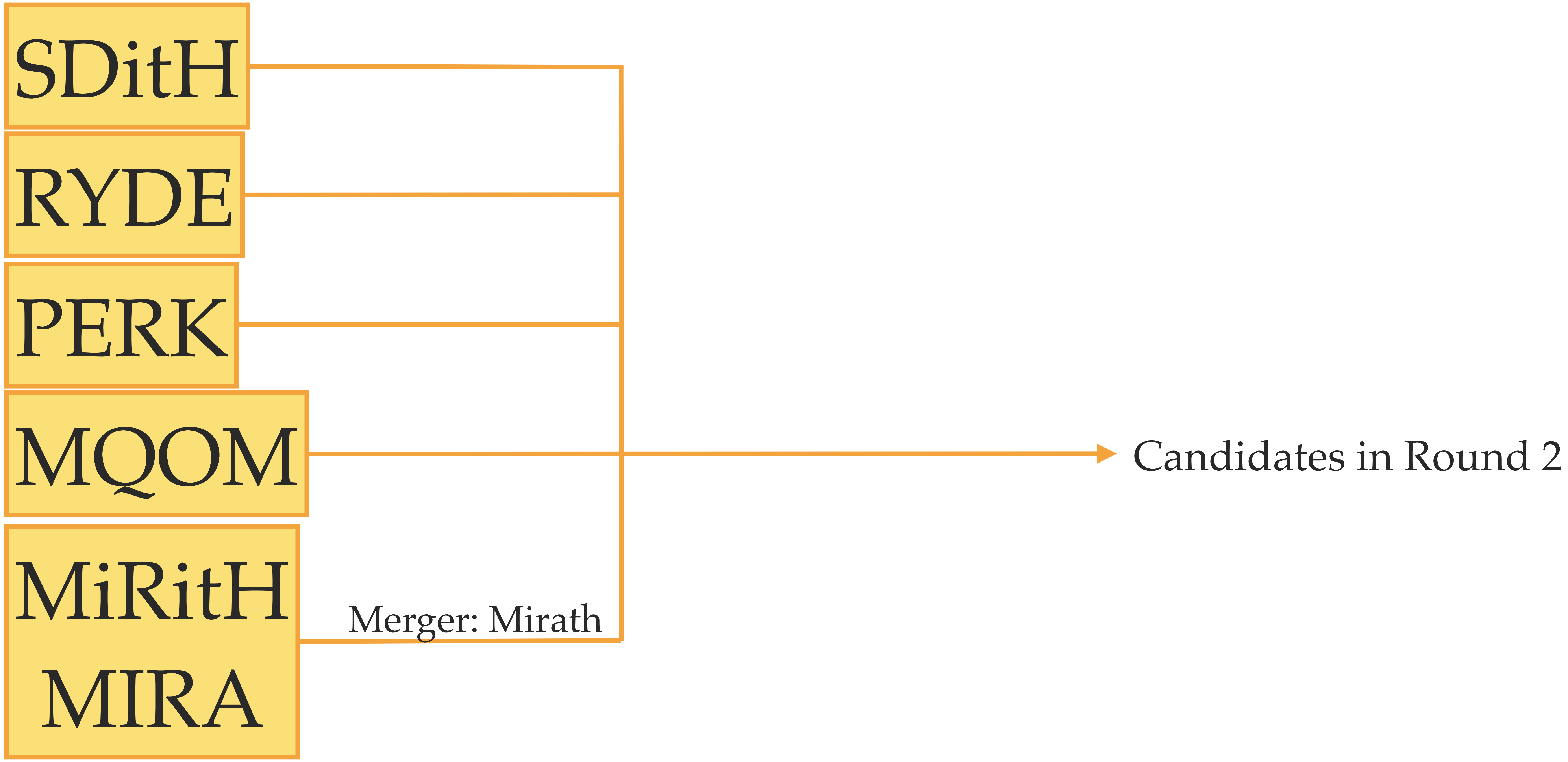
(4,2)

4 MPC instance of 2 main parties.

MPCitH in the NIST competition

SDitH	$\longrightarrow$ Syndrome decoding problem in the Hamming metric
RYDE	$\longrightarrow$ Syndrome decoding problem in the rank metric
PERK	$\longrightarrow$ Permuted kernel problem
MQOM	$\longrightarrow$ MQ problem
MiRitH	$\longrightarrow$ MinRank problem
MIRA	$\longrightarrow$ MinRank problem
Biscuit	$\longrightarrow$ MQ problem (with additional structure)
AIMer	$\longrightarrow$ MPC-friendly symmetric primitive

MPCitH in the NIST competition



Biscuit

AIMer