Post-Quantum Signatures from Multiparty Computation: Recent Advances

Thibauld Feneuil

PQCrypto 2023

August 17, 2023, College Park (USA)

CRYPTOEXPERTS
WE INNOVATE TO SECURE YOUR BUSINESS

SORBONNE UNIVERSITÉ

Table of Contents

- Introduction
- MPC-in-the-Head: general principle
- From MPC-in-the-Head to signatures
- Optimisations and variants
- Related works
- Conclusion

Some figures used in the following slides have been realised by Matthieu Rivain (CryptoExperts).

Introduction

How to build signature schemes?

Hash & Sign

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

Very hard to compute

- Short signatures
- "Trapdoor" in the public key

How to build signature schemes?

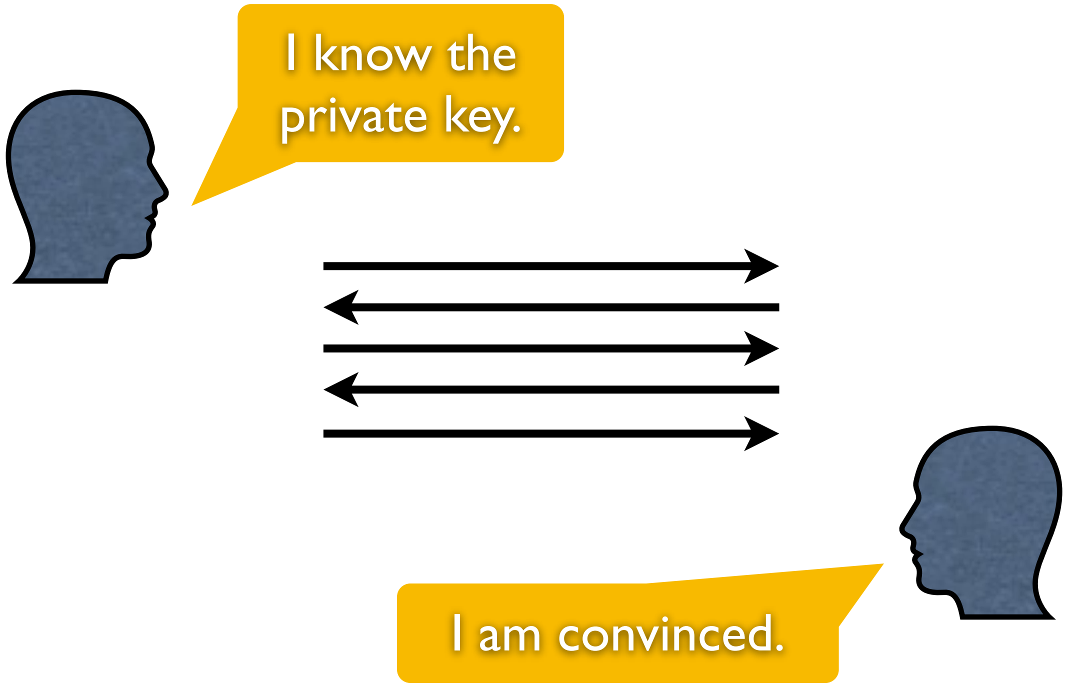
Hash & Sign

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

Very hard to compute

- Short signatures
- "Trapdoor" in the public key

From a zero-knowledge proof



- Large(r) signatures
- Short public key

How to build signature schemes?

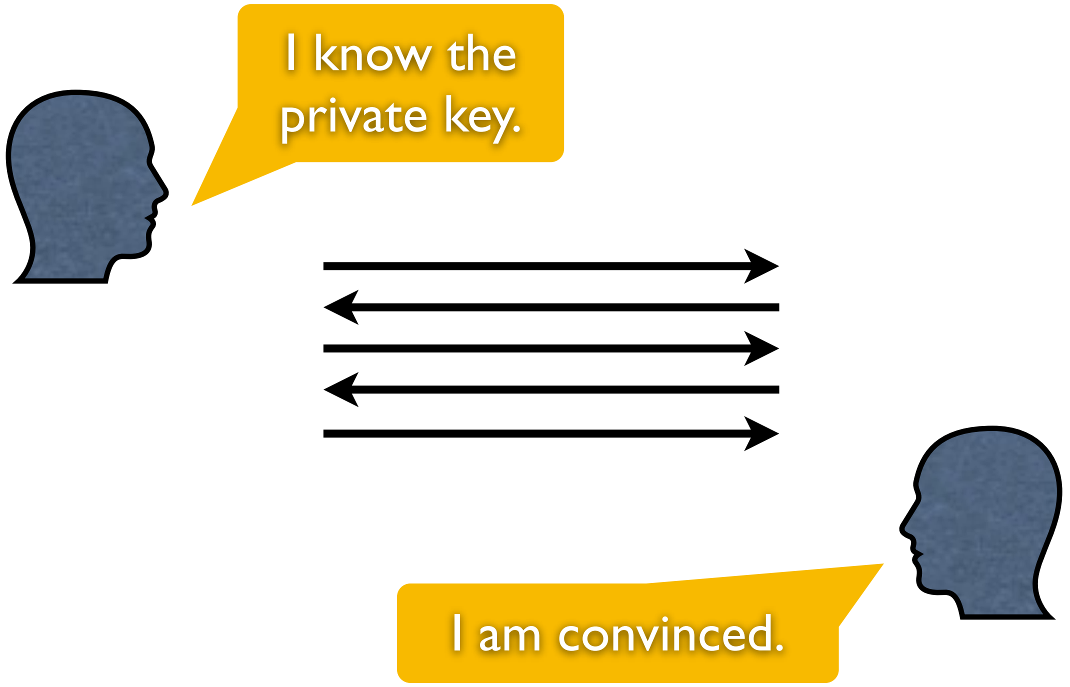
Hash & Sign

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

Very hard to compute

- Short signatures
- “Trapdoor” in the public key

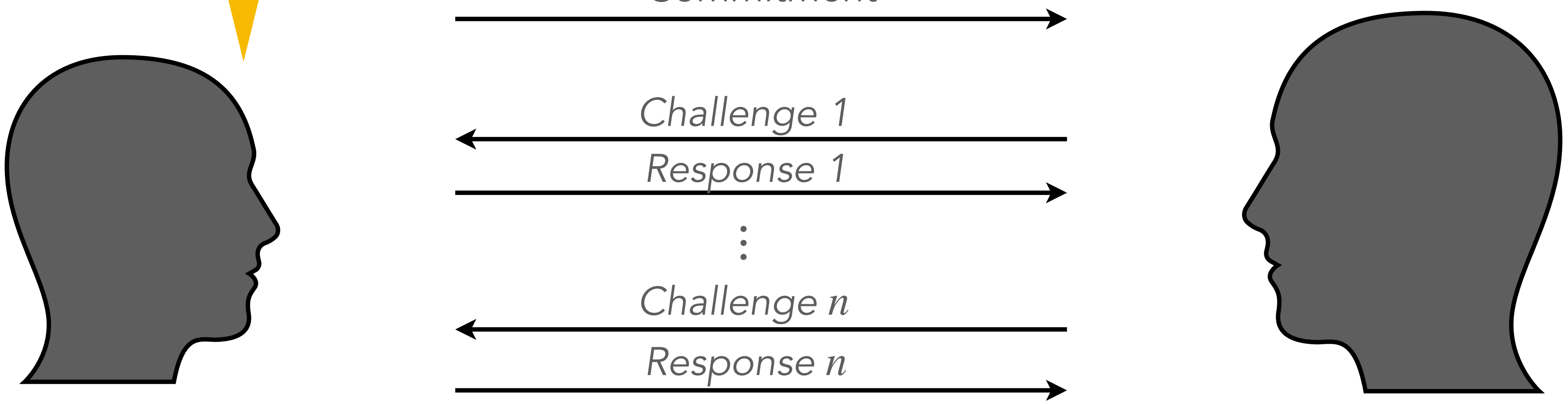
From a zero-knowledge proof



- Large(r) signatures
- Short public key

Proof of knowledge

I know x such that $F(x) = y$.



Commitment

Challenge 1

Response 1

$\vdots$

Challenge n

Response n

Prover

Verifier

I am convinced / I am not convinced.

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

MPC in the Head

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

MPC in the Head

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

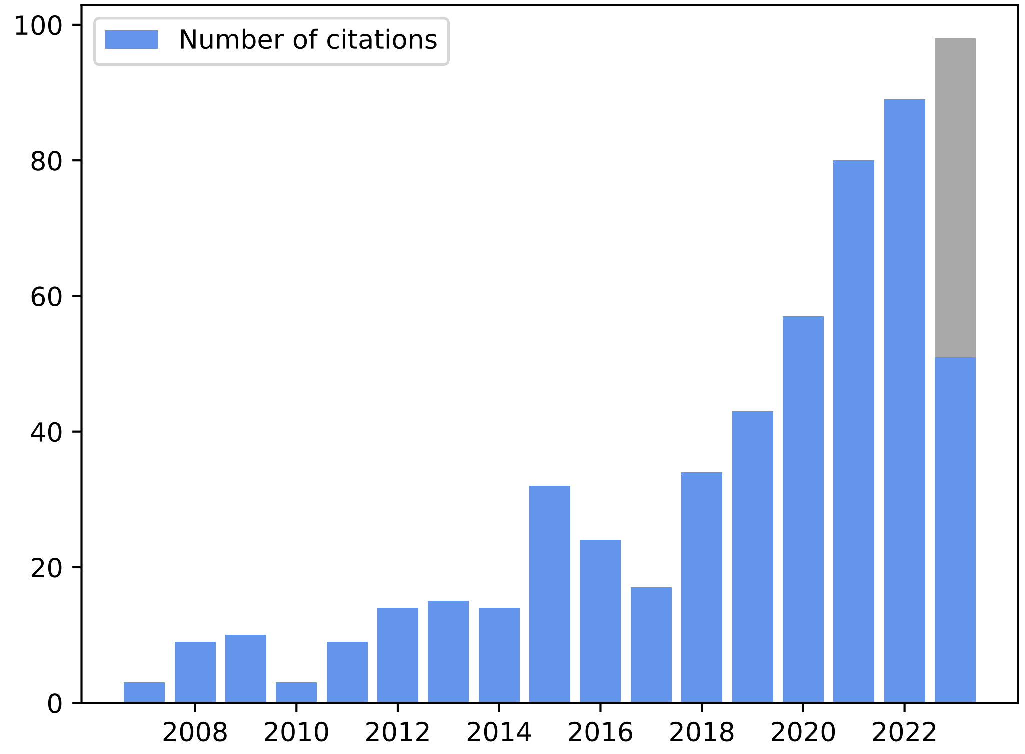


Figure: Number of citations to [IKOS07] by year

Source: Google Scholar

MPC in the Head

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

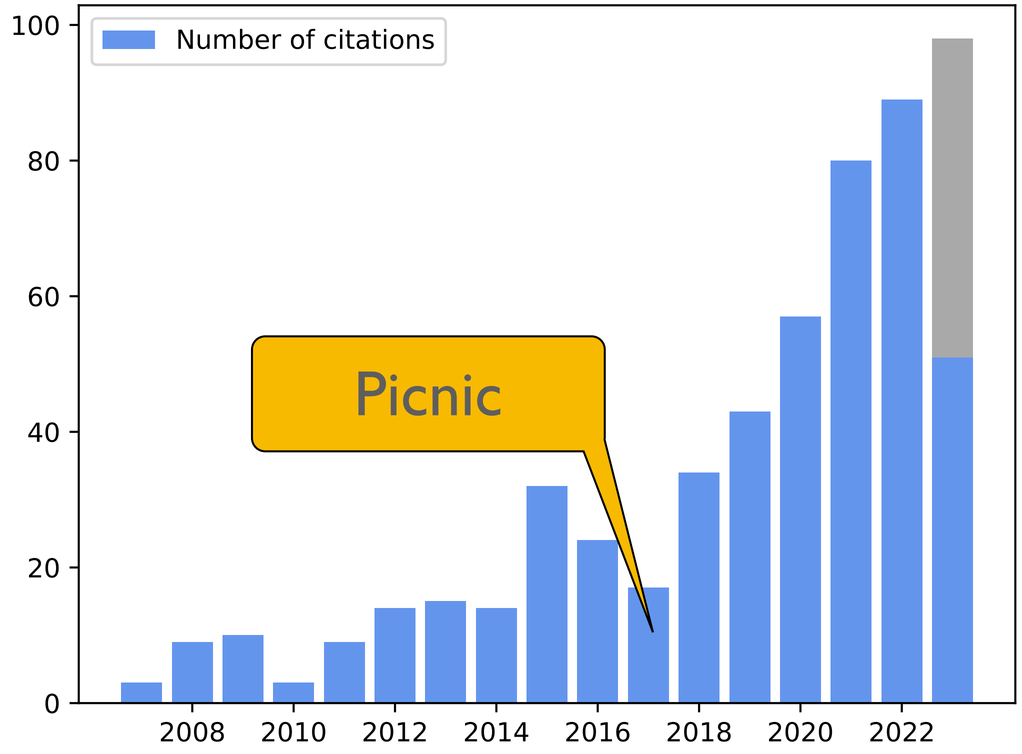


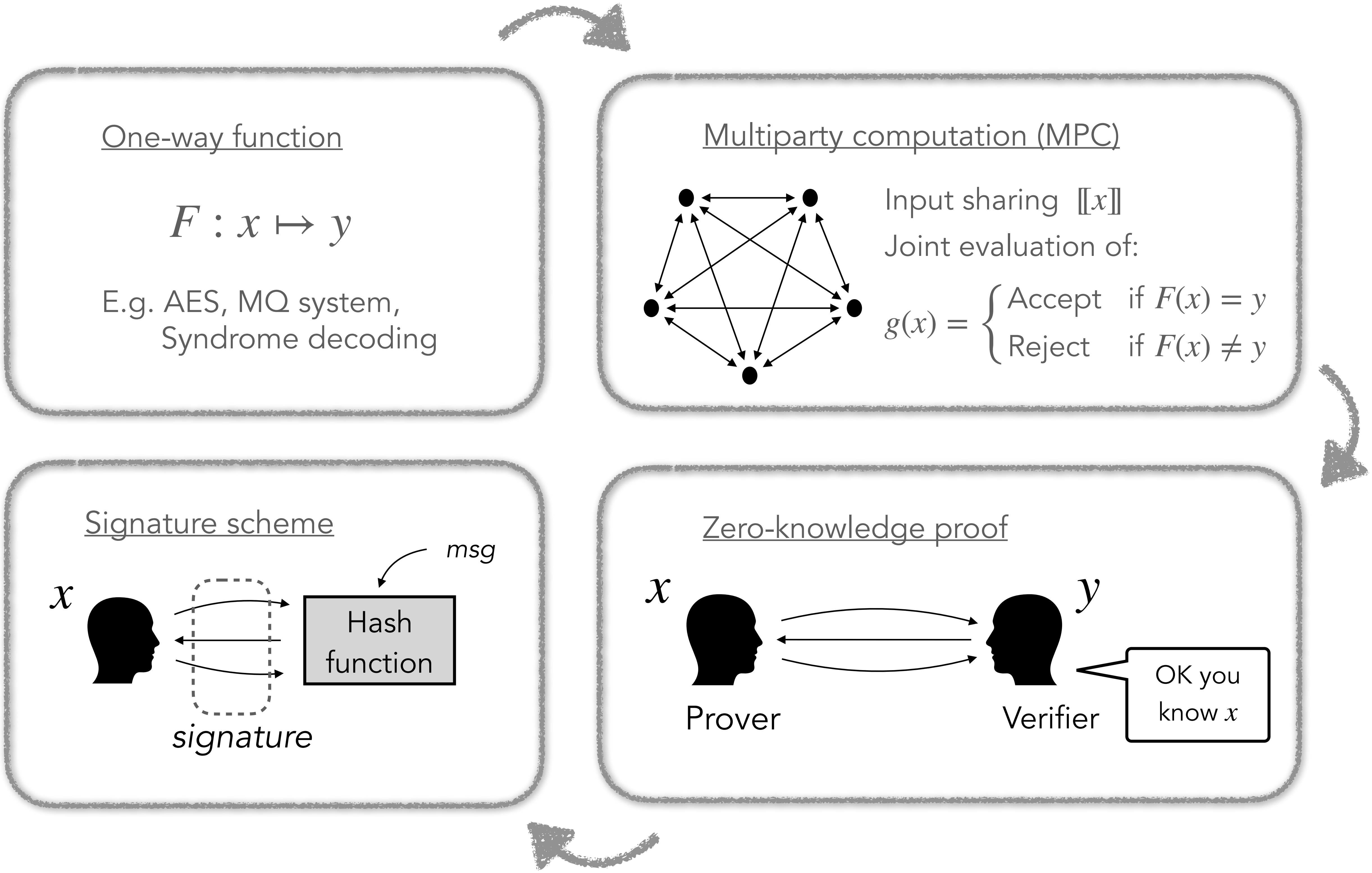
Figure: Number of citations to [IKOS07] by year

Source: Google Scholar

MPC in the Head

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

AIMer
Biscuit
MIRA
MiRitH
MQOM
PERK
RYDE
SDitH



One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

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

Joint evaluation of:

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

Zero-knowledge proof

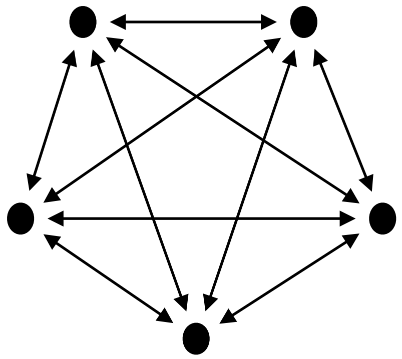
Signature scheme

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

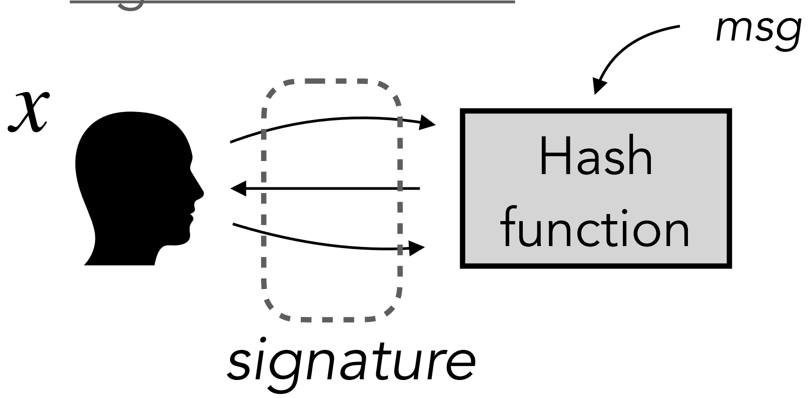


Input sharing $[\![x]\!]$

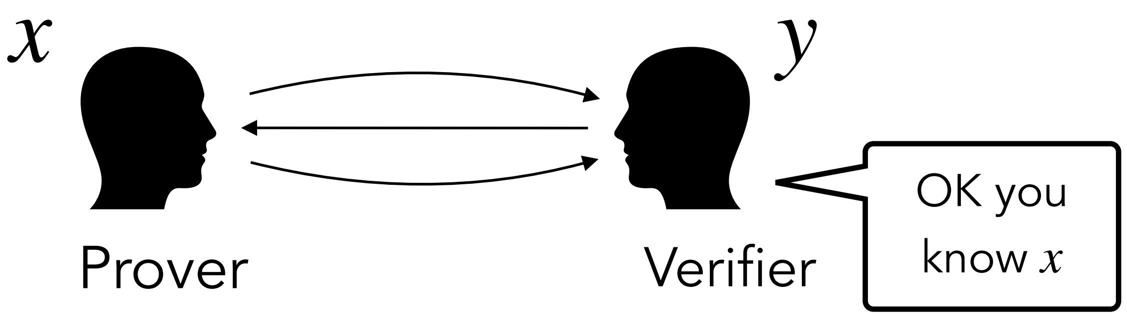
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof



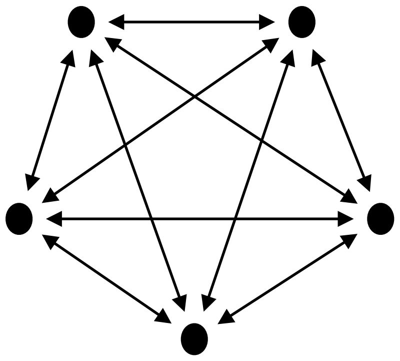
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ s.t. $x = [\![x]\!]_1 + \ldots + [\![x]\!]_N$

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

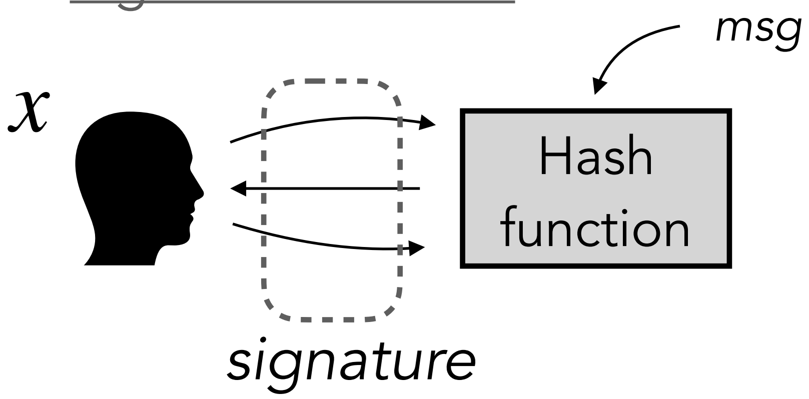


Input sharing $[\![x]\!]$

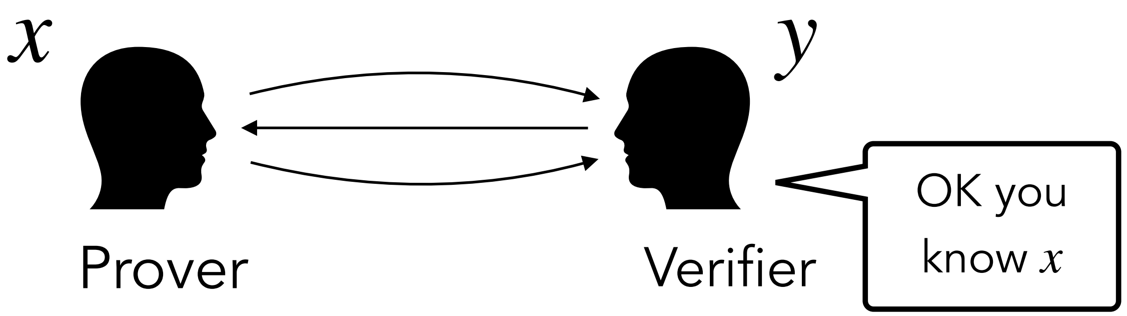
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

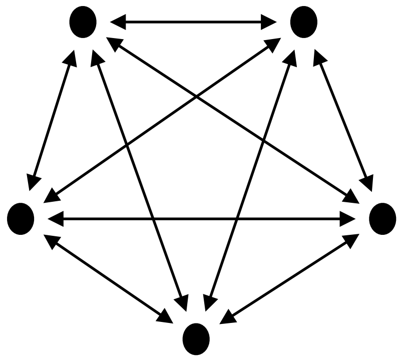


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

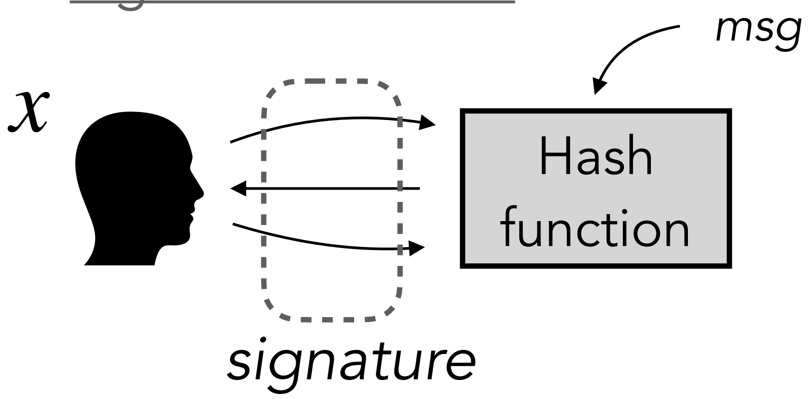


Input sharing $[\![x]\!]$

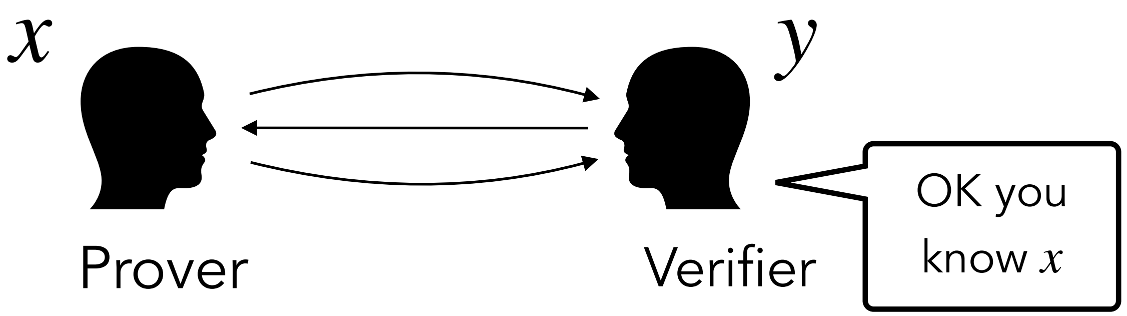
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

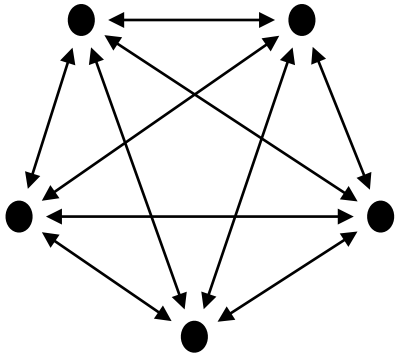


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

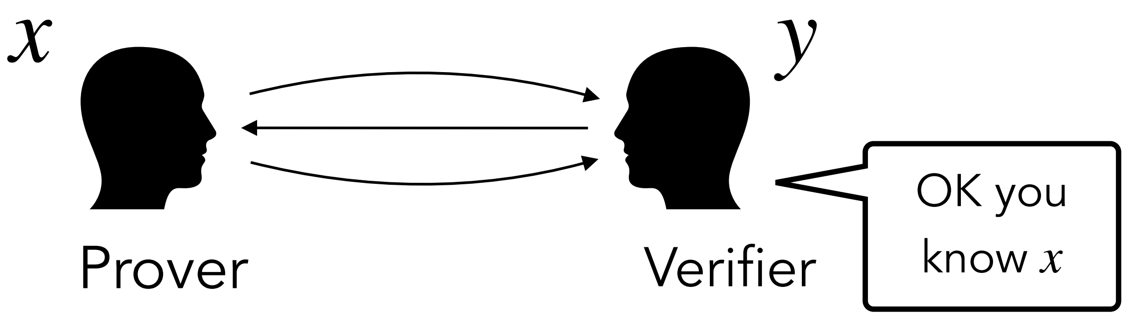


Input sharing $[\![x]\!]$

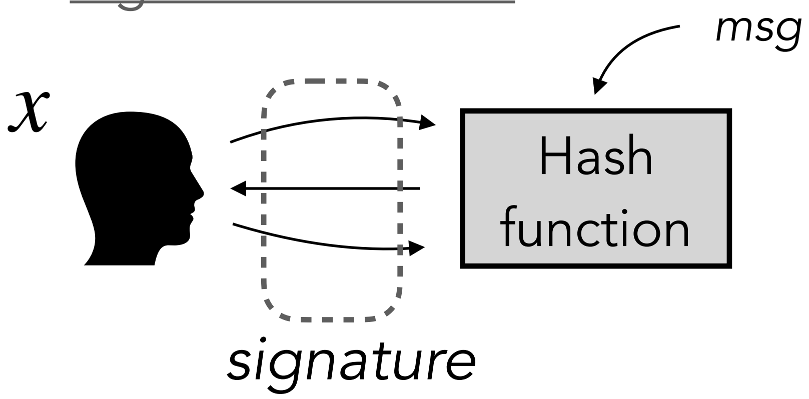
Joint evaluation of:

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

Zero-knowledge proof



Signature scheme

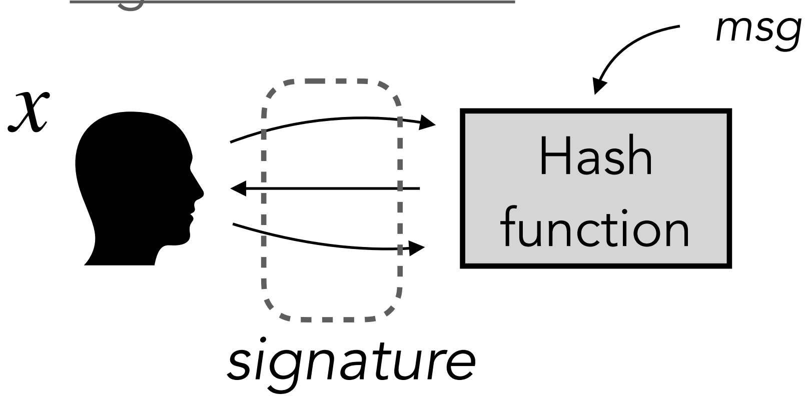


One-way function

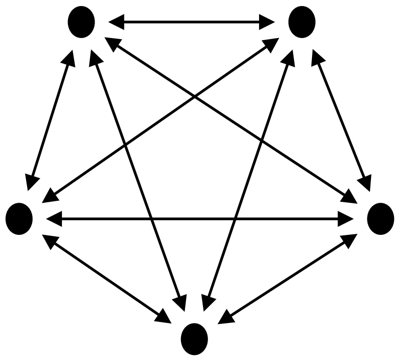
$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Signature scheme



Multiparty computation (MPC)



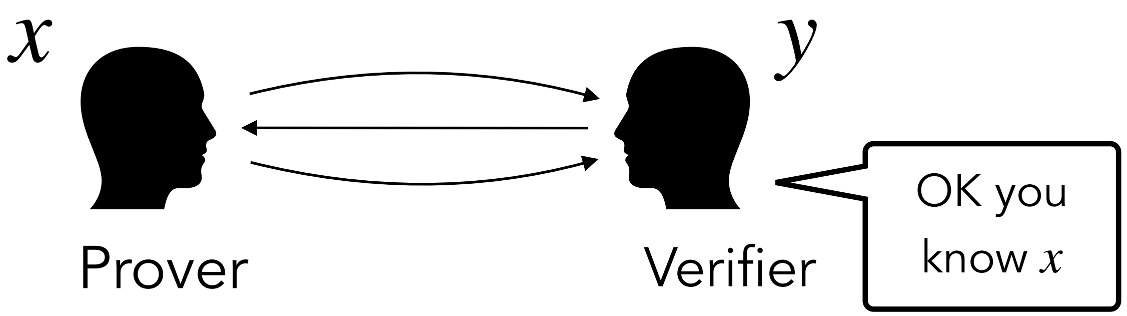
Input sharing $[\![x]\!]$

Joint evaluation of:

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

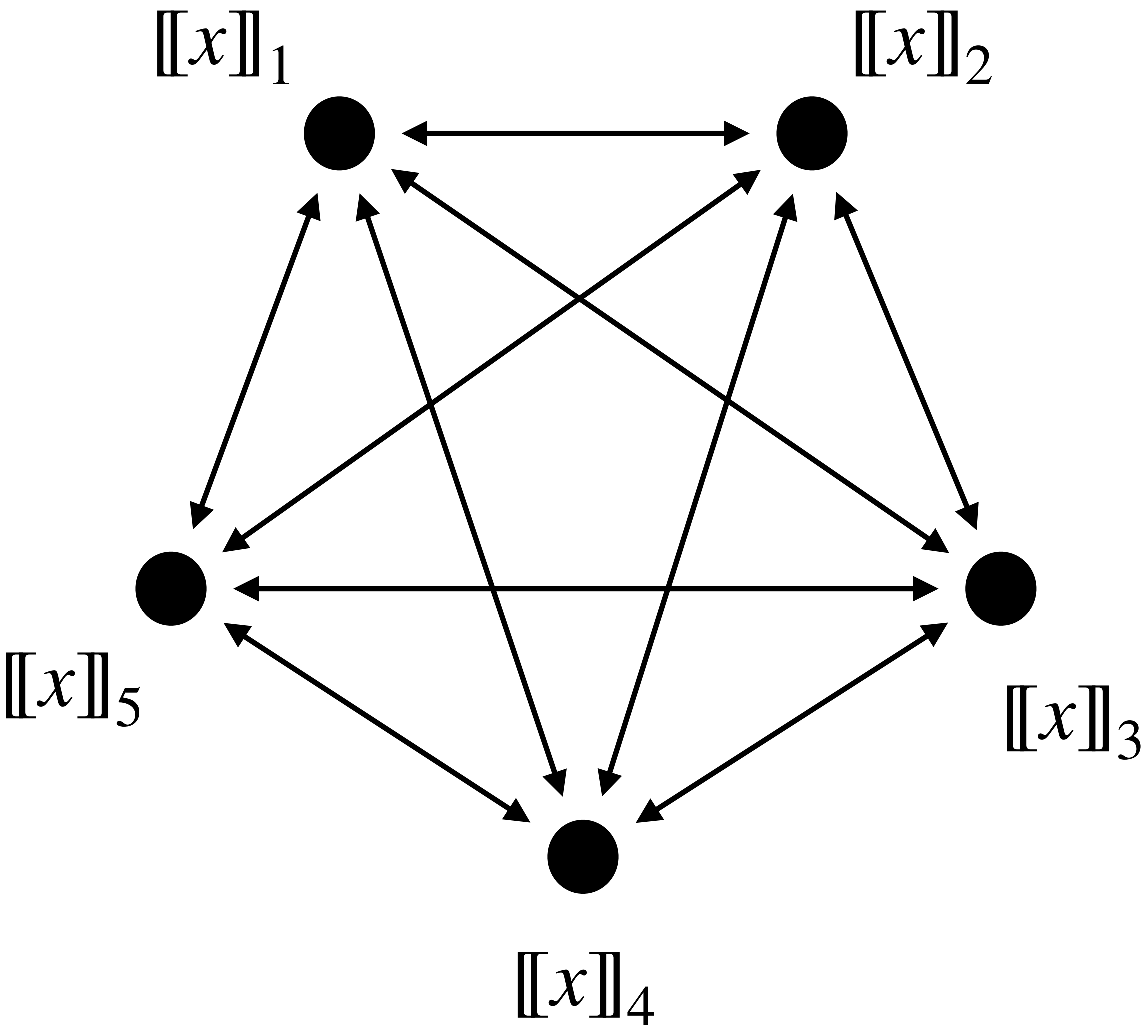
MPC-in-the-Head transform

Zero-knowledge proof



MPCitH: general principle

MPC model



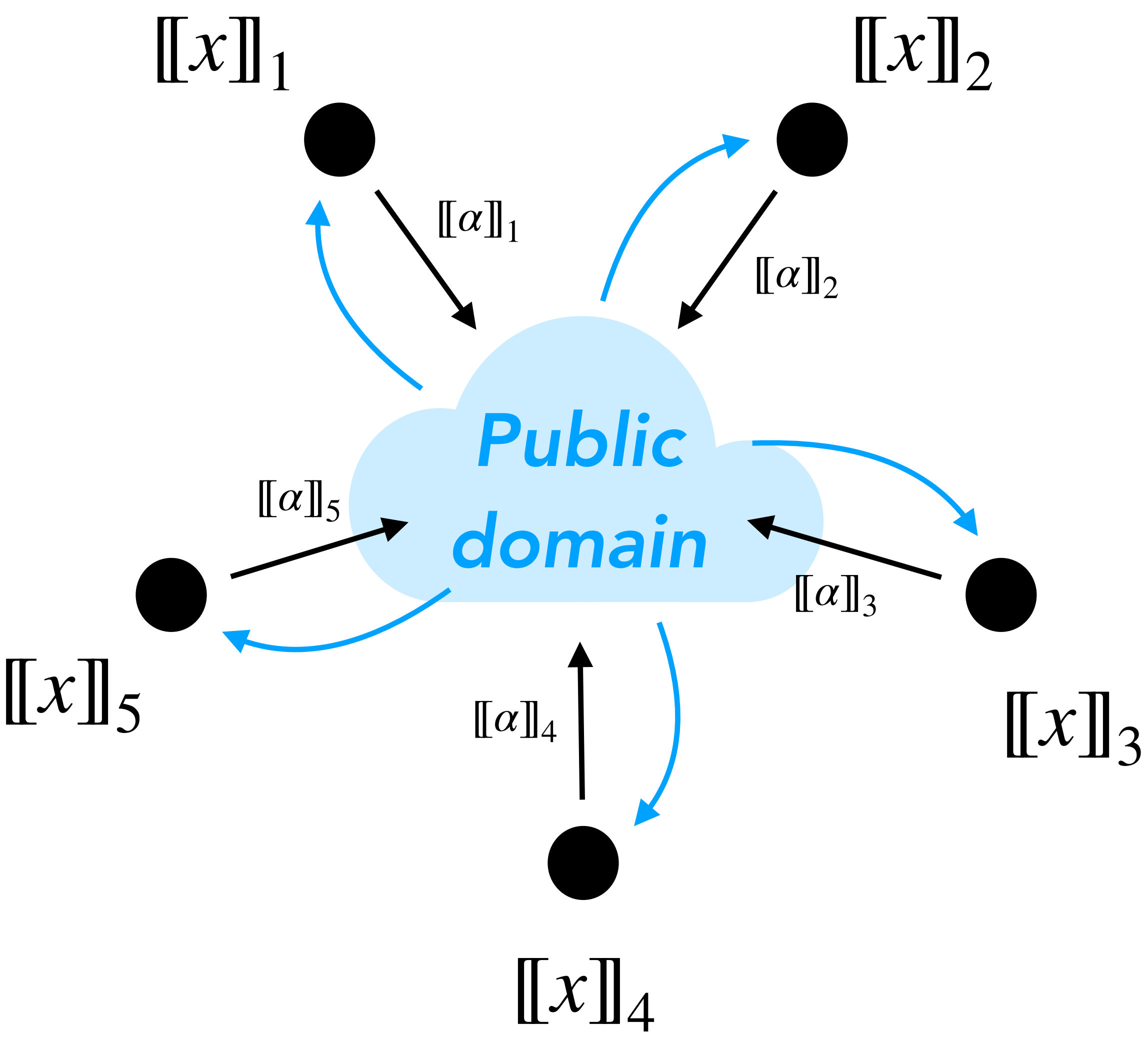
$$x = [\![x]\!]_1 + [\![x]\!]_2 + \ldots + [\![x]\!]_N$$

- **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

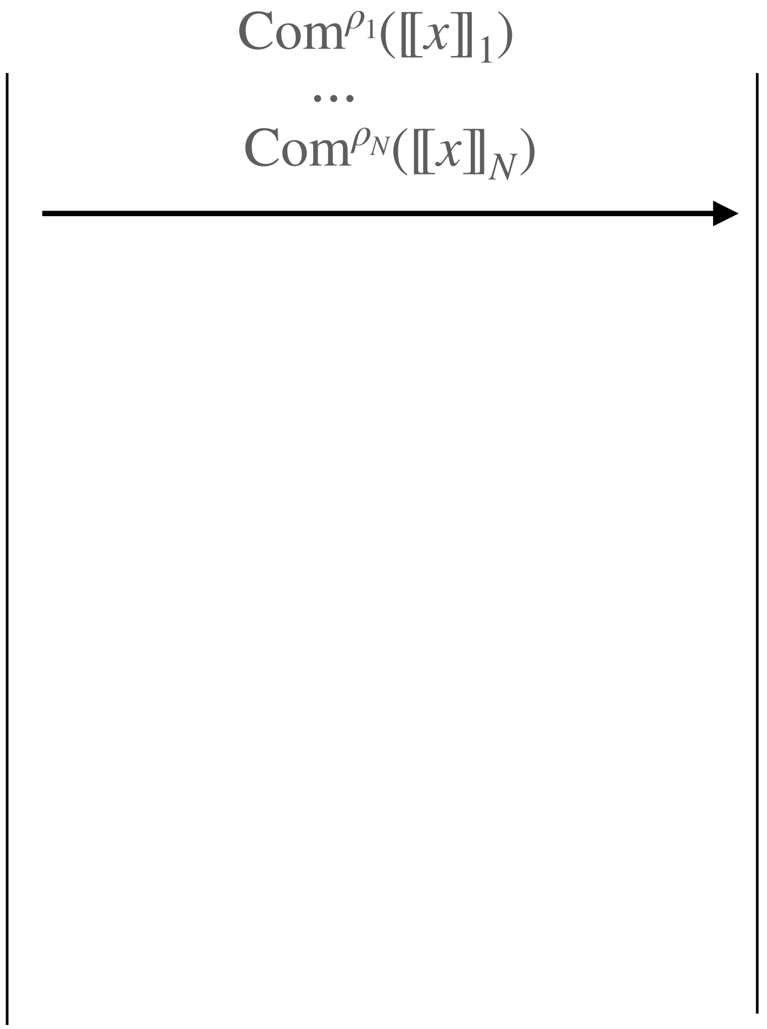
Prover

Verifier

MPCitH transform

① Generate and commit shares

$$\llbracket x \rrbracket = (\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$$



$\mathrm{Com}^{\rho_1}(\llbracket x \rrbracket_1)$

$\cdots$

$\mathrm{Com}^{\rho_N}(\llbracket x \rrbracket_N)$

Prover

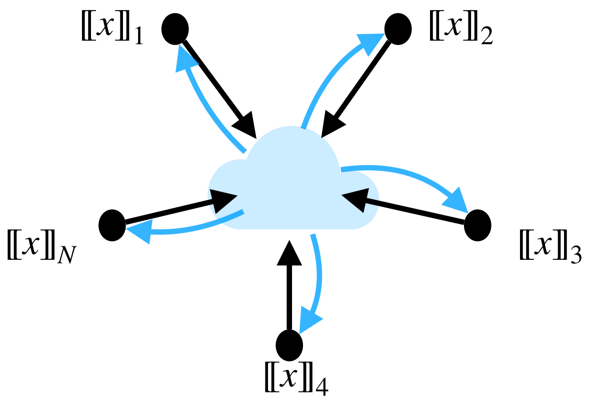
Verifier

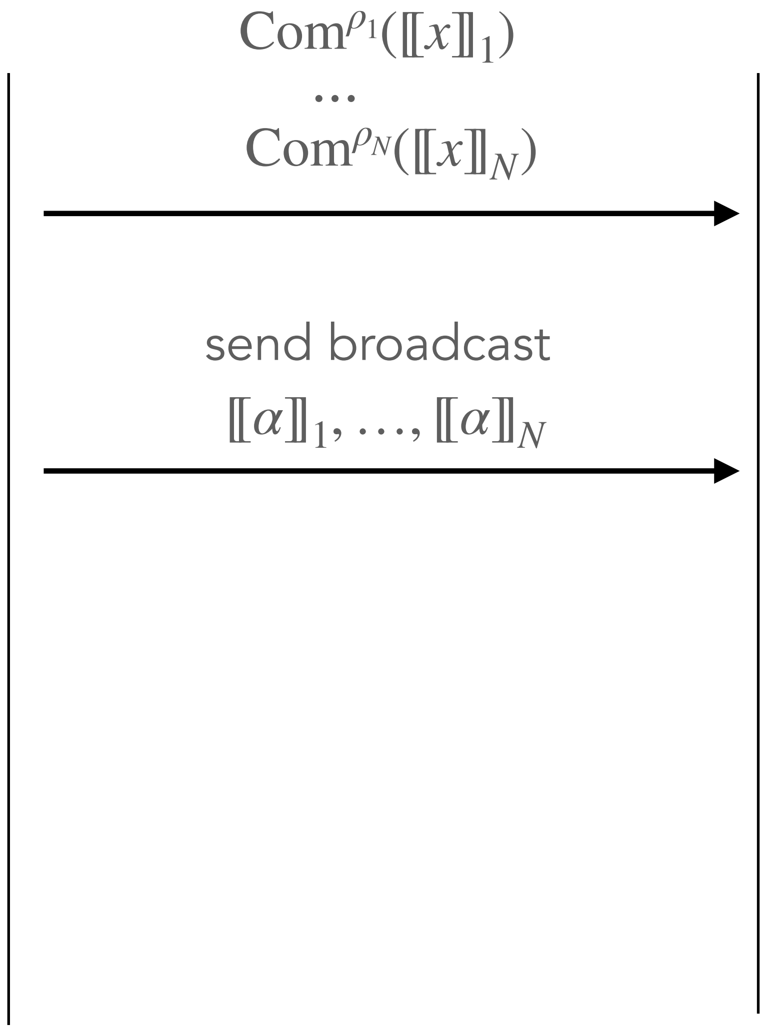
MPCitH transform

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

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

Prover

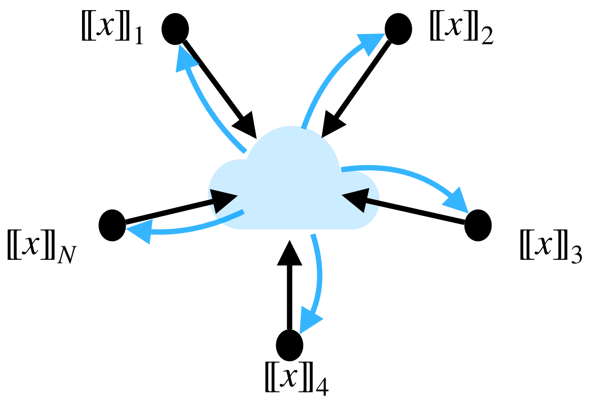
Verifier

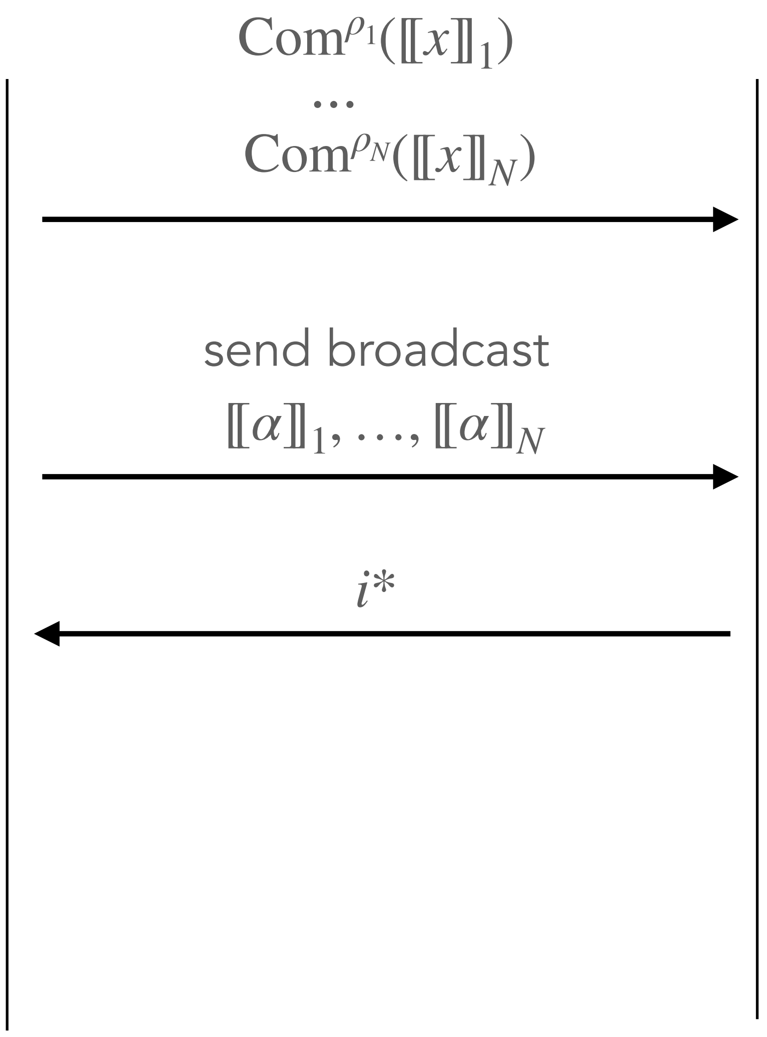
MPCitH transform

① Generate and commit shares

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

② Run MPC in their head





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

$\ldots$

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

send broadcast

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

i^*

③ Choose a random party

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

Prover

Verifier

MPCitH transform

① Generate and commit shares

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

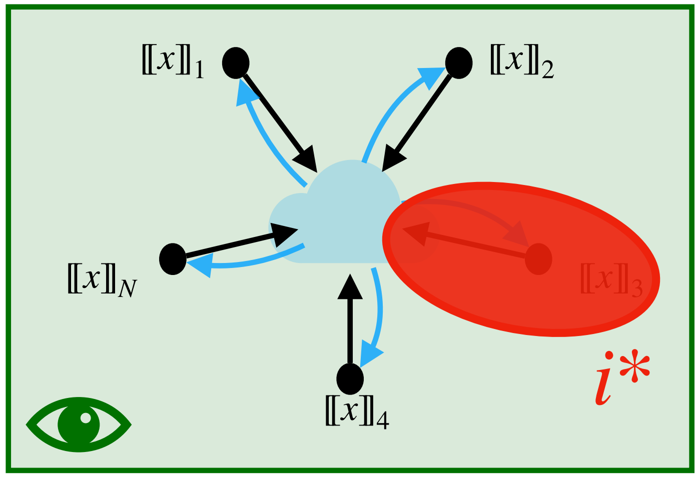
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

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

→

② Run MPC in their head



send broadcast

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

→

③ Choose a random party

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

i^*

←

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

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

→

Prover

Verifier

MPCitH transform

① Generate and commit shares

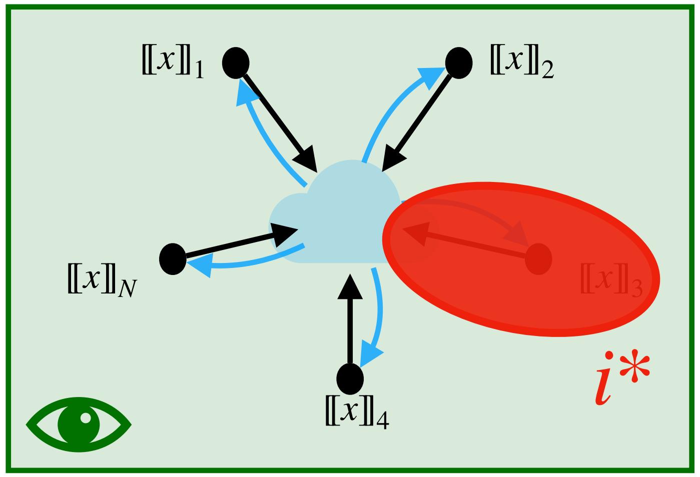
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

$\ldots$

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

② Run MPC in their head



send broadcast

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

③ Choose a random party

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

← i^*

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

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

⑤ Check $\forall i \neq i^*$

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

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

Prover

Verifier

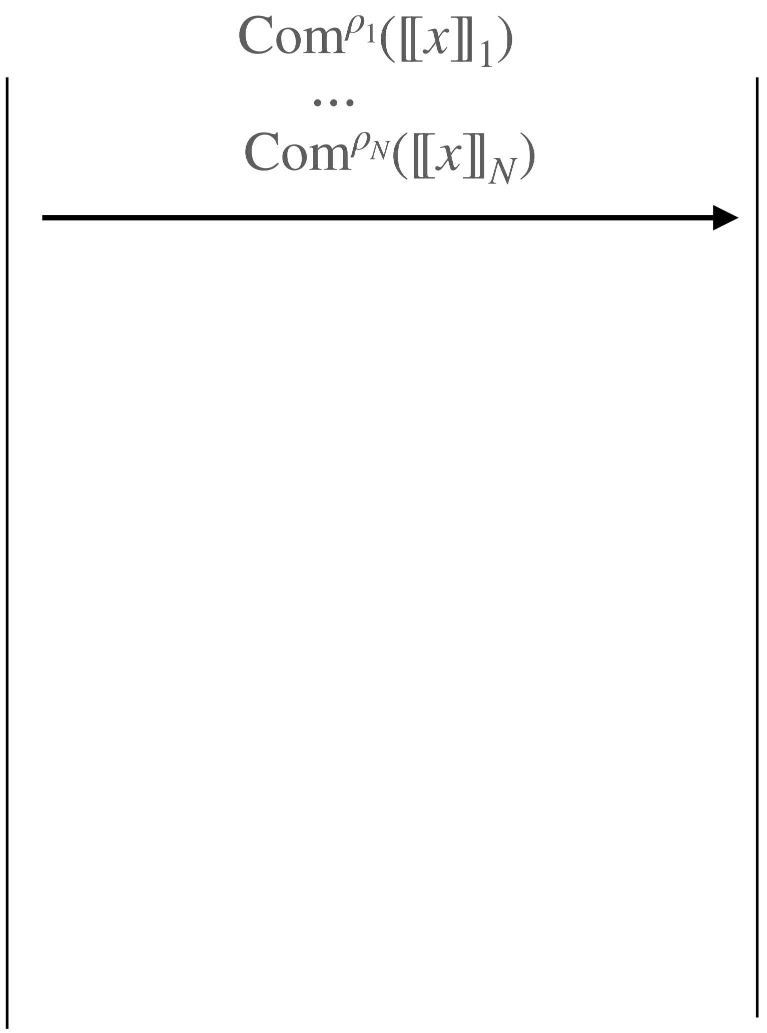
MPCitH transform

① Generate and commit shares

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

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

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



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

$\dots$

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

Malicious Prover

Verifier

MPCitH transform

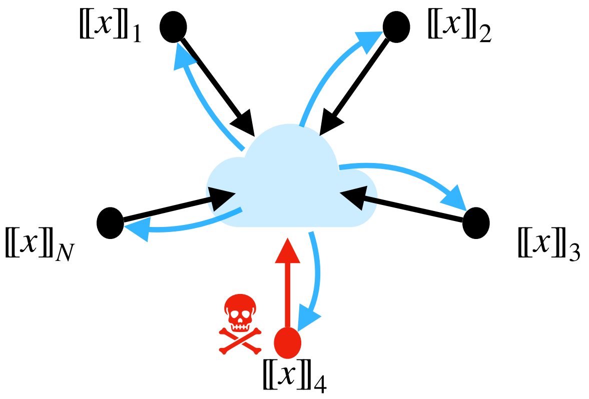
① Generate and commit shares

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

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

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

② Run MPC in their head



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

$\ldots$

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

→

send broadcast

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

→

Malicious Prover

Verifier

MPCitH transform

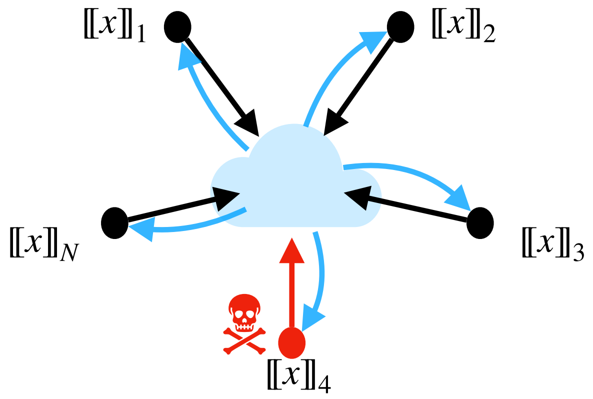
① Generate and commit shares

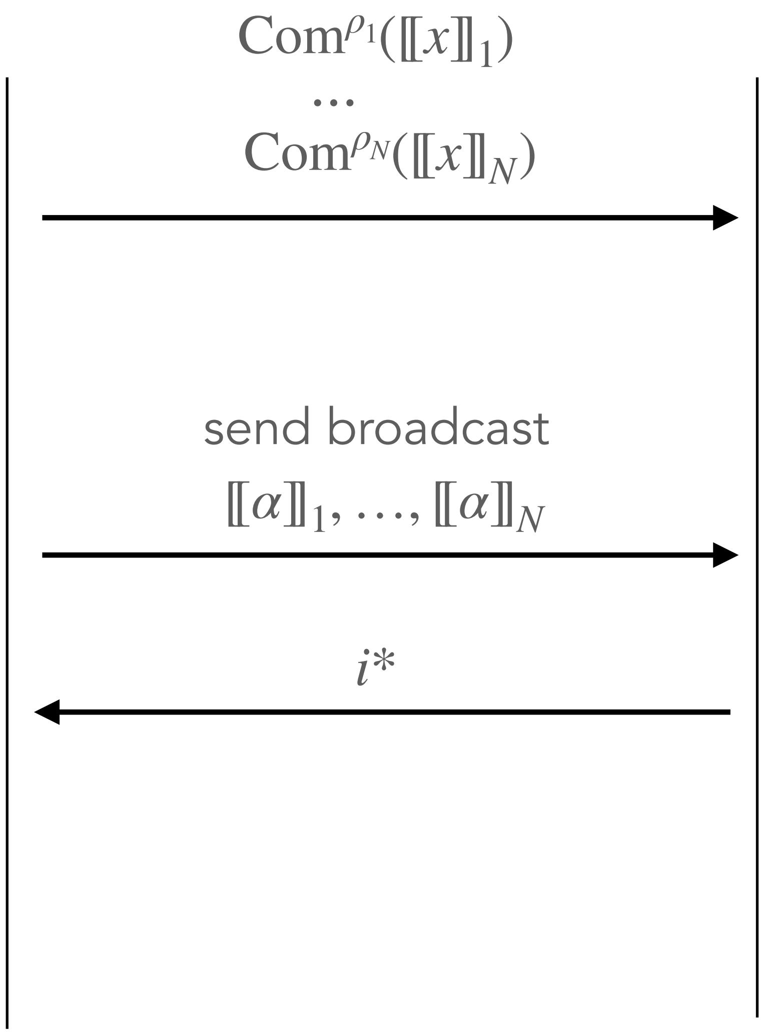
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

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

② Run MPC in their head





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

$\cdots$

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

send broadcast

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

i^*

③ Choose a random party

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

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

$\llbracket x \rrbracket = (\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_N)$

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

$x := \llbracket x \rrbracket_1 + \ldots + \llbracket x \rrbracket_N$

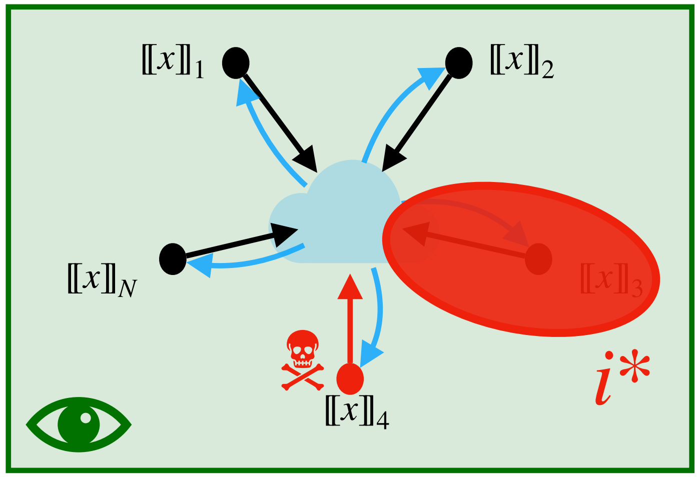
$\mathrm{Com}^{\rho_1}(\llbracket x \rrbracket_1)$

…

$\mathrm{Com}^{\rho_N}(\llbracket x \rrbracket_N)$

→

② Run MPC in their head


$\llbracket x \rrbracket_1$
$\llbracket x \rrbracket_2$
$\llbracket x \rrbracket_N$
$\llbracket x \rrbracket_3$
$\llbracket x \rrbracket_4$
i^*

send broadcast

$\llbracket \alpha \rrbracket_1, \ldots, \llbracket \alpha \rrbracket_N$

→

③ Choose a random party

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

i^*

←

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

$(\llbracket x \rrbracket_i, \rho_i)_{i \neq i^*}$

→

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

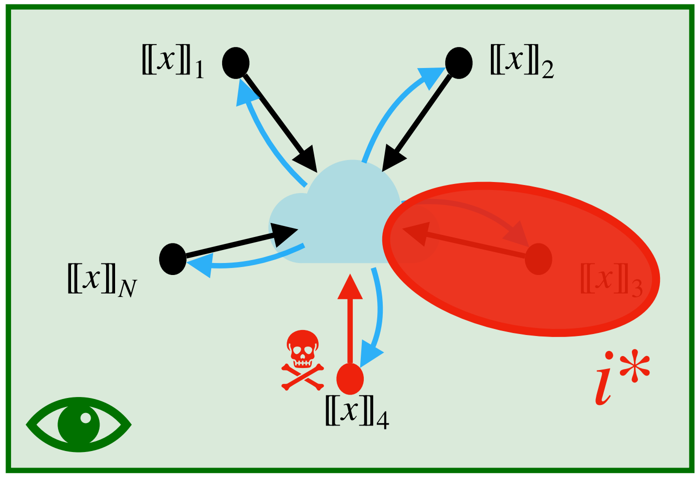
$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

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

…

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

② Run MPC in their head



send broadcast

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

③ Choose a random party

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

i^*

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

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

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

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

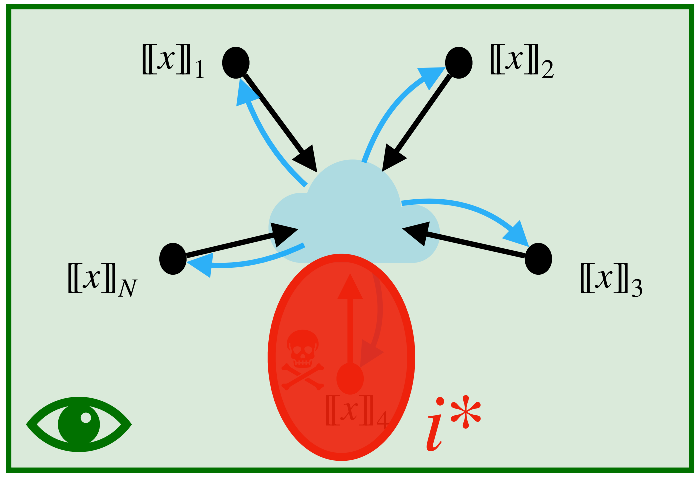
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\ldots$

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

→

② Run MPC in their head



send broadcast

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

→

③ Choose a random party

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

i^*

←

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

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

→

⑤ Check $\forall i \neq i^*$

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

Check $g(y, \alpha) =$ 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 $\left(\frac{1}{N}\right)^{\tau}$

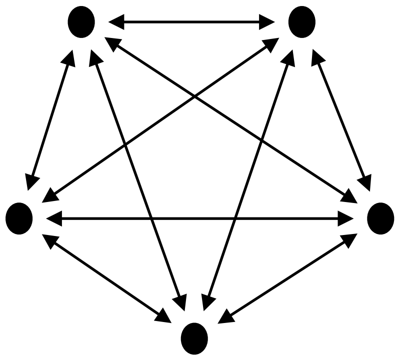
From MPC-in-the-Head to signatures

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

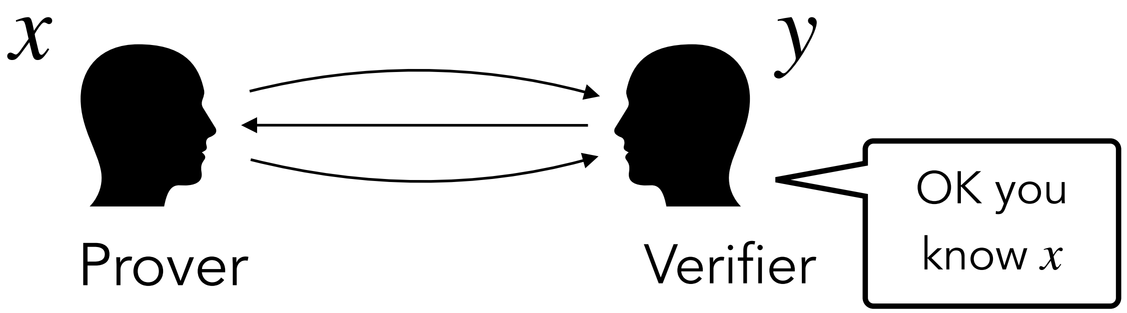


Input sharing $[\![x]\!]$

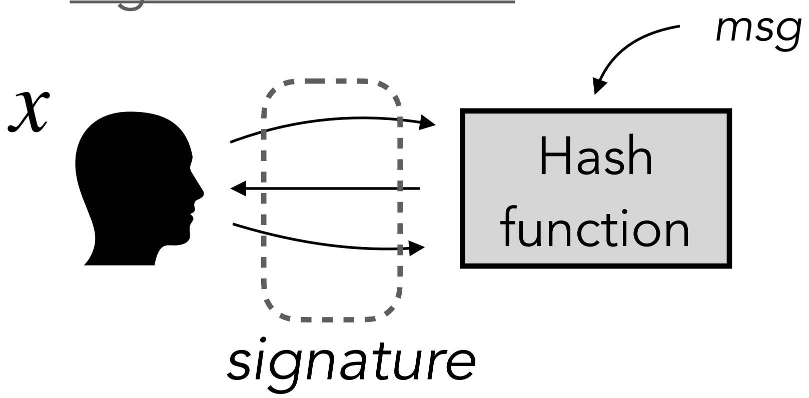
Joint evaluation of:

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

Zero-knowledge proof



Signature scheme

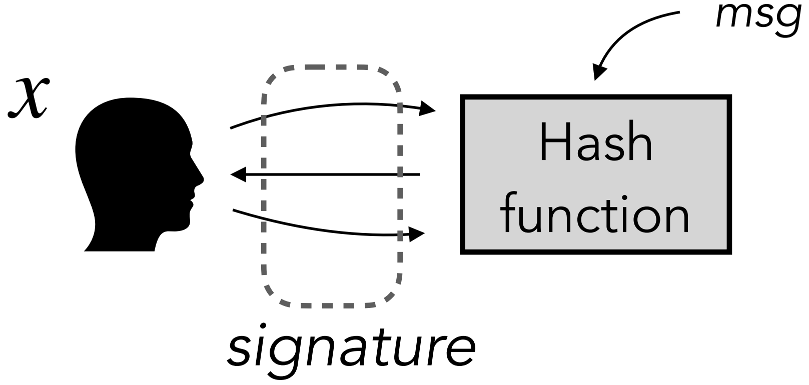


One-way function

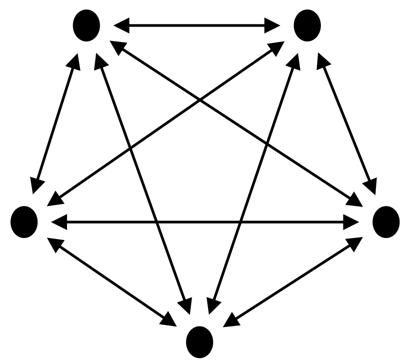
$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Signature scheme



Multiparty computation (MPC)



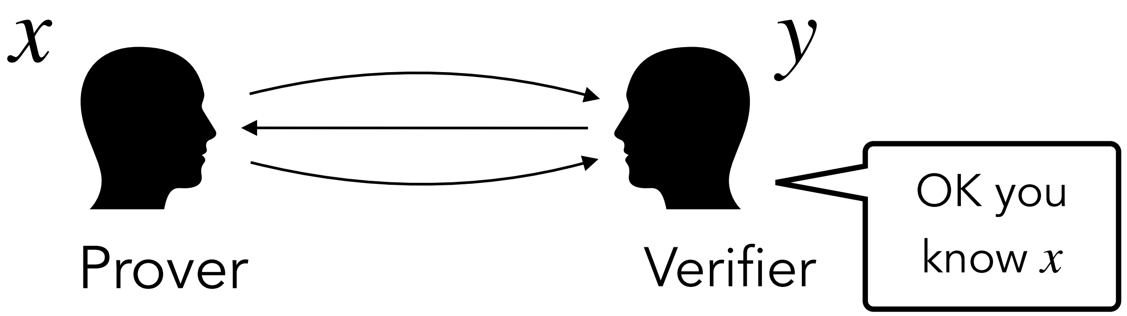
Input sharing $[\![x]\!]$

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

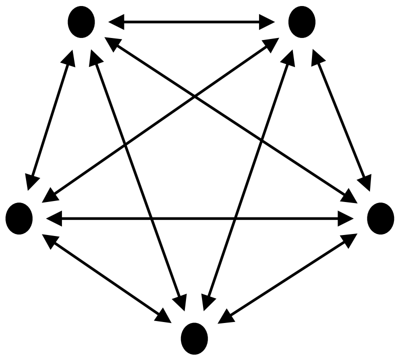


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

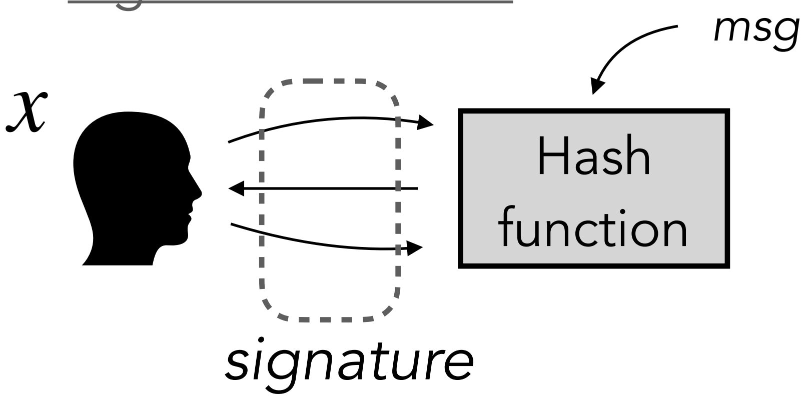


Input sharing $[\![x]\!]$

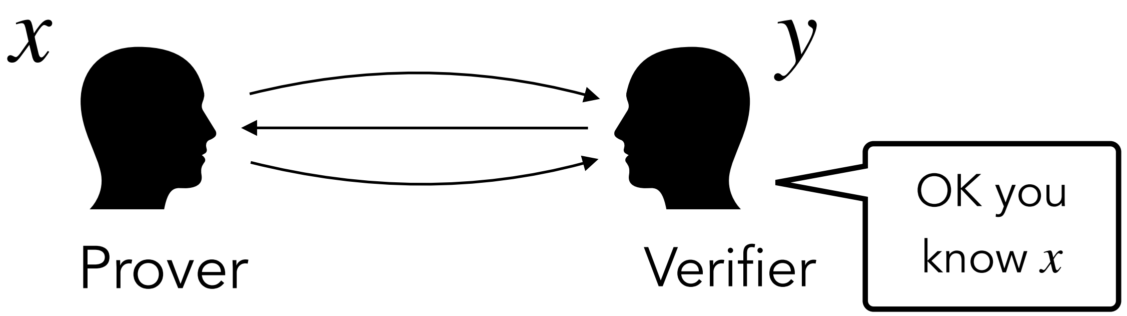
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof



One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Three approaches:

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

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Three approaches:

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

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Three approaches:

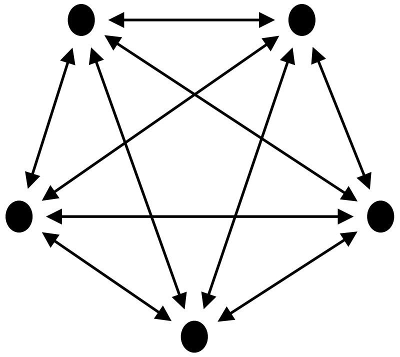
- Rely on standard symmetric primitives
- Rely on MPC-friendly symmetric primitives
- Rely on well-known hard problems *(non-exhaustive list)*
  - Syndrome Decoding: *SDitH* (2022), *RYDE* (2023)
  - MinRank: *MiRitH* (2022), *MIRA* (2023)
  - Multivariate Quadratic: *MQOM* (2023), *Biscuit* (2023)
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

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Three approaches:

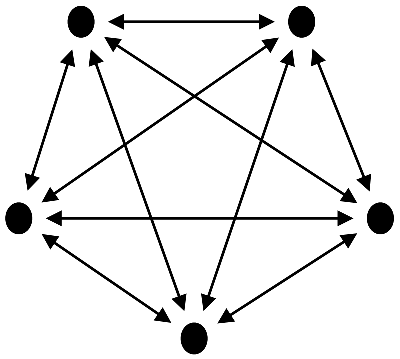
- Rely on standard symmetric primitives
- Rely on MPC-friendly symmetric primitives
- Rely on well-known hard problems

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Three approaches:

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

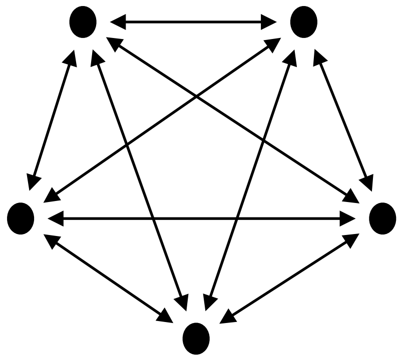
Expressed as an arithmetic circuit, enabling us to use existing MPCitH-based proof systems (as BN++)

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Three approaches:

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

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

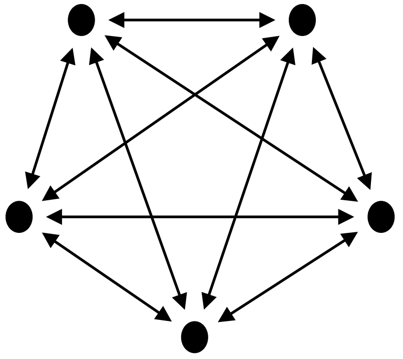
Should be rephrased to achieve interesting performances

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



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

Joint evaluation of:

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

Three approaches:

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

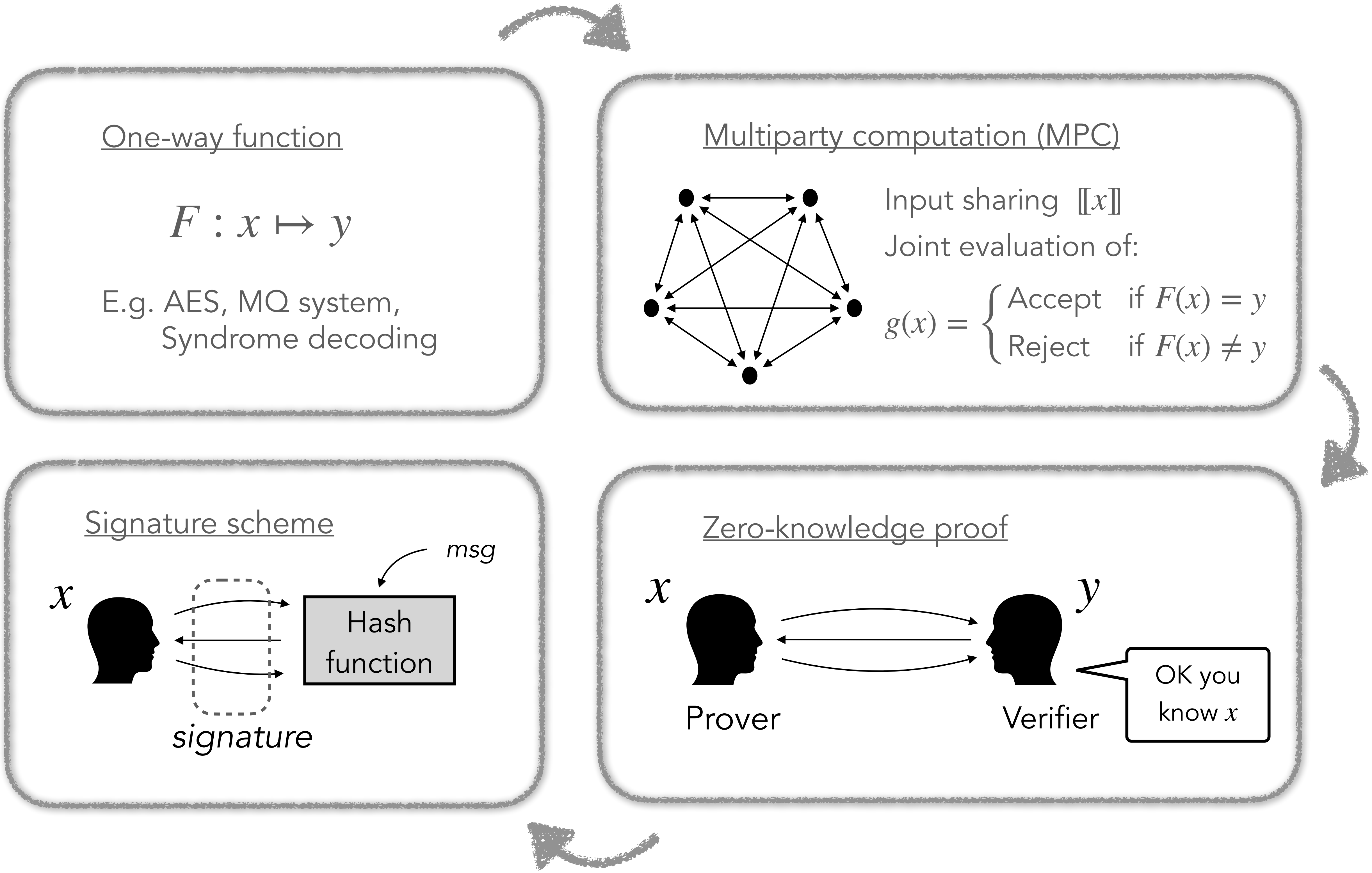
Expressed as an arithmetic circuit, enabling us to use existing MPCitH-based proof systems (as BN++)

Should be rephrased to achieve interesting performances

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

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

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



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

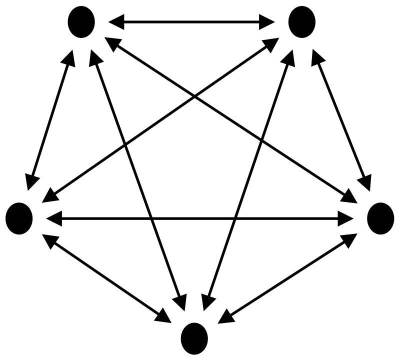
Signature scheme

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

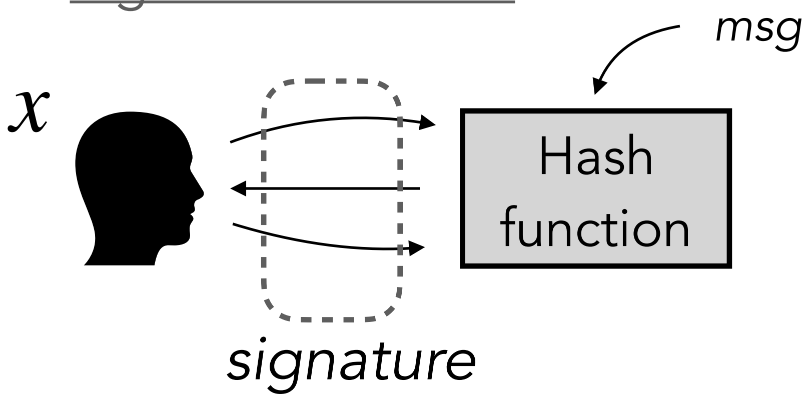


Input sharing $[\![x]\!]$

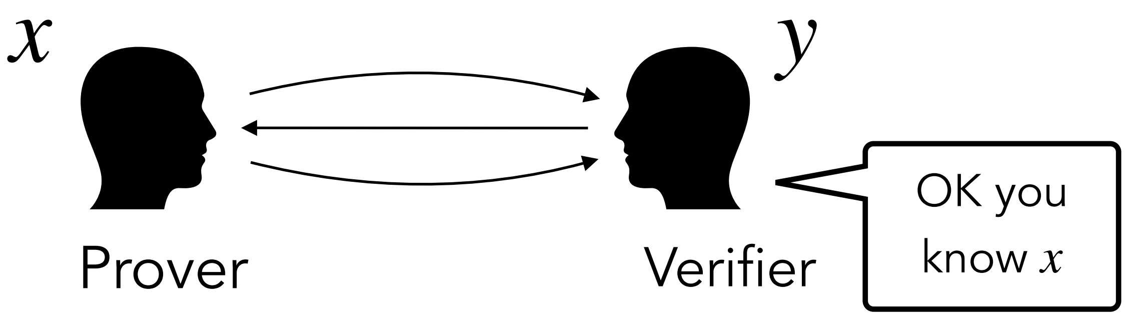
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof



Fiat-Shamir transform

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

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

Optimisations and variants

Optimisations and variants

Field $GF(251)$

With `SDitH-L1-gf251` as example.

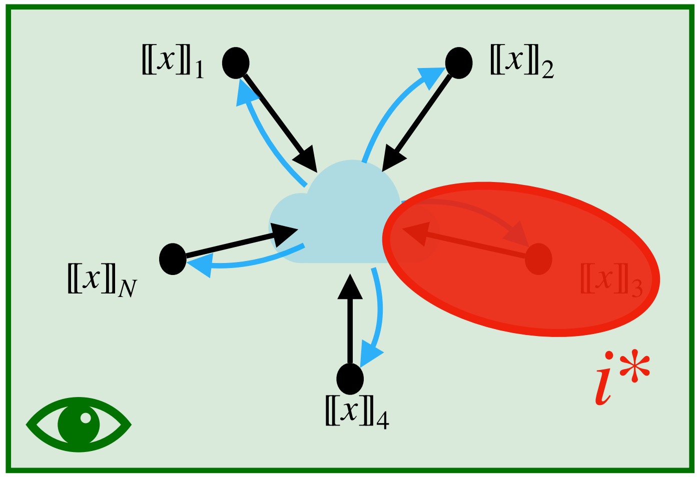
NIST Category I

MPCitH transform

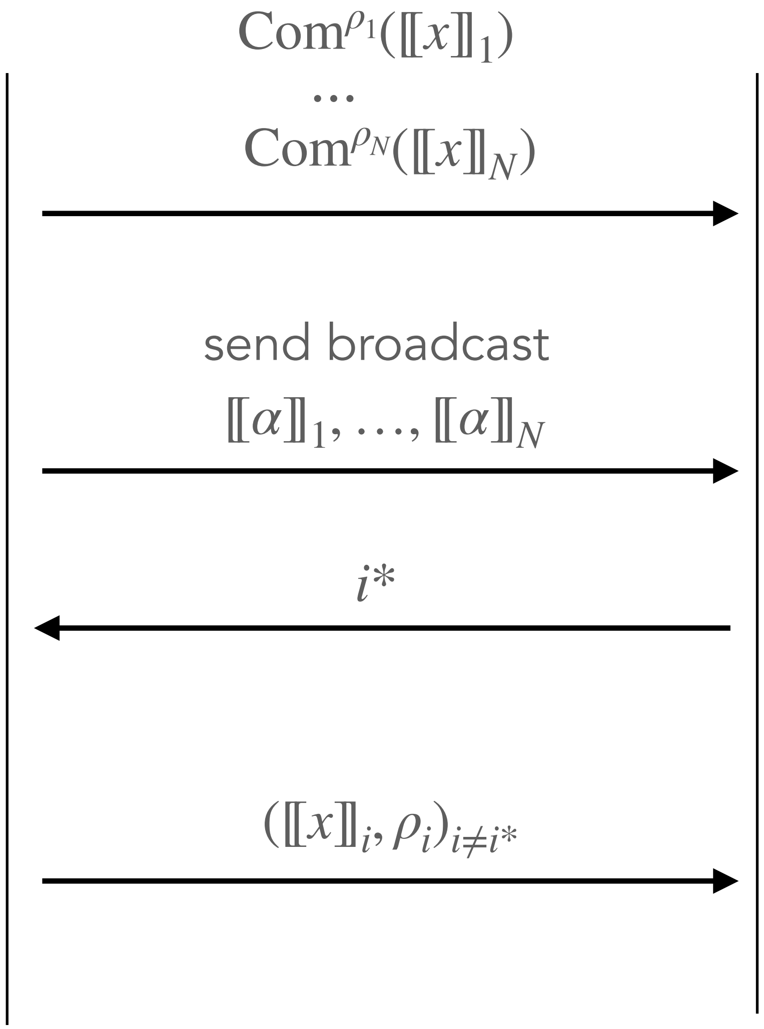
① Generate and commit shares

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

② Run MPC in their head



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


$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$
$\cdots$
$\mathrm{Com}^{\rho_N}([\![x]\!]_N)$
send broadcast
$[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$
i^*
$([\![x]\!]_i, \rho_i)_{i \neq i^*}$

③ Choose a random party

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

⑤ Check $\forall i \neq i^*$

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

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

Prover

Verifier

Naive MPCitH transformation

Size of the broadcast (per party)

Size of a commitment digest

Size of the MPC input (per party)

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

Number of repetitions to achieve the desired security level

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

Naive MPCitH transformation

Size of the broadcast (per party)

Size of a commitment digest

Size of the MPC input (per party)

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

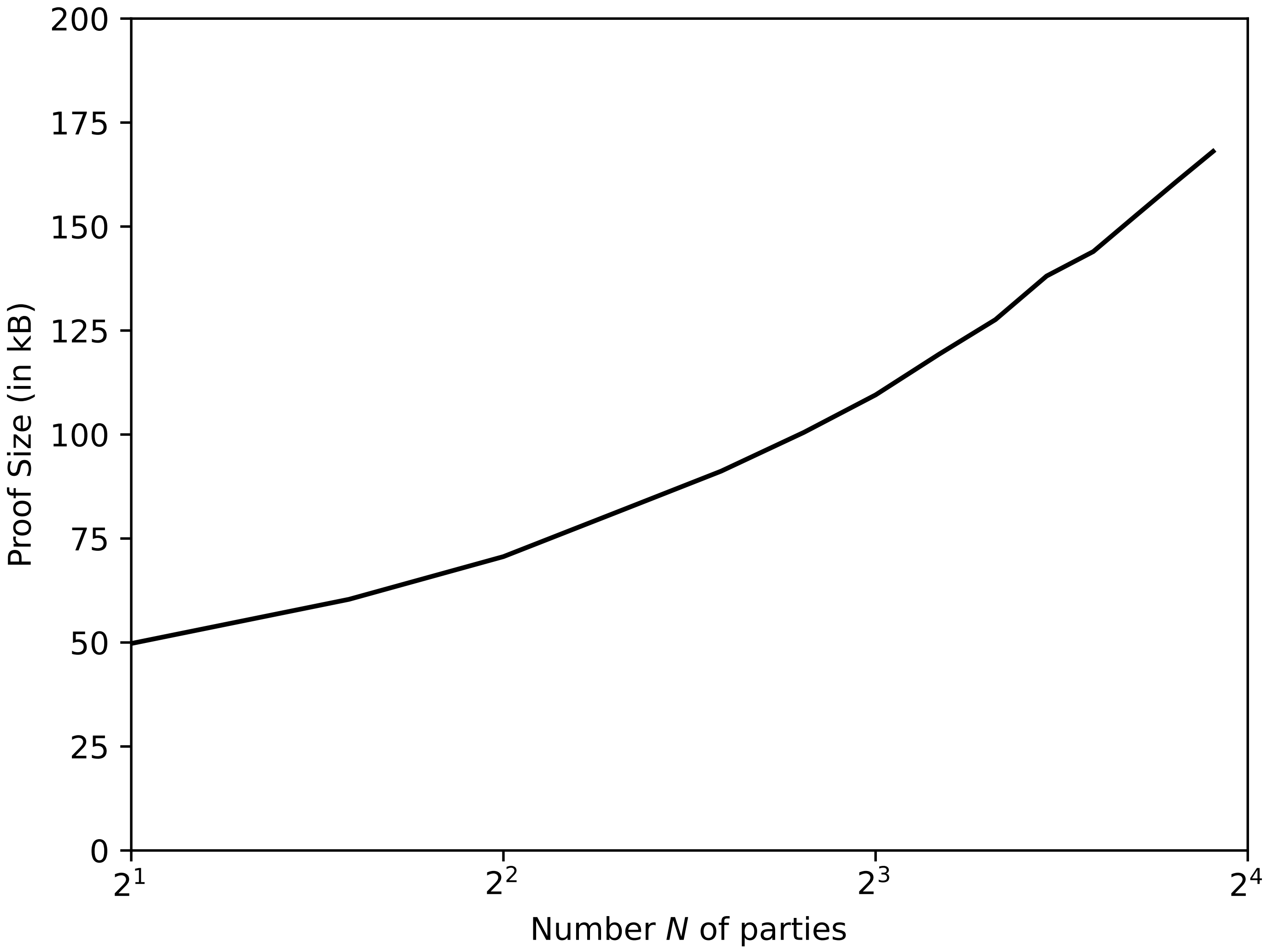
Number of repetitions to achieve the desired security level

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

`SDitH-L1-gf251:`

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

Naive MPCitH transformation



`SDitH-L1-gf251:`

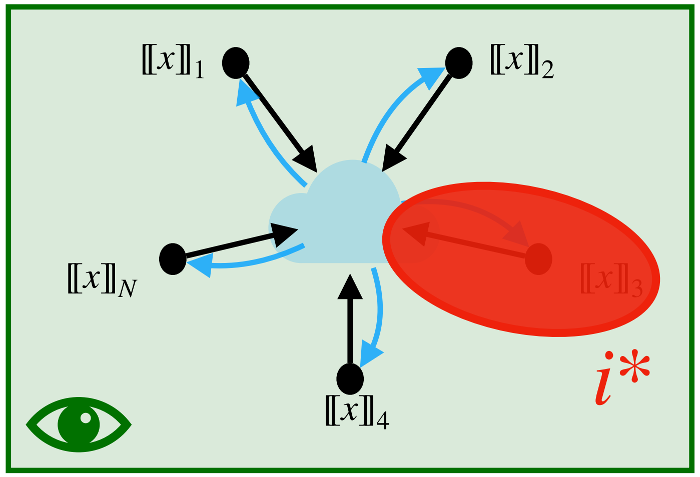
the input x of the MPC protocol is around **323** bytes,
The broadcast value α of the MPC protocol is around **36** bytes

MPCitH transform

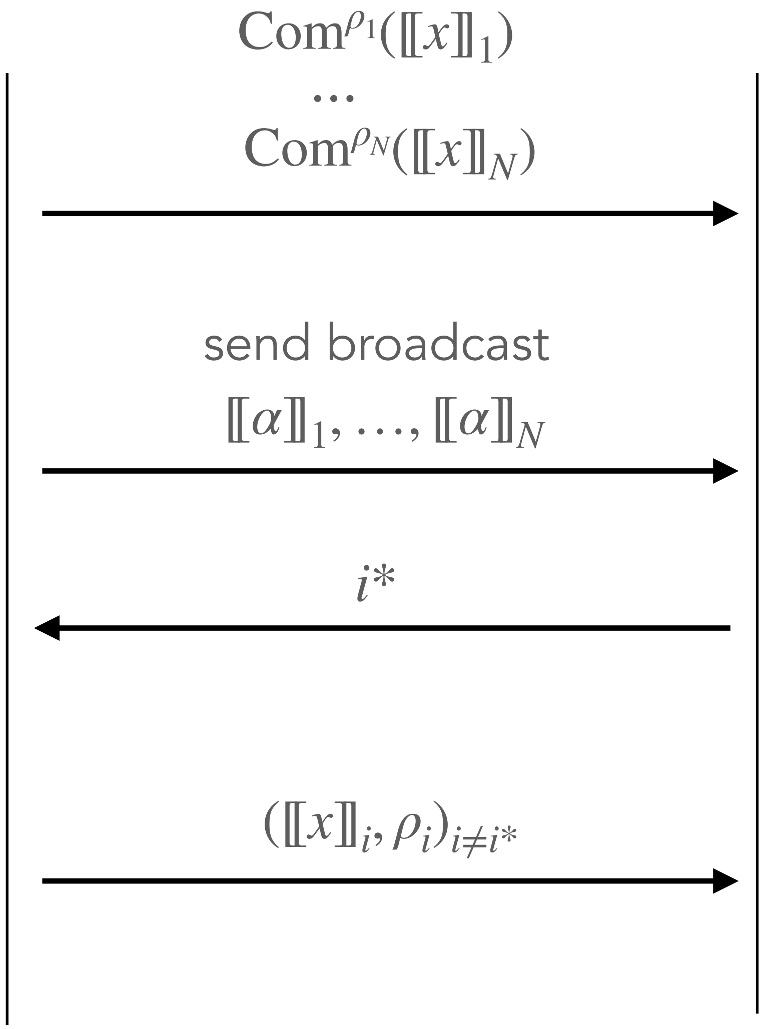
① Generate and commit shares

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

② Run MPC in their head



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



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

$\ldots$

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

send broadcast

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

i^*

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

③ Choose a random party

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

⑤ Check $\forall i \neq i^*$

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

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

Prover

Verifier

MPCitH transform

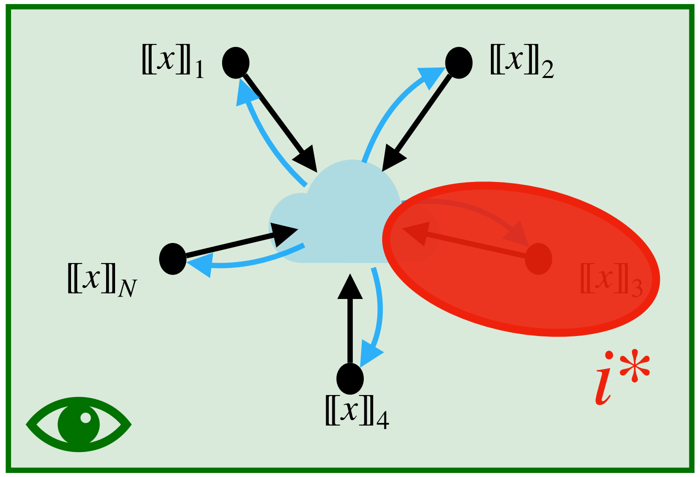
① Generate and commit shares

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

Compute

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

② Run MPC in their head



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

Prover

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

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

← i^*

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

③ Choose a random party

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

⑤ Compute $\forall i \neq i^*$

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

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

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

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

Verifier

MPCitH transform

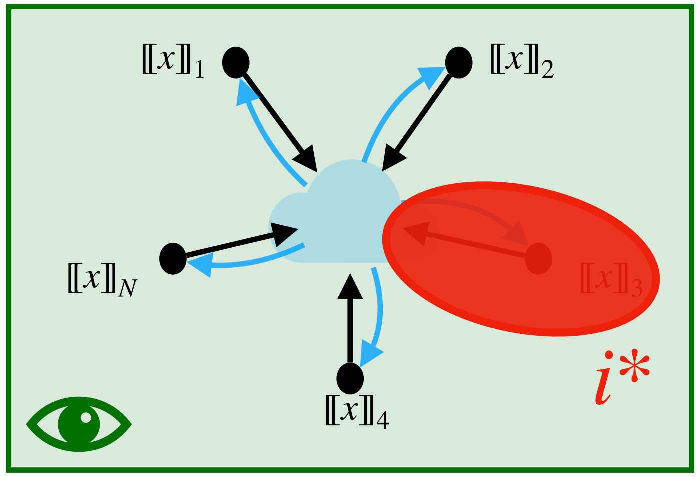
① Generate and commit shares

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

Compute

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

② Run MPC in their head



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

Prover

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

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

← i^*

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

③ Choose a random party

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

⑤ Compute $\forall i \neq i^*$

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

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

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

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

Verifier

Using a Seed Tree

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

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

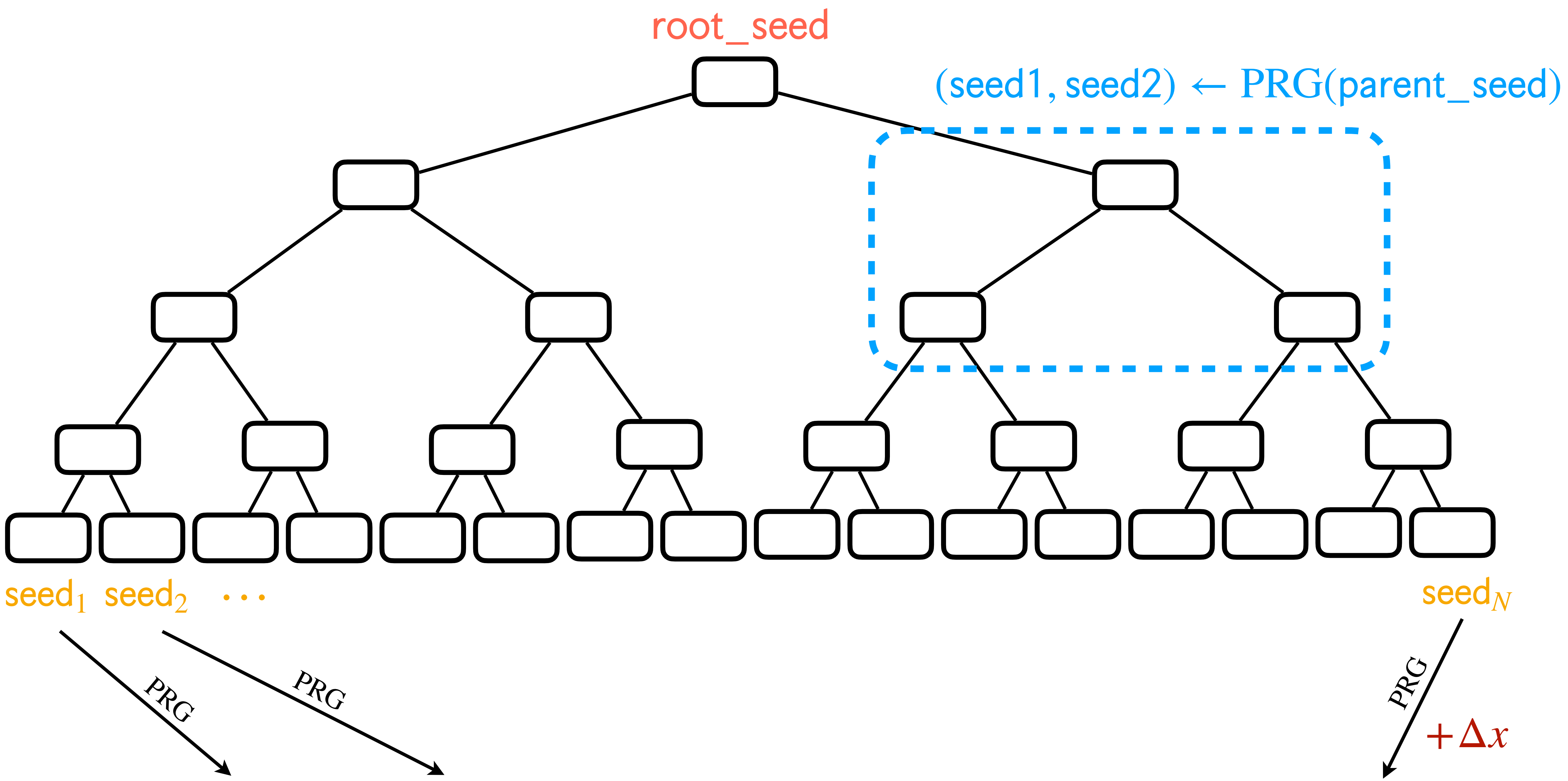
Using a Seed Tree

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

$$\begin{tikzcd}
 & \mathrm{seed}_1 \arrow[d, "\mathrm{PRG}"'] & & \mathrm{seed}_2 \arrow[d, "\mathrm{PRG}"'] & & \mathrm{seed}_3 \arrow[d, "\mathrm{PRG}"'] & & & & \mathrm{seed}_{N-1} \arrow[d, "\mathrm{PRG}"'] & & \mathrm{seed}_N \arrow[d, "\mathrm{PRG}"', "+\Delta x"] \\
x = & \llbracket x \rrbracket_1 & + & \llbracket x \rrbracket_2 & + & \llbracket x \rrbracket_3 & + & \ldots & + & \llbracket x \rrbracket_{N-1} & + & \llbracket x \rrbracket_N
\end{tikzcd}$$

Using a Seed Tree

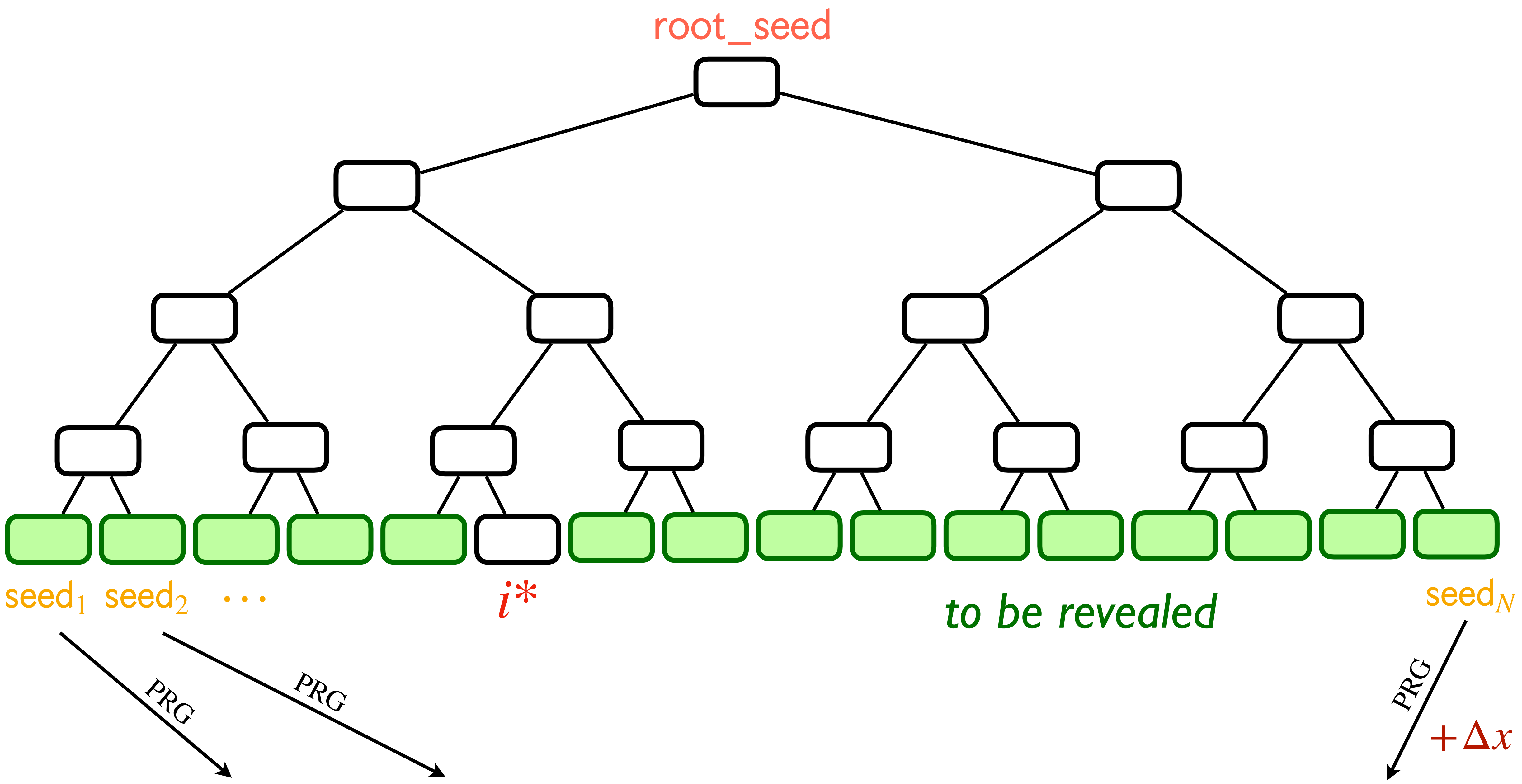
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



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

Using a Seed Tree

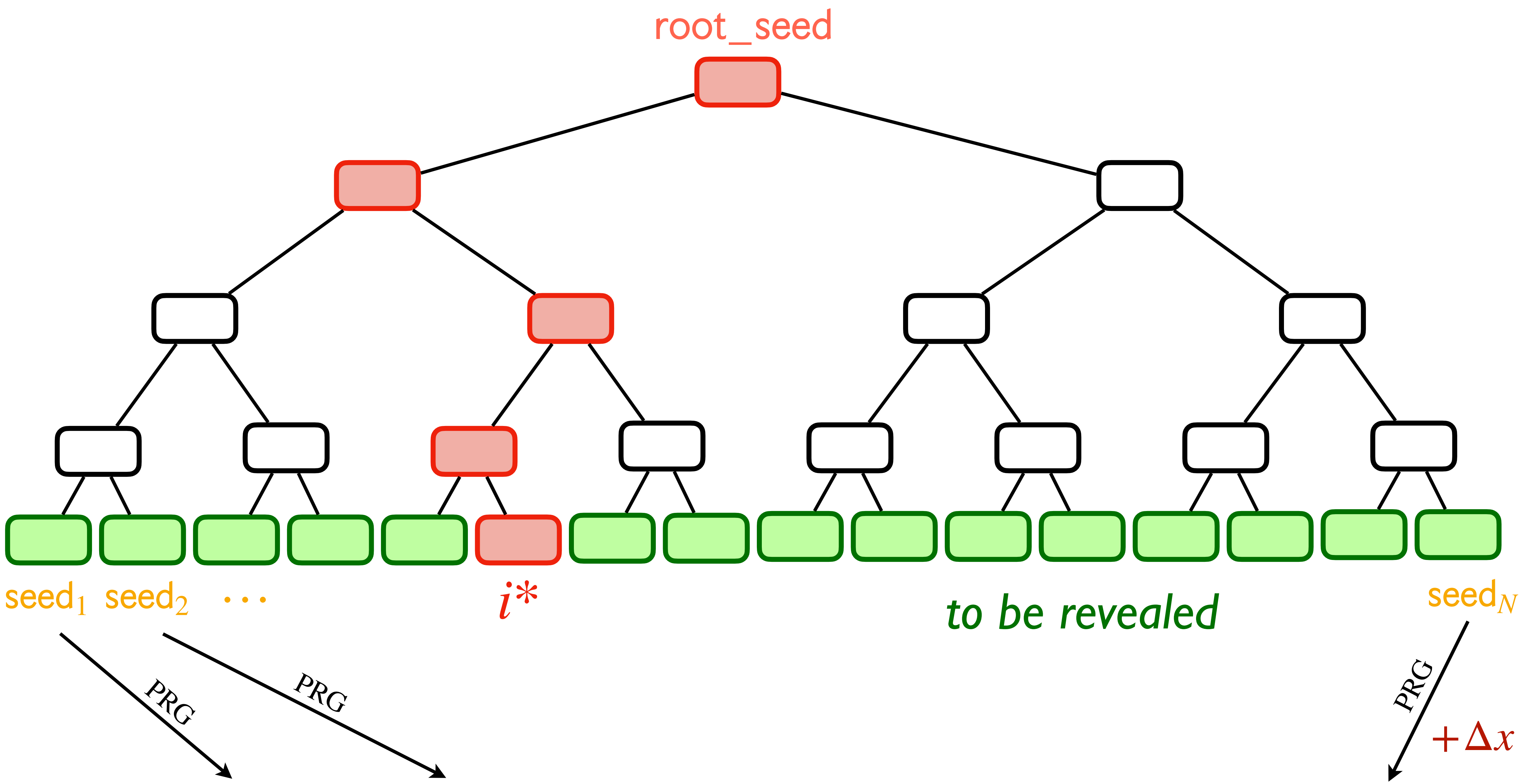
[KKW18] Katz, Kolesnikov, Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures" (CCS 2018)



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

Using a Seed Tree

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



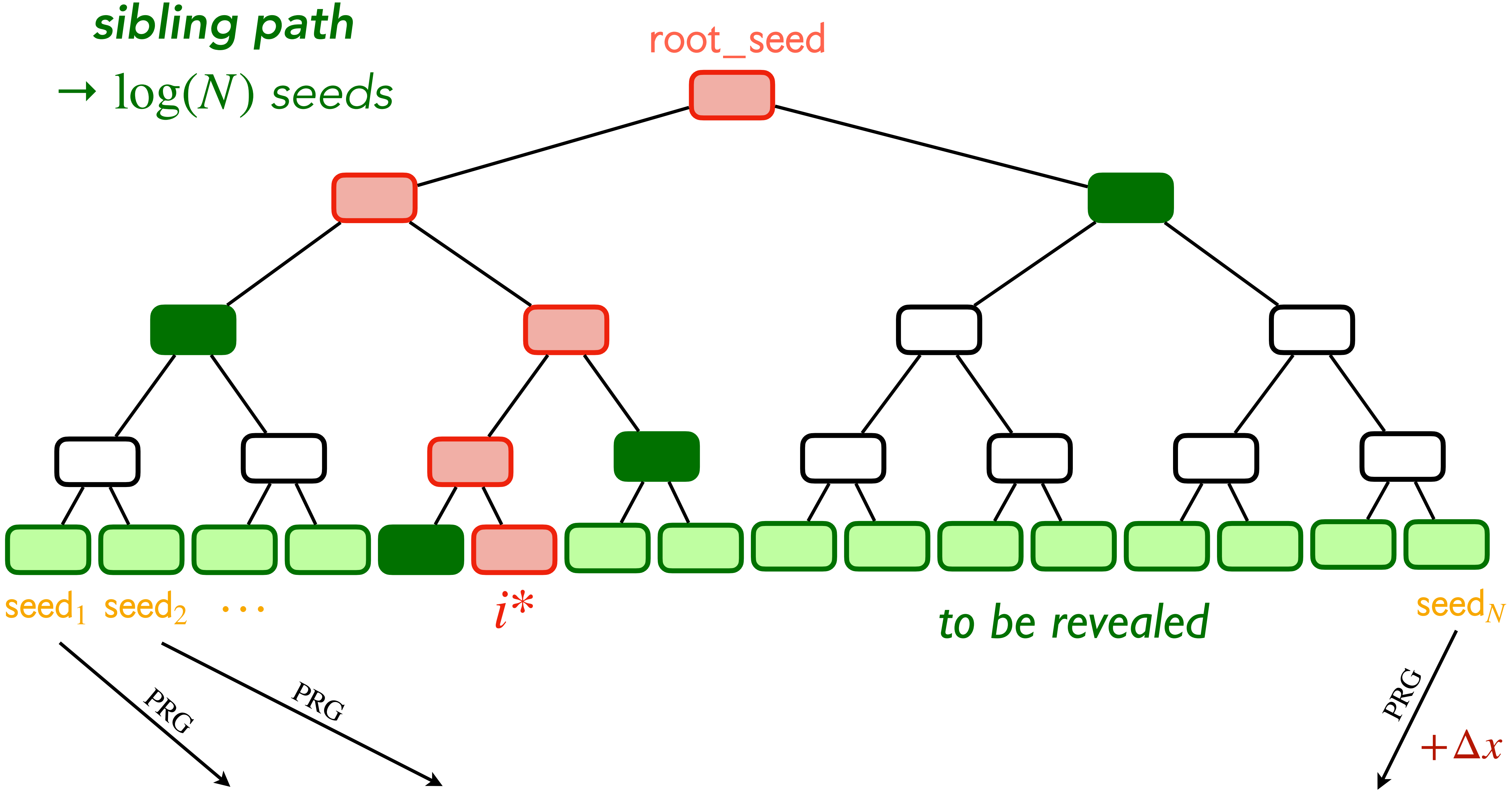
$$x = [\![x]\!]_1 + [\![x]\!]_2 + [\![x]\!]_3 + \ldots + [\![x]\!]_{N-1} + [\![x]\!]_N$$

Using a Seed Tree

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

sibling path

→ $\log(N)$ *seeds*



root_seed

seed$_1$ seed$_2$ $\cdots$

i^*

to be revealed

seed$_N$

PRG

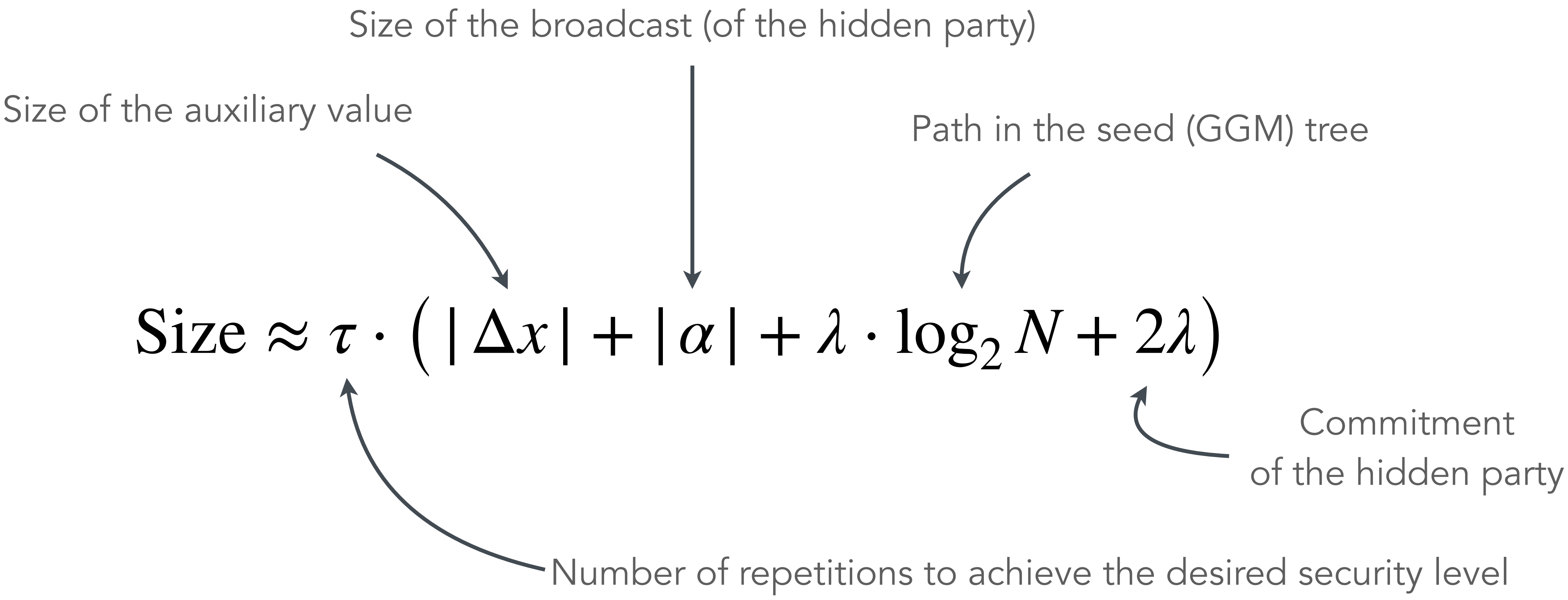
PRG

PRG

$+\Delta x$

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

Traditional MPCitH transformation



Size of the broadcast (of the hidden party)

Size of the auxiliary value

Path in the seed (GGM) tree

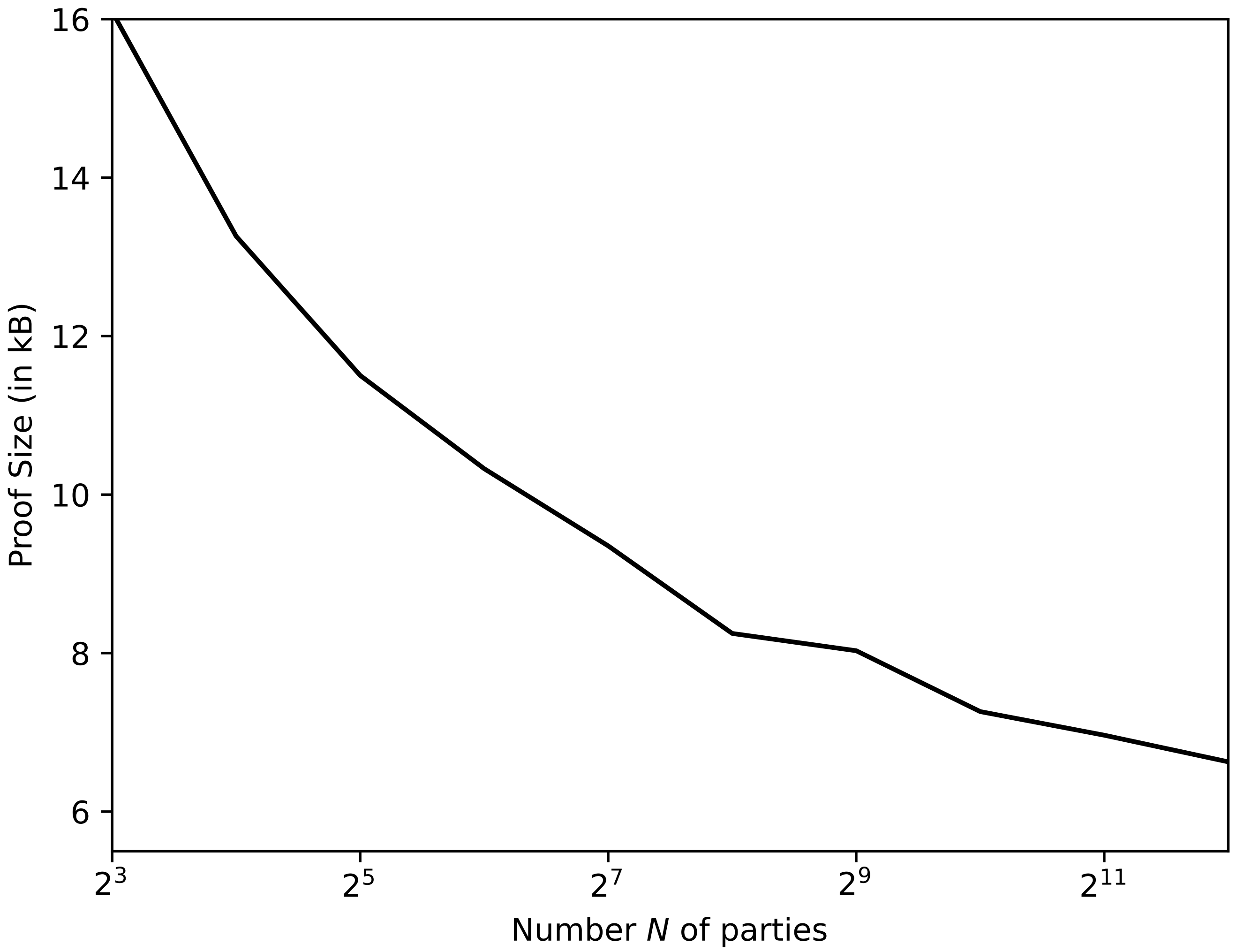
$$\text{Size} \approx \tau \cdot \big(|\Delta x| + |\alpha| + \lambda \cdot \log_2 N + 2\lambda \big)$$

Commitment
of the hidden party

Number of repetitions to achieve the desired security level

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

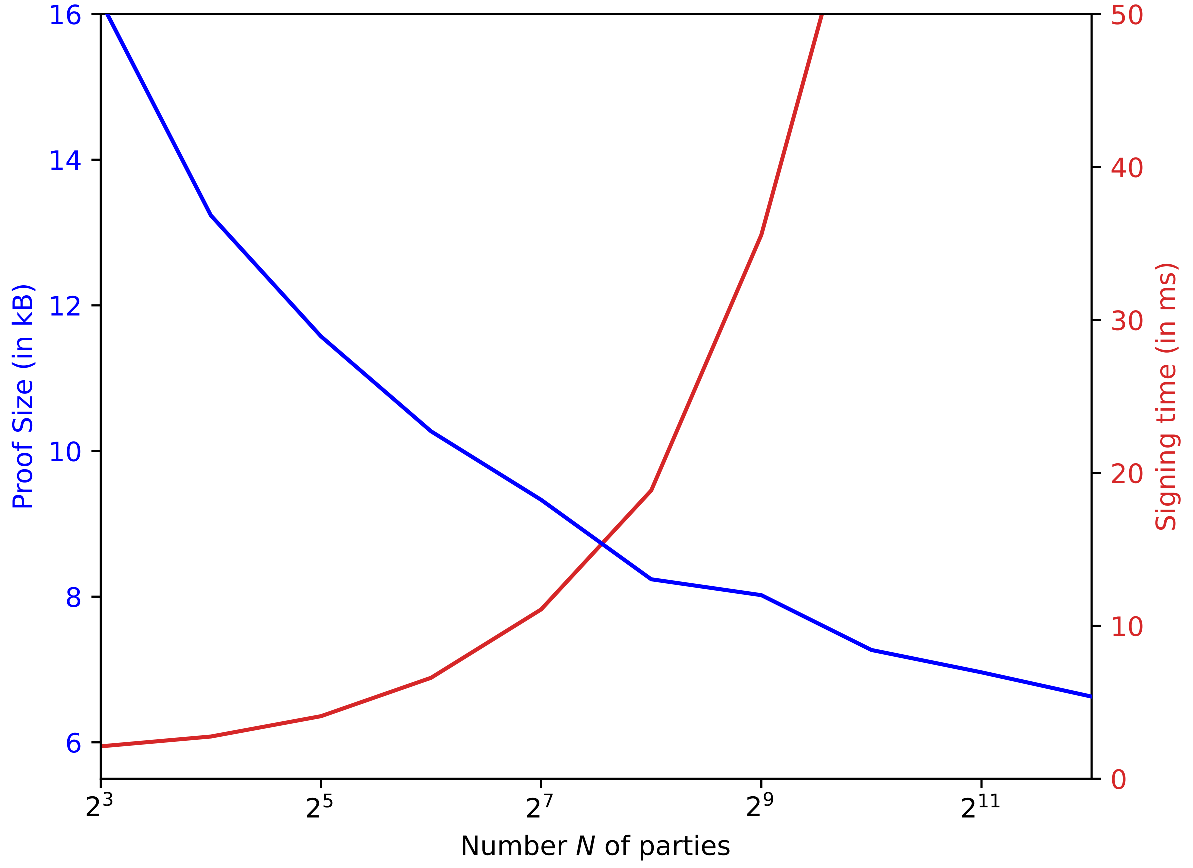
Traditional MPCitH transformation



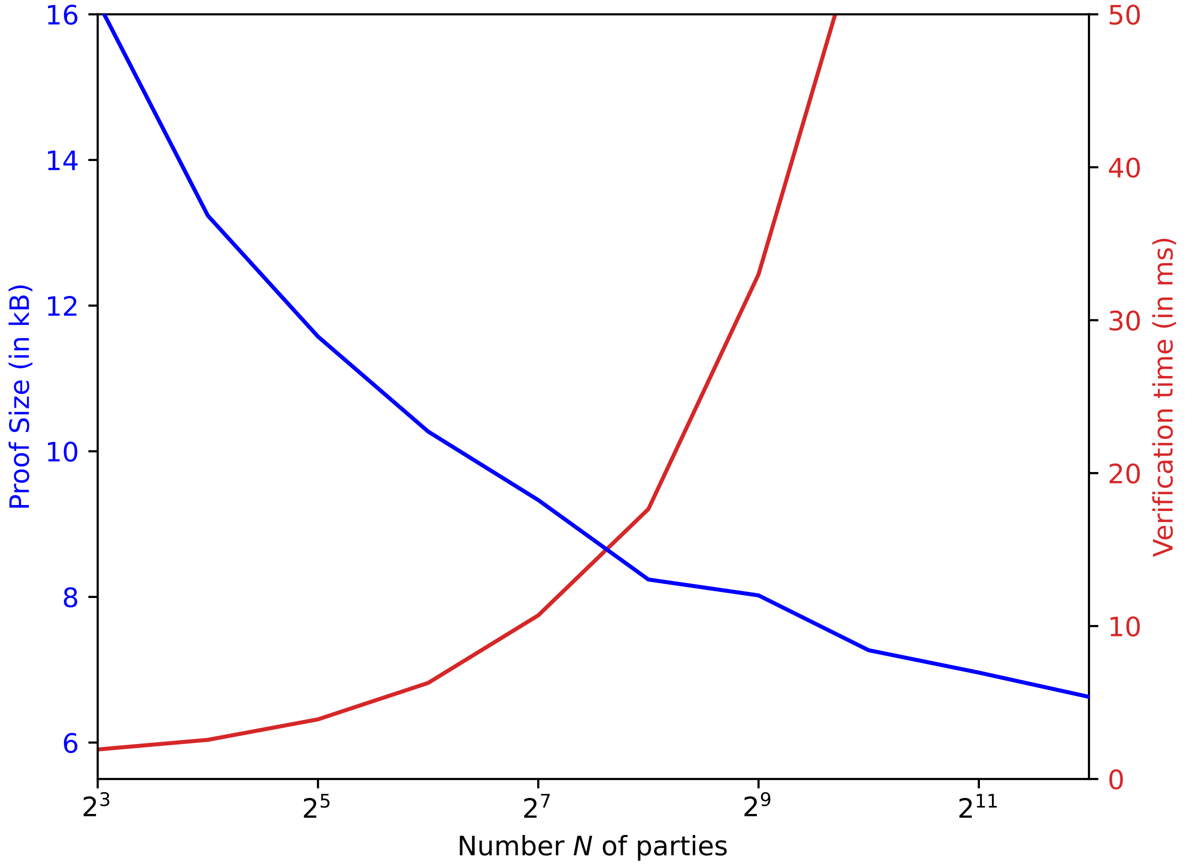
`SDitH-L1-gf251:`

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

Traditional MPCitH transformation



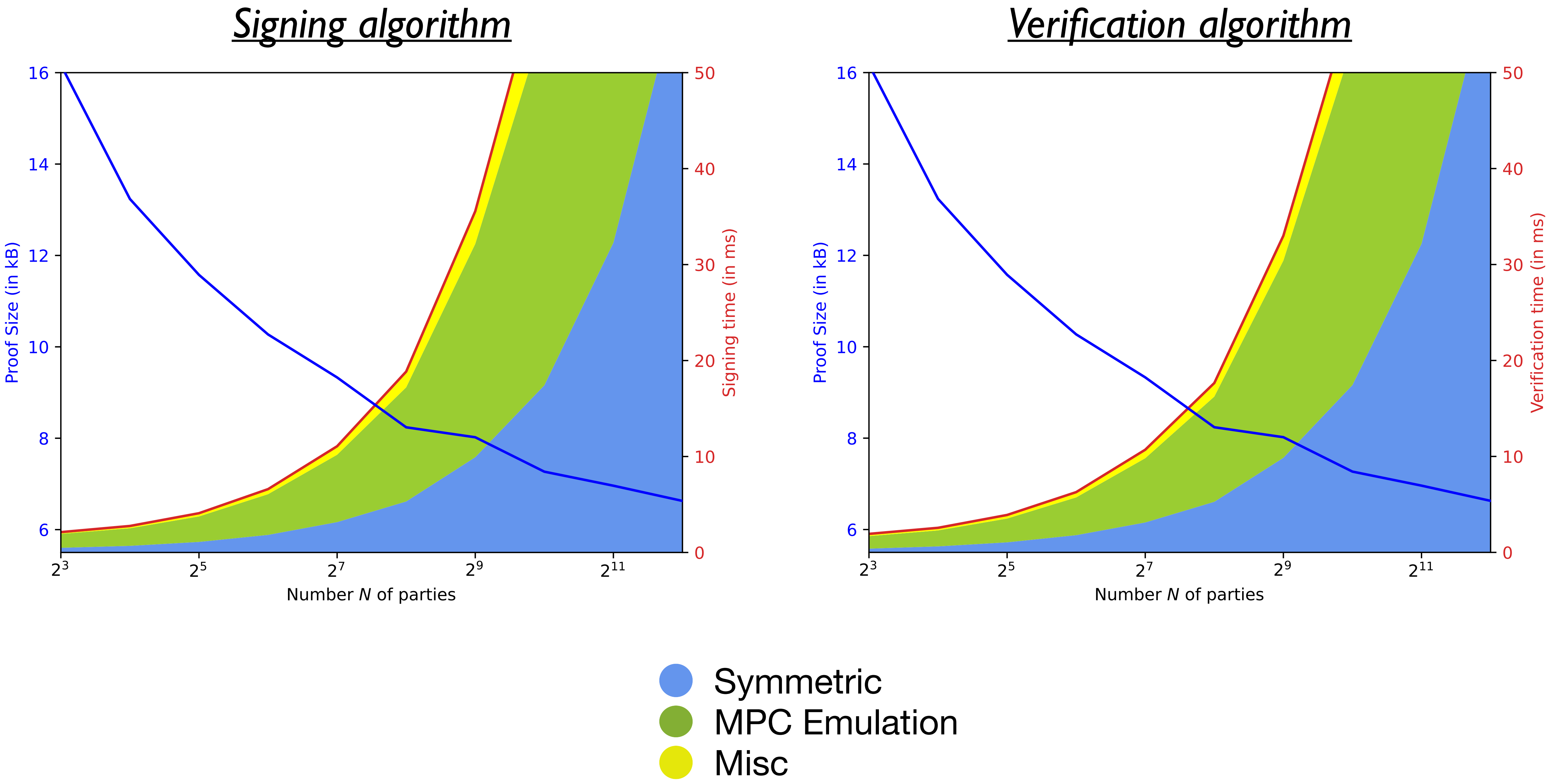
Signing algorithm



Verification algorithm

Running times @3.80Ghz

Traditional MPCitH transformation



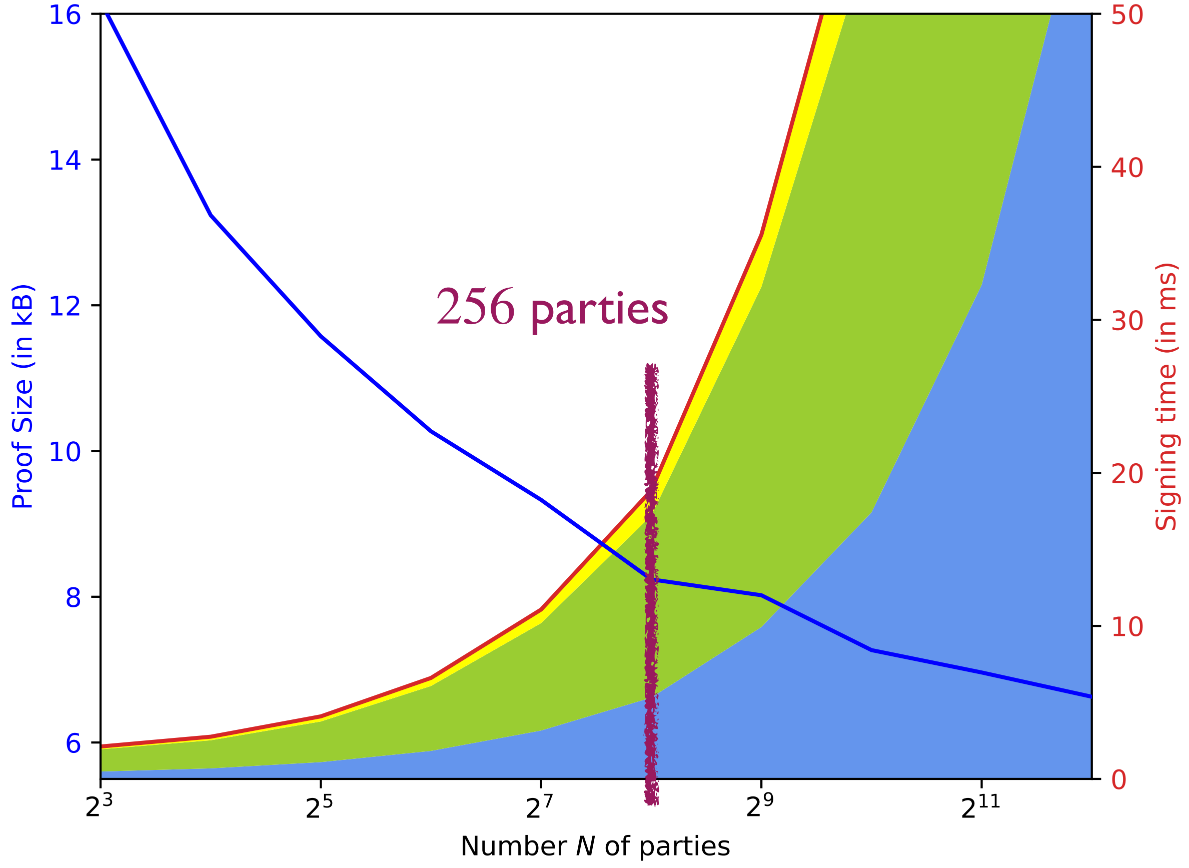
Signing algorithm

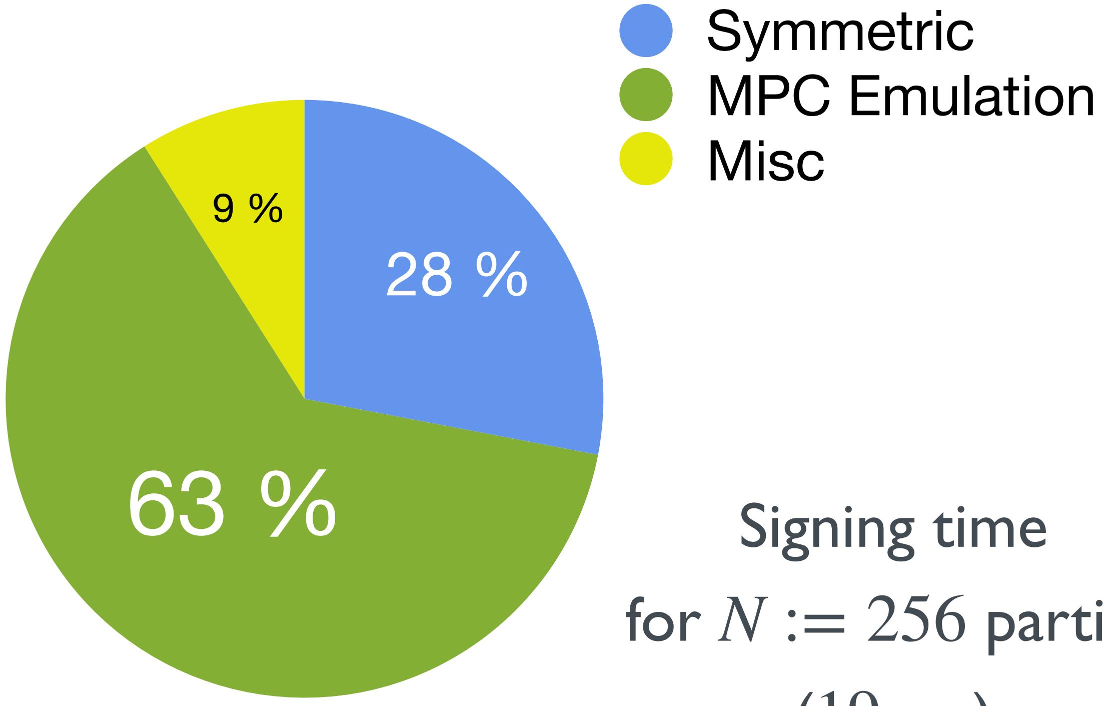
Verification algorithm

Running times @3.80Ghz

Traditional MPCitH transformation

Signing algorithm



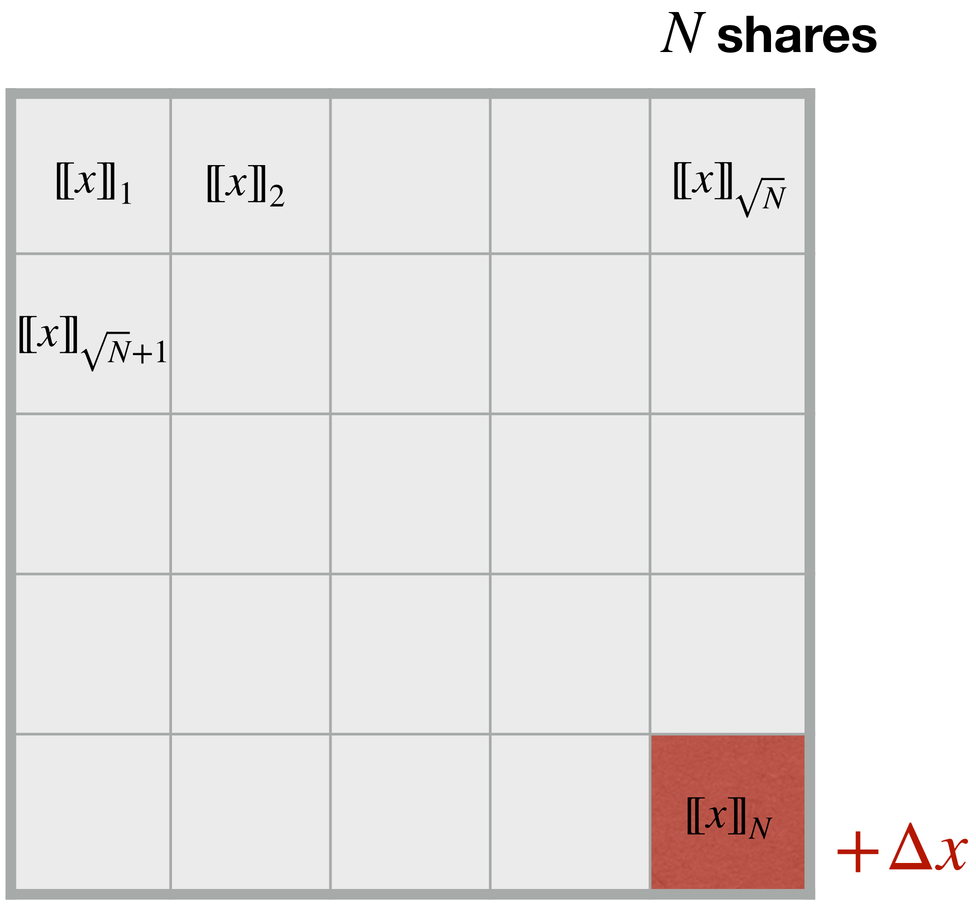


Signing time
for $N := 256$ parties
(19 ms)

Running times @3.80Ghz

The Hypercube Technique

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



N **shares**

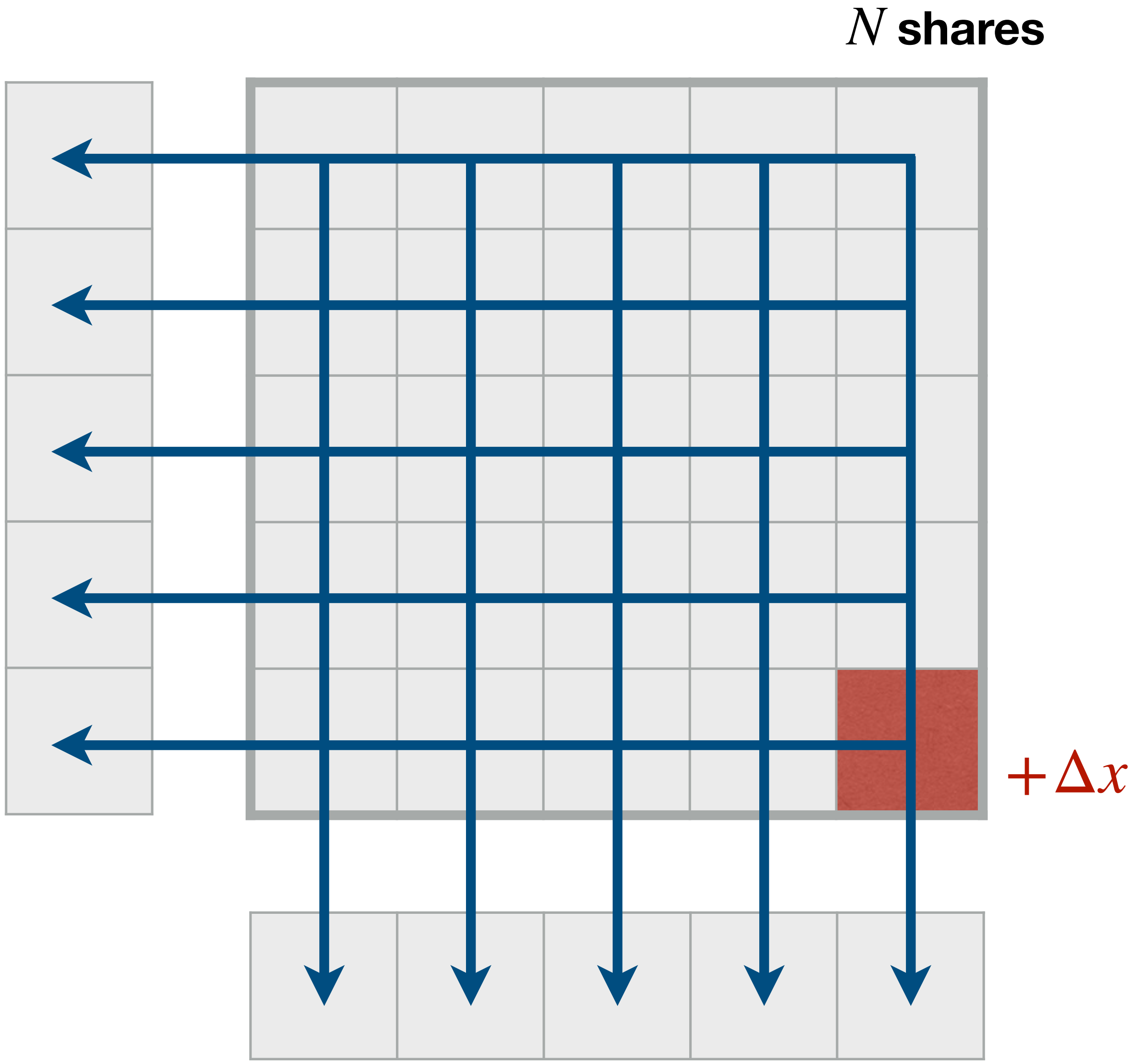
$[\![x]\!]_1$ $[\![x]\!]_2$ $[\![x]\!]_{\sqrt{N}}$

$[\![x]\!]_{\sqrt{N}+1}$

$[\![x]\!]_N$ $+\Delta x$

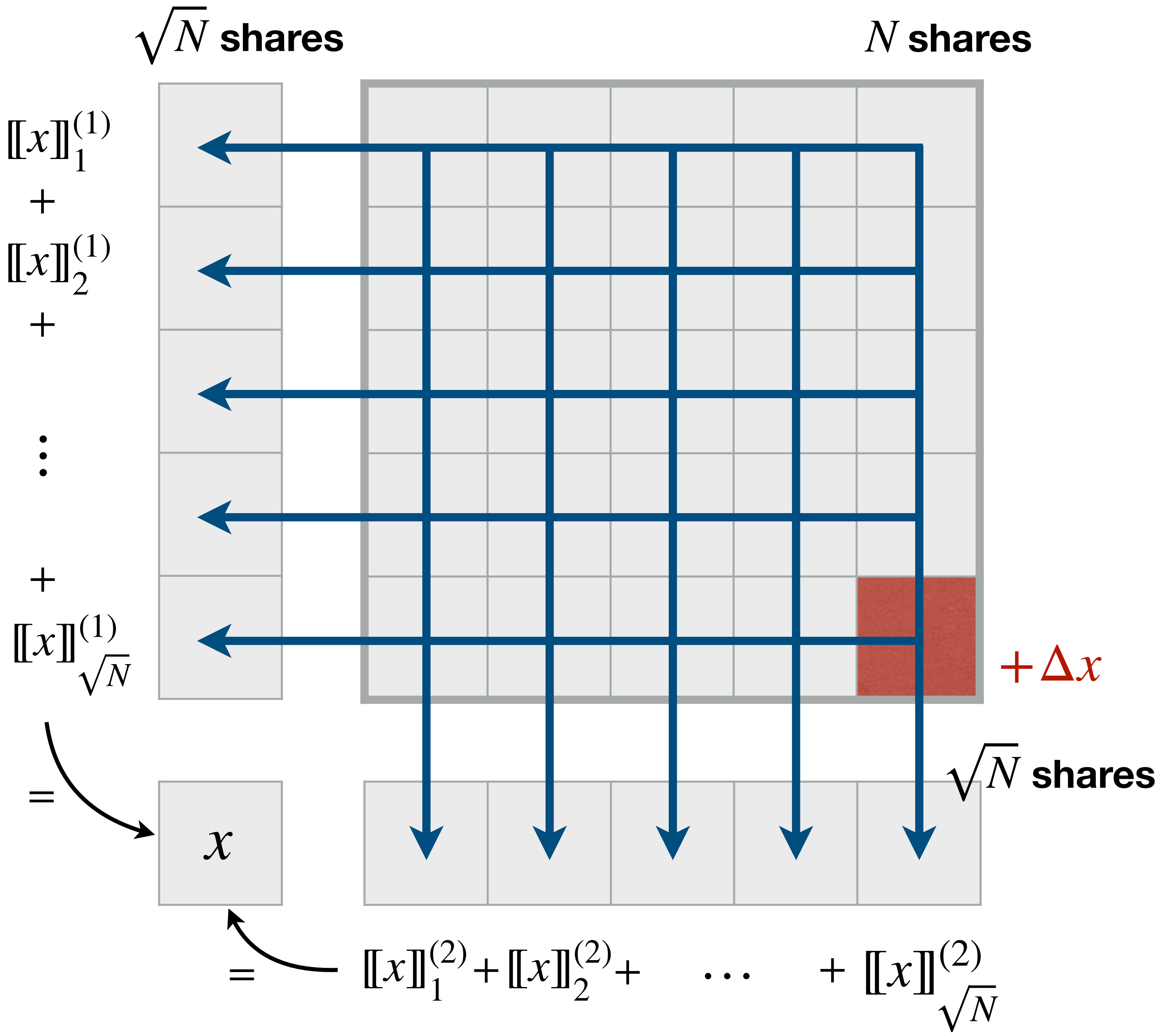
The Hypercube Technique

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



The Hypercube Technique

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

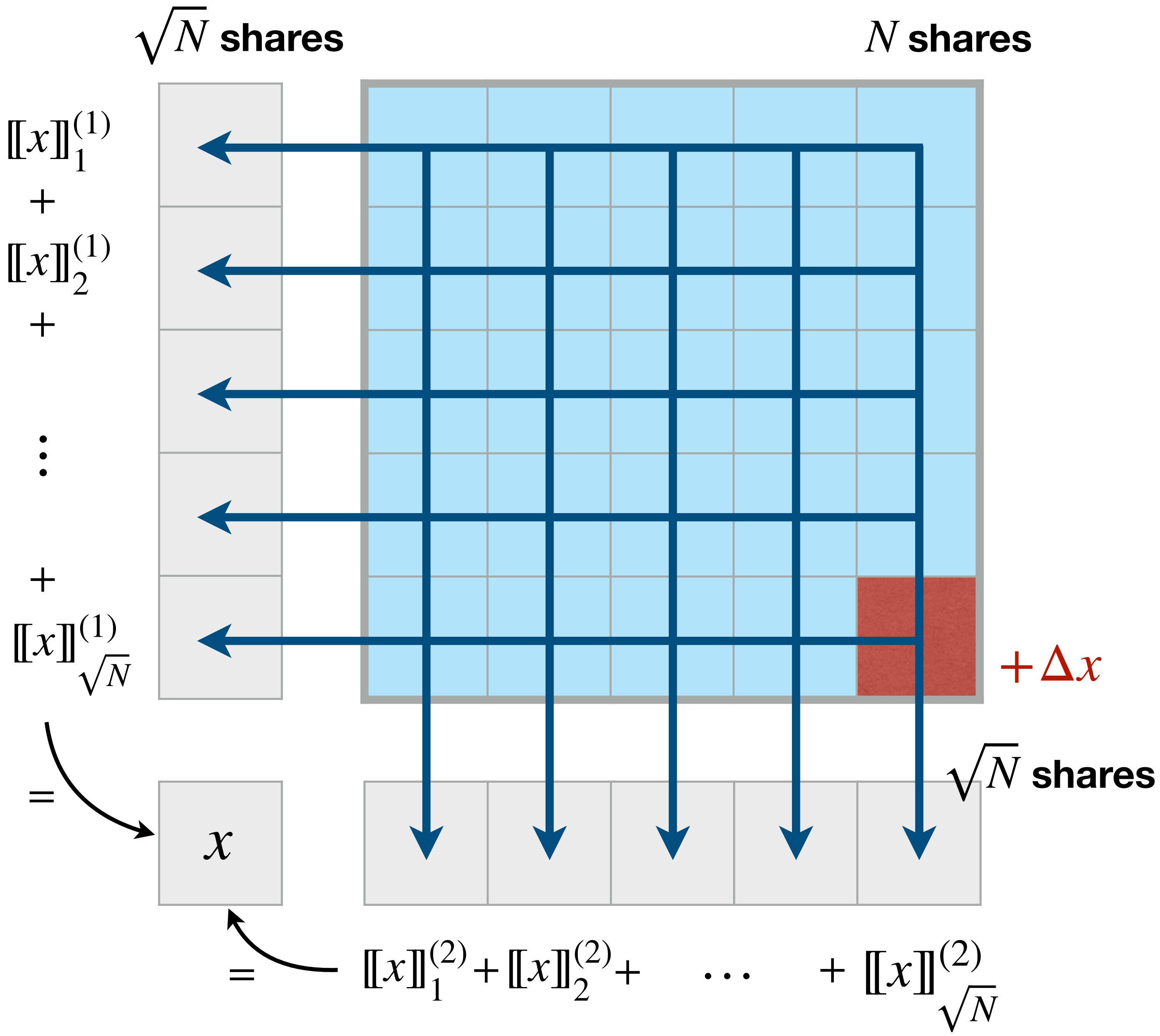


The Hypercube Technique

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

Traditional approach:

- Emulating the N-party protocol with inputs $[\![x]\!]_1, \ldots, [\![x]\!]_N$
- Chance of cheating $1/N$



The Hypercube Technique

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

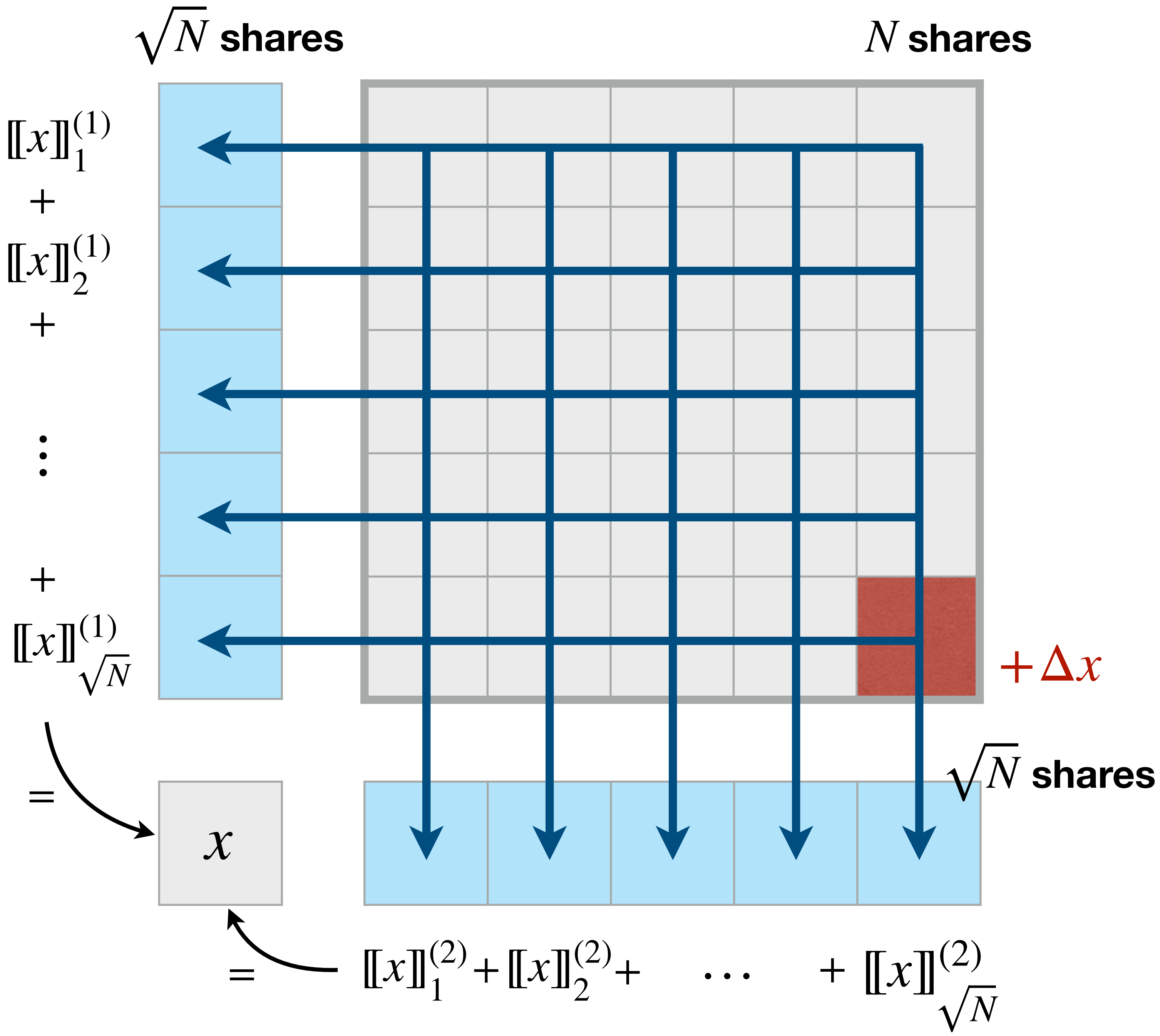
Traditional approach:

- Emulating the N-party protocol with inputs $[\![x]\!]_1, \dots, [\![x]\!]_N$
- Chance of cheating $1/N$

Hypercube technique:

- Emulating the $\sqrt{N}$-party protocol with inputs $[\![x]\!]_1^{(1)}, \dots, [\![x]\!]_{\sqrt{N}}^{(1)}$
- Emulating the $\sqrt{N}$-party protocol with inputs $[\![x]\!]_1^{(2)}, \dots, [\![x]\!]_{\sqrt{N}}^{(2)}$
- Chance of cheating

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



The Hypercube Technique

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

Traditional approach:

- Emulating the N-party protoc
  with inputs $[\![x]\!]_1, \ldots, [\![$
- Chance of cheating $1/$

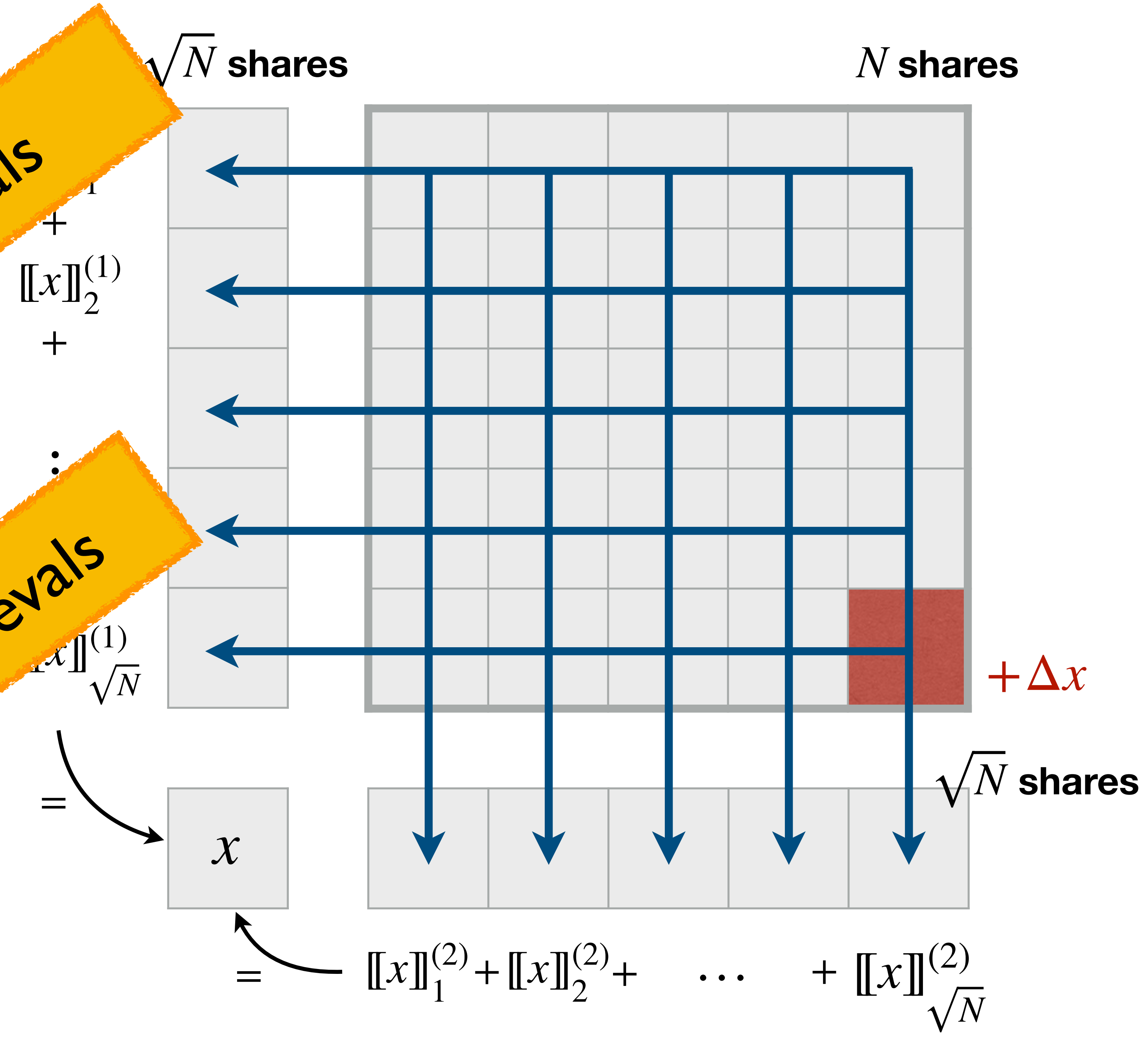
Hypercube technique:

- Emulating the $\sqrt{N}$-party protocol
  with inputs $[\![x]\!]_1^{(1)}, \ldots, [\![x]\!]_{\sqrt{N}}^{(1)}$
- Emulating the $\sqrt{N}$-party pro
  with inputs $[\![x]\!]_1^{(2)}, \ldots,$
- Chance of cheating

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

N evals

$2\sqrt{N}$ evals


$\sqrt{N}$ shares
N shares
$+$
$[\![x]\!]_2^{(1)}$
$+$
$\vdots$
$[\![x]\!]_{\sqrt{N}}^{(1)}$
$=$
x
$+\Delta x$
$\sqrt{N}$ shares
$= [\![x]\!]_1^{(2)} + [\![x]\!]_2^{(2)} + \cdots + [\![x]\!]_{\sqrt{N}}^{(2)}$

The Hypercube Technique

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

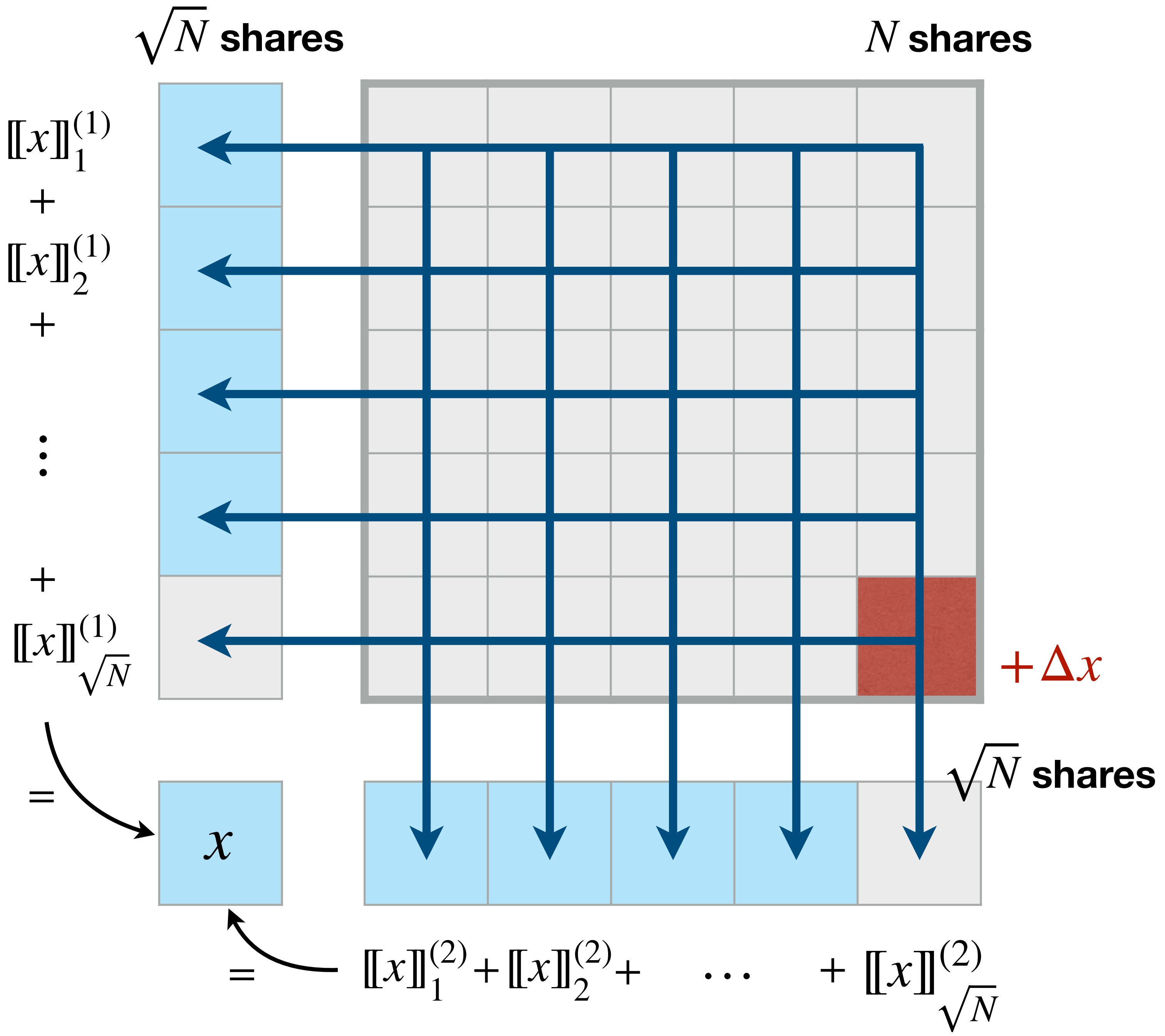
Traditional approach:

- Emulating the N-party protocol with inputs $[\![x]\!]_1, \dots, [\![x]\!]_N$
- Chance of cheating $1/N$

Hypercube technique:

- Emulating the $\sqrt{N}$-party protocol with inputs $[\![x]\!]^{(1)}_1, \dots, [\![x]\!]^{(1)}_{\sqrt{N}}$
- Emulating the $\sqrt{N}$-party protocol with inputs $[\![x]\!]^{(2)}_1, \dots, [\![x]\!]^{(2)}_{\sqrt{N}}$
- Chance of cheating

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



$\sqrt{N}$ shares
N shares
$[\![x]\!]^{(1)}_1$
+
$[\![x]\!]^{(1)}_2$
+
⋮
+
$[\![x]\!]^{(1)}_{\sqrt{N}}$
=
x
$+\Delta x$
$\sqrt{N}$ shares
=
$[\![x]\!]^{(2)}_1 + [\![x]\!]^{(2)}_2 + \cdots + [\![x]\!]^{(2)}_{\sqrt{N}}$

The Hypercube Technique

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

Traditional approach:

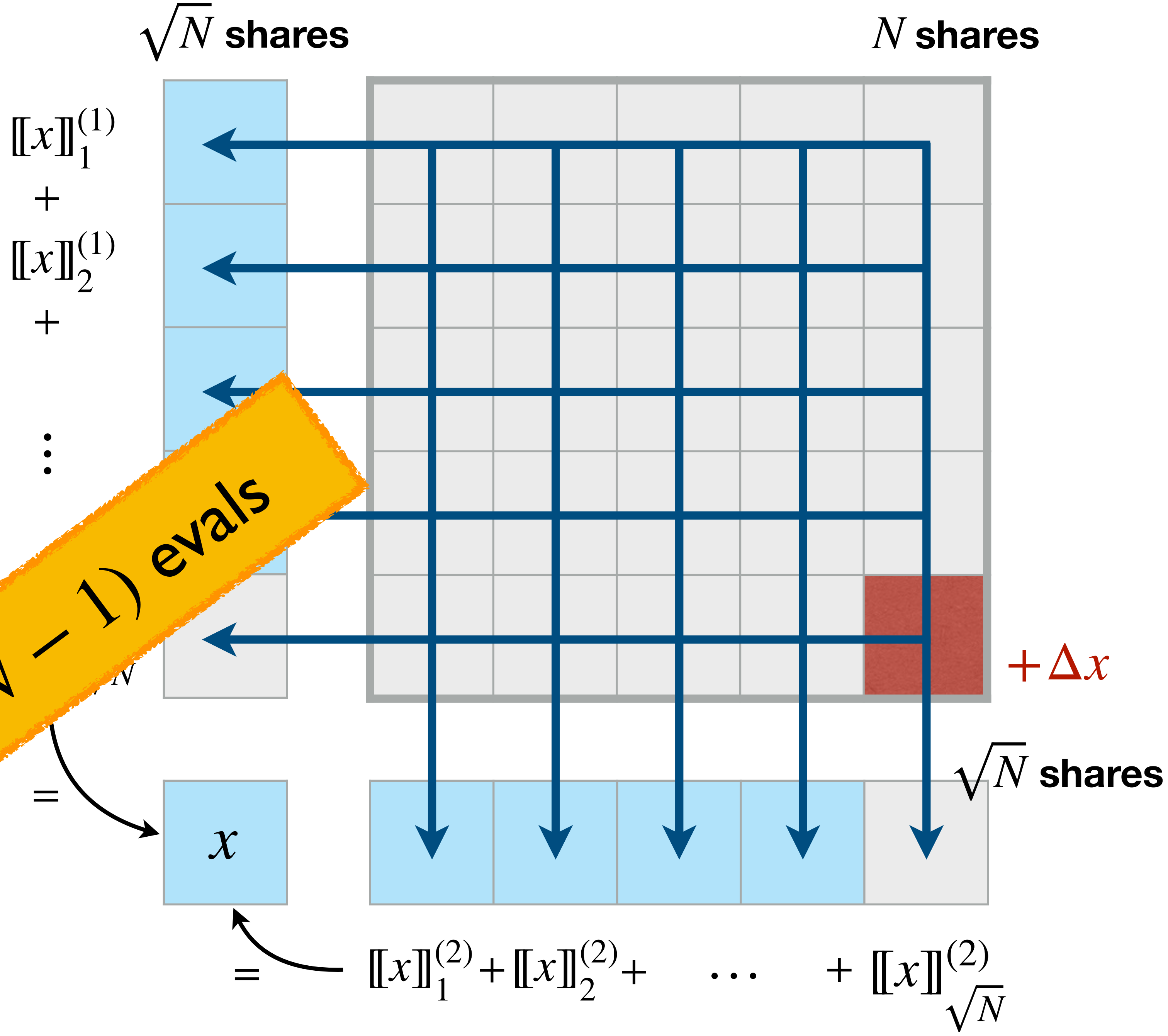
- Emulating the N-party protocol with inputs $[\![x]\!]_1, \dots, [\![x]\!]_N$
- Chance of cheating $1/N$

Hypercube technique:

- Emulating the $\sqrt{N}$-party protocol with inputs $[\![x]\!]^{(1)}_1, \dots, [\![x]\!]^{(1)}_{\sqrt{N}}$
- Emulating the $\sqrt{N}$-party protocol with inputs $[\![x]\!]^{(2)}_1, \dots, [\![x]\!]^{(2)}$
- Chance of cheating

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

$1 + 2(\sqrt{N} - 1)$ evals



The Hypercube Technique

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

<u>Previous slide</u>: square of side $\sqrt{N}$

The Hypercube Technique

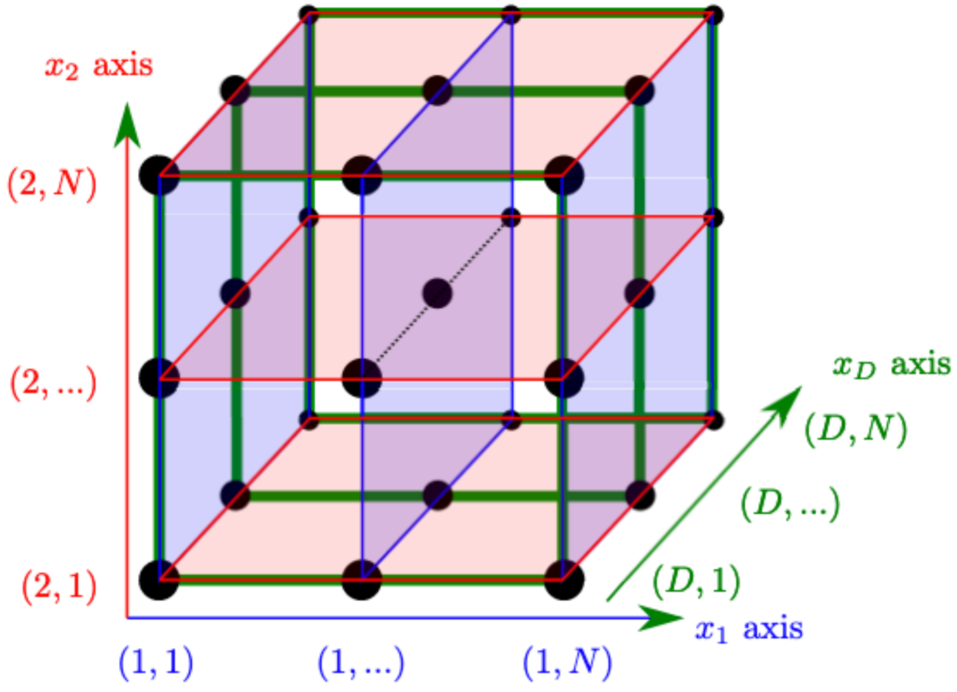
[AGHHJY23] Aguilar-Melchor, Gama, Howe, Hülsing, Joseph, Yue: "The Return of the SDitH" (Eurocrypt 2023)

Previous slide: square of side $\sqrt{N}$

The hypercube technique: hypercube of dimension $\log_2 N$ (each side has a size of 2)

Emulating $\log_2 N$ subprotocols with 2 parties.

Source: Figure from [AGHHJY23]



The $D \times N$ main party slices

The Hypercube Technique

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

Previous slide: square of side $\sqrt{N}$

The hypercube technique: hypercube of dimension $\log_2 N$ (each side has a size of 2)

Emulating $\log_2 N$ subprotocols with 2 parties.

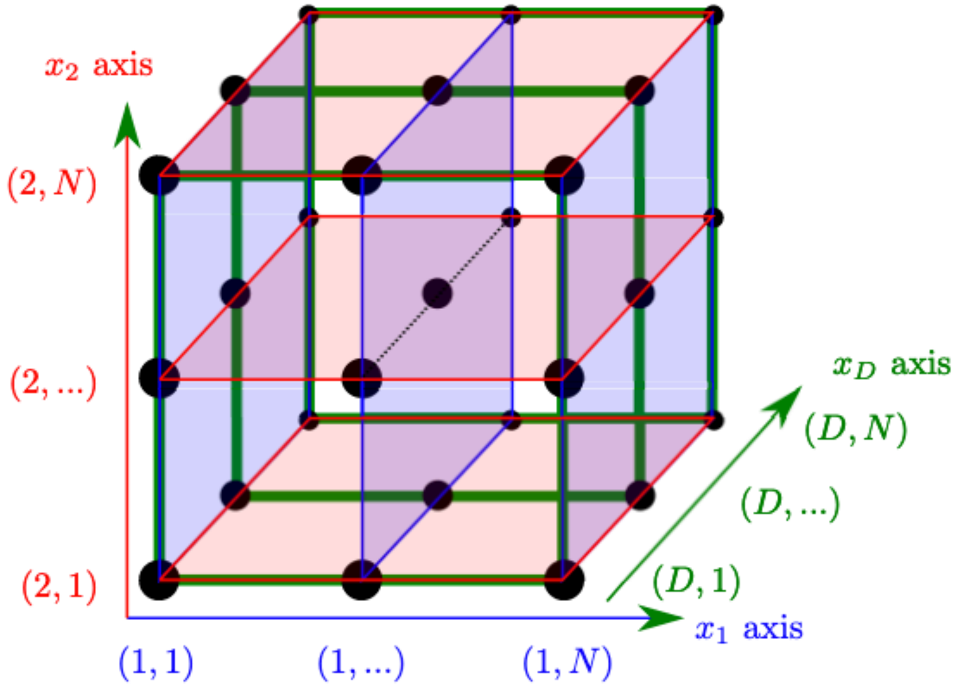
Soundness error:

$$\left(\frac{1}{2}\right)^{\log_2 N} = \frac{1}{N}$$

Emulation cost:

$2 \cdot \log_2 N$ parties

Source: Figure from [AGHHJY23]



The $D \times N$ main party slices

The Hypercube Technique

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

Previous slide: square of side $\sqrt{N}$

The hypercube technique: hypercube of dimension $\log_2 N$ (each side has a size of 2)

Emulating $\log_2 N$ subprotocols with 2 parties.

Soundness error:

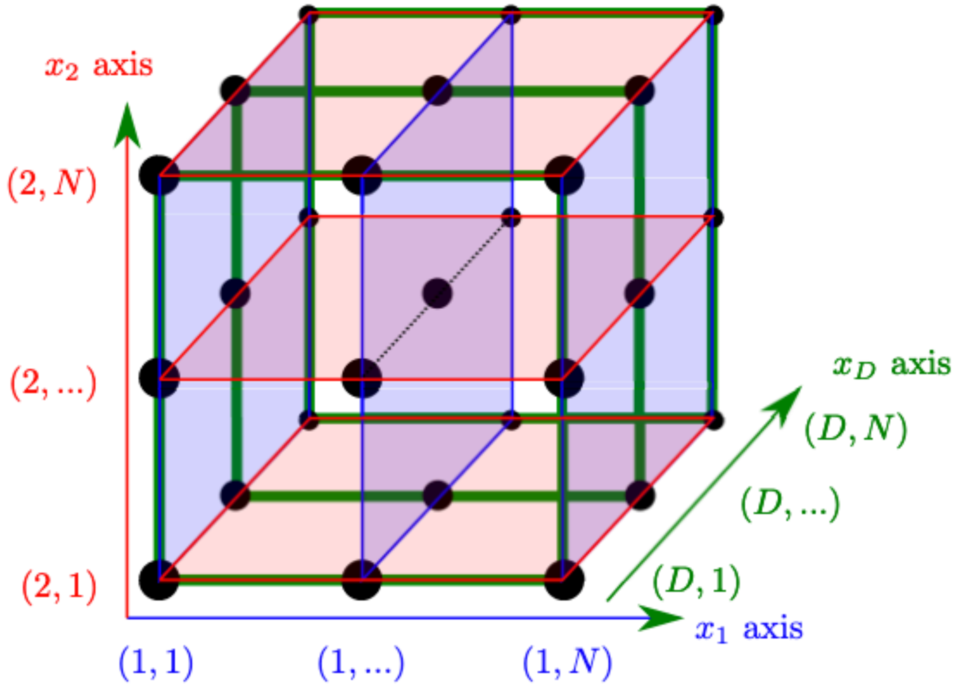
$$\left(\frac{1}{2}\right)^{\log_2 N} = \frac{1}{N}$$

Emulation cost:

~~$2 \cdot \log_2 N$ parties~~

$1 + \log_2 N$ parties

Source: Figure from [AGHHJY23]



The $D \times N$ main party slices

The Hypercube Technique

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

Traditional: N party emulations per repetition

↓

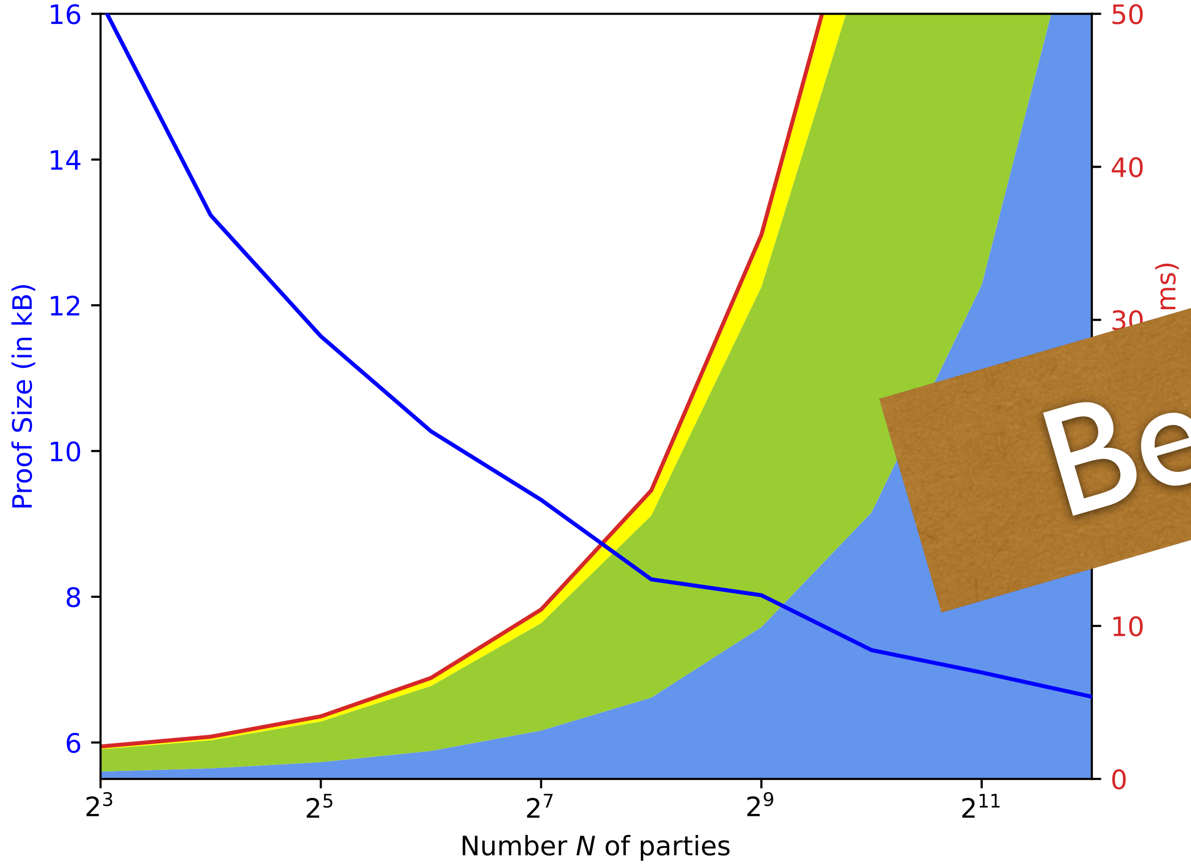
$N = 256$

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

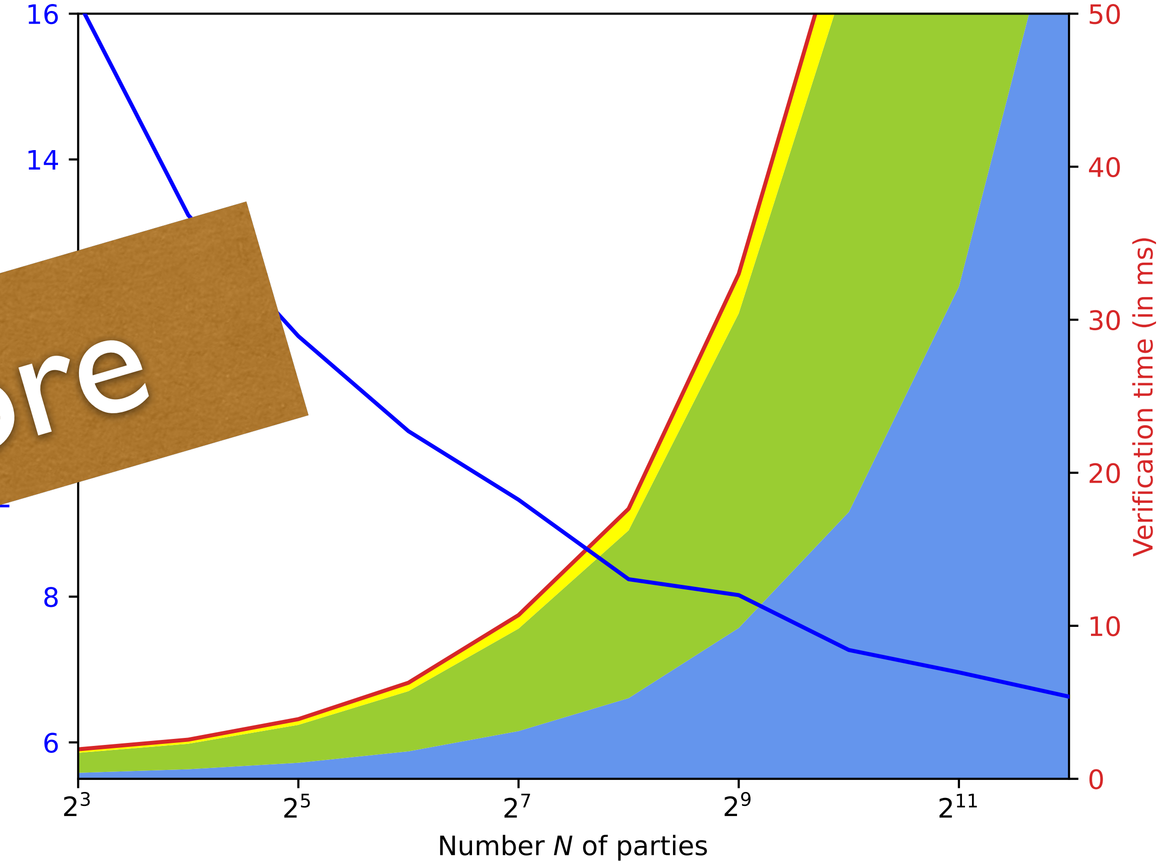
$1 + \log_2 N = 9$

The Hypercube Technique

Signing algorithm



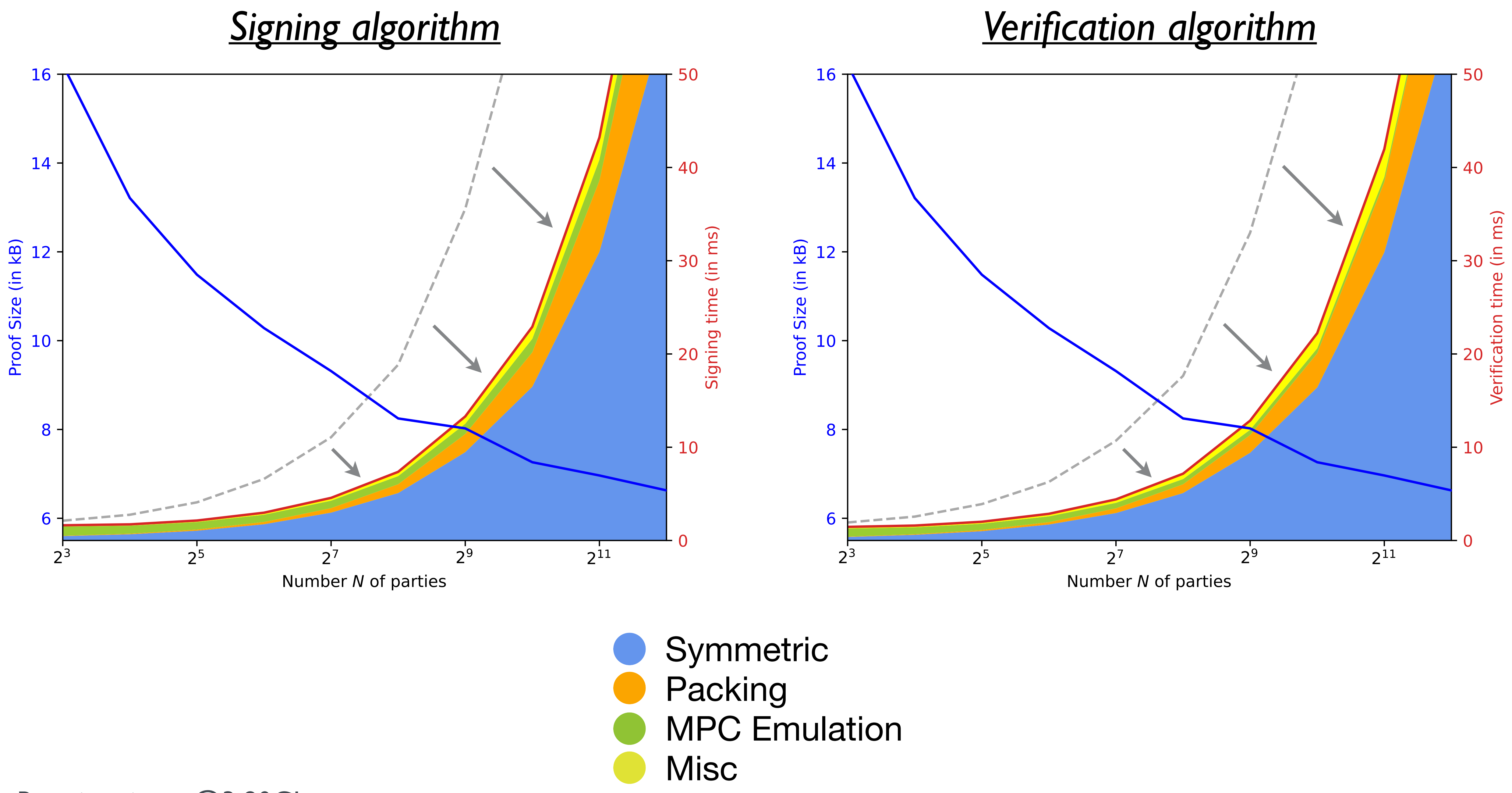
Verification algorithm



Before

Running times @3.80Ghz

The Hypercube Technique



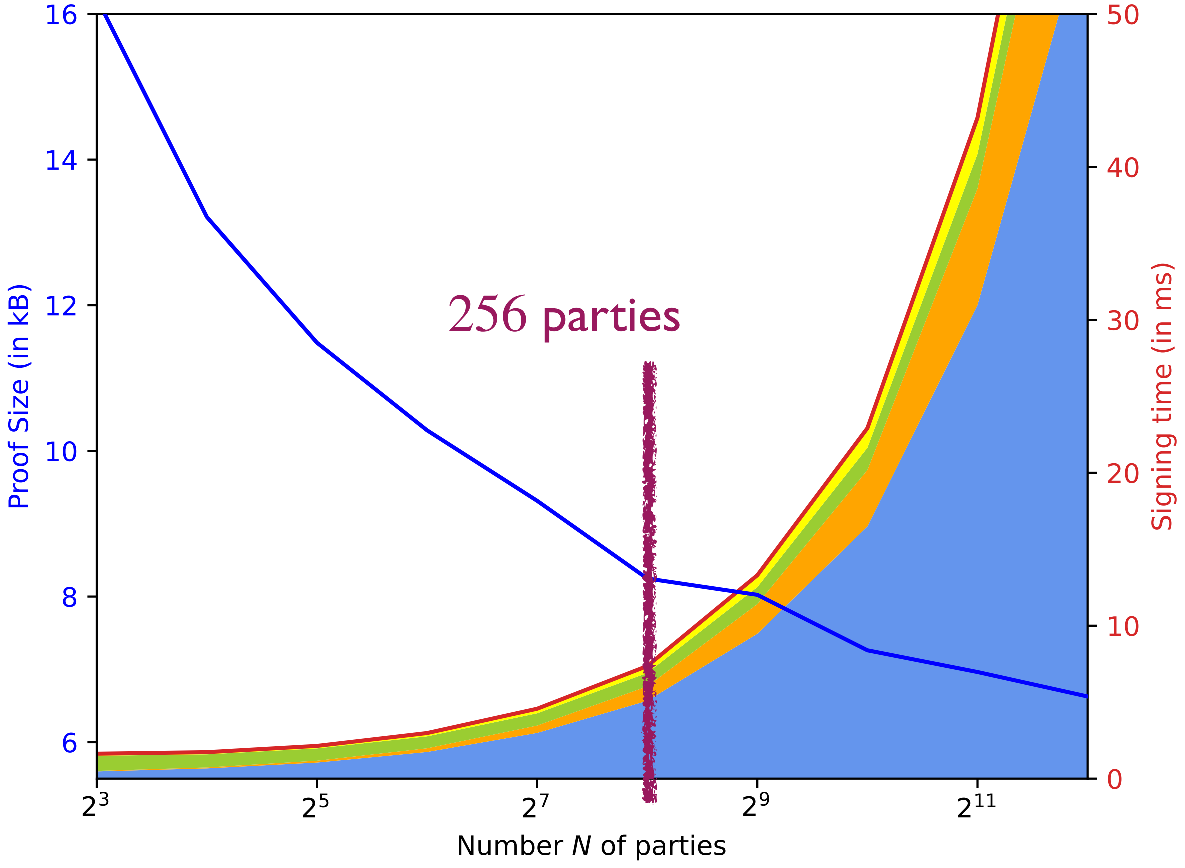
Signing algorithm

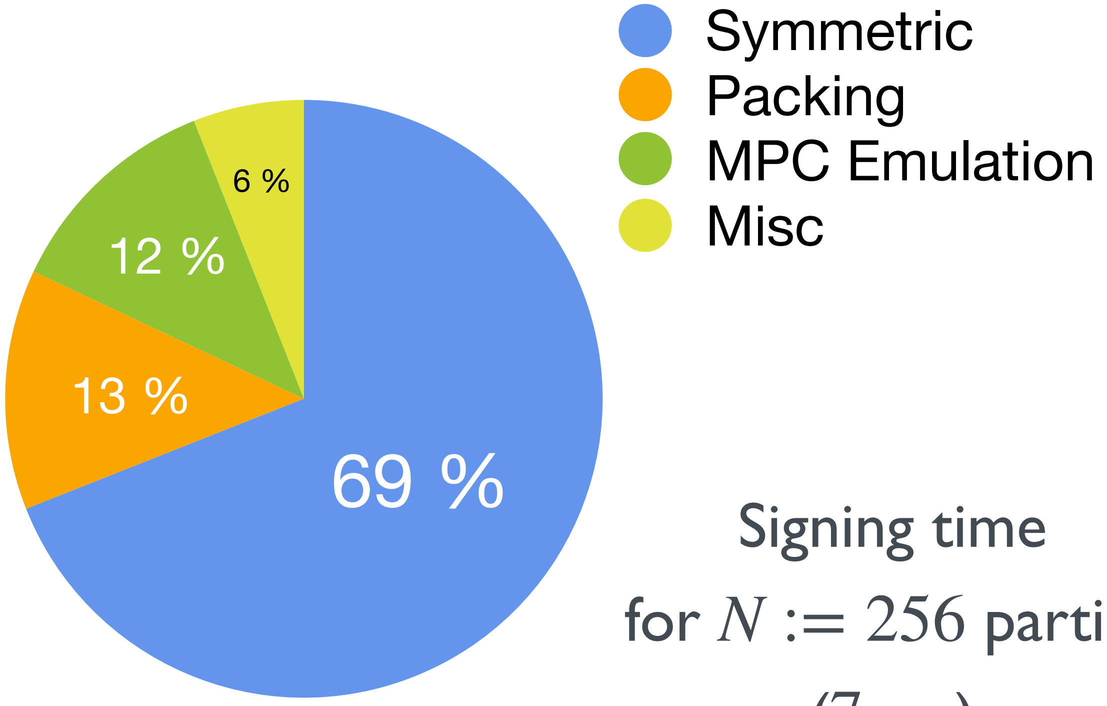
Verification algorithm

Running times @3.80Ghz

The Hypercube Technique

Signing algorithm





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

Running times @3.80Ghz

The Threshold Approach

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

In the *threshold* approach, we used an **low-threshold** sharing scheme. For example, the Shamir's (ℓ, 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

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

In the *threshold* approach, we used an **low-threshold** sharing scheme. For example, the Shamir's (ℓ, N)-secret sharing scheme.

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

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

The Threshold Approach

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

In the *threshold* approach, we used an **low-threshold** sharing scheme. For example, the Shamir's (ℓ, N)-secret sharing scheme.

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

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

Construction:

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

The Threshold Approach

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

In the *threshold* approach, we used an **low-threshold** sharing scheme. For example, the Shamir’s (ℓ, N)-secret sharing scheme.

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

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

Construction:

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

The Threshold Approach

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

In the *threshold* approach, we used an **low-threshold** sharing scheme. For example, the Shamir's (ℓ, N)-secret sharing scheme.

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

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

<u>Construction</u>:

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

The Threshold Approach

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

In the *threshold* approach, we used an **low-threshold** sharing scheme. For example, the Shamir's (ℓ, N)-secret sharing scheme.

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

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

<u>Construction</u>:

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

The Threshold Approach

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

In the *threshold* approach, we used an **low-threshold** sharing scheme. For example, the Shamir's (ℓ, N)-secret sharing scheme.

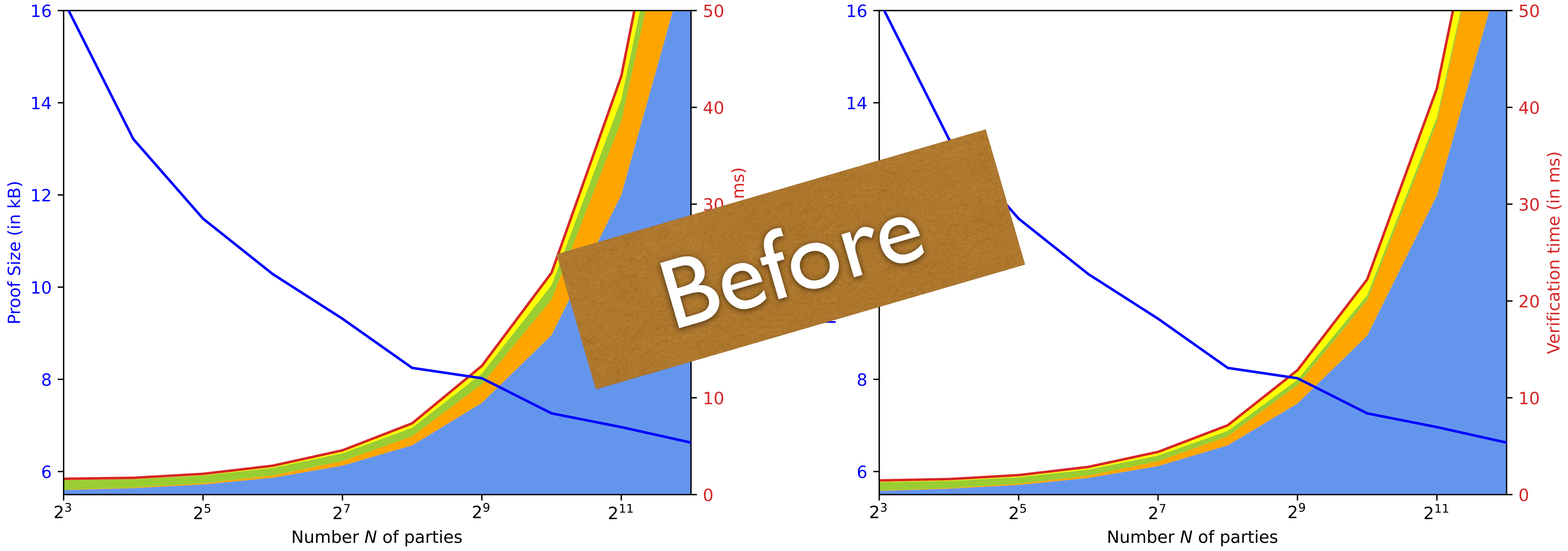
The prover reveals only ℓ shares to the verifier (instead of $N-1$).

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

Construction:

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

The Threshold Approach



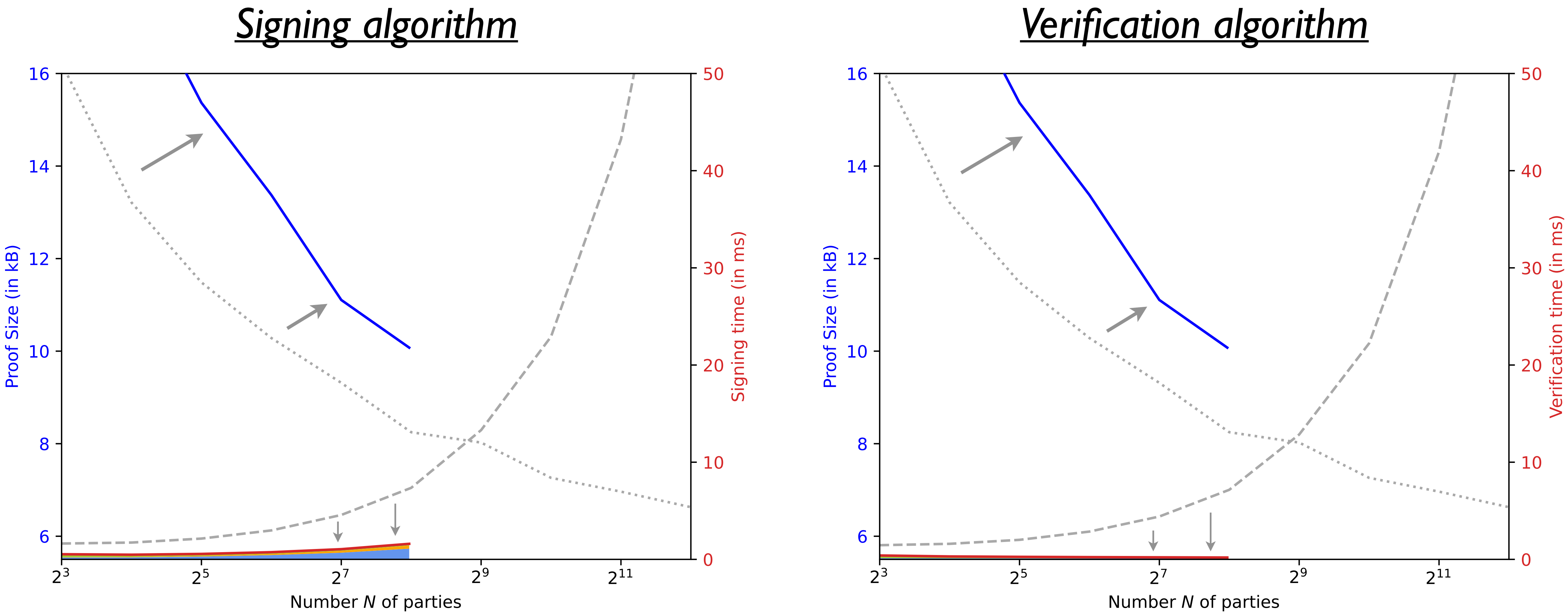
Signing algorithm

Verification algorithm

Before

Running times @3.80Ghz

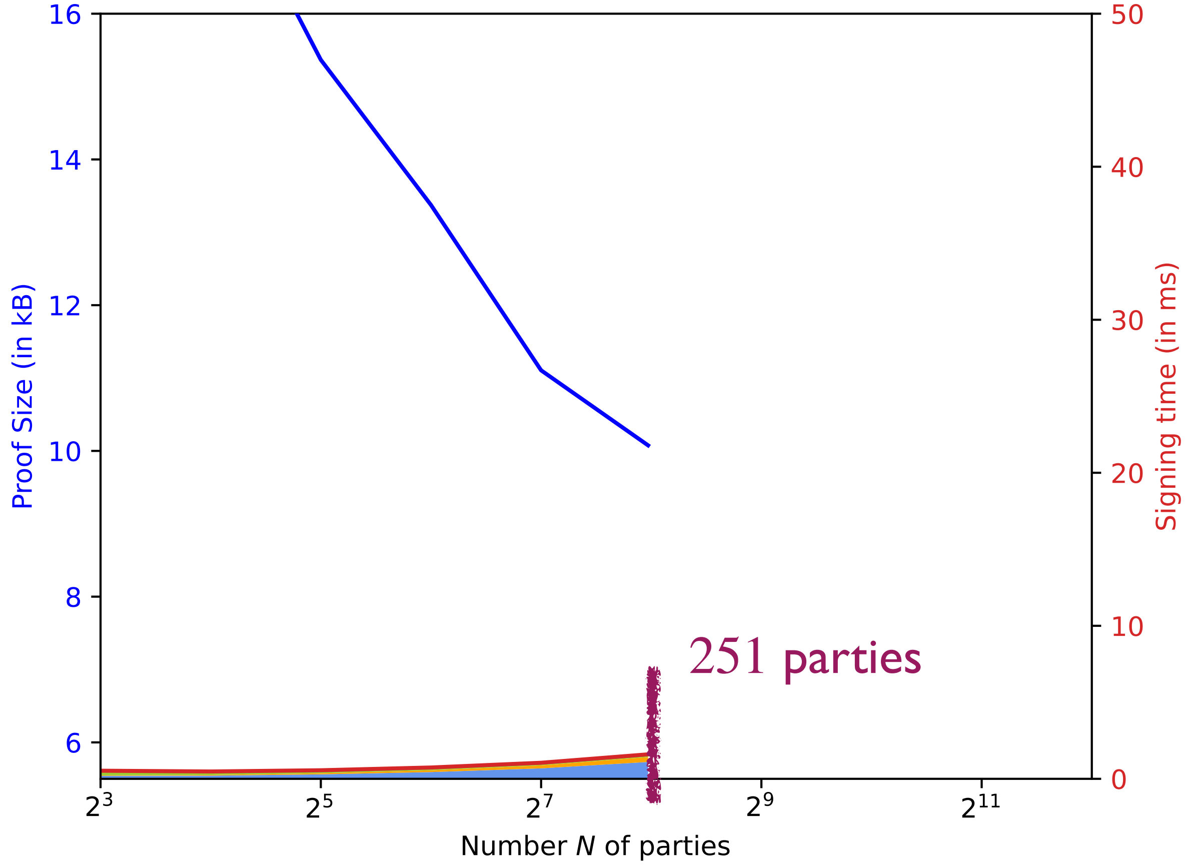
The Threshold Approach

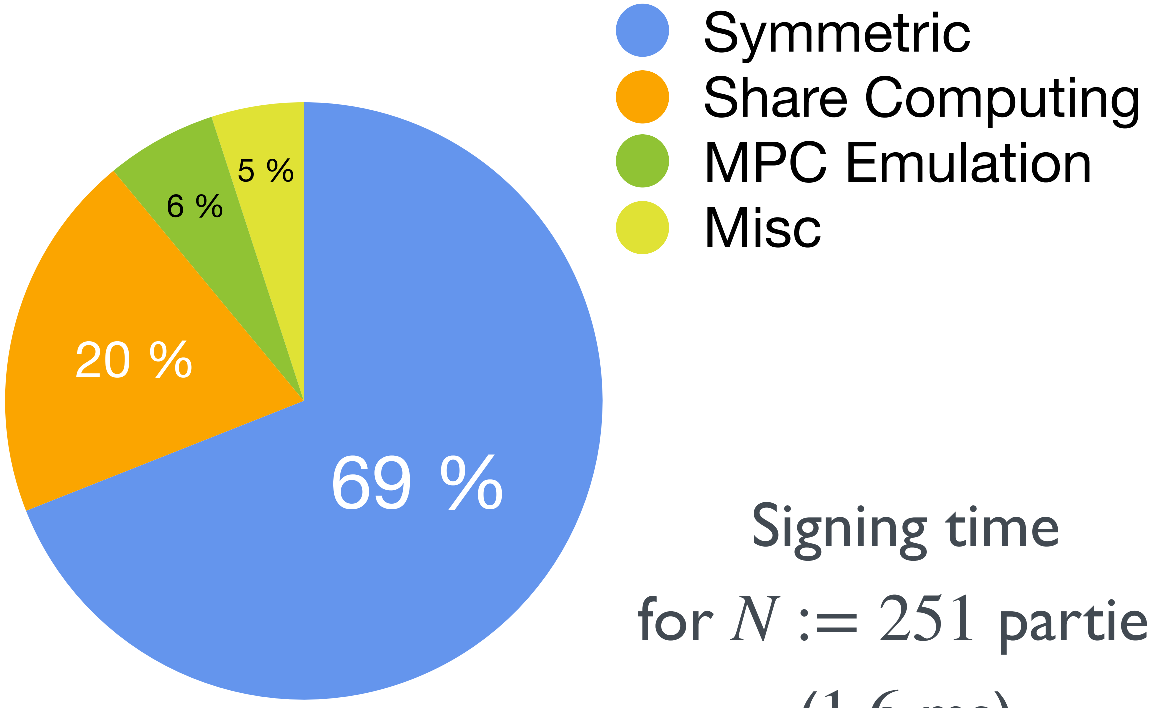


Running times @3.80Ghz

The Threshold Approach

Signing algorithm

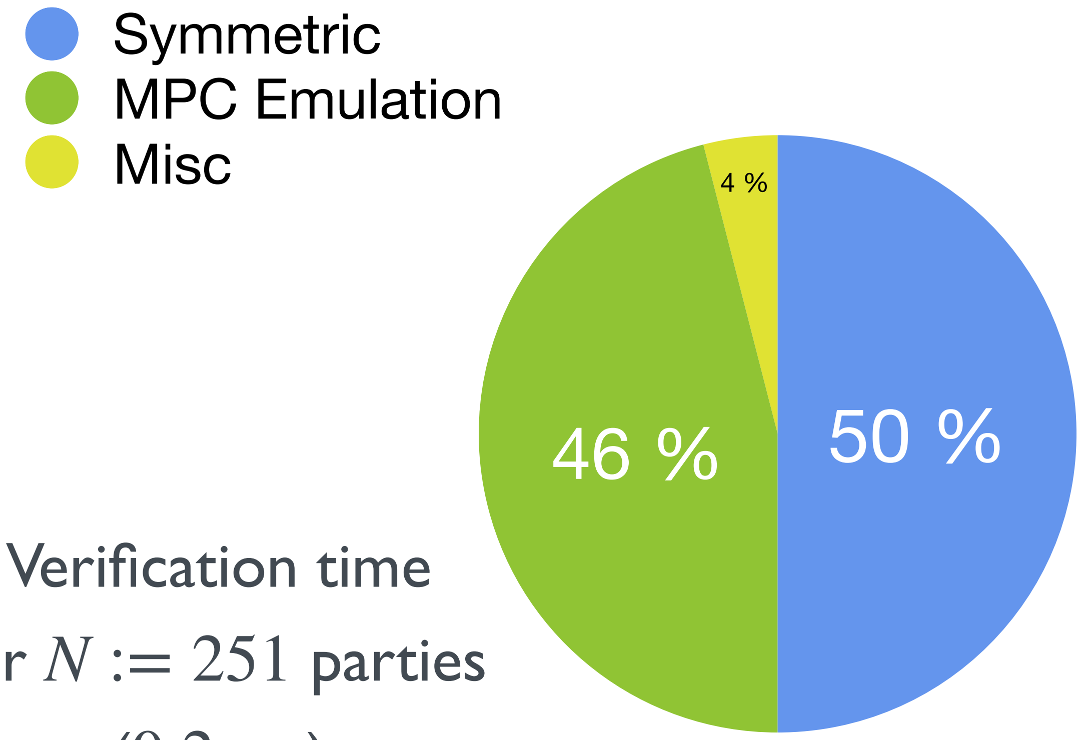




Signing time
for $N := 251$ parties
(1.6 ms)

Running times @3.80Ghz

The Threshold Approach

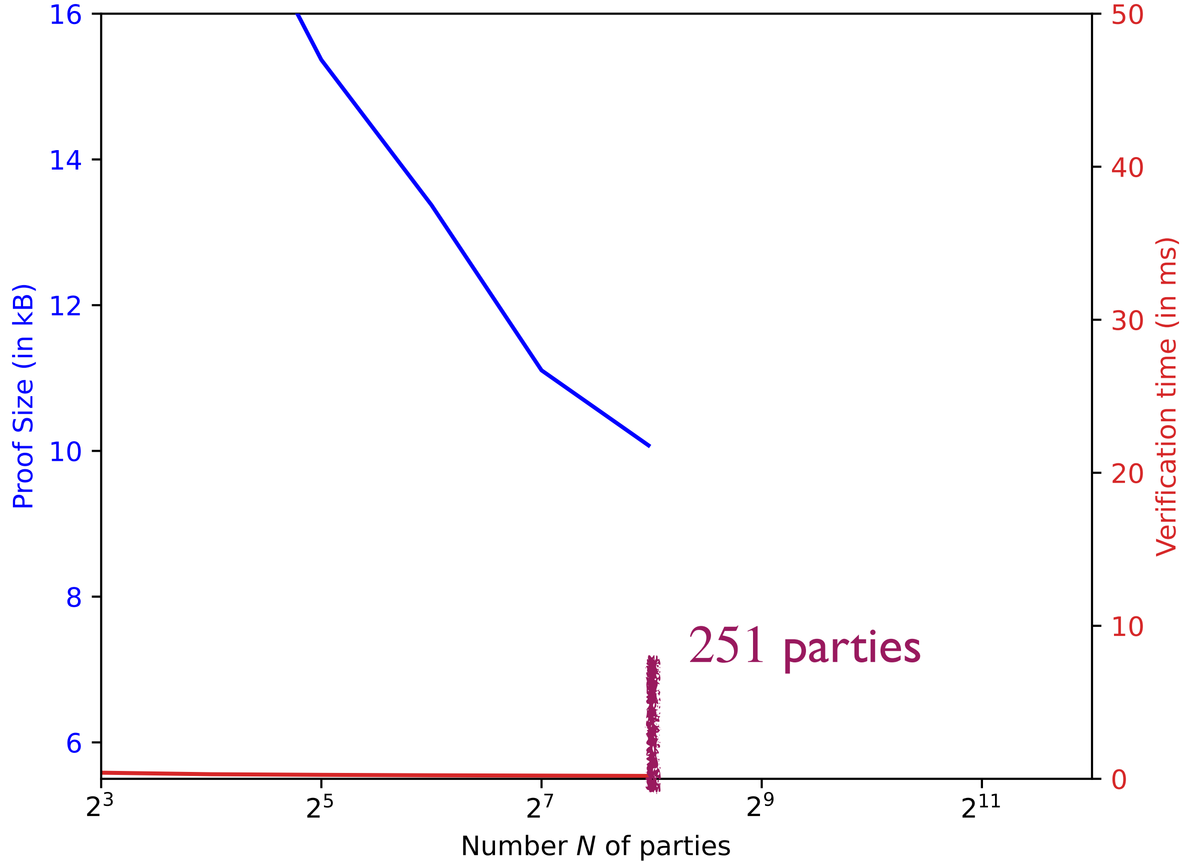


Verification time

for $N := 251$ parties

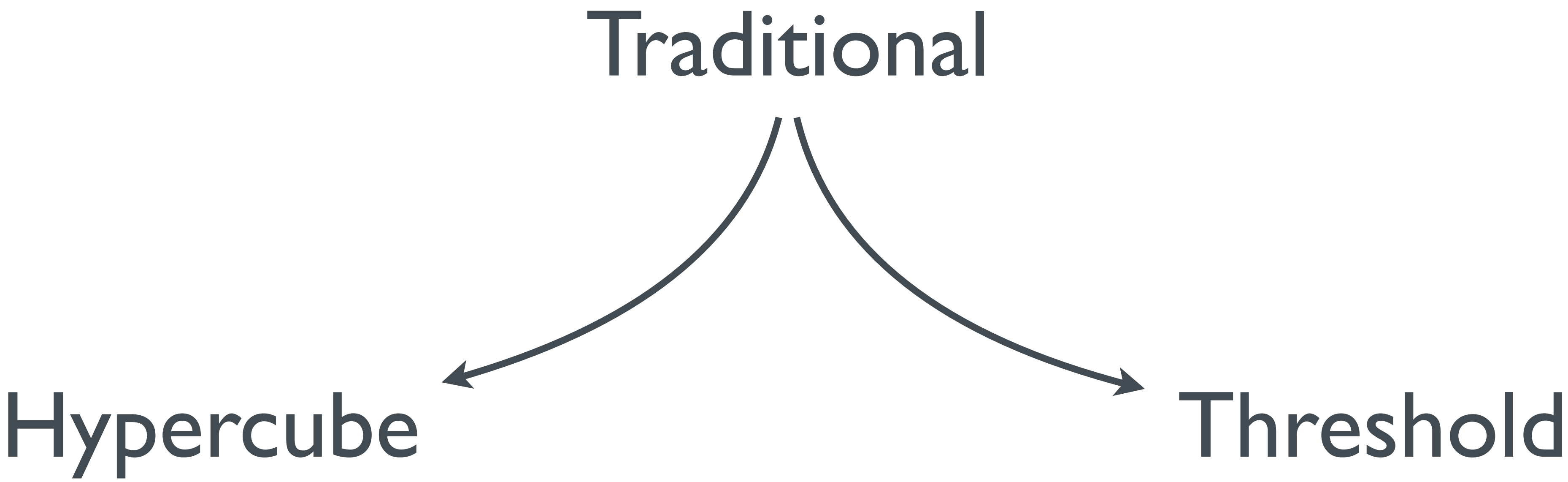
(0.2 ms)

Verification algorithm



Running times @3.80Ghz

The existing MPCitH transforms



Traditional

Hypercube

Threshold

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

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

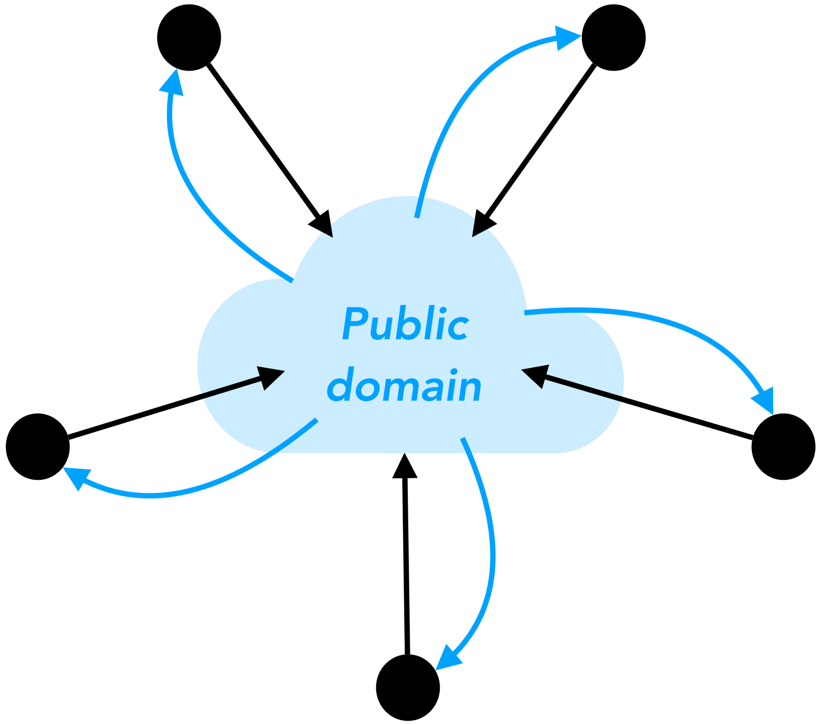
MPCitH-based NIST candidates

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

Related works

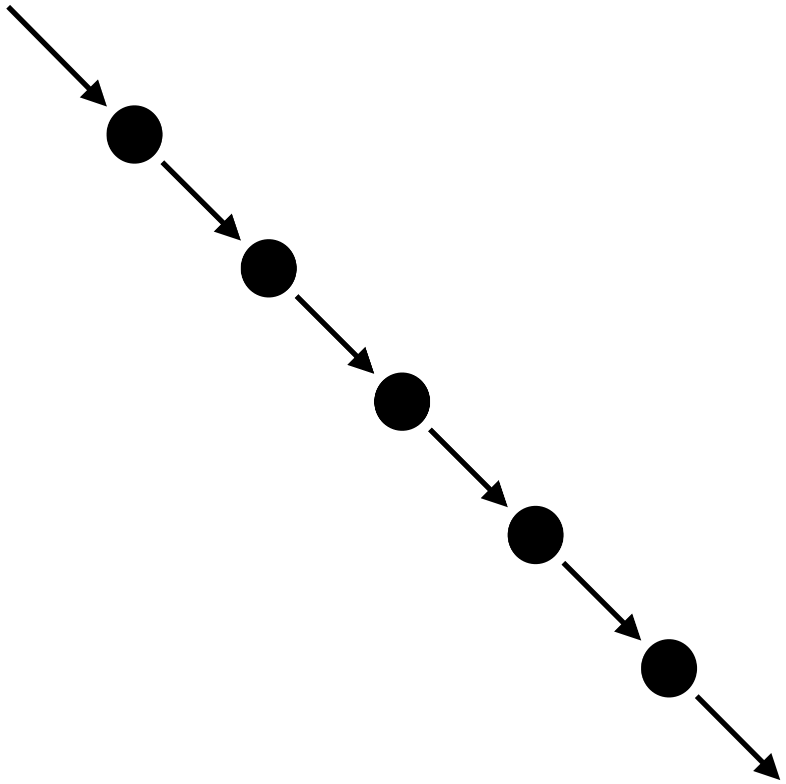
PERK: Shared Permutation on Permuted Kernel Problem

Standard MPC-in-the-Head



AIMer, Biscuit, MIRA, MiRitH
MQOM, RYDE, SDitH

Path-based MPC-in-the-Head



PERK

FAEST: VOLE-in-the-Head

VOLE: *vector oblivious linear evaluation*

"FAEST is the first AES-based signature scheme to be smaller than SPHINCS+"

Will be presented at Crypto'23 the 23rd August

Conclusion

Advantages and limitations

- Limitations
  - Relatively ***slow*** *(few milliseconds)*
    - Greedy use of symmetric cryptography
  - Relatively ***large*** signatures (*4-10 KB for L1*)
  - ***Quadratic*** growth in the security level

Advantages and limitations

- Limitations
  - Relatively ***slow*** *(few milliseconds)*
    - Greedy use of symmetric cryptography
  - Relatively ***large*** signatures (*4-10 KB for L1*)
  - ***Quadratic*** growth in the security level

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

Conclusion

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

Conclusion

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

- Perspectives
  - MPCitH transformations: new works in 2022 (hypercube, threshold)

    Could lead to follow-up works

  - Signatures with advanced functionalities:

    ring signatures, threshold signatures, multi-signatures,

    blind signatures, …

Conclusion

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

- Perspectives
  - MPCitH transformations: new works in 2022 (hypercube, threshold)

    Could lead to follow-up works

  - Signatures with advanced functionalities:

    ring signatures, threshold signatures, multi-signatures,

    blind signatures, …

Thank you for your attention.