Code-based Signatures from Secure Multiparty Computation

Thibauld Feneuil

SIAM AG23 — Advances in Code-based Signature

July 12, 2023, Eindhoven

CRYPTOEXPERTS
WE INNOVATE TO SECURE YOUR BUSINESS

SORBONNE
UNIVERSITÉ

Registration and travel support for this presentation was provided by the Eindhoven University of Technology.

Introduction

Proof of knowledge

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

Commitment

Challenge 1

Response 1

⋮

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

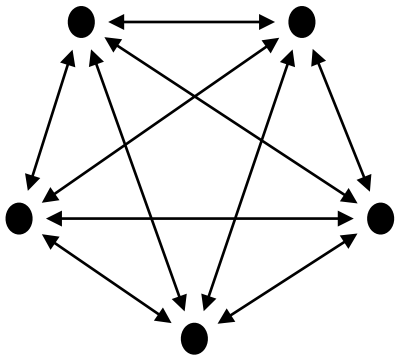
- **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)
- Turn an MPC protocol into a zero knowledge proof of knowledge
- **Generic**: can be apply to any cryptographic problem
- Convenient to build (candidate) **post-quantum signature** schemes
- **Picnic**: submission to NIST (2017)
- Recent NIST call (01/06/2023): 7 MPCitH schemes / 50 submissions

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

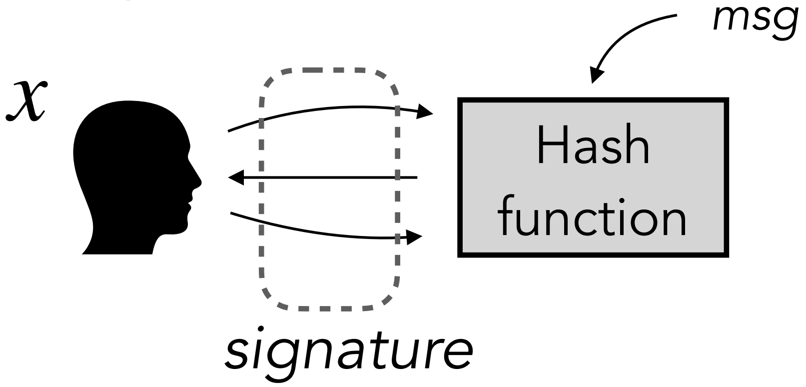


Input sharing $[\![x]\!]$

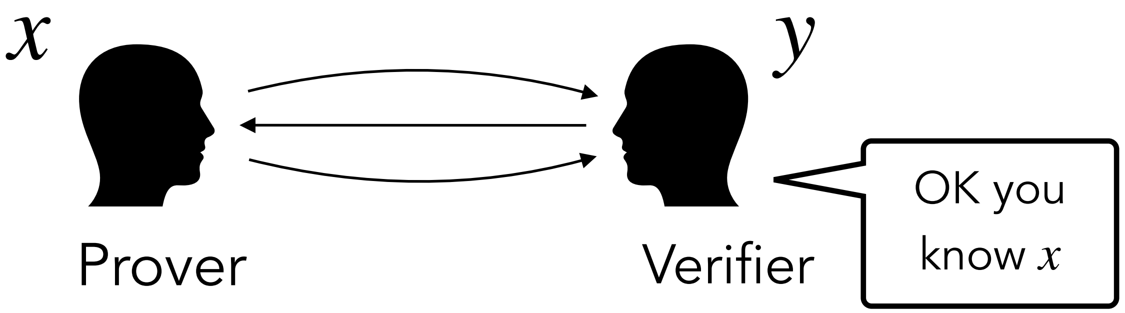
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

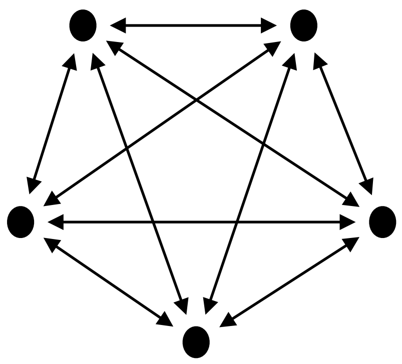


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

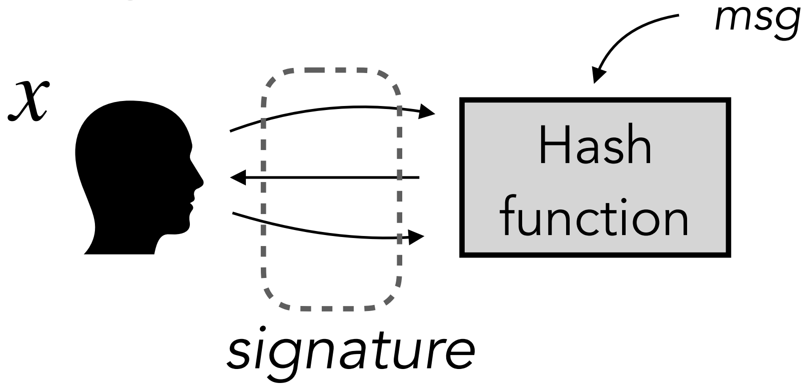


Input sharing $[\![x]\!]$

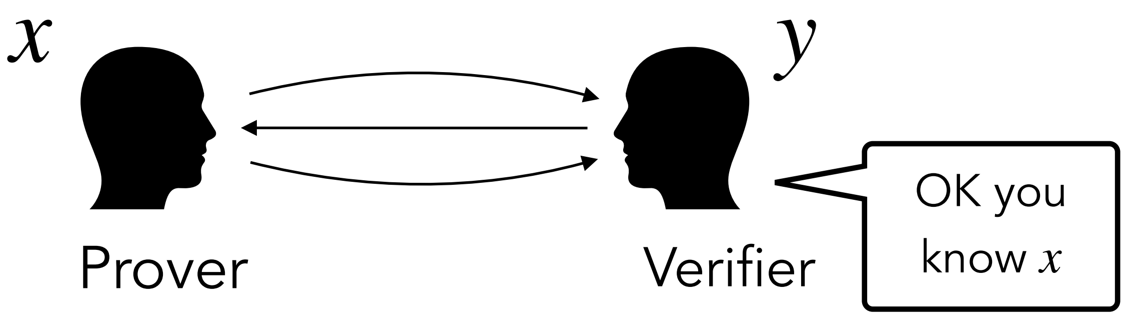
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof



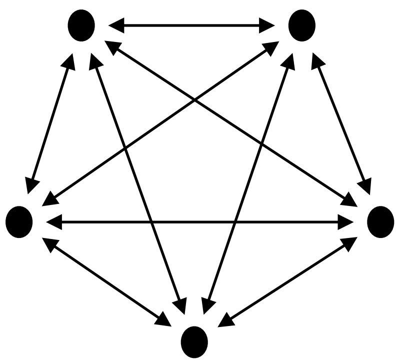
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$ s.t. $x = [\![x]\!]_1 + \ldots + [\![x]\!]_N$

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

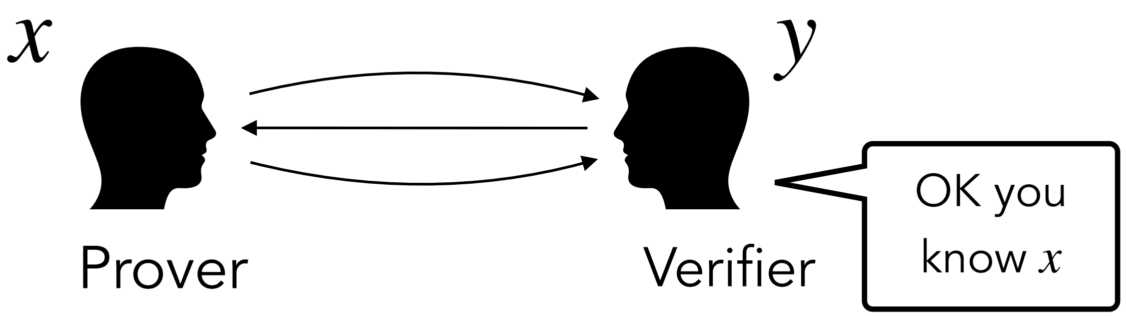


Input sharing $[\![x]\!]$

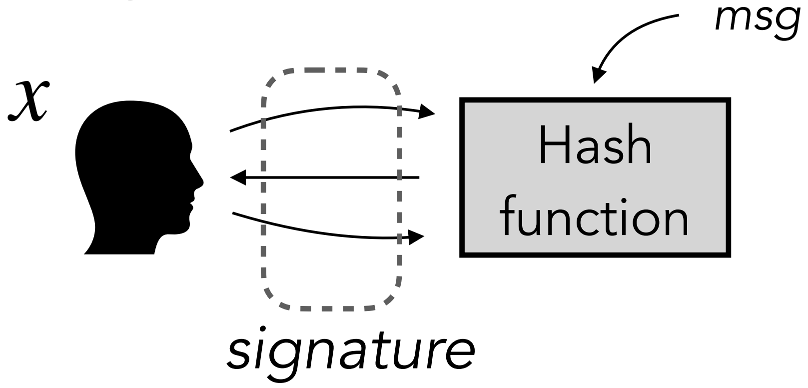
Joint evaluation of:

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

Zero-knowledge proof



Signature scheme

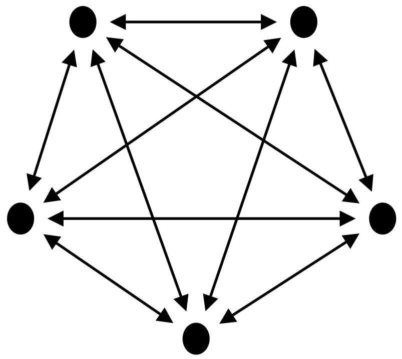


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

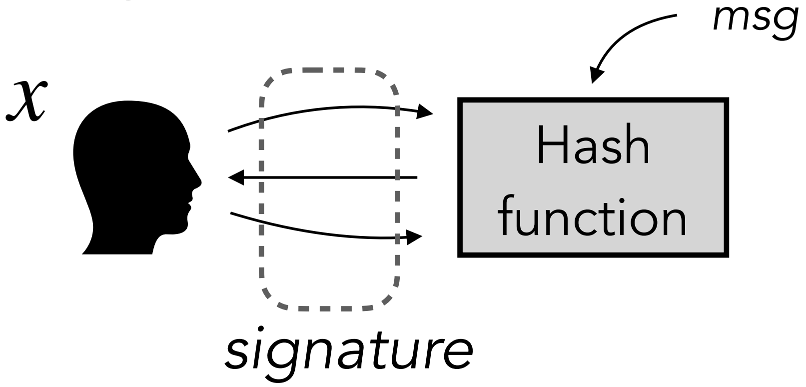


Input sharing $[\![x]\!]$

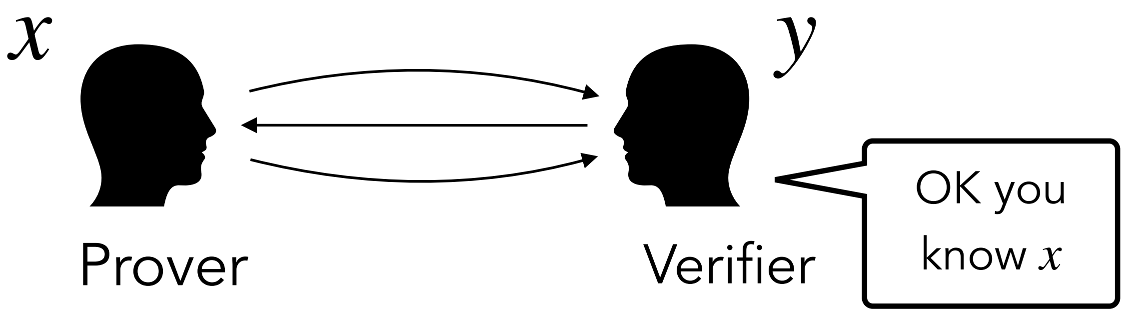
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

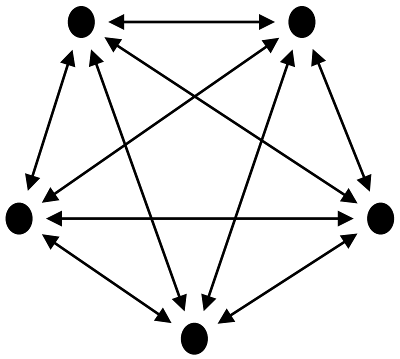


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

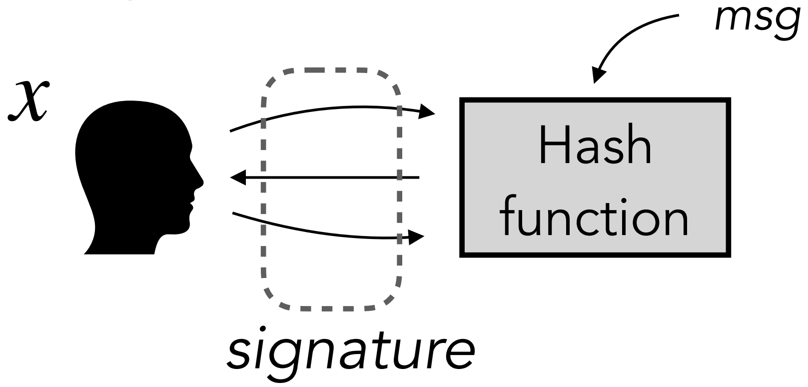


Input sharing $[\![x]\!]$

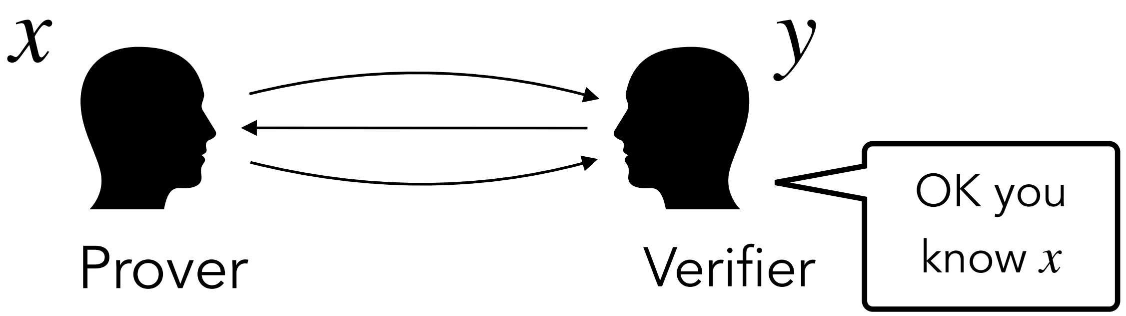
Joint evaluation of:

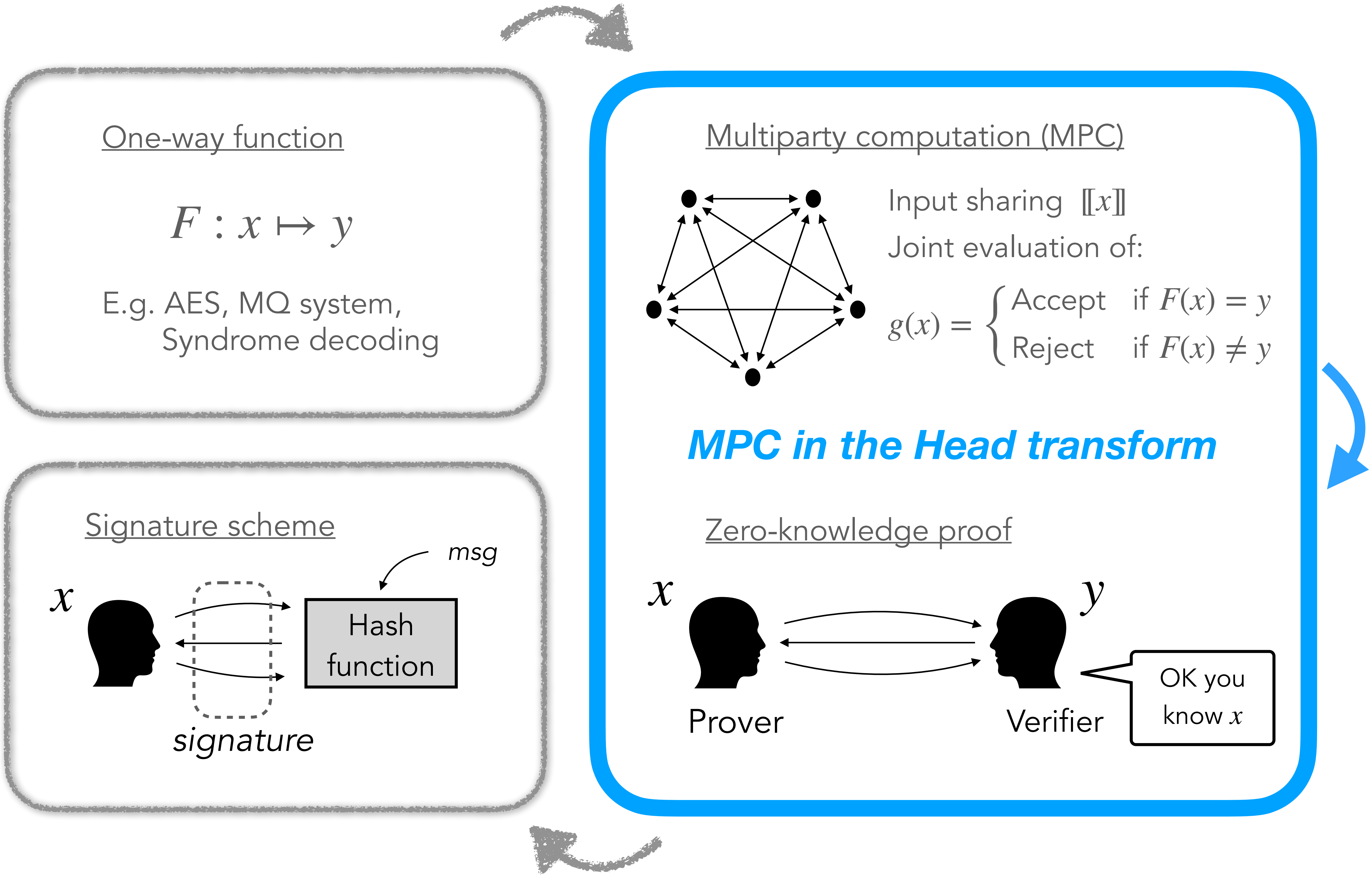
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Signature scheme



Zero-knowledge proof





One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Signature scheme

Multiparty computation (MPC)

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

Joint evaluation of:

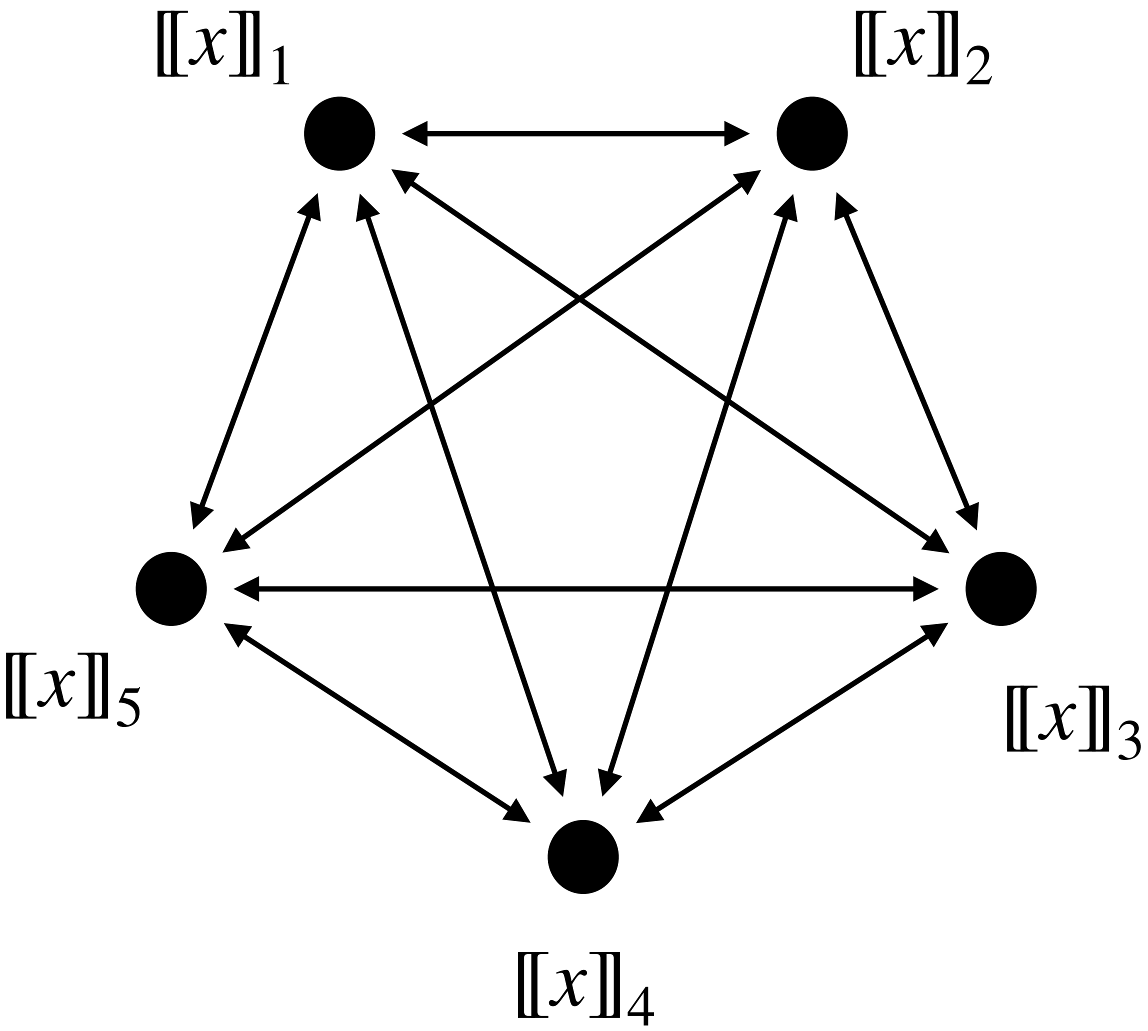
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

MPC in the Head transform

Zero-knowledge proof

MPCitH: general principle

MPC model

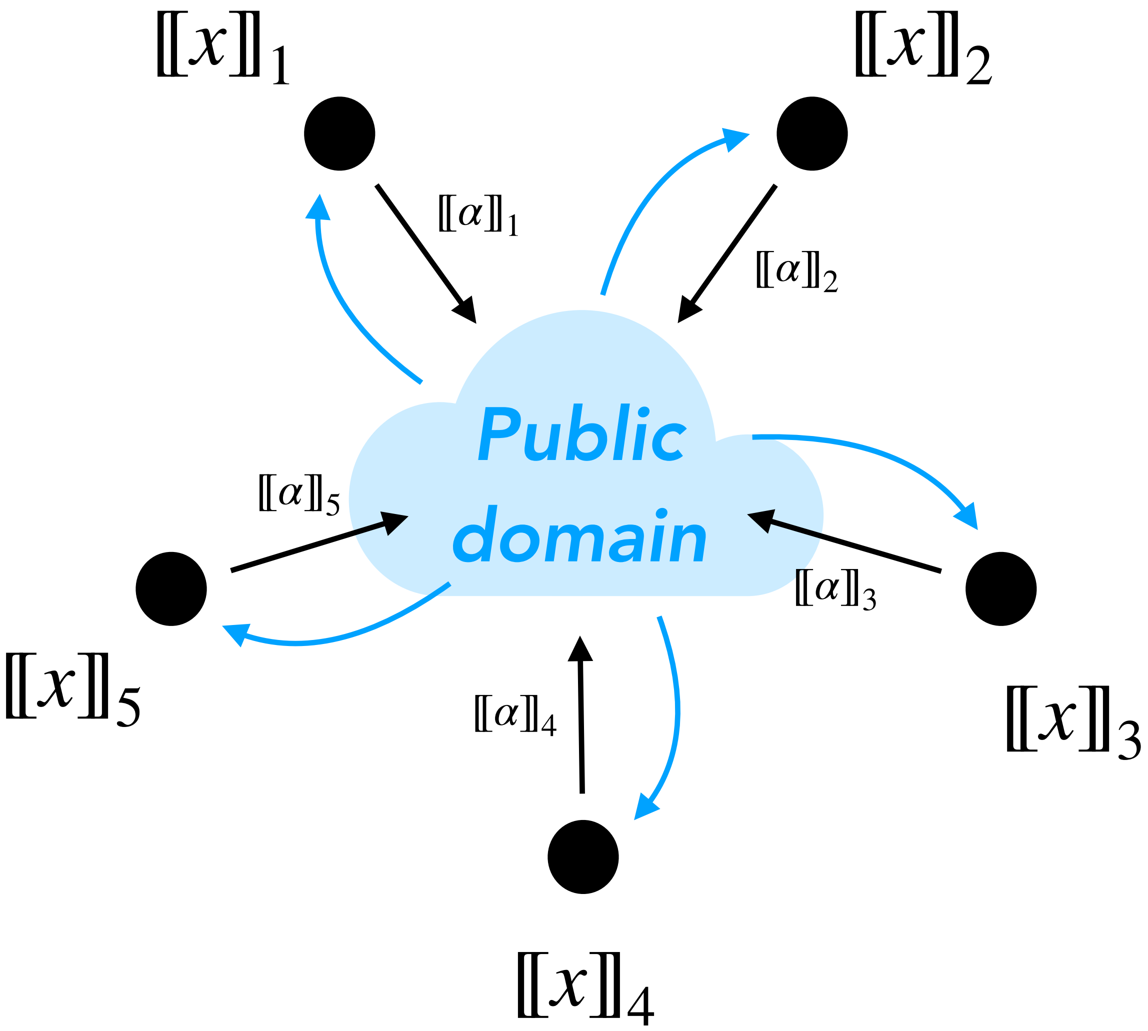


- **Jointly compute**

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

- $(N-1)$ **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



- **Jointly compute**

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

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

MPCitH transform

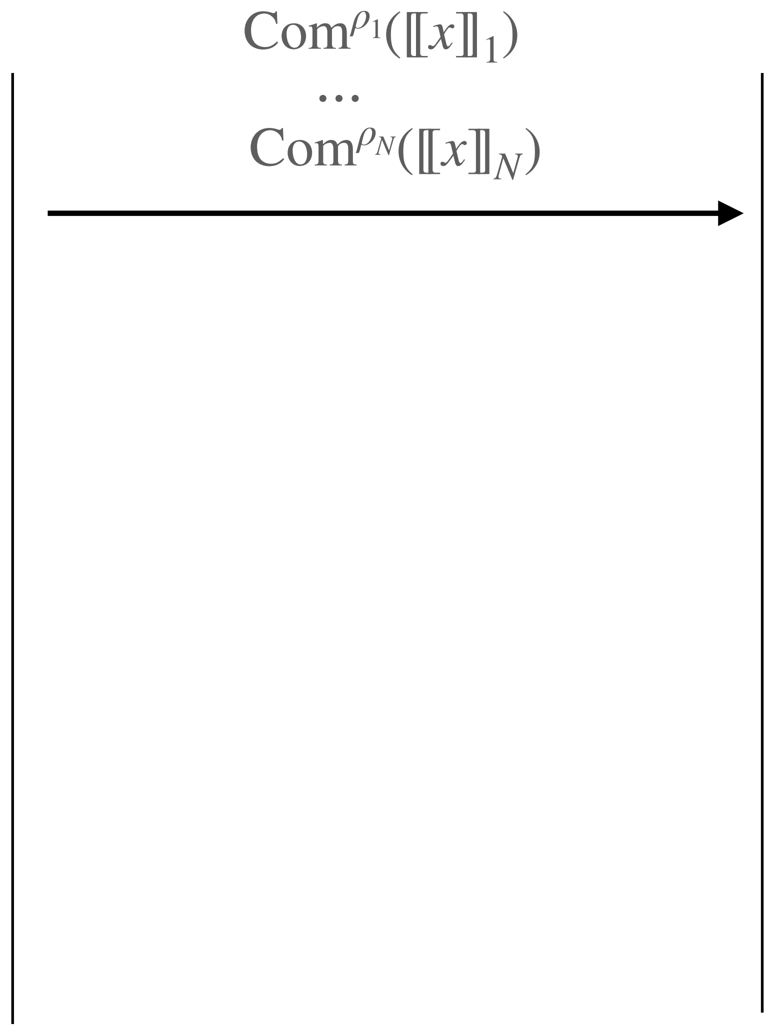
Prover

Verifier

MPCitH transform

① Generate and commit shares

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



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

$\ldots$

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

Prover

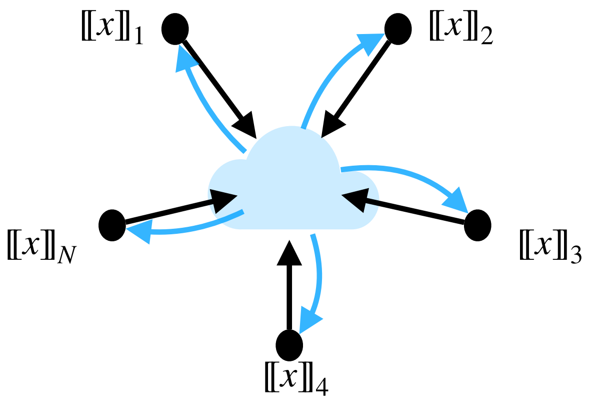
Verifier

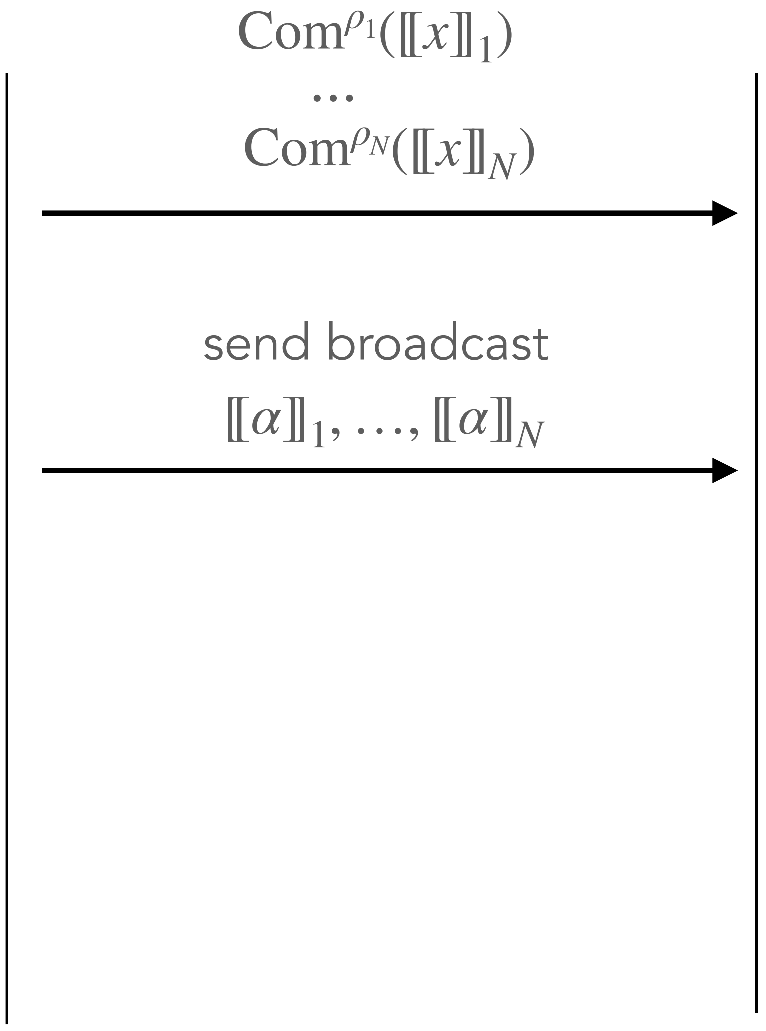
MPCitH transform

① Generate and commit shares

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

② Run MPC in their head





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

$\ldots$

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

send broadcast

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

Prover

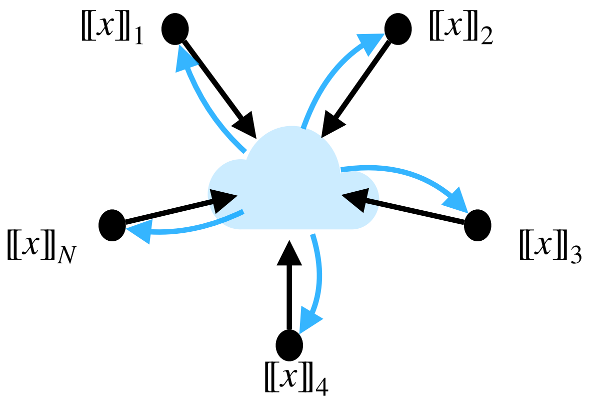
Verifier

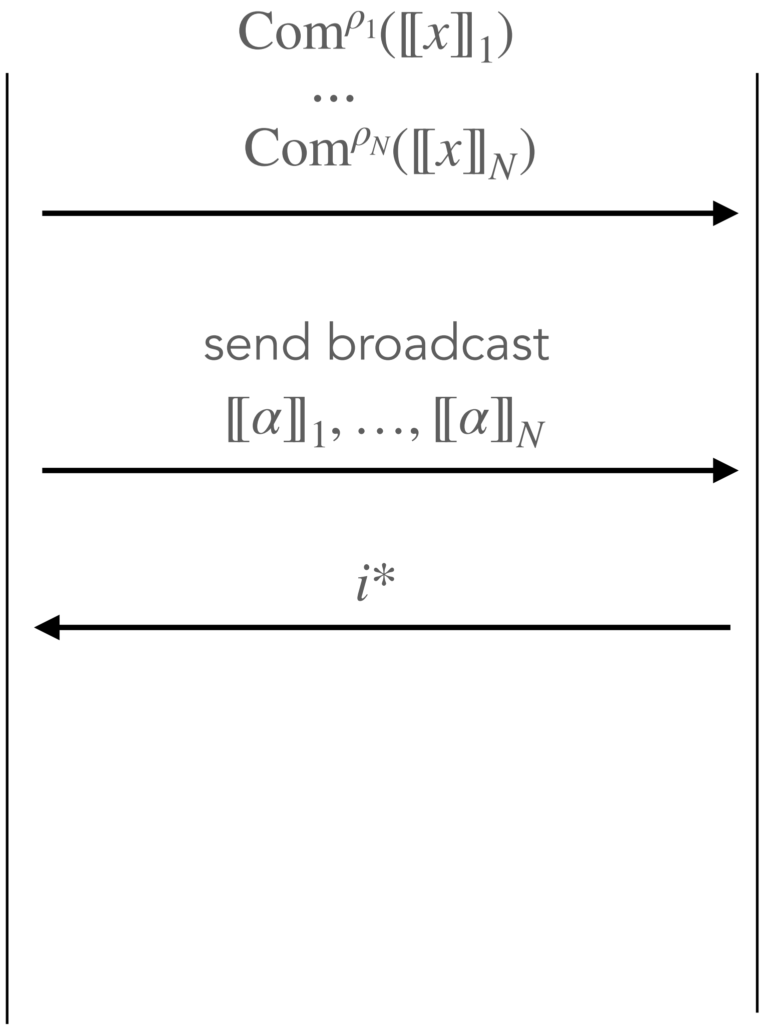
MPCitH transform

① Generate and commit shares

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

② Run MPC in their head





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

$\ldots$

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

send broadcast

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

i^*

③ Choose a random party

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

Prover

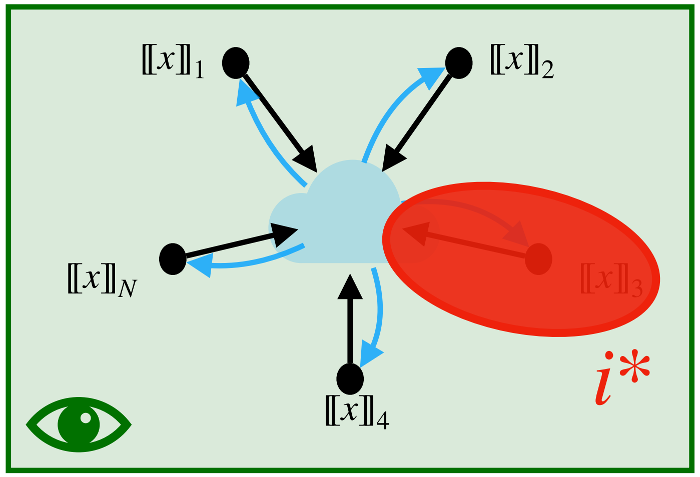
Verifier

MPCitH transform

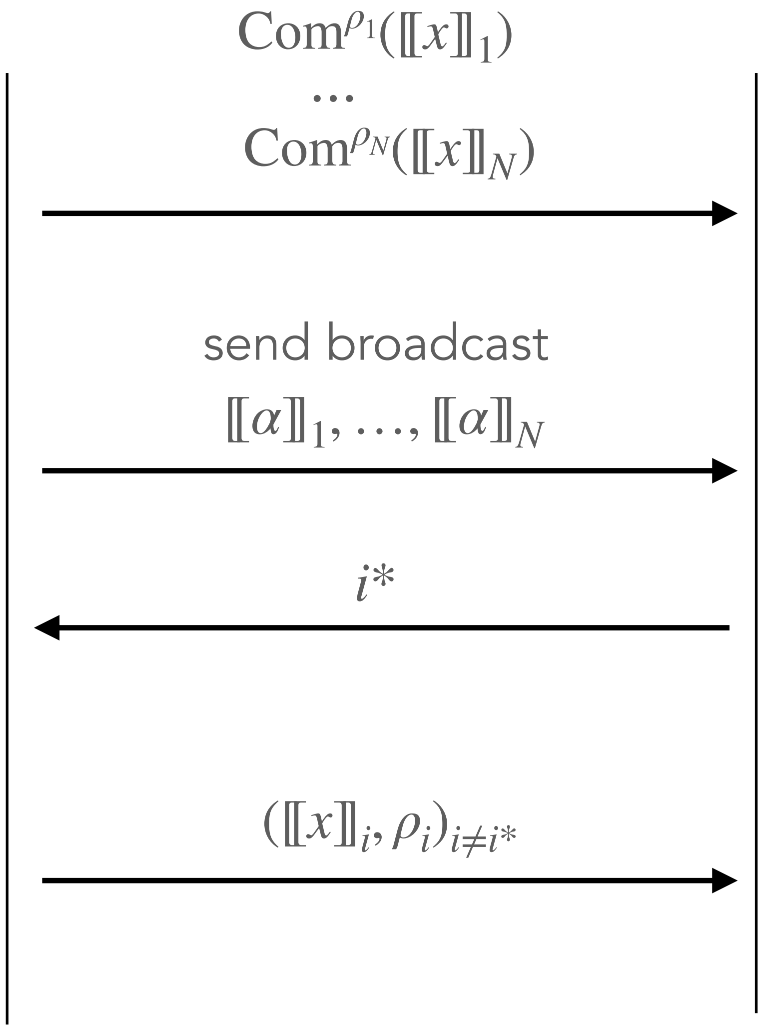
① Generate and commit shares

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

② Run MPC in their head



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



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

$\cdots$

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

send broadcast

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

i^*

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

③ Choose a random party

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

Prover

Verifier

MPCitH transform

① Generate and commit shares

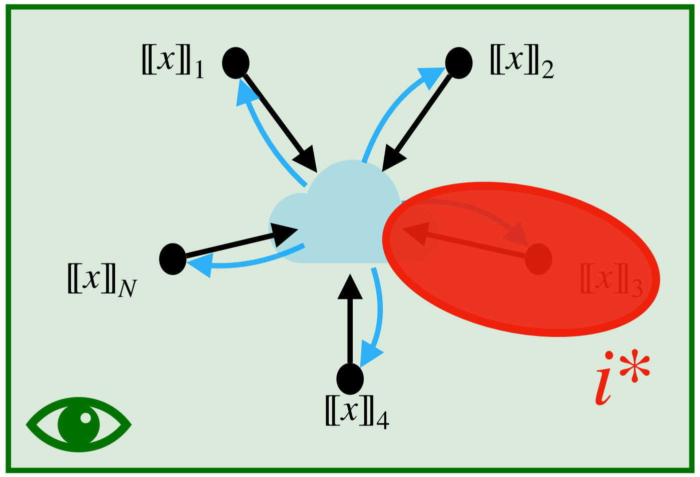
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

$\ldots$

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

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

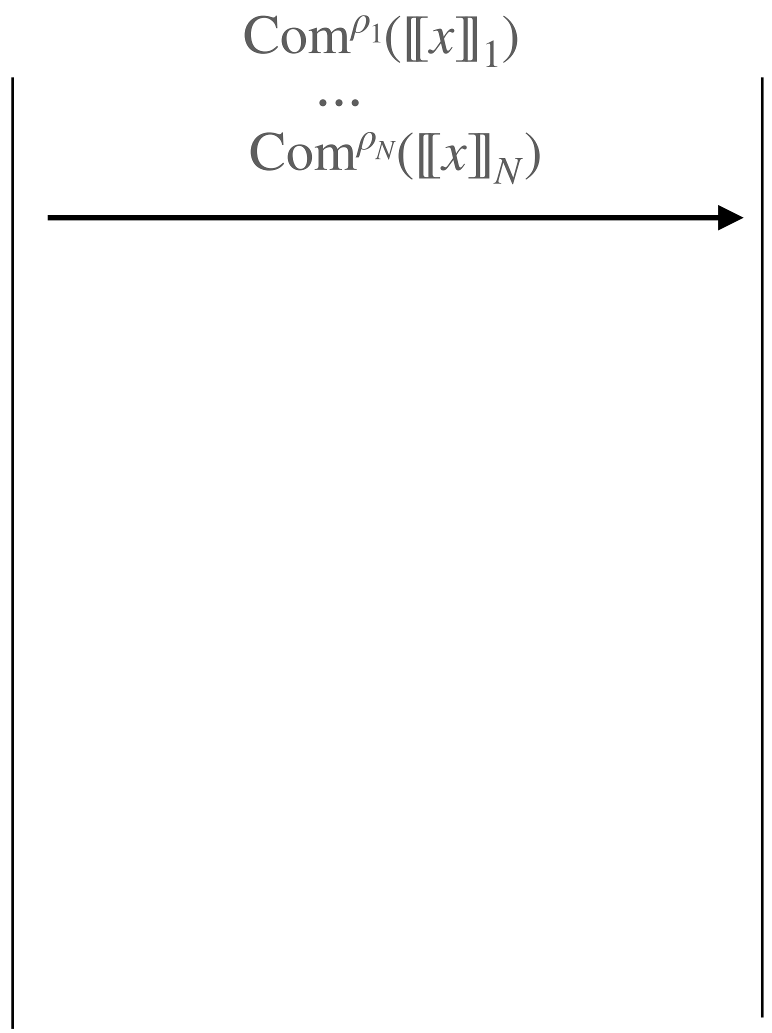
MPCitH transform

① Generate and commit shares

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

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

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



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

$\ldots$

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

Malicious Prover

Verifier

MPCitH transform

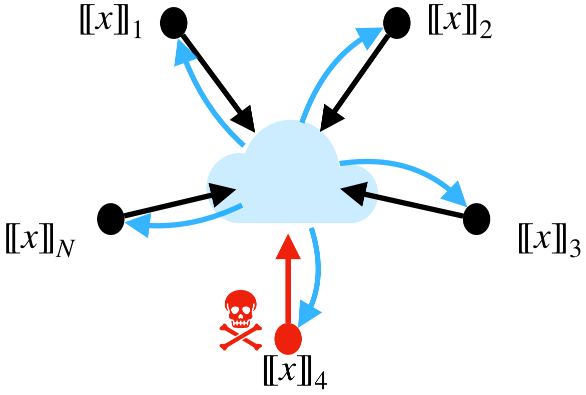
① Generate and commit shares

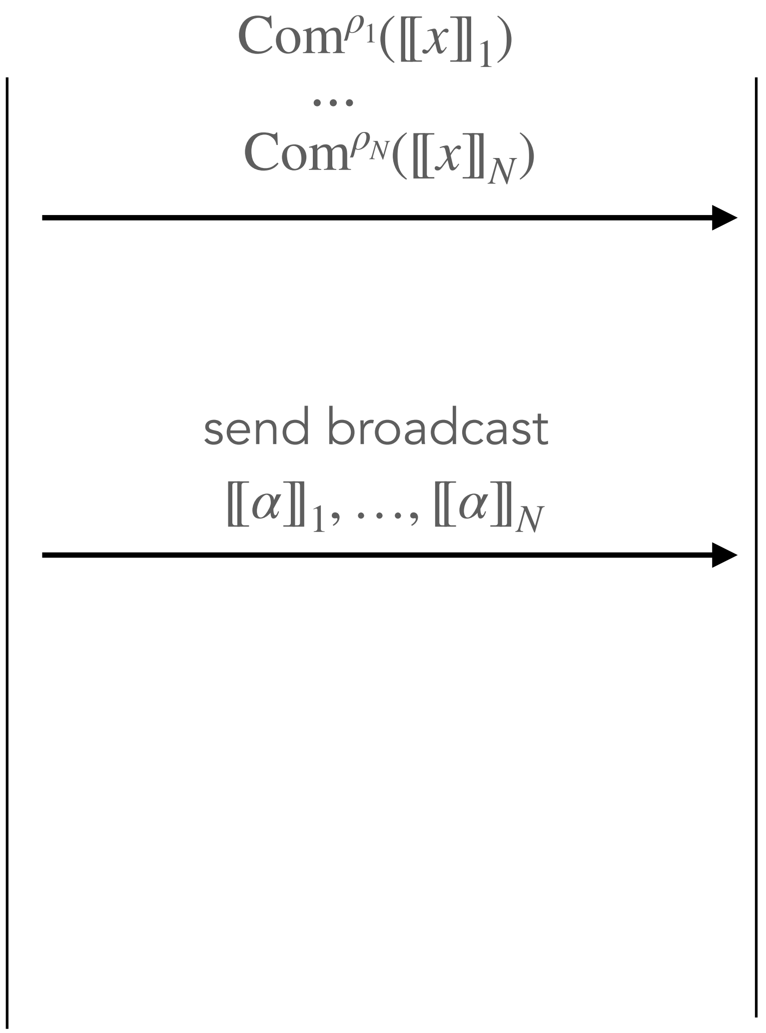
$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N)$

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

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

② Run MPC in their head





Malicious Prover

Verifier

MPCitH transform

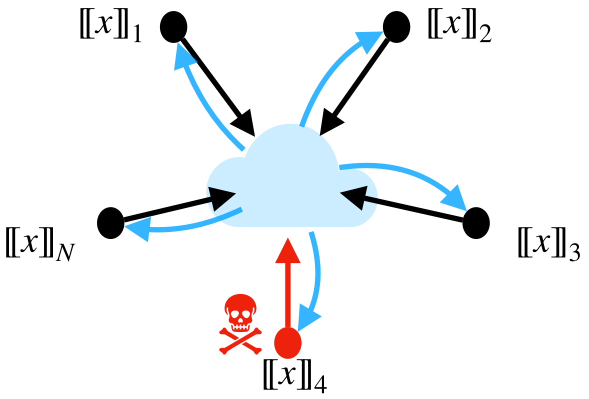
① Generate and commit shares

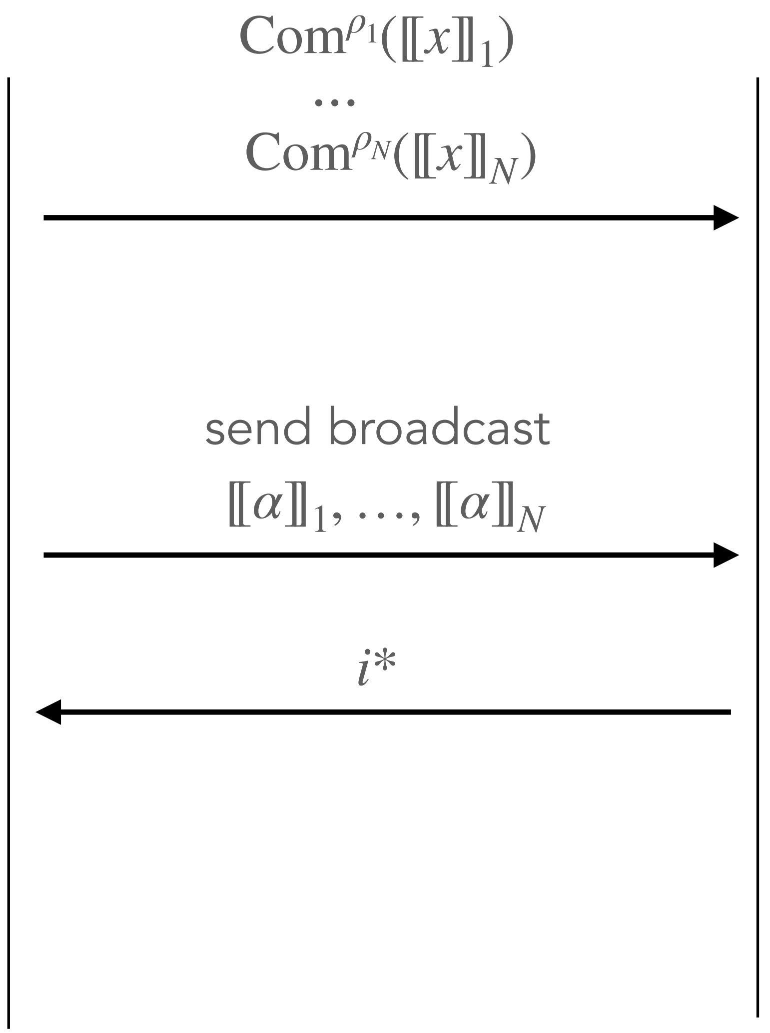
$[\![x]\!] = ([\![x]\!]_1, \dots, [\![x]\!]_N)$

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

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

② Run MPC in their head





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

$\dots$

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

send broadcast

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

③ Choose a random party

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

i^*

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

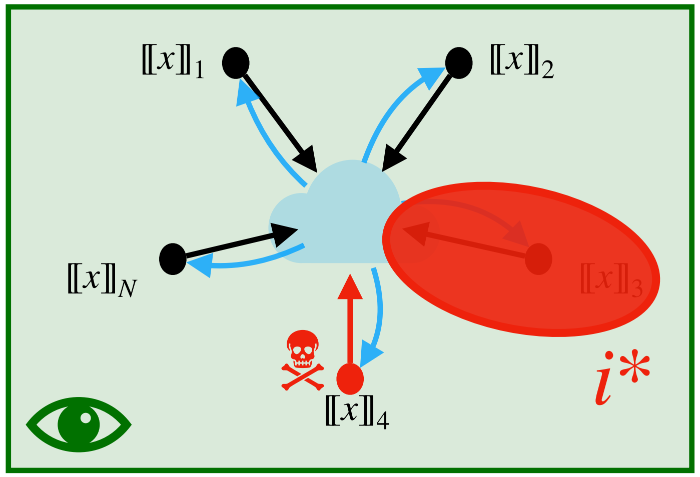
$x := [\![x]\!]_1 + \ldots + [\![x]\!]_N$

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

$\ldots$

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

② Run MPC in their head



send broadcast

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

③ Choose a random party

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

i^*

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

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

Malicious Prover

Verifier

MPCitH transform

① Generate and commit shares

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

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

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

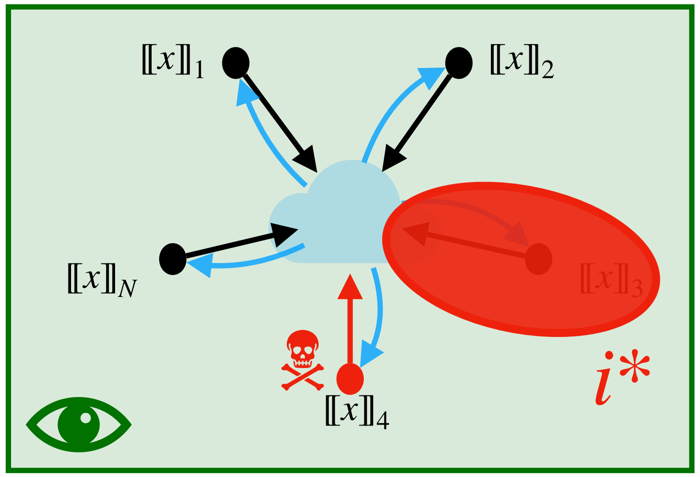
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\cdots$

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

→

② Run MPC in their head



send broadcast

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

→

③ Choose a random party

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

i^*

←

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

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

→

⑤ Check $\forall i \neq i^*$

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

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

Malicious Prover

Verifier

❌ **Cheating detected!**

MPCitH transform

① Generate and commit shares

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

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

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

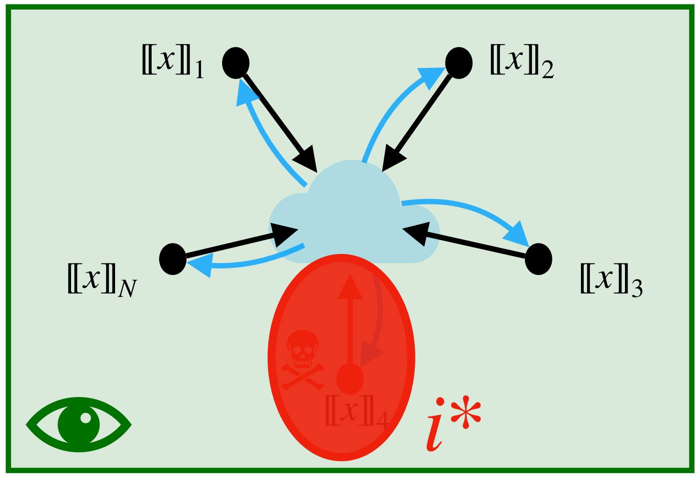
$\mathrm{Com}^{\rho_1}([\![x]\!]_1)$

$\dots$

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

$\longrightarrow$

② Run MPC in their head



send broadcast

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

$\longrightarrow$

③ Choose a random party

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

i^*

$\longleftarrow$

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

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

$\longrightarrow$

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:**

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

- **Parallel repetition**

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

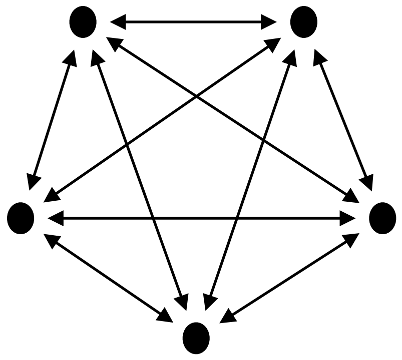
Code-based signature schemes

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

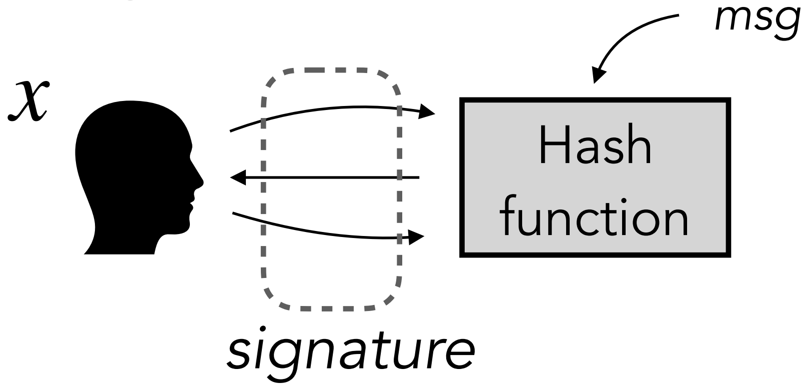


Input sharing $[\![x]\!]$

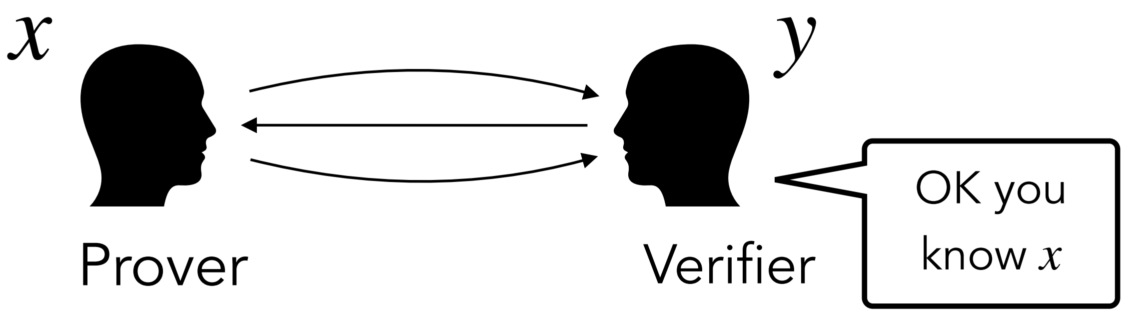
Joint evaluation of:

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

Signature scheme



Zero-knowledge proof

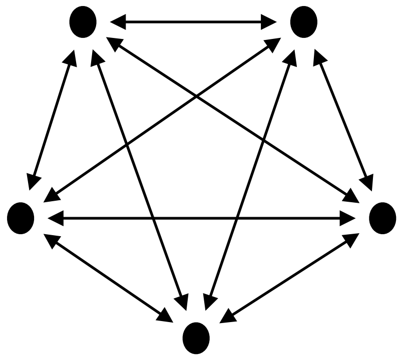


One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



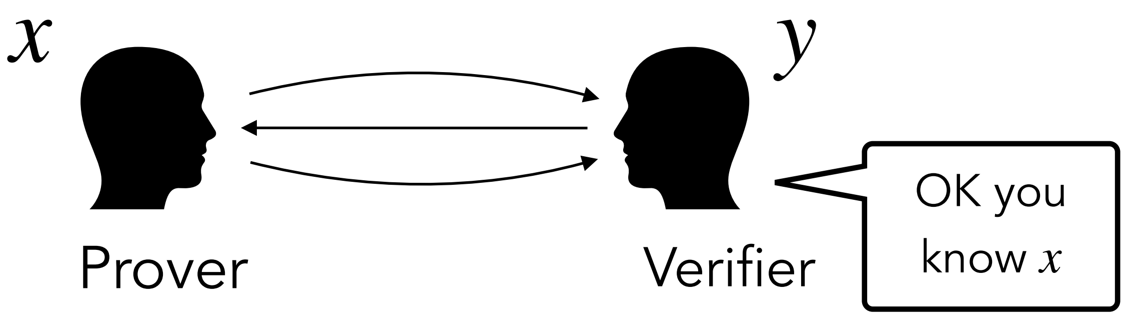
Input sharing $[\![x]\!]$

Joint evaluation of:

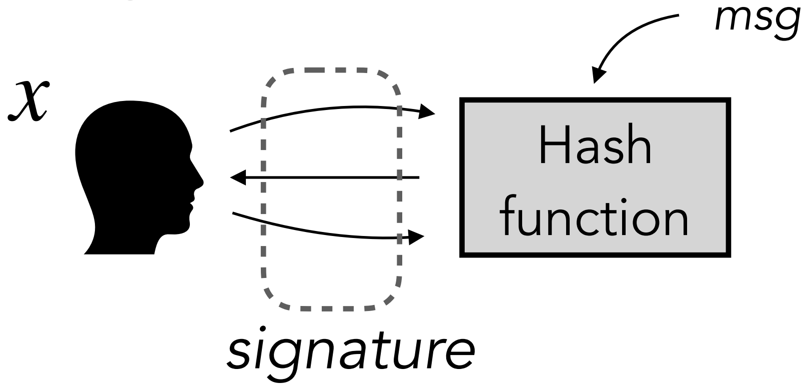
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

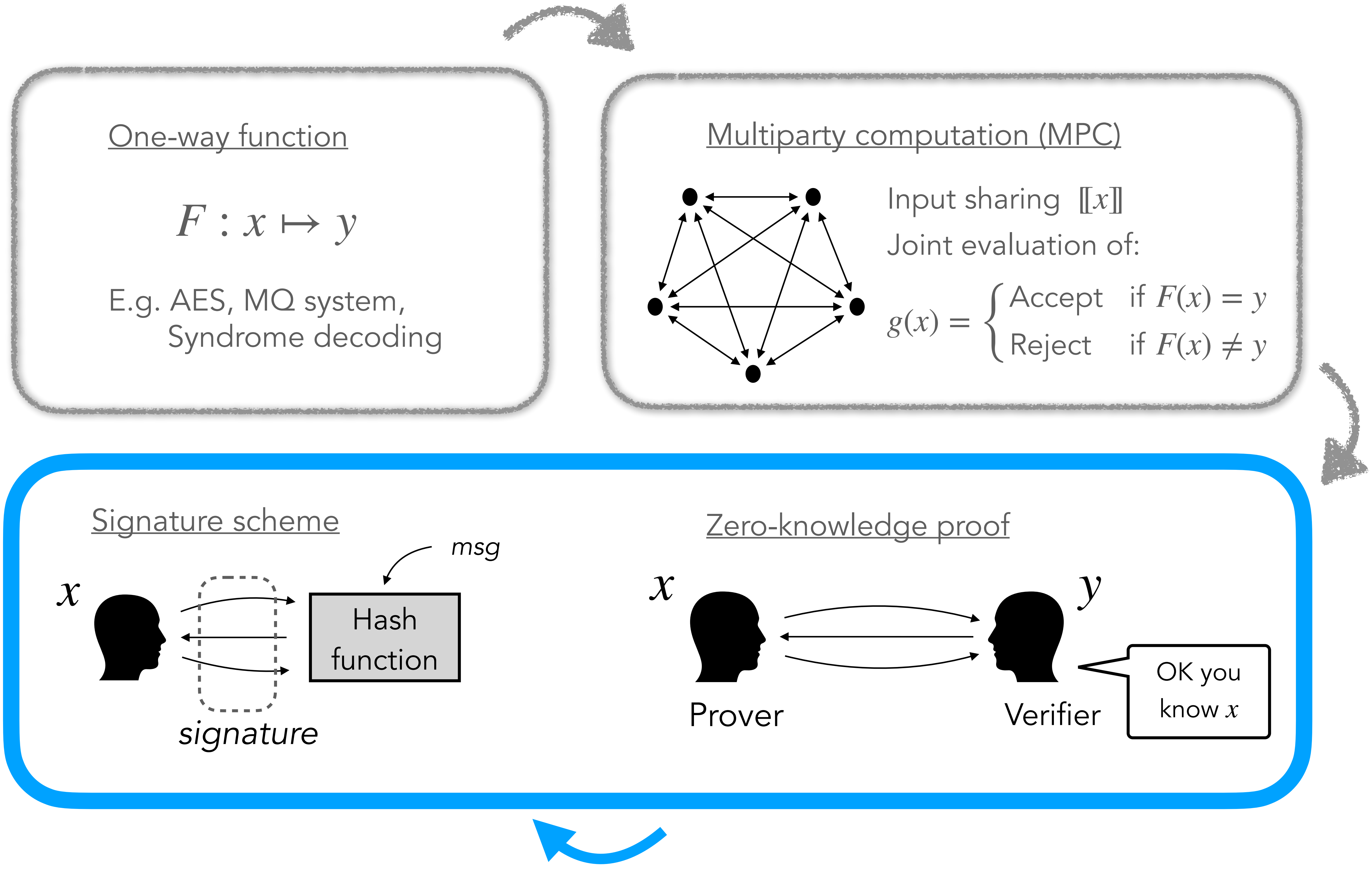
MPC in the Head transform

Zero-knowledge proof



Signature scheme





One-way function

$F : x \mapsto y$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

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

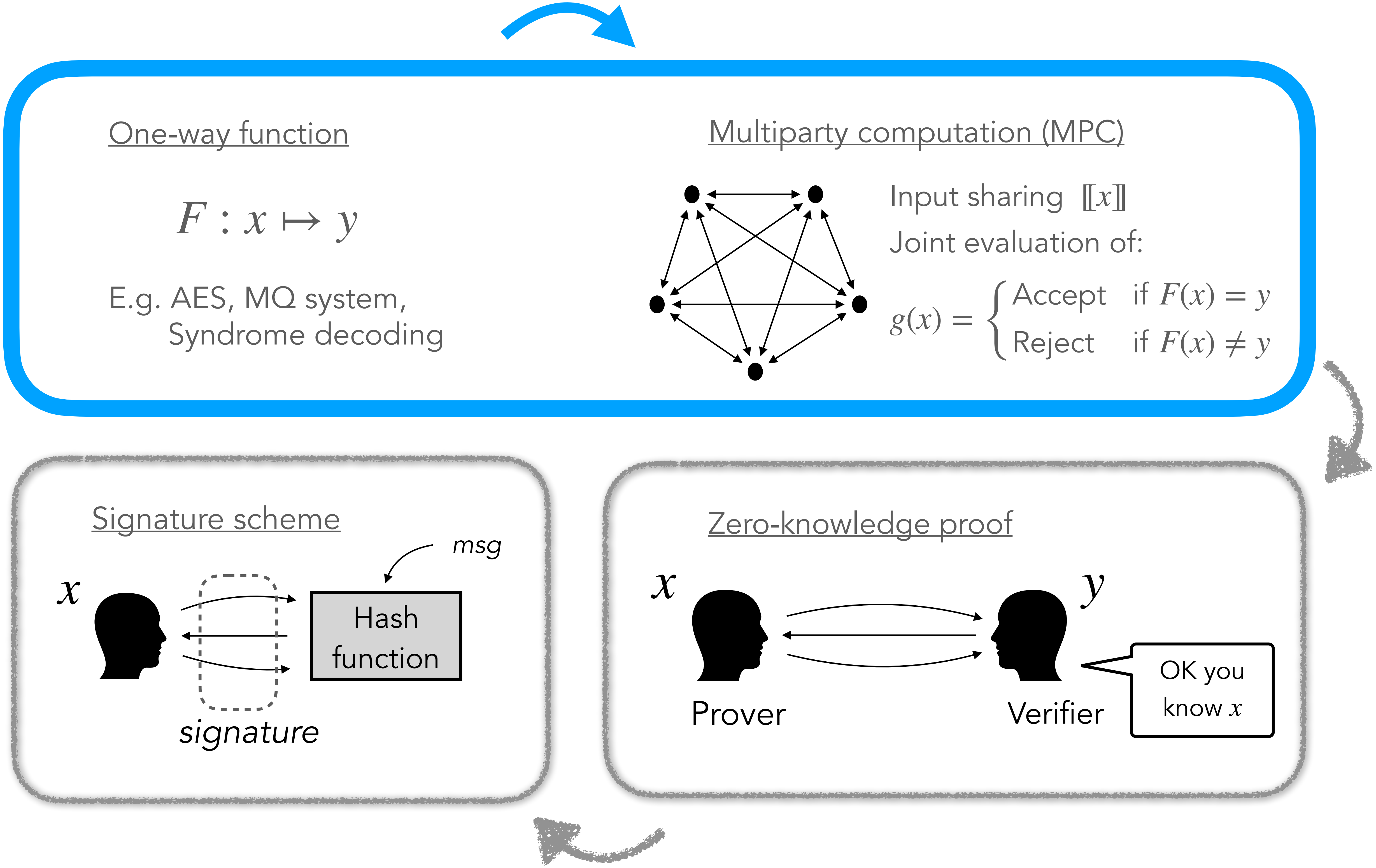
Joint evaluation of:

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

Signature scheme

Zero-knowledge proof

Fiat-Shamir transform



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

Submitted code-based candidates at NIST call

Syndrome Decoding Problem
in *Hamming* metric

Syndrome Decoding Problem
in *rank* metric

Given a matrix H and a vector y, find x such that $y = Hx$ and

$$\mathrm{wt}_H(x) \leq w$$

$$\mathrm{wt}_R(x) \leq r$$

$(x \in \mathbb{F}_{q^m}^n)$

Submitted code-based candidates at NIST call

Syndrome Decoding Problem
in *Hamming* metric

Syndrome Decoding Problem
in *rank* metric

Given a matrix H and a vector y, find x such that $y = Hx$ and

$$\mathrm{wt}_H(x) \leq w$$

$$\mathrm{wt}_R(x) \leq r$$

$(x \in \mathbb{F}_{q^m}^n)$

The MPC protocol checks that the shared input x satisfies $y = Hx$ and

there exists a degree-w polynomial Q such that

$$\forall i,\ x_i \cdot Q(\gamma_i) = 0.$$

there exists a degree-q^r q-polynomial $L := \sum_{i=0}^{r} L_i X^{q^i}$ such that

$$\forall i,\ L(x_i) = 0.$$

[FJR22] Feneuil T., Joux A., Rivain M. *Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs*. Crypto 2022

[Fen22] Feneuil T. *Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP*. ePrint 2022-1512

Submitted code-based candidates at NIST call

Syndrome Decoding Problem in *Hamming* metric

Syndrome Decoding Problem in *rank* metric

Given a matrix H and a vector y, find x such that $y = Hx$ and

$$\mathrm{wt}_H(x) \leq w$$

$$\mathrm{wt}_R(x) \leq r$$

$(x \in \mathbb{F}_{q^m}^n)$

The MPC protocol checks that the shared input x satisfies $y = Hx$ and

there exists a degree-w polynomial Q such that

$$\forall i,\ x_i \cdot Q(\gamma_i) = 0.$$

$$\Big\downarrow \textit{ MPCitH + FS}$$

SD-in-the-Head (SDitH)

C. Aguilar Melchor, T. Feneuil, N. Gama, S. Gueron, J. Howe, D. Joseph, A. Joux, E. Persichetti, T. Randrianarisoa, M. Rivain, D. Yue.

there exists a degree-q^r q-polynomial $L := \sum_{i=0}^{r} L_i X^{q^i}$ such that

$$\forall i,\ L(x_i) = 0.$$

$$\Big\downarrow \textit{ MPCitH + FS}$$

RYDE

N. Aragon, M. Bardet, L. Bidoux, J.-J. Chi-Domínguez, V. Dyseryn, T. Feneuil, P. Gaborit, A. Joux, M. Rivain, J.-P. Tillich, A. Vinçotte.

Performances

Shamir's sharing

	Short Instance			Fast Instance		
	$\|\text{sig}\|$	t_{sign}	t_{verify}	$\|\text{sig}\|$	t_{sign}	t_{verify}
SDitH-256	8.3	13.4	12.5	10.1	5.1	1.6
SDitH-251	8.3	22.1	21.2	10.1	4.4	0.6
RYDE	6.0	23.4	20.1	7.4	5.4	4.4

Additive sharing

NIST Category I
Isochronous implementations
Size in kilobytes, timing in Mcycles
@2.60GHz: 1 millisecond $\approx$ *2.6 Mcycles*

Performances

- *How it scales for higher security levels?*

	SDitH			**RYDE**		
	|pk|	***Short***	***Fast***	|pk|	***Short***	***Fast***
Category I	120 B	8.3 KB	10.1 KB	86 B	6.0 KB	7.4 KB
Category III	183 B	19.2 KB	24.9 KB	131 B	12.9 KB	16.4 KB
Category V	234 B	33.4 KB	43.9 KB	188 B	22.8 KB	29.1 KB

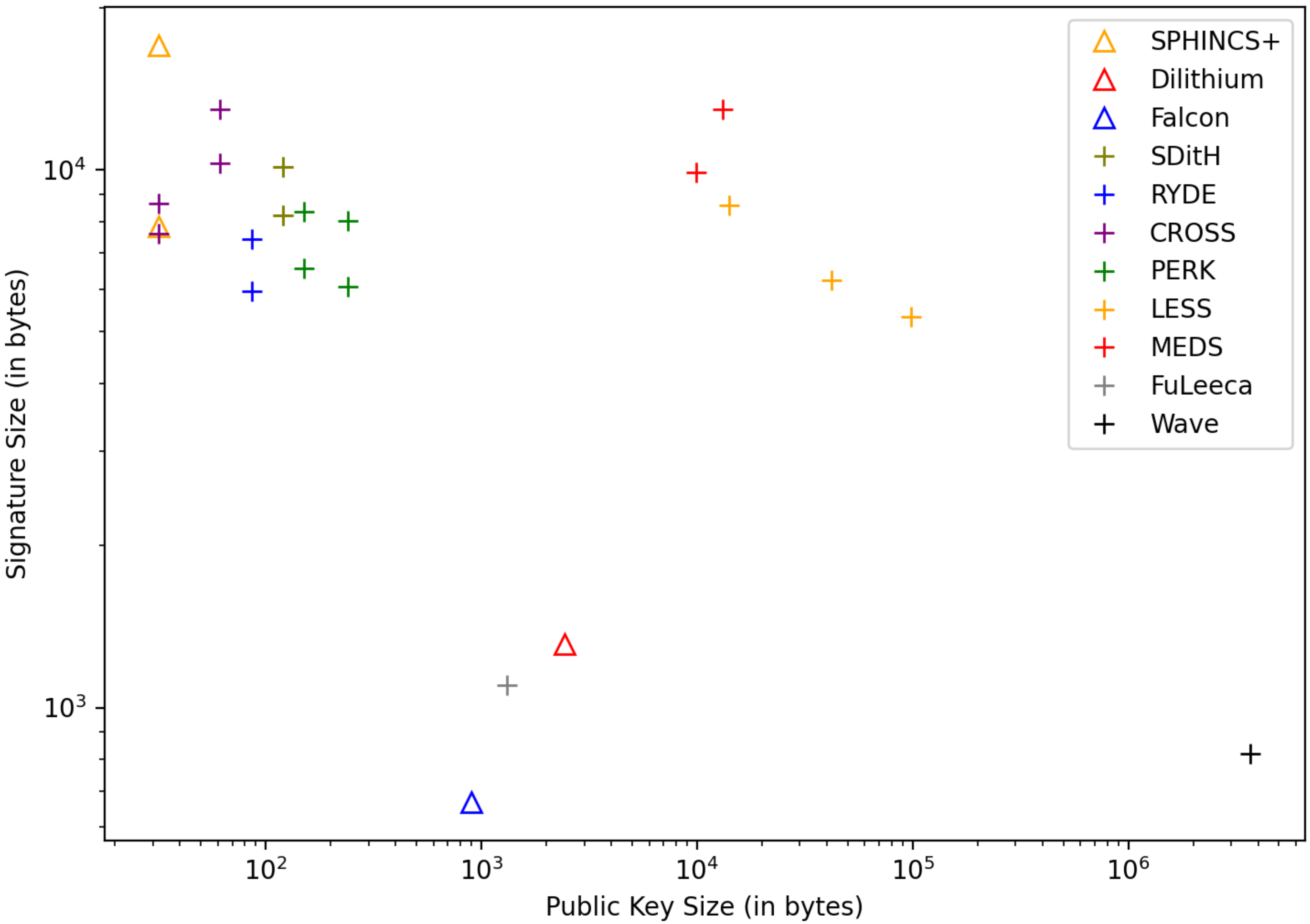
Comparison

NIST Category I
Size in bytes, timing in Mcycles

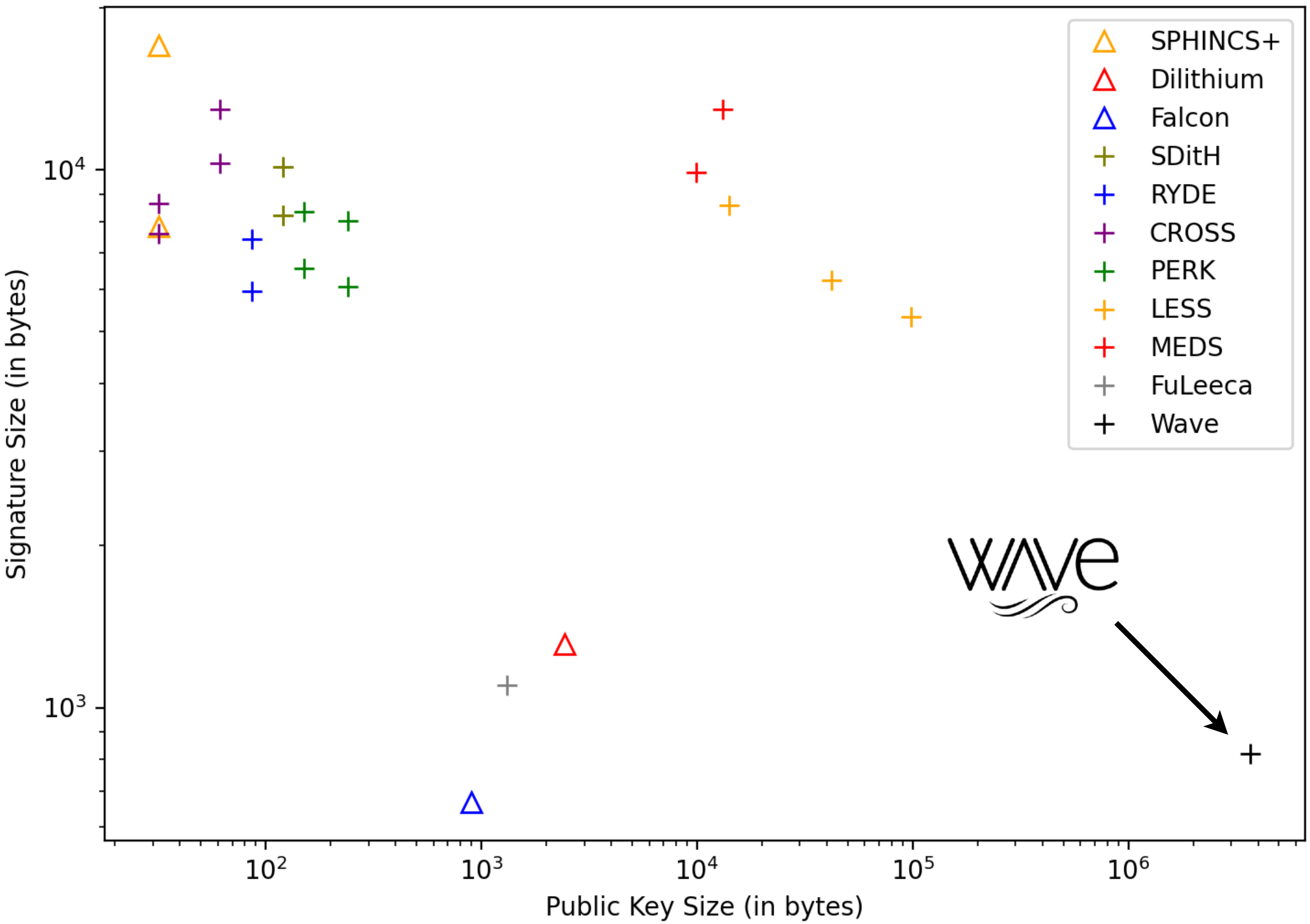
	$\lvert pk\rvert$	$\lvert sig\rvert$	$\lvert sig\rvert+\lvert pk\rvert$	t_{sign}	t_{verify}
SDitH-256-short	120	8241	8361	13.4	12.5
SDitH-251-short	120	8241	8361	22.1	21.2
SDitH-256-fast	120	10117	10237	5.1	1.6
SDitH-251-fast	120	10117	10237	4.4	0.6
RYDE-short	86	5956	6042	23.4	20.1
RYDE-fast	86	7446	7532	5.4	4.4
PERK-short3	150	6560	6710	39	27
PERK-short5	240	6060	6300	36	25
PERK-fast3	150	8350	8500	7.6	5.3
PERK-fast5	240	8030	8270	7.2	5.1
CROSS-fast	61	12944	13005	6.8	3.2
CROSS-small	61	10304	10365	22.0	10.3
CROSS-G-fast	32	8665	8697	3.1	2.1
CROSS-G-small	32	7625	7657	11.0	7.8
LESS-1b	13700	8400	22100	263.6	271.4
LESS-1i	41100	6100	47200	254.3	263.4
LESS-1s	95900	5200	101100	206.6	213.4
MEDS-9923	9923	9896	19819	518.1	515.6
MEDS-13220	13220	12976	26196	88.9	46.0
FuLeeca	1318	1100	2418	1846.8	1.26
Wave-822	3677390	822	3678212	1160	1.23

`https://wave-sign.org/`
`https://www.meds-pqc.org/`
`https://www.less-project.com/`
`https://www.cross-crypto.com/`
`https://pqc-perk.org/`
`https://pqc-ryde.org/`
`https://sdith.org/`
`https://www.ce.cit.tum.de/.../fuleeca/`

Comparison

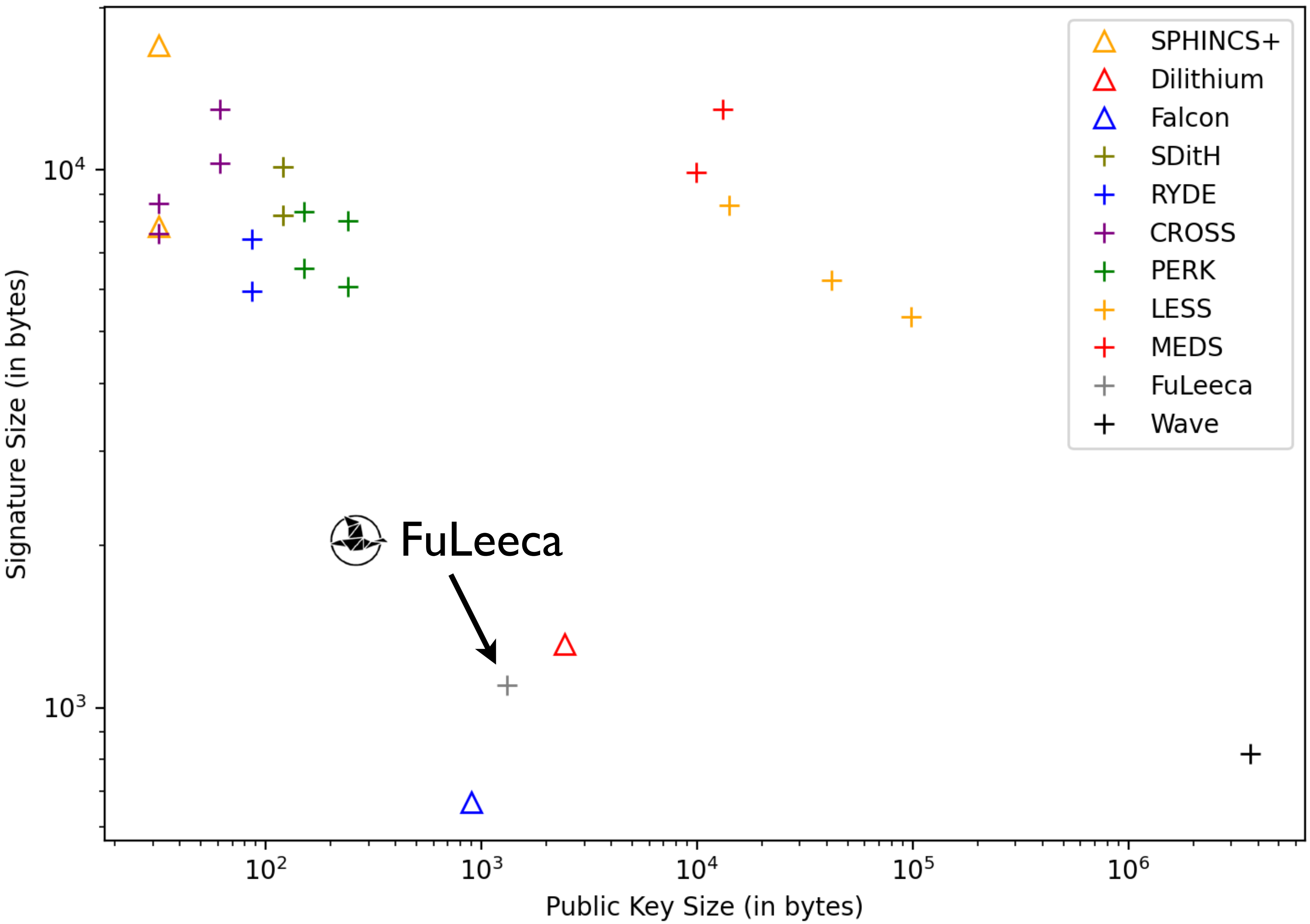

Signature Size (in bytes)
Public Key Size (in bytes)
10^3
10^4
10^2
10^3
10^4
10^5
10^6
△ SPHINCS+
△ Dilithium
△ Falcon
+ SDitH
+ RYDE
+ CROSS
+ PERK
+ LESS
+ MEDS
+ FuLeeca
+ Wave

Comparison



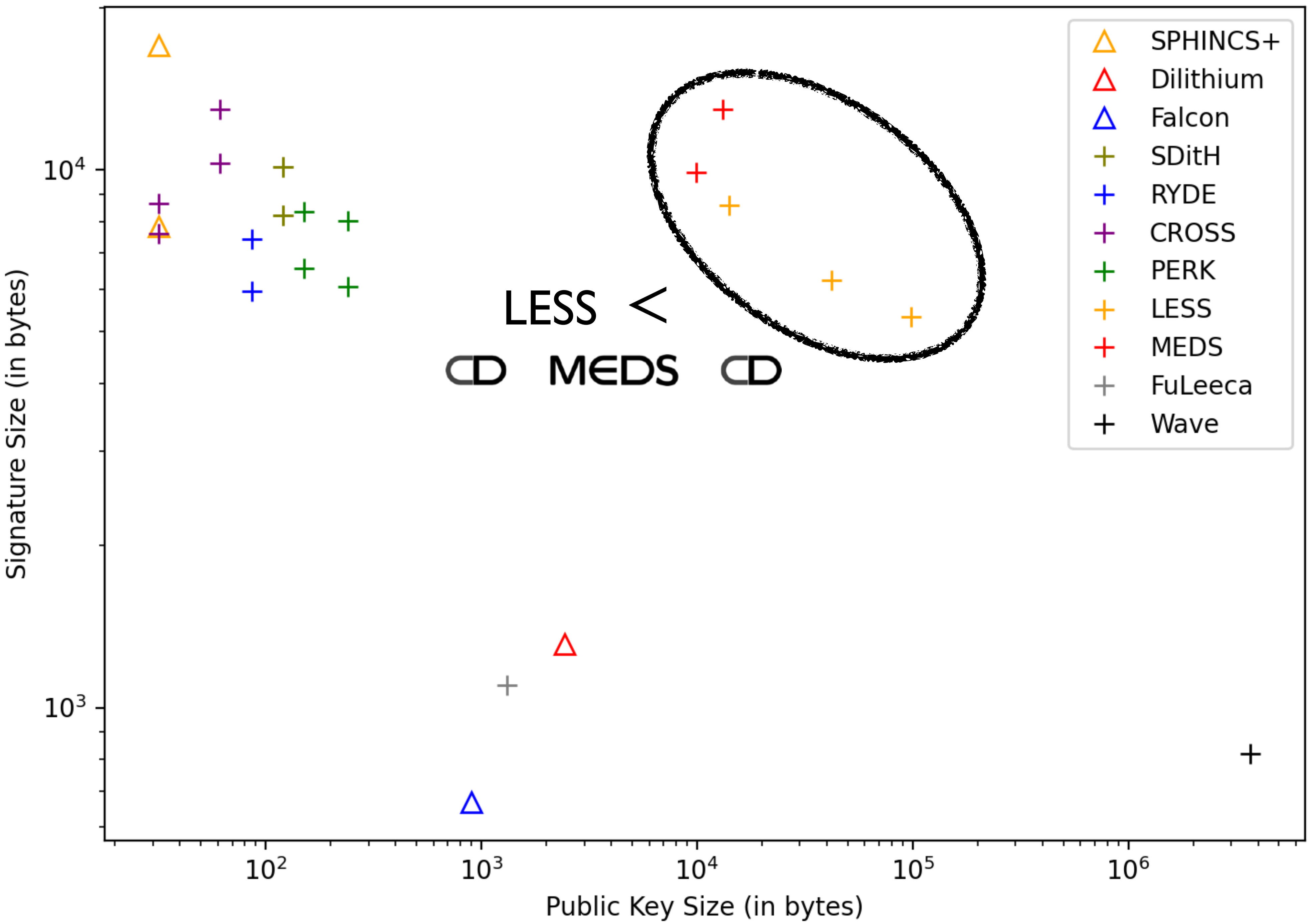
SPHINCS+
Dilithium
Falcon
SDitH
RYDE
CROSS
PERK
LESS
MEDS
FuLeeca
Wave
WAVE
Signature Size (in bytes)
10^4
10^3
10^2
10^3
10^4
10^5
10^6
Public Key Size (in bytes)

Comparison

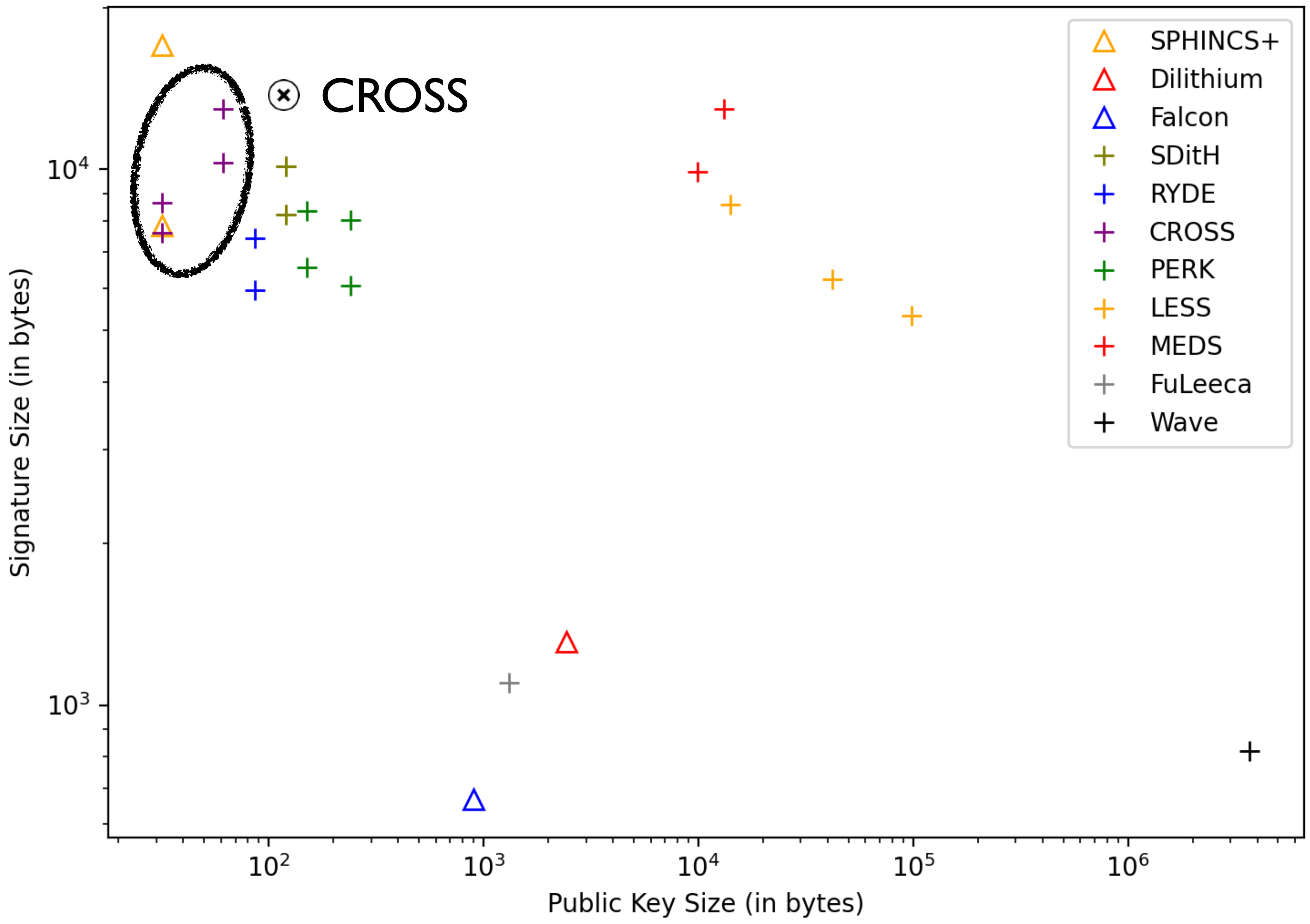


Signature Size (in bytes)
10^4
10^3
Public Key Size (in bytes)
10^2
10^3
10^4
10^5
10^6
SPHINCS+
Dilithium
Falcon
SDitH
RYDE
CROSS
PERK
LESS
MEDS
FuLeeca
Wave
FuLeeca

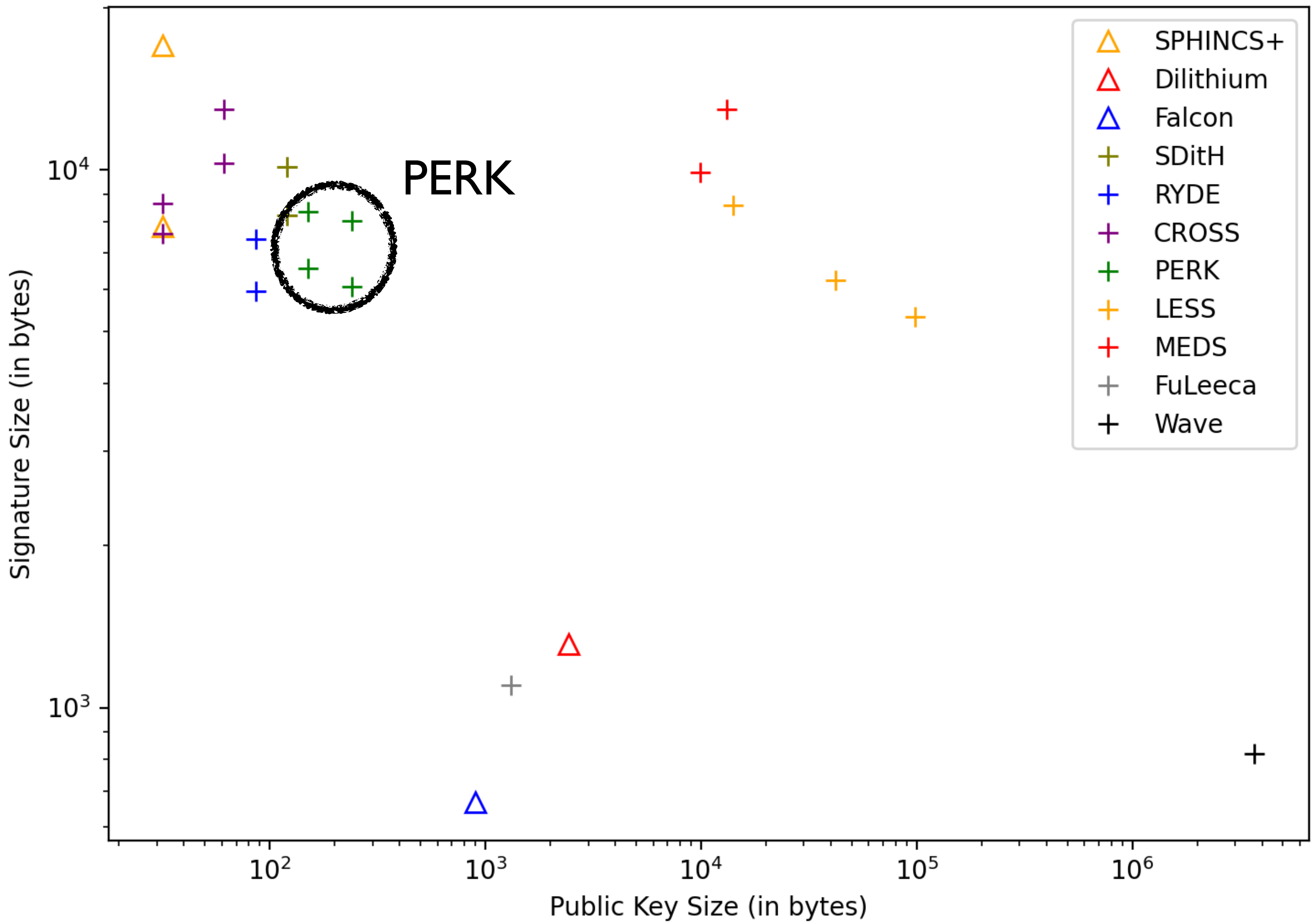
Comparison


SPHINCS+
Dilithium
Falcon
SDitH
RYDE
CROSS
PERK
LESS
MEDS
FuLeeca
Wave
LESS <
MEDS
Signature Size (in bytes)
10^4
10^3
Public Key Size (in bytes)
10^2
10^3
10^4
10^5
10^6

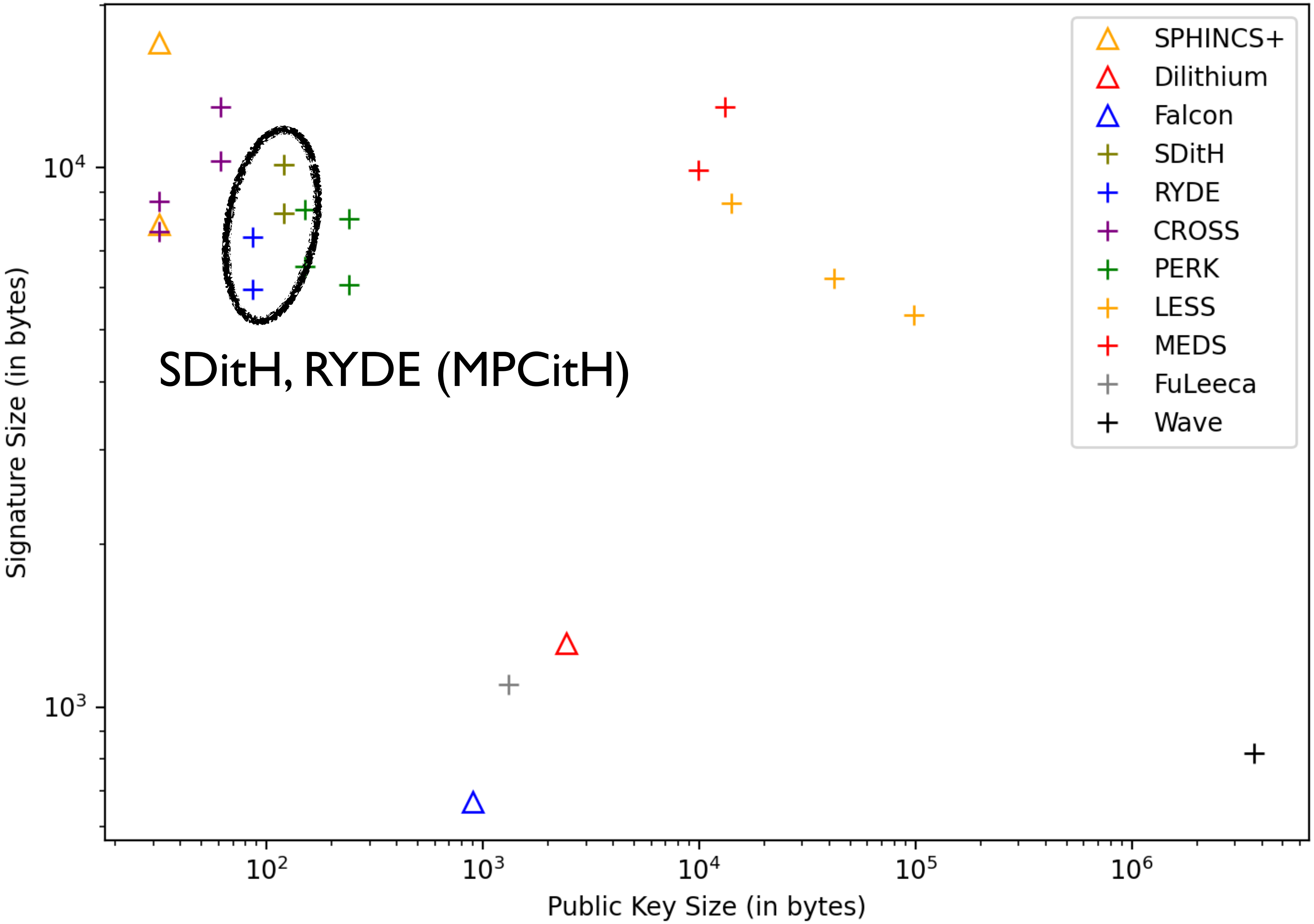
Comparison


CROSS
Signature Size (in bytes)
10^4
10^3
Public Key Size (in bytes)
10^2
10^3
10^4
10^5
10^6
SPHINCS+
Dilithium
Falcon
SDitH
RYDE
CROSS
PERK
LESS
MEDS
FuLeeca
Wave

Comparison

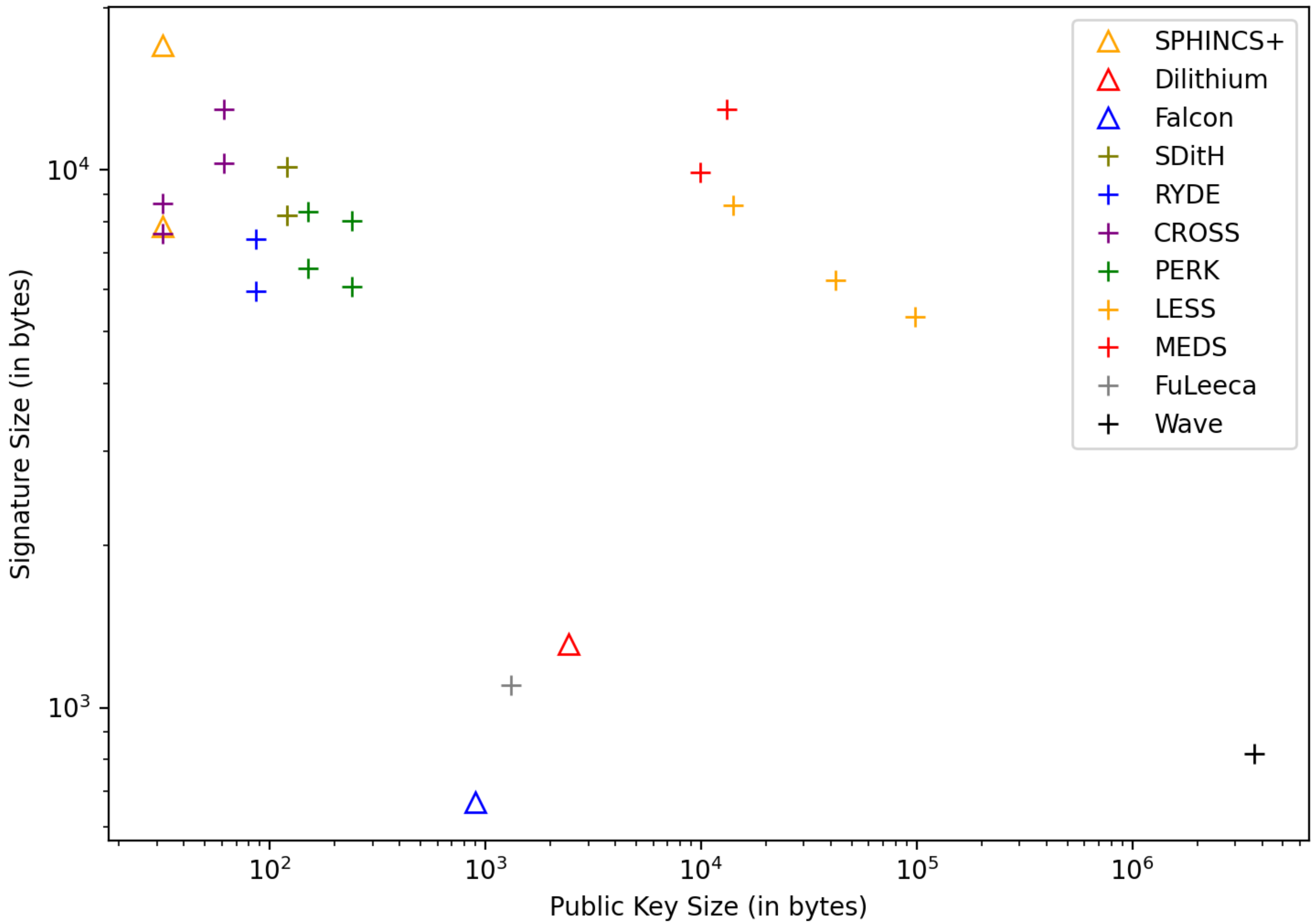

PERK
Signature Size (in bytes)
10^4
10^3
10^2
10^3
10^4
10^5
10^6
Public Key Size (in bytes)
SPHINCS+
Dilithium
Falcon
SDitH
RYDE
CROSS
PERK
LESS
MEDS
FuLeeca
Wave

Comparison



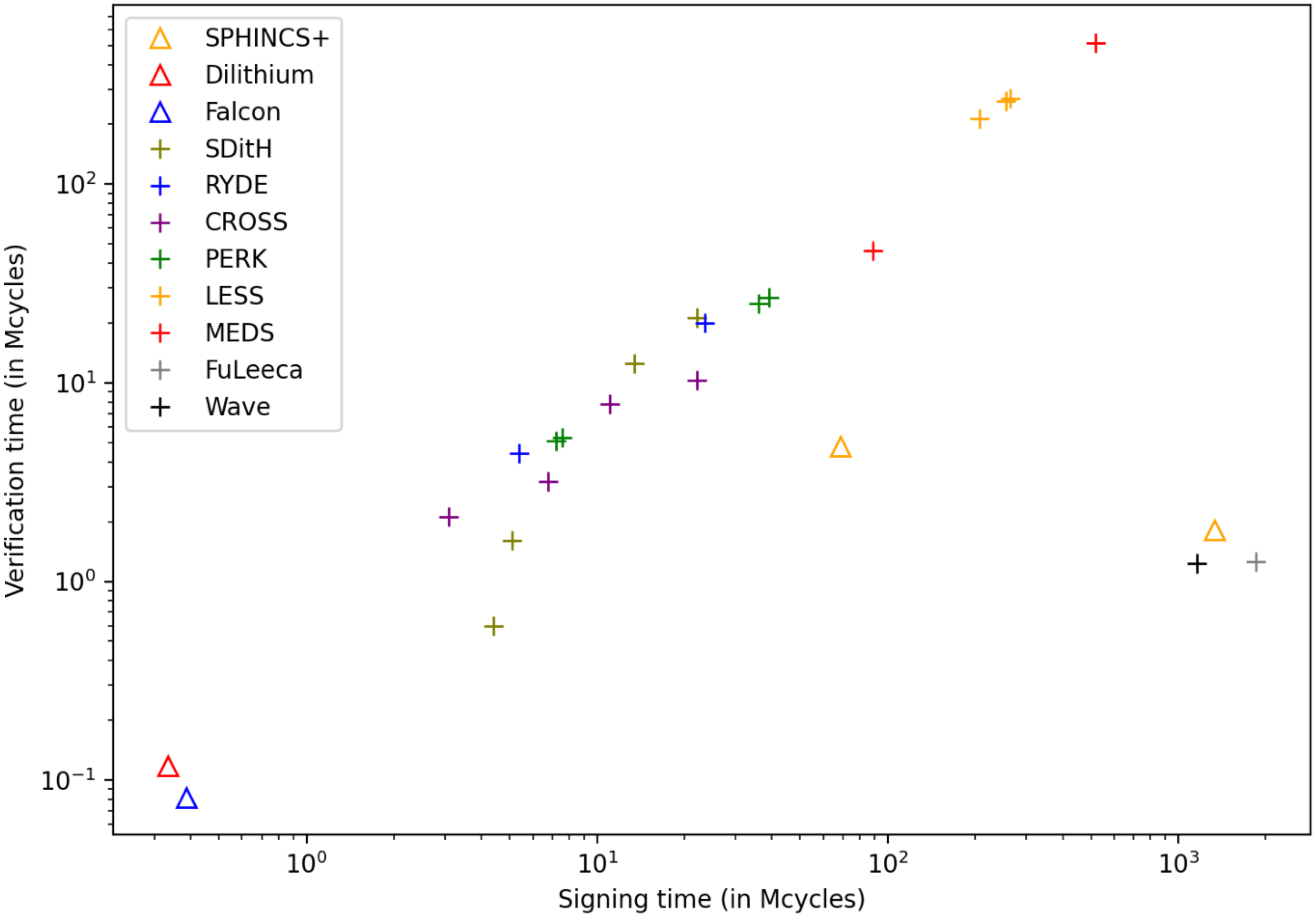
SDitH, RYDE (MPCitH)

Comparison


Signature Size (in bytes)
10^4
10^3
10^2
10^3
10^4
10^5
10^6
Public Key Size (in bytes)
△ SPHINCS+
△ Dilithium
△ Falcon
+ SDitH
+ RYDE
+ CROSS
+ PERK
+ LESS
+ MEDS
+ FuLeeca
+ Wave

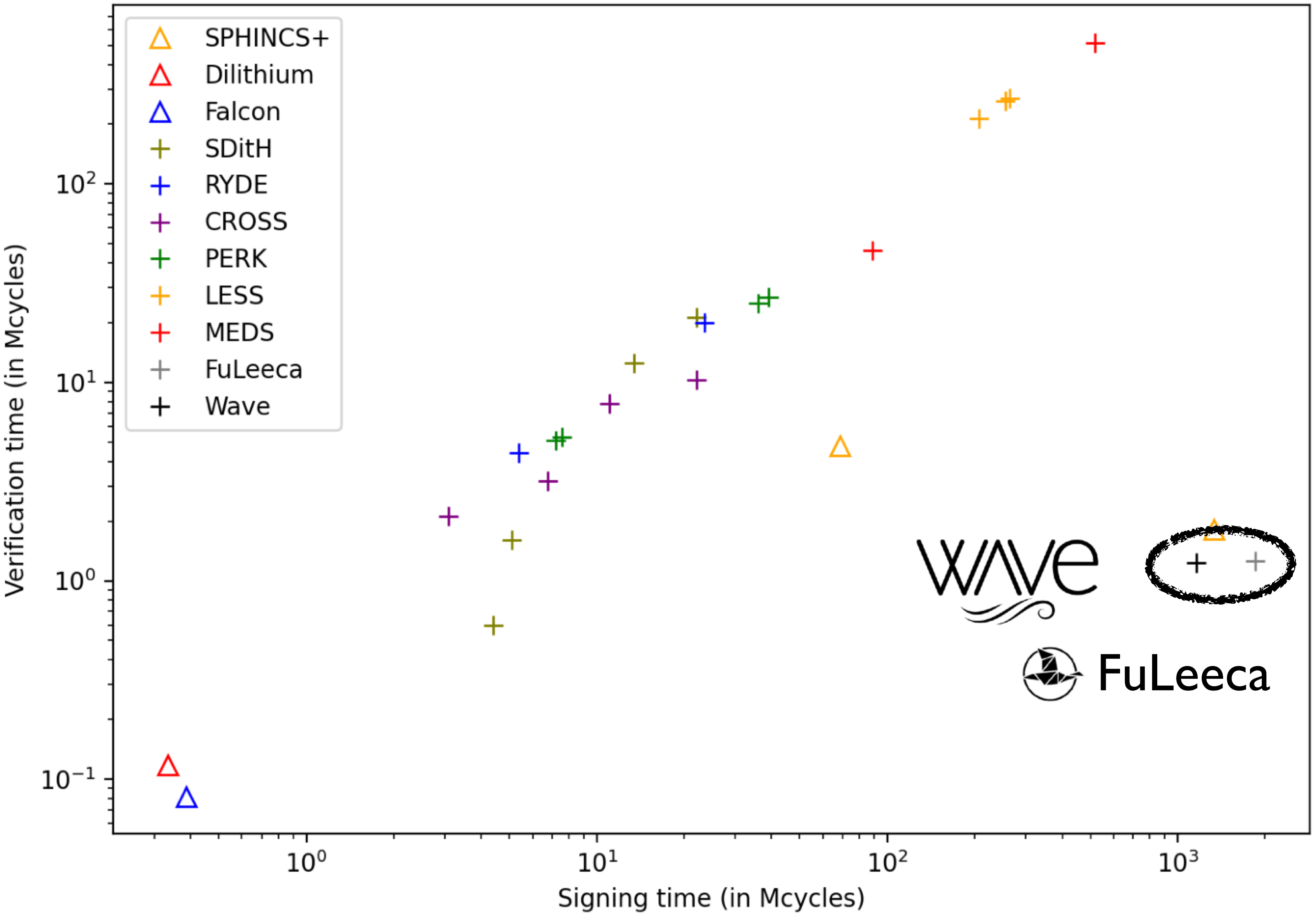
Comparison

Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.



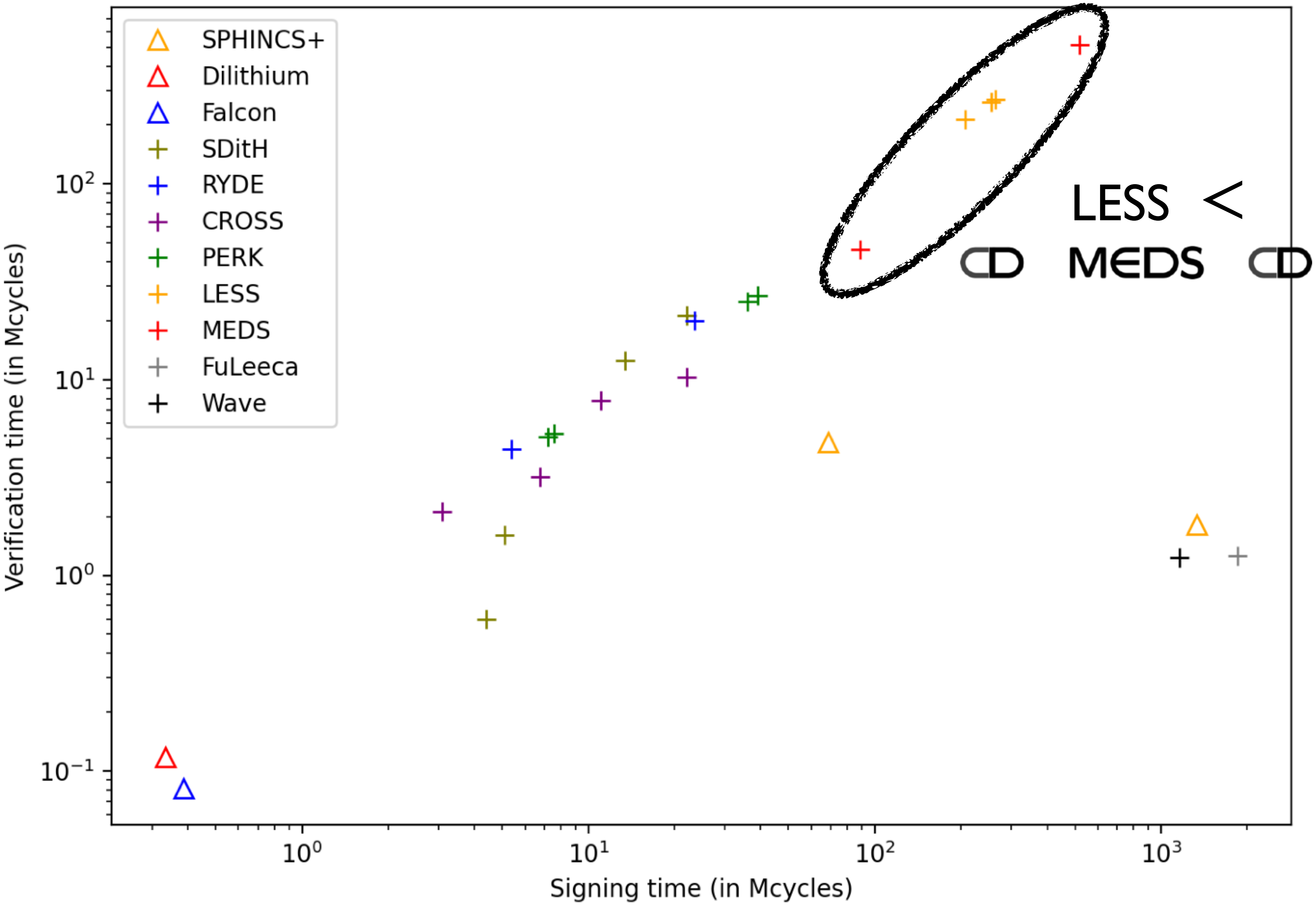
Comparison

⚠ *Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.*



Comparison

Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.

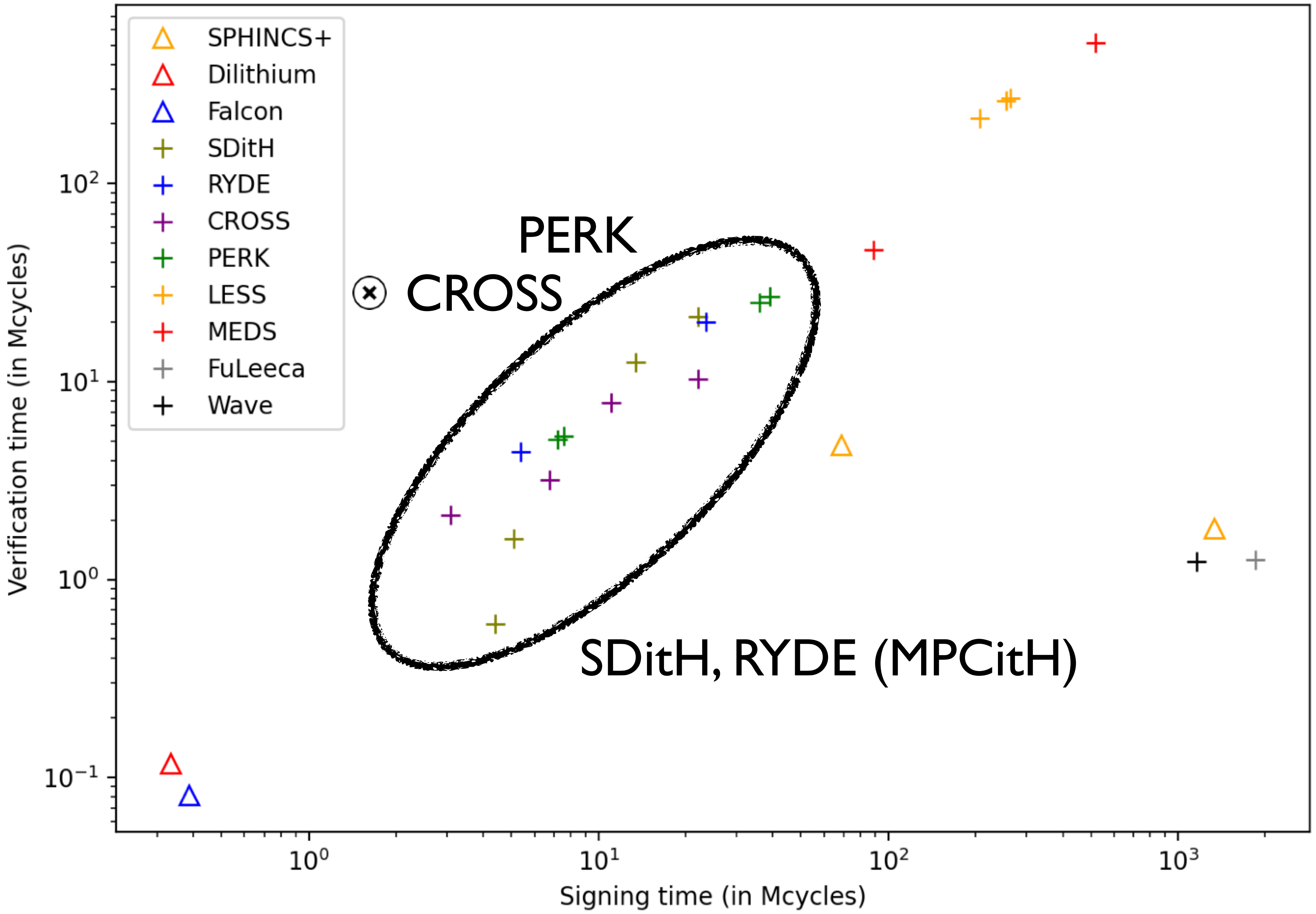


LESS <

MEDS

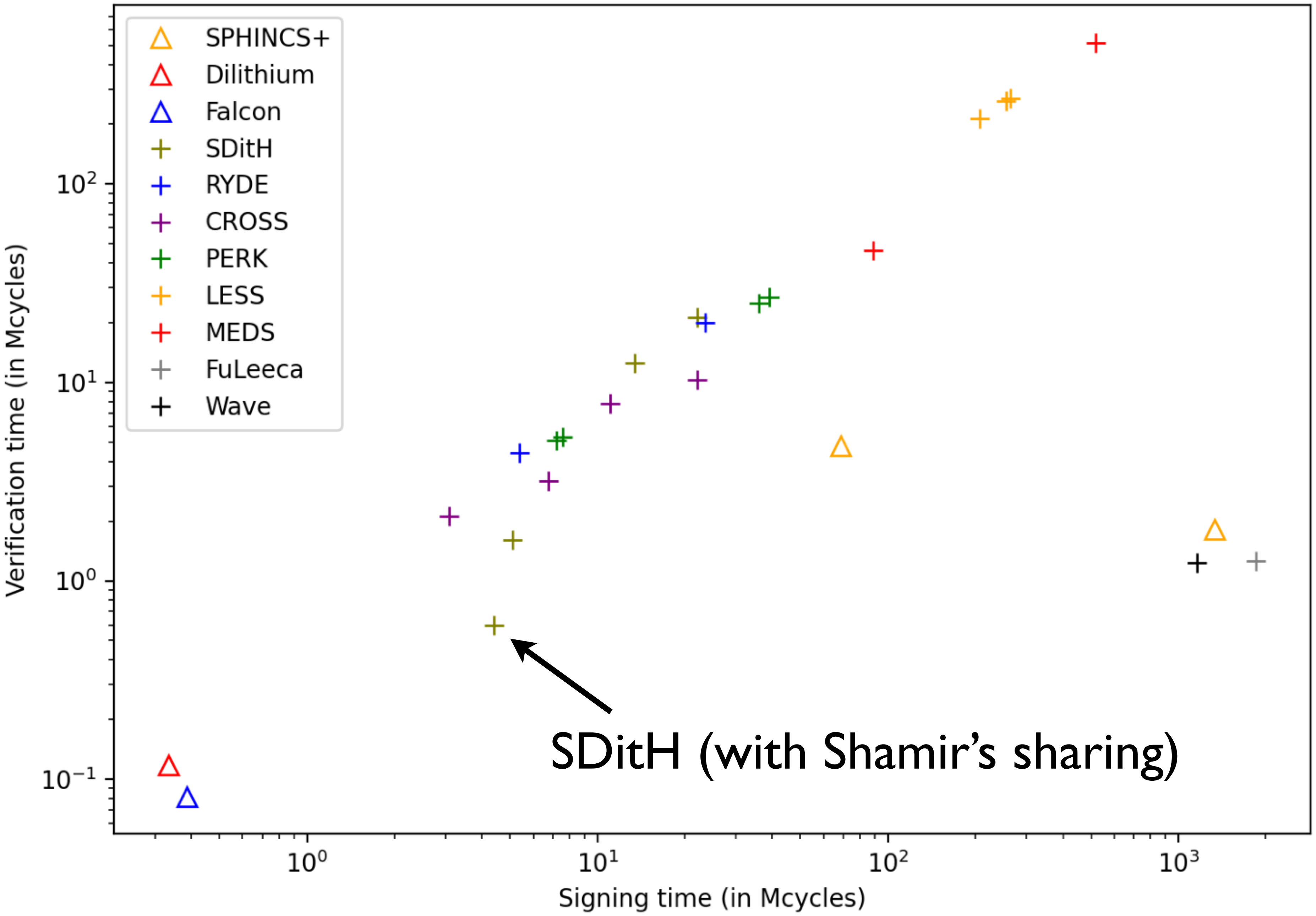
Comparison

⚠ *Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.*



Comparison

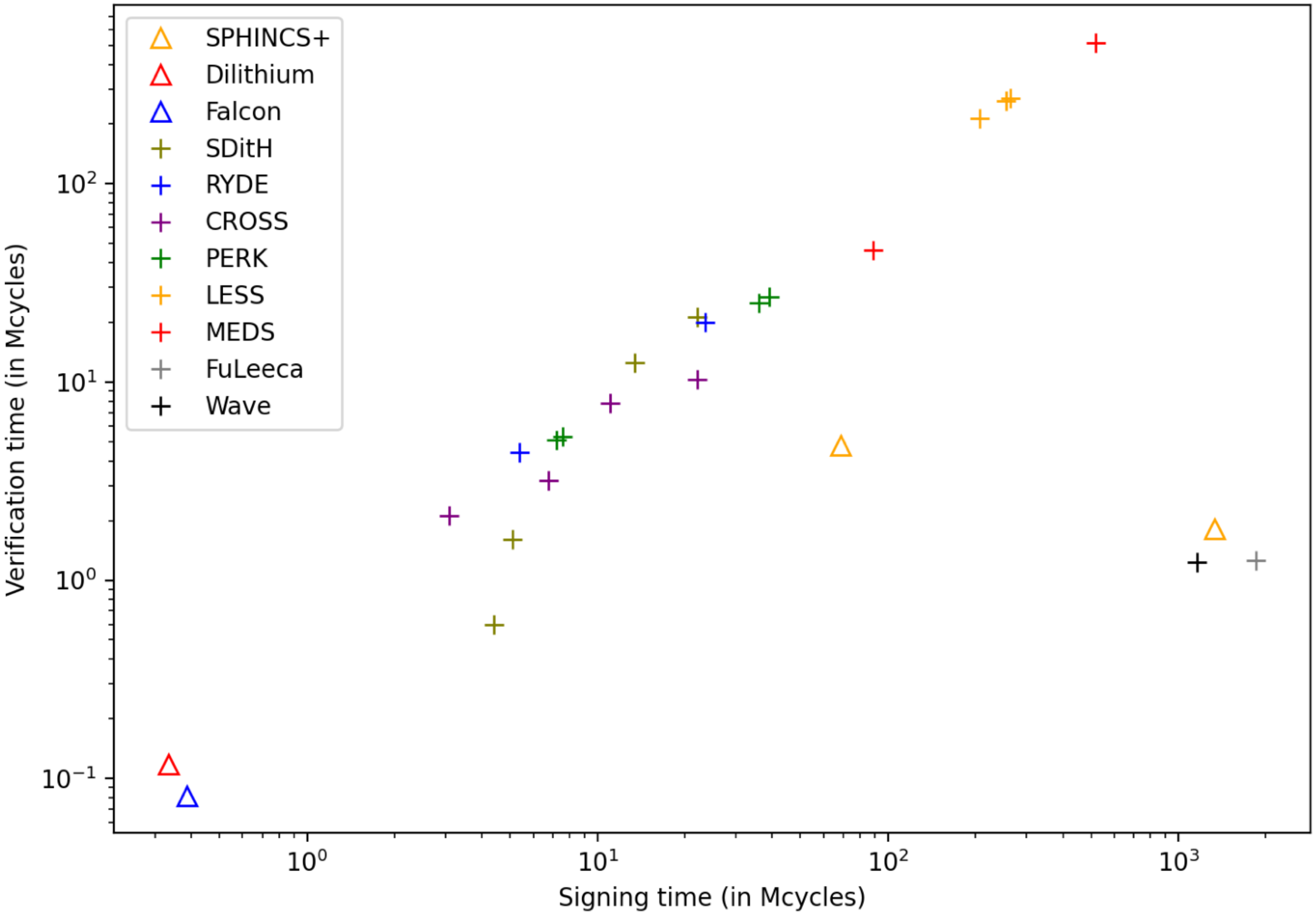
Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.



SDitH (with Shamir’s sharing)

Comparison

⚠ *Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.*



Advantages and limitations

- Limitations
  - Relatively ***slow*** *(few milliseconds)*
    - Greedy use of symmetric cryptography
  - Relatively ***large*** signatures (*5-10 KB for L1*)
  - ***Quadratic*** growth in the security level

- Advantages
  - ***Conservative*** hardness assumption:
    - Old problems, no structure, no trapdoor
  - ***Small*** (public) keys
  - Highly ***parallelizable***
  - ***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
    - *No structure* in the security assumption

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
    - *No structure* in the security assumption

- Perspectives
  - *Additive-based MPCitH:* stable
  - *Low-threshold-based MPCitH*: new approach, could lead to follow-up works

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
    - *No structure* in the security assumption

- Perspectives
  - *Additive-based MPCitH:* stable
  - *Low-threshold-based MPCitH*: new approach, could lead to follow-up works

- Remark:
  - Can be applied to one-way functions from other research fields: multivariate quadratic problem, MinRank problem, …

Conclusion

SD-in-the-Head (SDitH)

C. Aguilar Melchor, T. Feneuil, N. Gama, S. Gueron,
J. Howe, D. Joseph, A. Joux, E. Persichetti,
T. Randrianarisoa, M. Rivain, D. Yue.

Website: `https://sdith.org/`
Email: `consortium@sdith.org`

RYDE

N. Aragon, M. Bardet, L. Bidoux, J.-J. Chi-Domínguez,
V. Dyseryn, T. Feneuil, P. Gaborit, A. Joux,
M. Rivain, J.-P. Tillich, A. Vinçotte.

Website: `https://pqc-ryde.org/`
Email: `team@pqc-ryde.org`