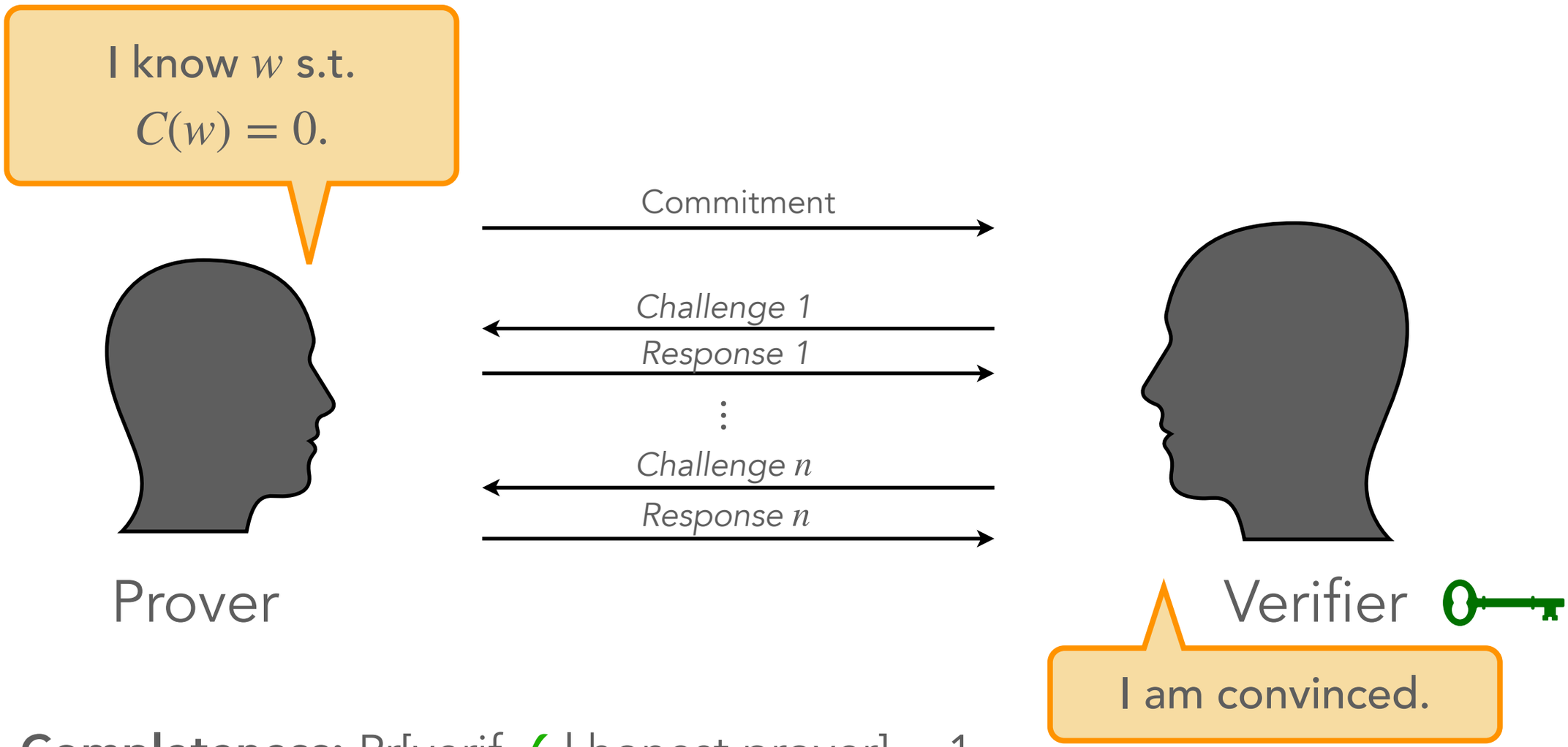
Beyond SNARKs: Hash-Based Zero-Knowledge for Post-Quantum Signatures

Thibauld Feneuil

P³S³ Summer School

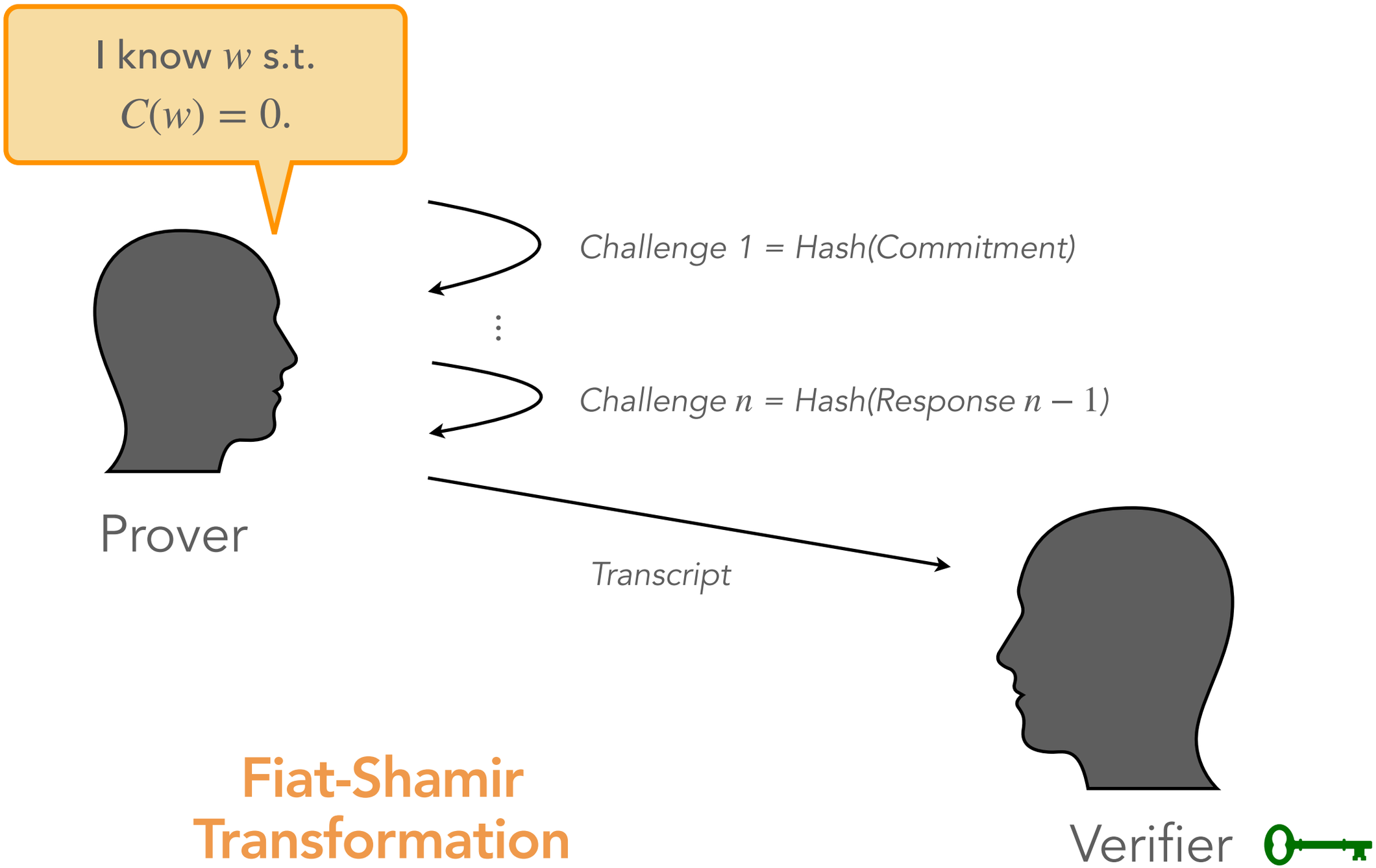
June 2, 2026 — Orsay (France)

Zero-Knowledge Proof



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

Zero-Knowledge Proof

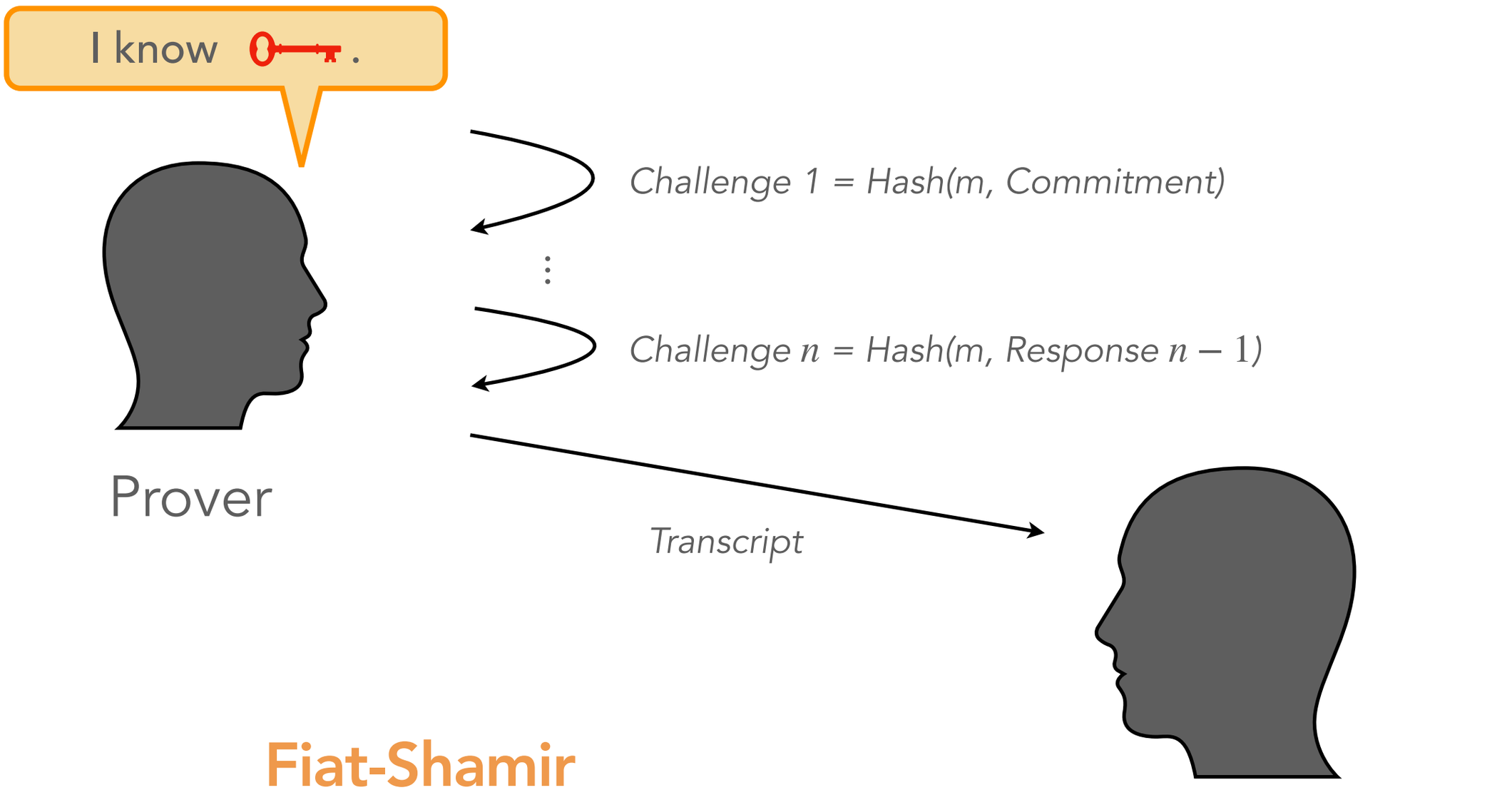


Fiat-Shamir Transformation

Identification Scheme

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

Identification Scheme



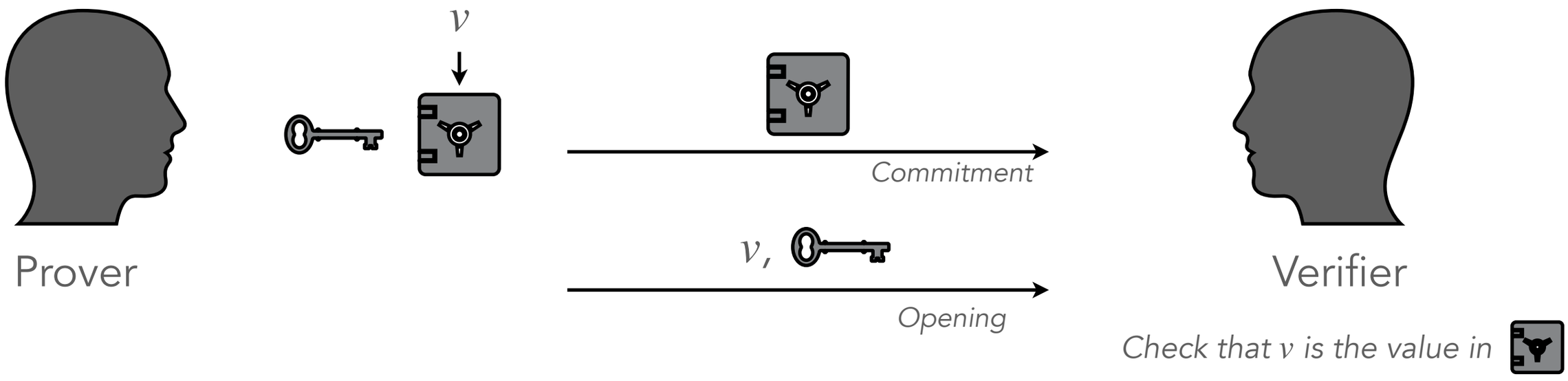
Fiat-Shamir Transformation

m: message to sign

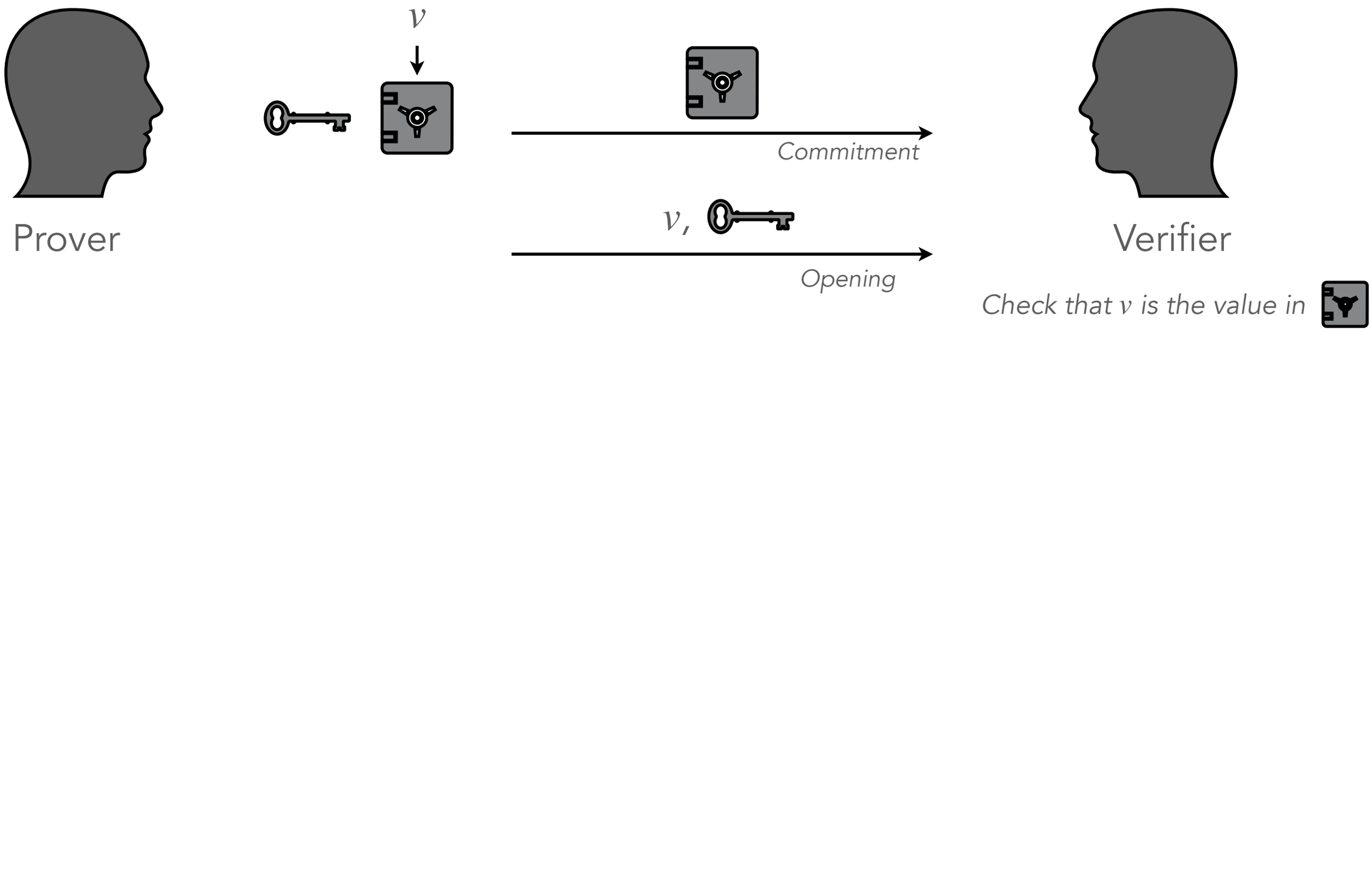
Commitment Scheme

- **Commitment** algorithm/procedure. A prover can commit to a chosen value v while keeping it *hidden* to other people (*hiding property*).

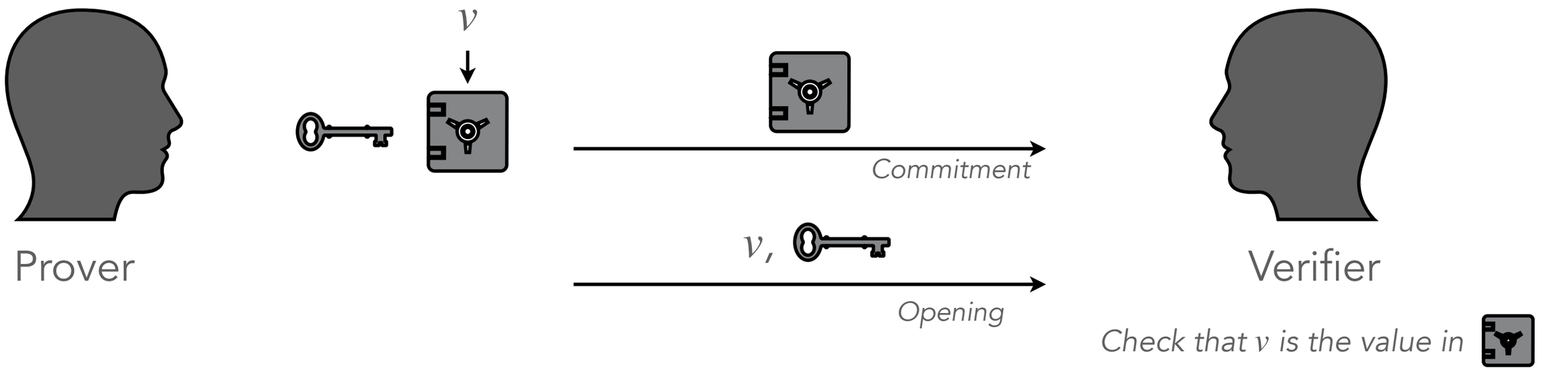
- **Opening** algorithm/procedure. The prover can reveal the value v and prove that the revealed value is the one which has been committed through the commitment procedure. It should be impossible for the prover to reveal a value $v' \neq v$ while convincing the verifier that v' is the committed value (binding property).



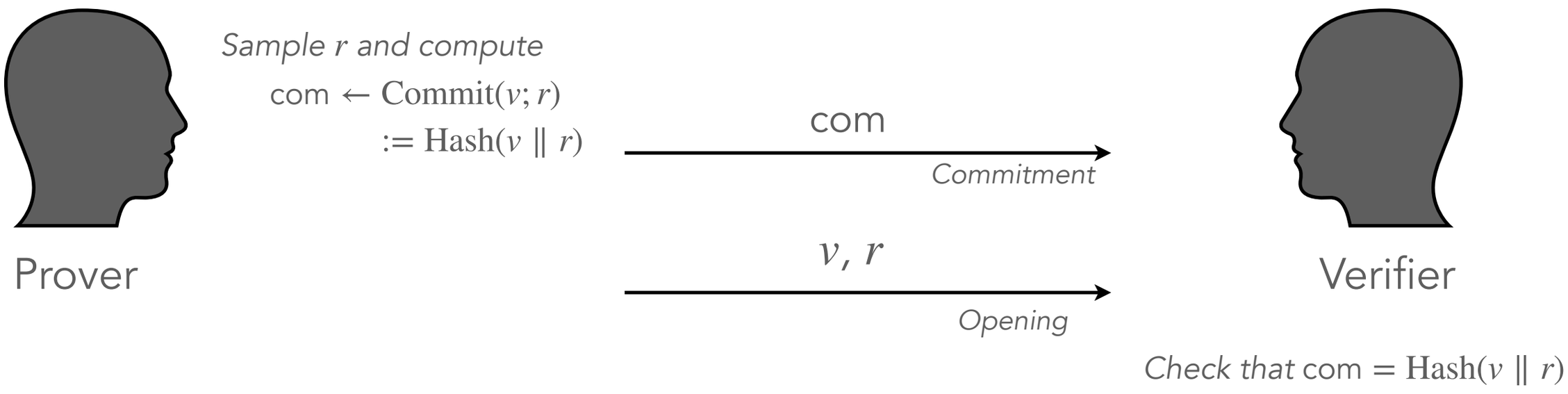
Commitment Scheme



Commitment Scheme



***Hash-based Commitment Scheme*:**



Polynomial Commitment Scheme

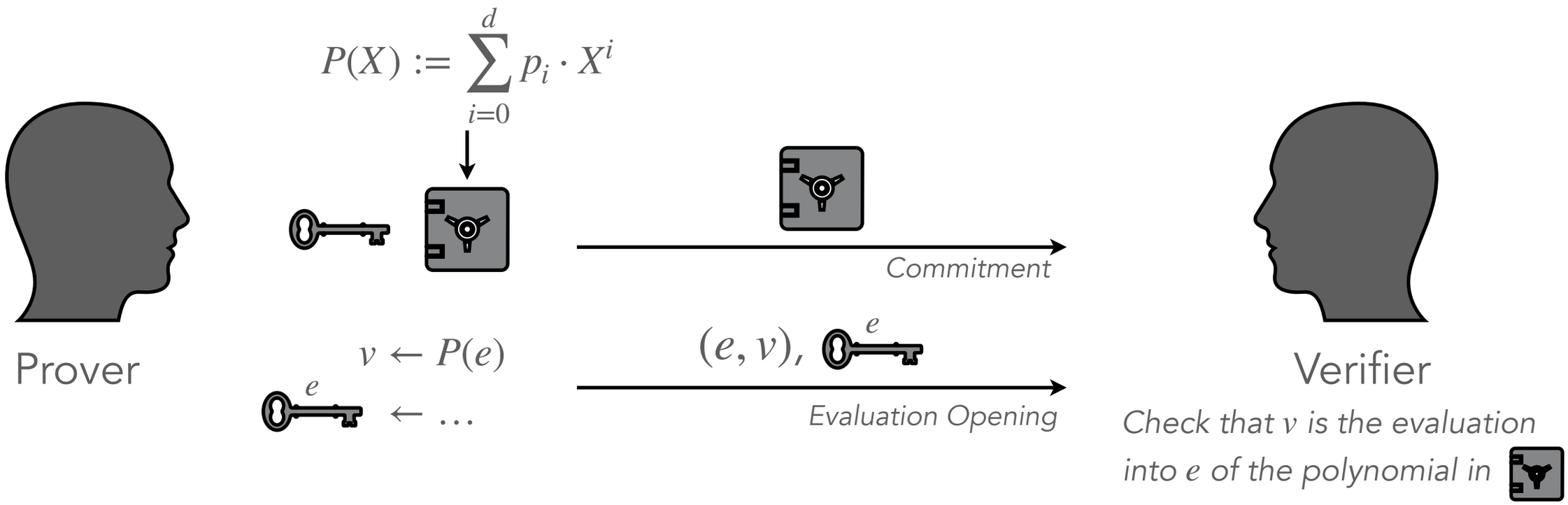
- We want to commit a **polynomial** $P(X) := p_0 + p_1 \cdot X + \ldots + p_d \cdot X^d$:
    - Using a standard commitment scheme, the opening procedure would consist of revealing the *entire polynomial* in a verifiable way.

Polynomial Commitment Scheme

- We want to commit a **polynomial** $P(X) := p_0 + p_1 \cdot X + \ldots + p_d \cdot X^d$:
  - Using a standard commitment scheme, the opening procedure would consist of revealing the *entire polynomial* in a verifiable way.
  - Using a **polynomial commitment scheme**, the opening procedure would consist of ***some evaluations of the committed polynomials*** in a verifiable way, while keeping the other evaluations **hidden** to the verifier.

Polynomial Commitment Scheme

- We want to commit a **polynomial** $P(X) := p_0 + p_1 \cdot X + \ldots + p_d \cdot X^d$:
  - Using a standard commitment scheme, the opening procedure would consist of revealing the *entire polynomial* in a verifiable way.
  - Using a **polynomial commitment scheme**, the opening procedure would consist of ***some evaluations of the committed polynomials*** in a verifiable way, while keeping the other evaluations **hidden** to the verifier.



Goals

Provide an overview of the techniques used in hash-based zero-knowledge proof systems, especially when proving small statements.

Provide an overview of the rationale design of the MPCitH signatures (candidates to standardization).

Table of Contents

Context / Motivation

Blueprint of Hash-based Proof Systems

I know $s_1, \ldots, s_{|s|}$ such that

$$C(s_1, \ldots, s_{|s|}) = 0$$

where C represents any computation.

Prover

Prove it!

Verifier

Blueprint of Hash-based Proof Systems

I know $s_1, \ldots, s_{|s|}$ such that

$$C(s_1, \ldots, s_{|s|}) = 0$$

where C represents any computation.

Prover

Strictly equivalent to prove

Verifier

I know some polynomials $P_1(X), \ldots, P_n(X)$ such that

$$\begin{cases} F_1\left(P_1(X), \ldots, P_n(X)\right) & = 0 \\ & \vdots \\ F_m\left(P_1(X), \ldots, P_n(X)\right) & = 0, \end{cases}$$

where $F_1, \ldots, F_m$ are polynomial relations.

Prove it!

Prove it!

PIOP-friendly Statement

Blueprint of Hash-based Proof Systems

Prover

Build some polynomials

$$P_1(X), \ldots, P_n(X)$$

satisfying some polynomial relations

$$\begin{cases} F_1\left(P_1(X), \ldots, P_n(X)\right) & = 0 \\ & \vdots \\ F_m\left(P_1(X), \ldots, P_n(X)\right) & = 0, \end{cases}$$

Verifier

$\xrightarrow{\mathrm{PCom}(P_1, \ldots, P_n)}$

Sample a point e from $\mathscr{C}$.

$\xleftarrow{e}$

Build an opening proof π for

$$v_1 := P_1(e), \ldots, v_n := P_n(e).$$

$\xrightarrow{(v_1, \ldots, v_n), \pi}$

Check that π is a valid opening proof for e.

Check that

$$\begin{cases} F_1(v_1, \ldots, v_n) & = 0 \\ & \vdots \\ F_m(v_1, \ldots, v_n) & = 0, \end{cases}$$

Blueprint of Hash-based Proof Systems

Prover

Build some polynomials

$$P_1(X), \ldots, P_n(X)$$

satisfying some polynomial relations

$$\begin{cases} F_1\left(P_1(X), \ldots, P_n(X)\right) & = 0 \\ & \vdots \\ F_m\left(P_1(X), \ldots, P_n(X)\right) & = 0, \end{cases}$$

Verifier

$\xrightarrow{\mathrm{PCom}(P_1, \ldots, P_n)}$

Sample a point e from $\mathscr{C}$.

$\xleftarrow{e}$

Build an opening proof π for

$$v_1 := P_1(e), \ldots, v_n := P_n(e).$$

$\xrightarrow{(v_1, \ldots, v_n), \pi}$

Check that π is a valid opening proof for e.

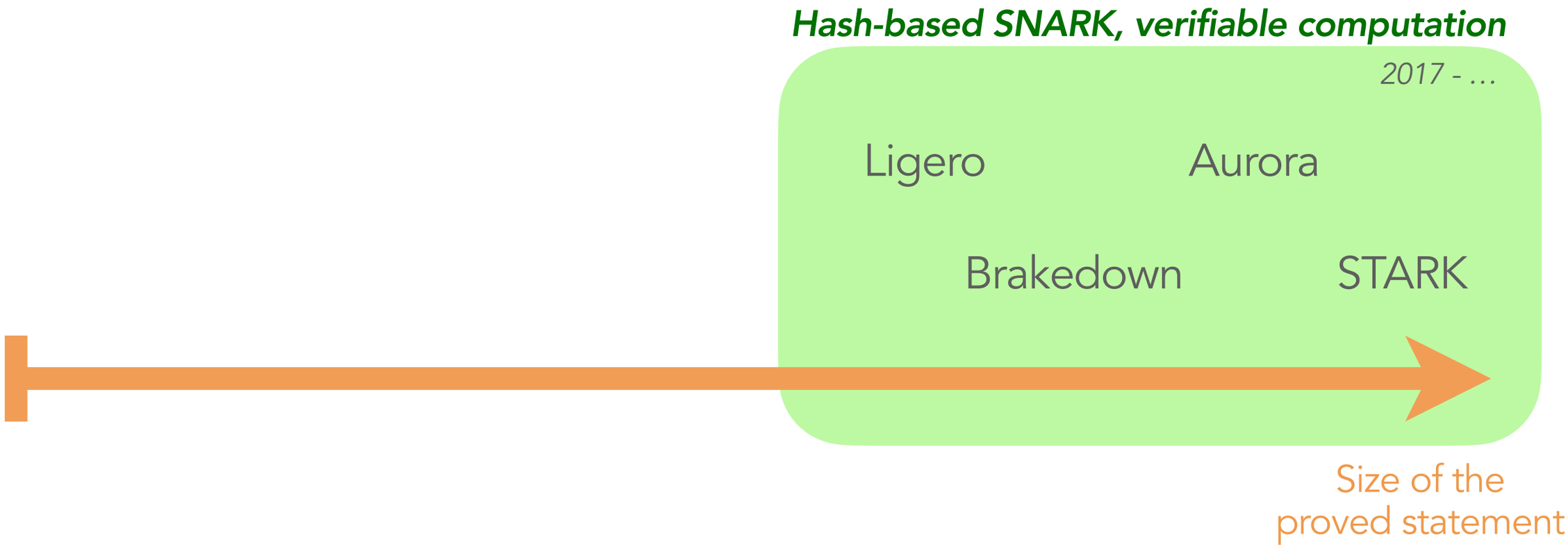
Check that

$$\begin{cases} F_1(v_1, \ldots, v_n) & = 0 \\ & \vdots \\ F_m(v_1, \ldots, v_n) & = 0, \end{cases}$$

Hash-based SNARK: Ligero, Aurora, STARK, Brakedown, …

Post-quantum signatures: FAEST, MQOM v2, SDitH v2, …

Blueprint of Hash-based Proof Systems



Blueprint of Hash-based Proof Systems

Hash-based SNARK, verifiable computation

2017 - …

Ligero

Aurora

Brakedown

STARK

Size of the
proved statement

Example:

"y is obtained by inferring
from data x using AI model C."

Blueprint of Hash-based Proof Systems

Hash-based SNARK, verifiable computation

2017 - …

Ligero

Aurora

Brakedown

STARK

Size of the
proved statement

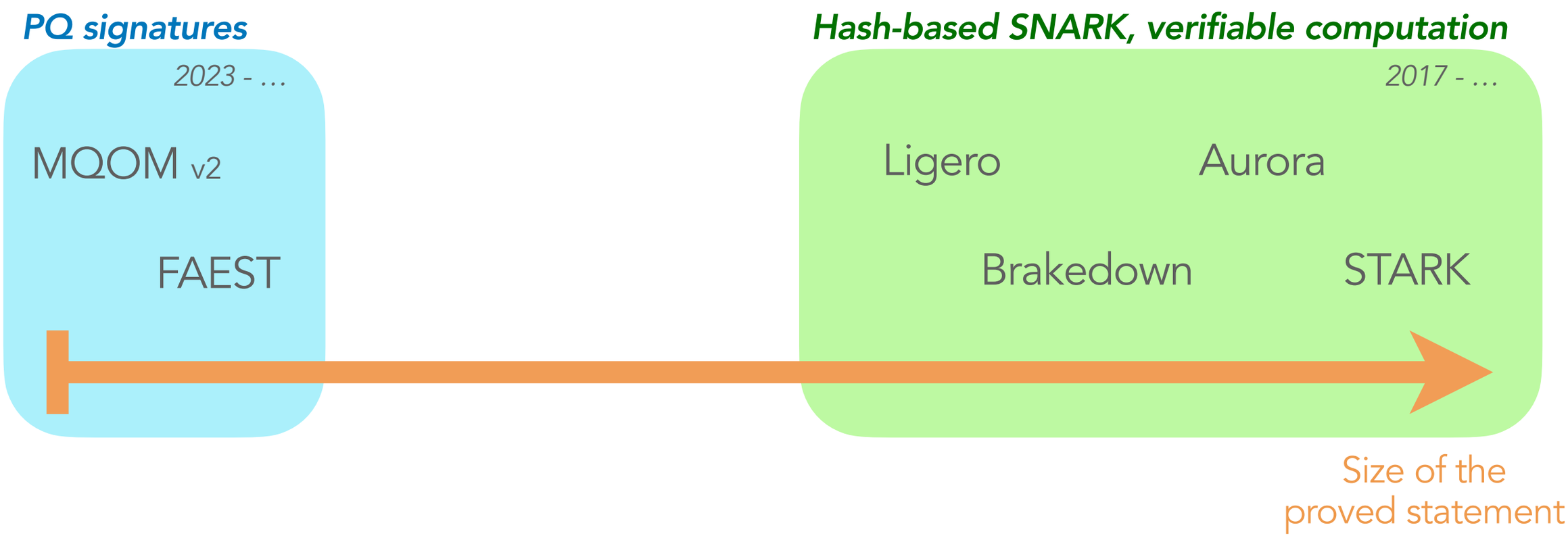
Example:

"y is obtained by inferring
from data x using AI model C."

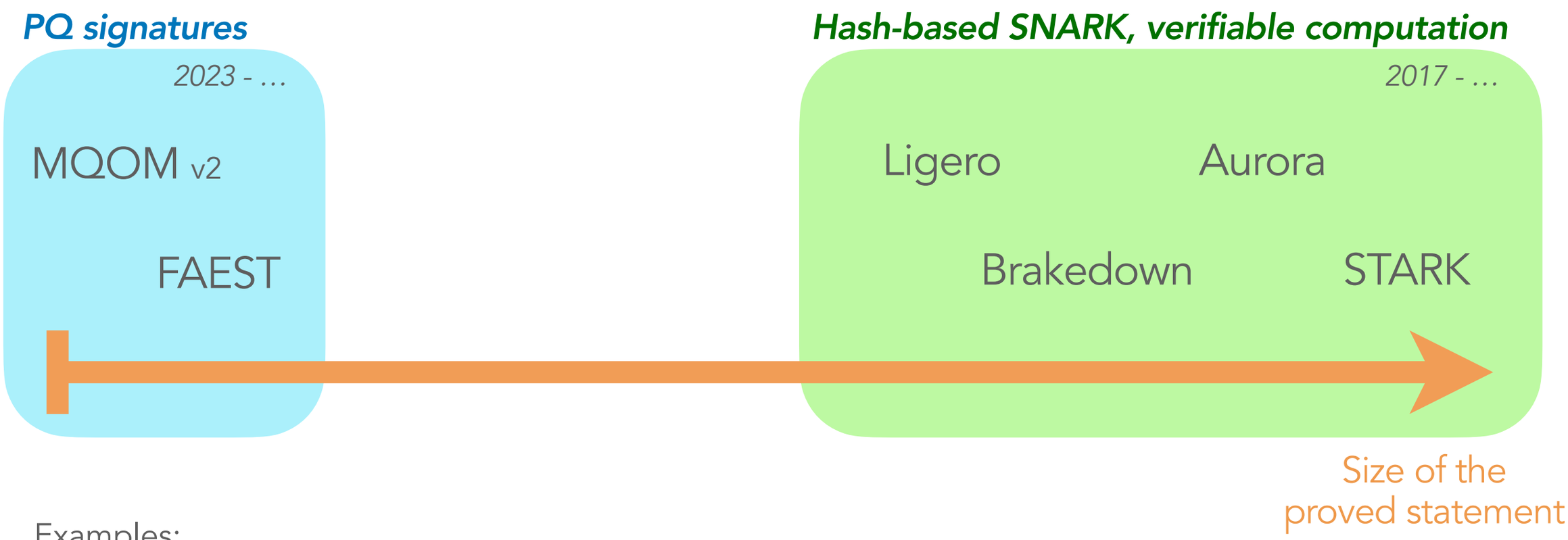
Properties:

- Succinctness: ***required***
- Zero-Knowledge: ***optional***

Blueprint of Hash-based Proof Systems



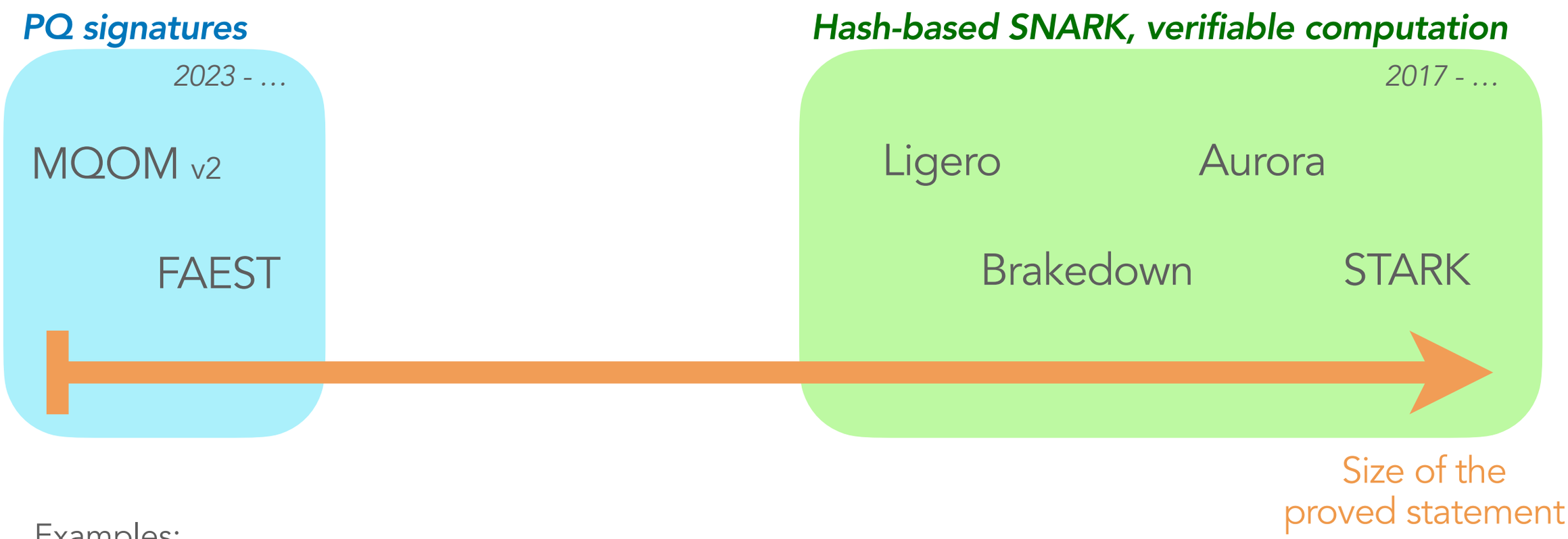
Blueprint of Hash-based Proof Systems



Examples:

- **FAEST**: Given (x, y), I know k such that $y = \mathrm{AES}_k(x)$
- **MQOM**: Given a multivariate system $\mathscr{F}$, I know x such that $\mathscr{F}(x) = 0$

Blueprint of Hash-based Proof Systems



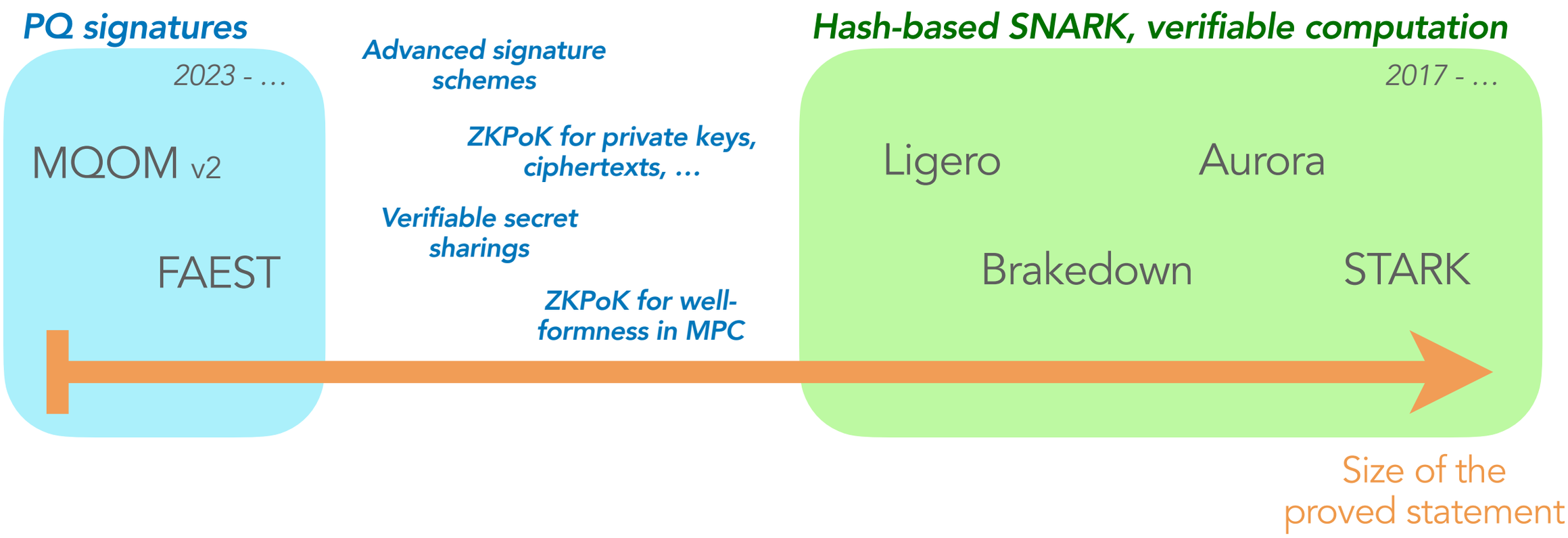
Examples:

- **FAEST**: Given (x, y), I know k such that $y = \mathrm{AES}_k(x)$
- **MQOM**: Given a multivariate system $\mathscr{F}$, I know x such that $\mathscr{F}(x) = 0$

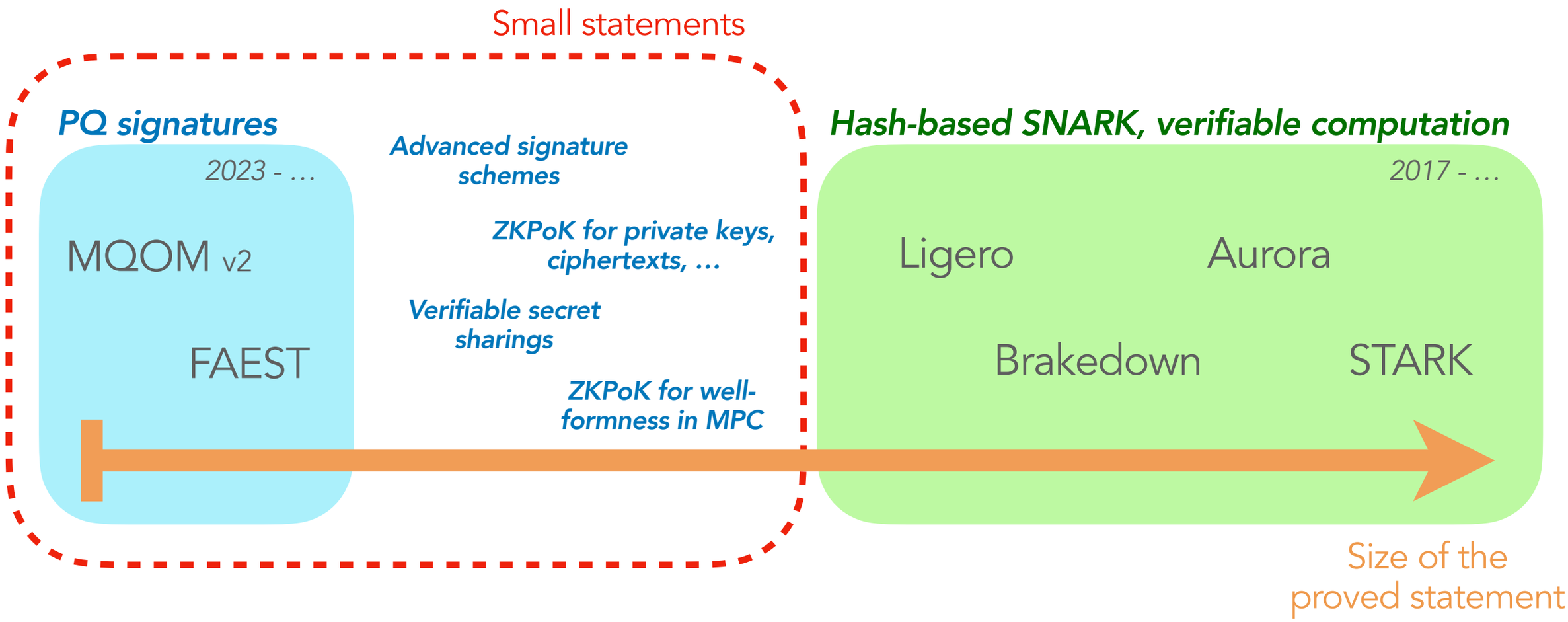
Properties:

- Zero-Knowledge: ***required***
- Succinctness: ***optional***

Blueprint of Hash-based Proof Systems



Blueprint of Hash-based Proof Systems



Zero-Knowledge is ***required***
Succinctness is ***optional***

Succinctness is ***required***
Zero-Knowledge is ***optional***

Basic Proof System for Polynomial Constraints

I know $w_1, \ldots, w_n$ such that

$$\begin{cases} f_1(w_1, \ldots, w_n) = 0 \\ \vdots \\ f_m(w_1, \ldots, w_n) = 0, \end{cases}$$

where $f_1, \ldots, f_m$ are public **degree-d polynomials**.

Prover

Prove it!

Verifier

Basic Proof System for Polynomial Constraints

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

Verifier

Prover

Basic Proof System for Polynomial Constraints

Verifier

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$
$$\vdots$$
$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

Prover

Basic Proof System for Polynomial Constraints

Verifier

Prover

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$
$$\vdots$$
$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

Well-defined!

$$\forall j,\ \ f_j(P_1(0), \ldots, P_n(0)) = f_j(w_1, \ldots, w_n) = 0$$

Basic Proof System for Polynomial Constraints

Verifier

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$
$$\vdots$$
$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

$\text{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

Prover

Basic Proof System for Polynomial Constraints

Prover

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$
$$\vdots$$
$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

⑤ Reveal the evaluation

- for all (i, k), $v^P_{k,i} := P_i(r_k)$
- for all (j, k), $v^Q_{k,j} := Q_j(r_k)$

Verifier

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

$\mathrm{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

← $r_1, \ldots, r_\ell$

$\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k, \{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ →

Basic Proof System for Polynomial Constraints

Prover

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$
$$\vdots$$
$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

⑤ Reveal the evaluation

- for all (i, k), $v^P_{k,i} := P_i(r_k)$
- for all (j, k), $v^Q_{k,j} := Q_j(r_k)$

$\mathrm{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

← $r_1, \ldots, r_\ell$

$\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k, \{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ →

Verifier

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

① For all i, choose a degree-ℓ polynomial $P_i(X)$. There exist j^* such that

$$f_{j^*}(P_1(0), \ldots, P_n(0)) \neq 0.$$

② Choose some polynomials $Q_1, \ldots, Q_m$. We know that

$$X \cdot Q_{j^*}(X) \neq f_{j^*}(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

⑤ Reveal the evaluation

- for all (i, k), $v^P_{k,i} := P_i(r_k)$
- for all (j, k), $v^Q_{k,j} := Q_j(r_k)$

Malicious Prover 😈

Verifier

Soundness Analysis

$\text{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

← $r_1, \ldots, r_\ell$

$\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k, \{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ →

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

① For all i, choose a degree-ℓ polynomial $P_i(X)$. There exist j^* such that

$$f_{j^*}(P_1(0), \ldots, P_n(0)) \neq 0.$$

② Choose some polynomials $Q_1, \ldots, Q_m$. We know that

$$X \cdot Q_{j^*}(X) \neq f_{j^*}(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

⑤ Reveal the evaluation

- for all (i,k), $v^P_{k,i} := P_i(r_k)$
- for all (j,k), $v^Q_{k,j} := Q_j(r_k)$

Evaluation into 0

$= 0$ $\neq 0$

Malicious Prover 😈

Verifier

Soundness Analysis

$\mathrm{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

← $r_1, \ldots, r_\ell$

$\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k, \{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ →

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

Malicious Prover 😈

① For all i, choose a degree-ℓ polynomial $P_i(X)$. There exist j^* such that

$$f_{j^*}(P_1(0), \ldots, P_n(0)) \neq 0.$$

② Choose some polynomials $Q_1, \ldots, Q_m$. We know that

$$X \cdot Q_{j^*}(X) \neq f_{j^*}(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

$\mathrm{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

⑤ Reveal the evaluation

- for all (i,k), $v^P_{k,i} := P_i(r_k)$
- for all (j,k), $v^Q_{k,j} := Q_j(r_k)$

← $r_1, \ldots, r_\ell$

→ $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k, \{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$

Verifier

Soundness Analysis

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

$$r_k \cdot v^Q_{k,j^*} = f_{j^*}(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

Verifier

① For all i, choose a degree-ℓ polynomial $P_i(X)$. There exist j^* such that

$$f_{j^*}(P_1(0), \ldots, P_n(0)) \neq 0.$$

② Choose some polynomials $Q_1, \ldots, Q_m$. We know that

$$X \cdot Q_{j^*}(X) \neq f_{j^*}(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

$\text{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

$r_1, \ldots, r_\ell$

$\}_k$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Soundness Analysis

Schwartz-Zippel Lemma: Let D be the **non-zero** degree-$(d \cdot \ell)$ polynomial defined as

$$D := X \cdot Q_{j^*}(X) - f_{j^*}(P_1(X), \ldots, P_n(X))$$

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

$$r_k \cdot v^Q_{k,j^*} = f_{j^*}(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

Verifier

① For all i, choose a degree-ℓ polynomial $P_i(X)$. There exist j^* such that

$$f_{j^*}(P_1(0), \ldots, P_n(0)) \neq 0.$$

② Choose some polynomials $Q_1, \ldots, Q_m$. We know that

$$X \cdot Q_{j^*}(X) \neq f_{j^*}(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

$\text{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

$r_1, \ldots, r_\ell$

$\}_k$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Soundness Analysis

Schwartz-Zippel Lemma: Let D be the **non-zero** degree-$(d \cdot \ell)$ polynomial defined as

$$D := X \cdot Q_{j^*}(X) - f_{j^*}(P_1(X), \ldots, P_n(X))$$

We have

$$\Pr[\text{verification passes}] = \Pr\left[\forall k, D(r_k) = 0 \mid \{r_k\}_k \subset_\$ \mathscr{C}\right] \leq$$

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

$$r_k \cdot v^Q_{k,j^*} = f_{j^*}(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

Verifier

① For all i, choose a degree-ℓ polynomial $P_i(X)$. There exist j^* such that

$$f_{j^*}(P_1(0), \ldots, P_n(0)) \neq 0.$$

② Choose some polynomials $Q_1, \ldots, Q_m$. We know that

$$X \cdot Q_{j^*}(X) \neq f_{j^*}(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

$\text{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

$r_1, \ldots, r_\ell$

$\{\}_k$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Soundness Analysis

Schwartz-Zippel Lemma: Let D be the **non-zero** degree-$(d \cdot \ell)$ polynomial defined as

$$D := X \cdot Q_{j^*}(X) - f_{j^*}(P_1(X), \ldots, P_n(X))$$

We have

$$\Pr[\text{verification passes}] = \Pr\left[\forall k, D(r_k) = 0 \mid \{r_k\}_k \subset_\$ \mathscr{C}\right] \leq \frac{\binom{d \cdot \ell}{\ell}}{\binom{|\mathscr{C}|}{\ell}}.$$

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

$$r_k \cdot v^Q_{k,j^*} = f_{j^*}(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

Verifier

Zero-Knowledge Analysis

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$
$$\vdots$$
$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

$\text{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

← $r_1, \ldots, r_\ell$

⑤ Reveal the evaluation

- for all (i, k), $v^P_{k,i} := P_i(r_k)$
- for all (j, k), $v^Q_{k,j} := Q_j(r_k)$

→ $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k, \{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

Prover

Basic Proof System for Polynomial Constraints

Prover

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$

$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$

⑤ Reveal the evaluation

- for all (i, k), $v^P_{k,i} := P_i(r_k)$
- for all (j, k), $v^Q_{k,j} := Q_j(r_k)$

Revealing ℓ evaluations of $P_i(X)$ leaks no information about w_i.

$\mathrm{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

← $r_1, \ldots, r_\ell$

$\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k, \{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ →

Verifier

Zero-Knowledge Analysis

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

Verifier

Zero-Knowledge Analysis

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$
$$\vdots$$
$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

$\mathrm{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$ →

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

← $r_1, \ldots, r_\ell$

⑤ Reveal the evaluation

- for all (i,k), $v^P_{k,i} := P_i(r_k)$
- for all (j,k), $v^Q_{k,j} := Q_j(r_k)$

← $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

For all k, the evaluation $Q_j(r_k)$ is fully determined by r_k and $(v^P_{k,1}, \ldots, v^P_{k,n})$.

Prover

Basic Proof System for Polynomial Constraints

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.

② Build the polynomials $Q_1, \ldots, Q_m$ such that

$$X \cdot Q_1(X) = f_1(P_1(X), \ldots, P_n(X))$$
$$\vdots$$
$$X \cdot Q_m(X) = f_m(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

⑤ Reveal the evaluation

- for all (i, k), $v^P_{k,i} := P_i(r_k)$
- for all (j, k), $v^Q_{k,j} := Q_j(r_k)$

Hiding Polynomial Commitment Scheme

Prover

Verifier

Zero-Knowledge Analysis

$\mathrm{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$

$r_1, \ldots, r_\ell$

$\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k, \{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Check that, for all k,

$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

Basic Proof System for Polynomial Constraints

I know $w_1, \ldots, w_n$ such that

$$\begin{cases} f_1(w_1, \ldots, w_n) & = 0 \\ & \vdots \\ f_m(w_1, \ldots, w_n) & = 0, \end{cases}$$

where $f_1, \ldots, f_m$ are public **degree-d polynomials**.

Prove it!

Prover

Verifier

Number of opened evaluations

$$\textbf{Soundness Error} = \frac{\binom{d \cdot \ell}{\ell}}{\binom{|\mathscr{C}|}{\ell}}$$

Probability that a malicious prover can convince the verifier.

Size of the challenge space that contains all the possible opened evaluations

Achieving a Negligible Soundness Error

Number of opened evaluations

$$\textbf{Soundness Error} = \frac{\binom{d \cdot \ell}{\ell}}{\binom{|\mathscr{C}|}{\ell}}$$

Probability that a malicious prover can convince the verifier.

Size of the challenge space that contains all the possible opened evaluations

Assume that you want to achieve a negligible soundness error (e.g. 2^{-128})

Achieving a Negligible Soundness Error

Number of opened evaluations

$$\textbf{Soundness Error} = \frac{\binom{d \cdot \ell}{\ell}}{\binom{|\mathscr{C}|}{\ell}}$$

Probability that a malicious prover can convince the verifier.

Size of the challenge space that contains all the possible opened evaluations

Assume that you want to achieve a negligible soundness error (e.g. 2^{-128})

- Take the challenge set $\mathscr{C}$ exponentially large (with $\ell = 1$).

$$\text{Soundness Error} = \frac{d}{|\mathscr{C}|}$$

-

Achieving a Negligible Soundness Error

Number of opened evaluations

$$\textbf{Soundness Error} = \frac{\binom{d \cdot \ell}{\ell}}{\binom{|\mathscr{C}|}{\ell}}$$

Probability that a malicious prover can convince the verifier.

Size of the challenge space that contains all the possible opened evaluations

Assume that you want to achieve a negligible soundness error (e.g. 2^{-128})

- Take the challenge set $\mathscr{C}$ exponentially large (with $\ell = 1$).

$$\text{Soundness Error} = \frac{d}{|\mathscr{C}|}$$

- If not possible (or too costly), take $\ell > 1$ such that $\binom{|\mathscr{C}|}{\ell}$ is exponentially high.
-

Achieving a Negligible Soundness Error

Number of opened evaluations

$$\textbf{Soundness Error} = \frac{\binom{d \cdot \ell}{\ell}}{\binom{|\mathscr{C}|}{\ell}}$$

Probability that a malicious prover can convince the verifier.

Size of the challenge space that contains all the possible opened evaluations

Assume that you want to achieve a negligible soundness error (e.g. 2^{-128})

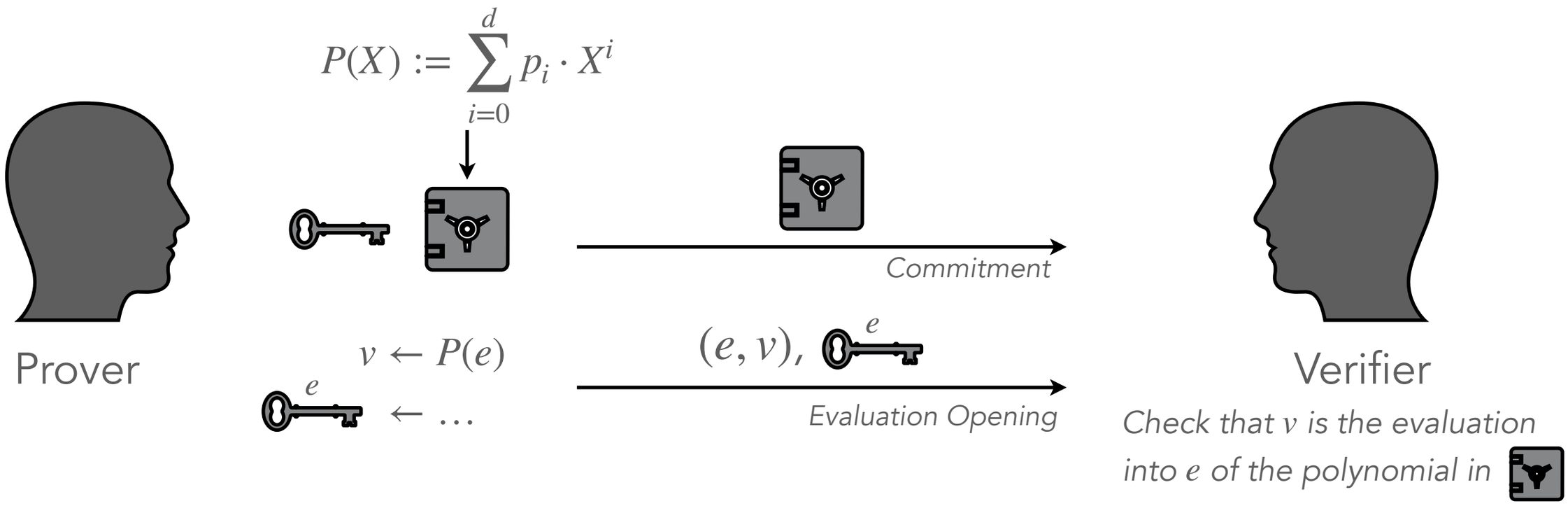
- Take the challenge set $\mathscr{C}$ exponentially large (with $\ell = 1$).

$$\text{Soundness Error} = \frac{d}{|\mathscr{C}|}$$

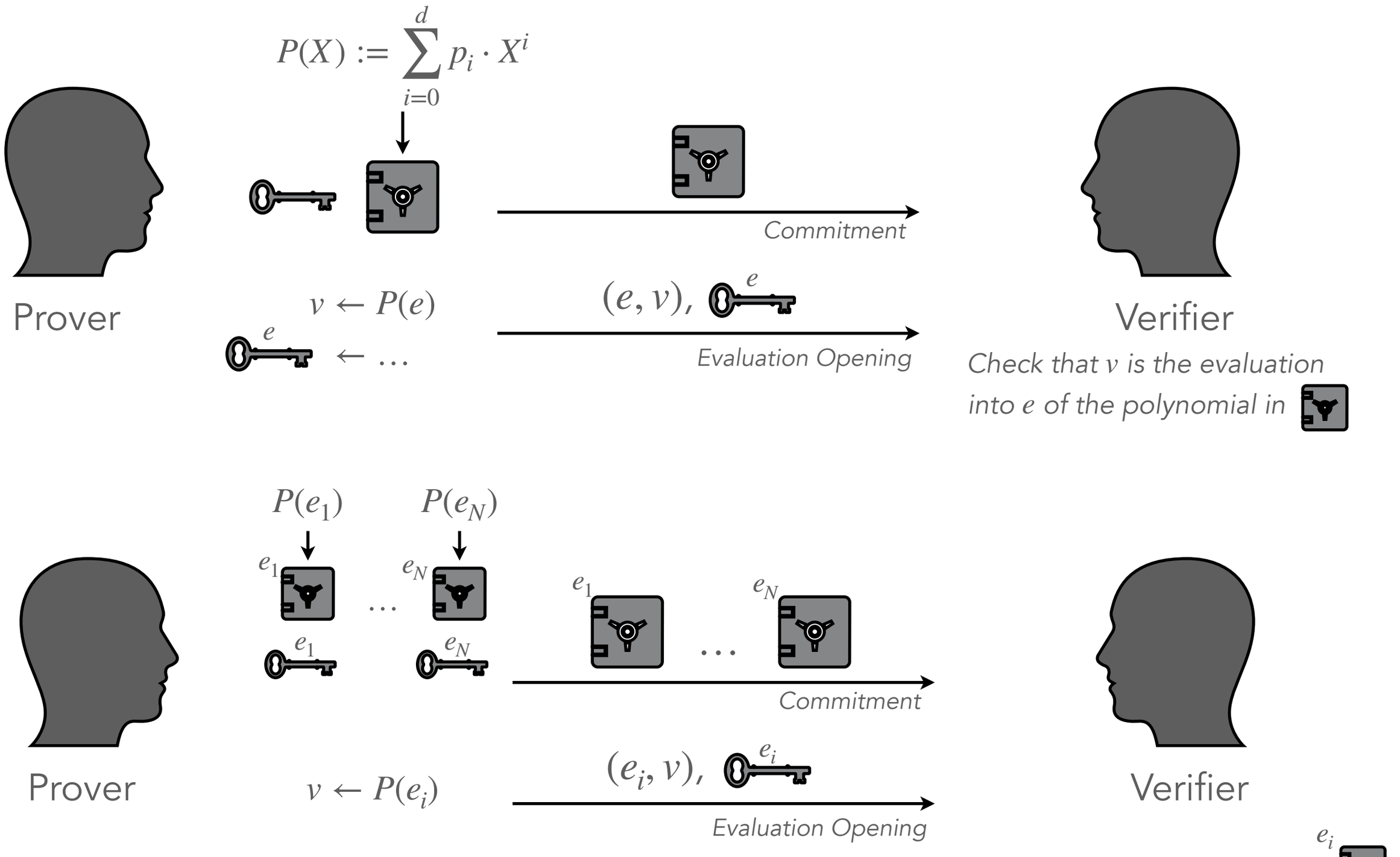
- If not possible (or too costly), take $\ell > 1$ such that $\binom{|\mathscr{C}|}{\ell}$ is exponentially high.
- Last option: rely on ***parallel repetitions*** of the protocol.

Hash-based Polynomial Commitments

Polynomial Commitment Scheme

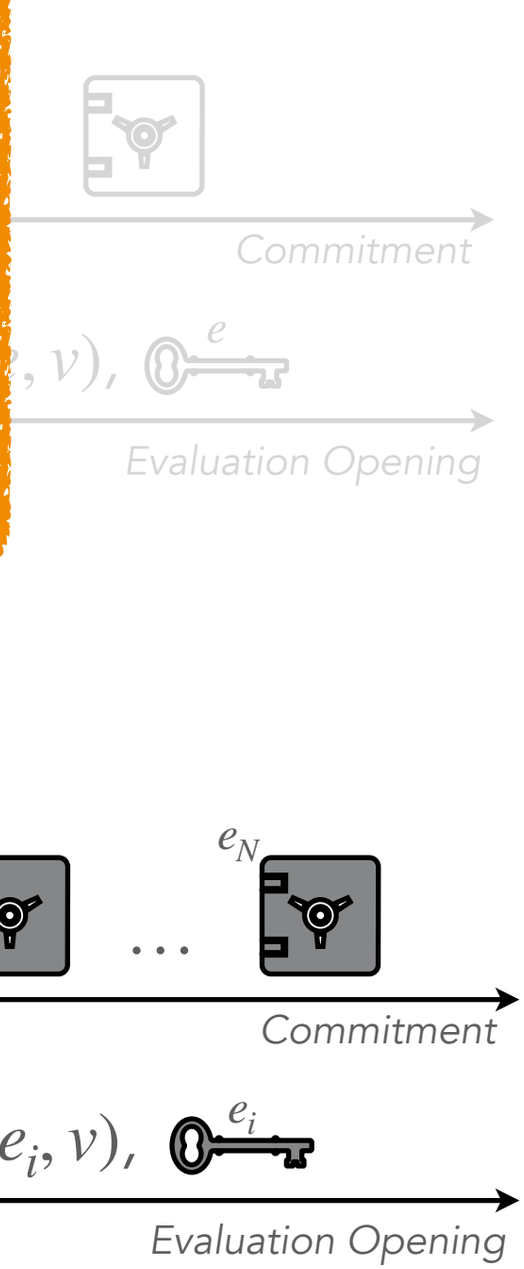


Polynomial Commitment Scheme



Polynomial Commitment Scheme

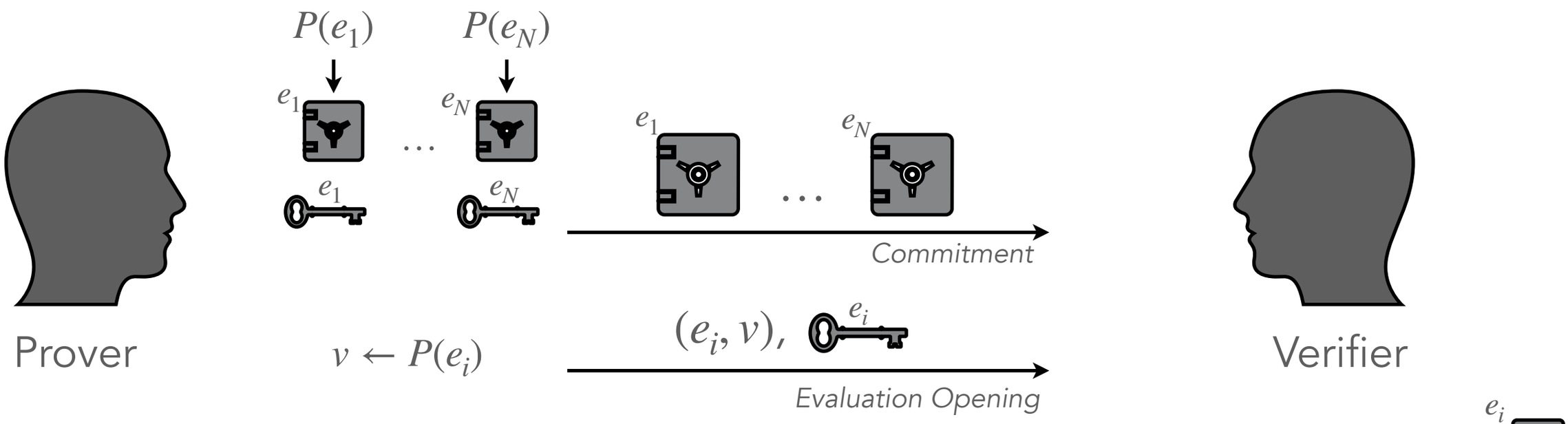
Performance issue:
If the number N of possible evaluations is ***large***, it will be impracticable.



Polynomial Commitment Scheme

Performance issue:
If the number N of possible evaluations is ***large***, it will be impracticable.

Security issue:
The verifier has ***no guarantee*** that the committed evaluations form a polynomial of the right degree.



Basic Proof System for Polynomial Constraints

Verifier

① For all i, choose a degree-ℓ polynomial $P_i(X)$. There exist j^* such that

$$f_{j^*}(P_1(0), \ldots, P_n(0)) \neq 0.$$

② Choose some polynomials $Q_1, \ldots, Q_m$. We know that

$$X \cdot Q_{j^*}(X) \neq f_{j^*}(P_1(X), \ldots, P_n(X))$$

③ Commit the polynomials $(P_1, \ldots, P_n)$ and $(Q_1, \ldots, Q_m)$.

$\text{PCom}(P_1, \ldots, P_n, Q_1, \ldots, Q_m)$

④ Choose ℓ random evaluation points $r_1, \ldots, r_\ell \in \mathscr{C} \subset \mathbb{F}$

$r_1, \ldots, r_\ell$

$\{\}_k$

⑥ Check that $\{v^P_{k,1}, \ldots, v^P_{k,n}\}_k$ and $\{v^Q_{k,1}, \ldots, v^Q_{k,m}\}_k$ are consistent with the commitment.

Soundness Analysis

Schwartz-Zippel Lemma: Let D be the **non-zero** degree-$(d \cdot \ell)$ polynomial defined as

$$D := X \cdot Q_{j^*}(X) - f_{j^*}(P_1(X), \ldots, P_n(X))$$

We have

$$\Pr[\text{verification passes}] = \Pr\left[\forall k, D(r_k) = 0 \mid \{r_k\}_k \subset_\$ \mathscr{C}\right] \leq \frac{\binom{d \cdot \ell}{\ell}}{\binom{|\mathscr{C}|}{\ell}}.$$

Check that, for all k,

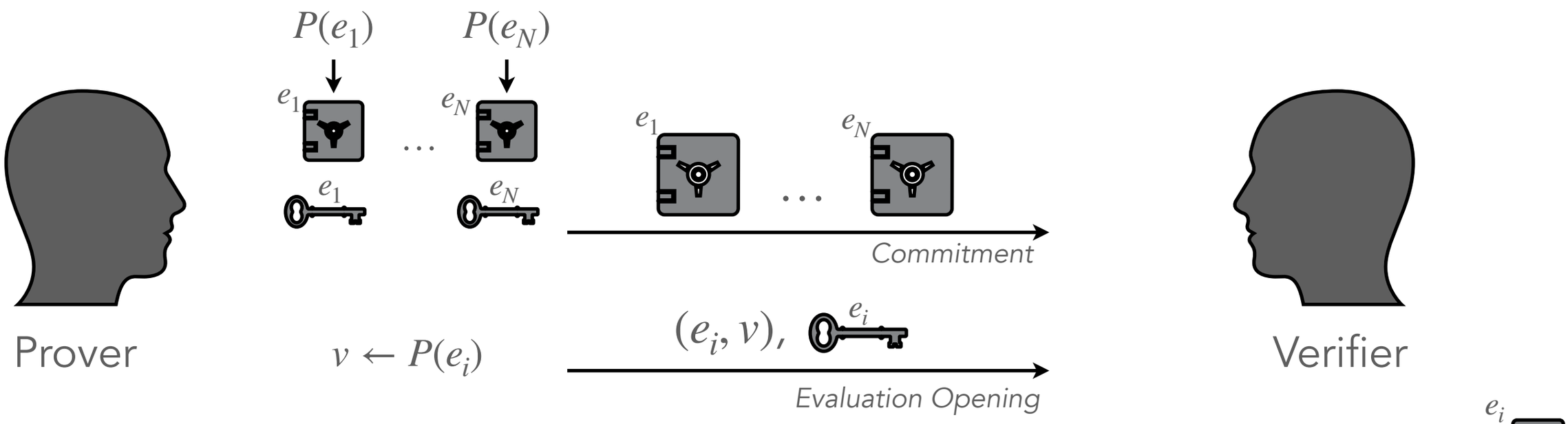
$$r_k \cdot v^Q_{k,1} = f_1(v^P_{k,1}, \ldots, v^P_{k,n})$$
$$\vdots$$
$$r_k \cdot v^Q_{k,m} = f_m(v^P_{k,1}, \ldots, v^P_{k,n})$$

$$r_k \cdot v^Q_{k,j^*} = f_{j^*}(v^P_{k,1}, \ldots, v^P_{k,n})$$

Polynomial Commitment Scheme

Performance issue:
If the number N of possible evaluations is ***large***, it will be impracticable.

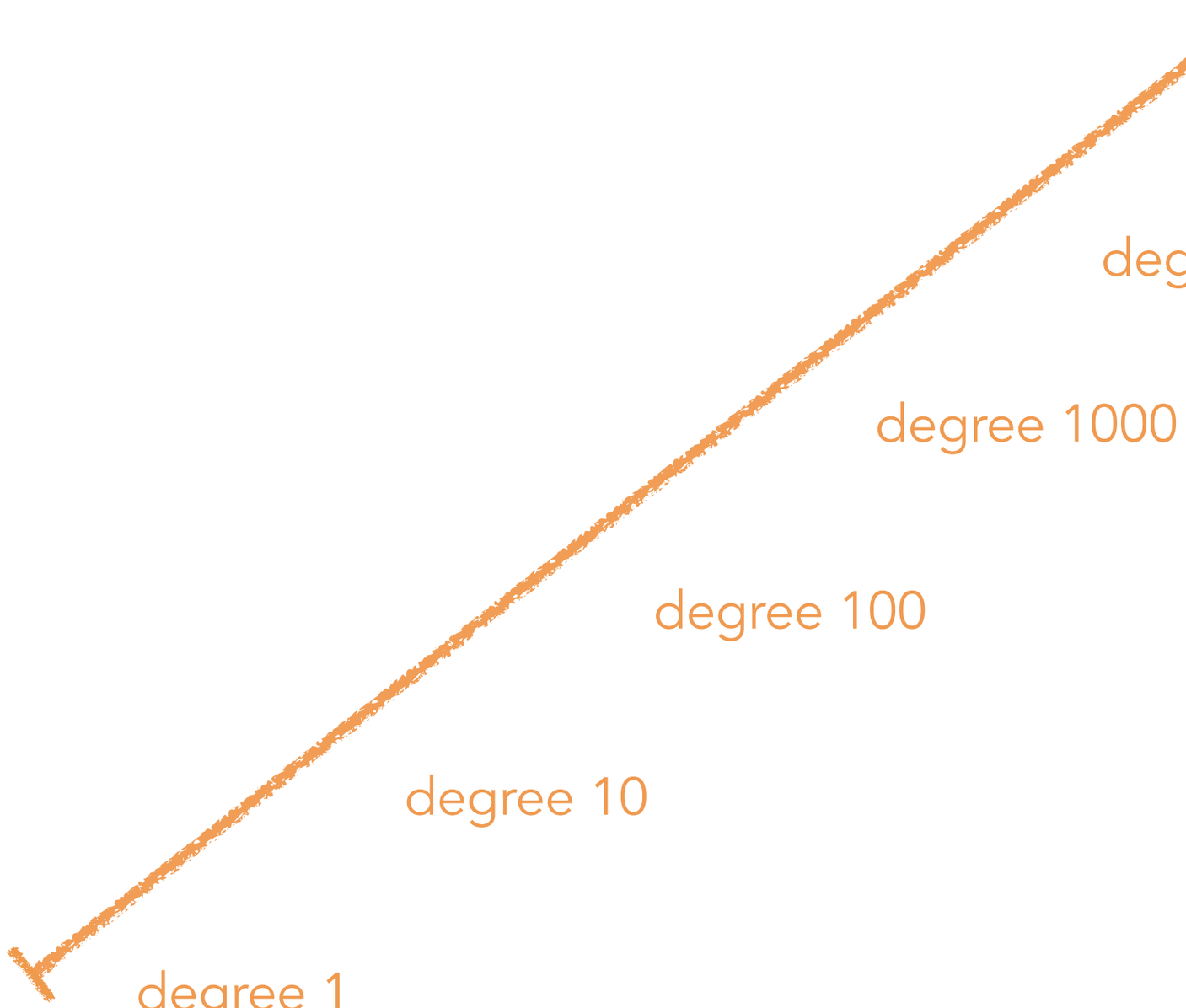
Security issue:
The verifier has ***no guarantee*** that the committed evaluations form a polynomial of the right degree.



How to commit to polynomials?

(using symmetric primitives)

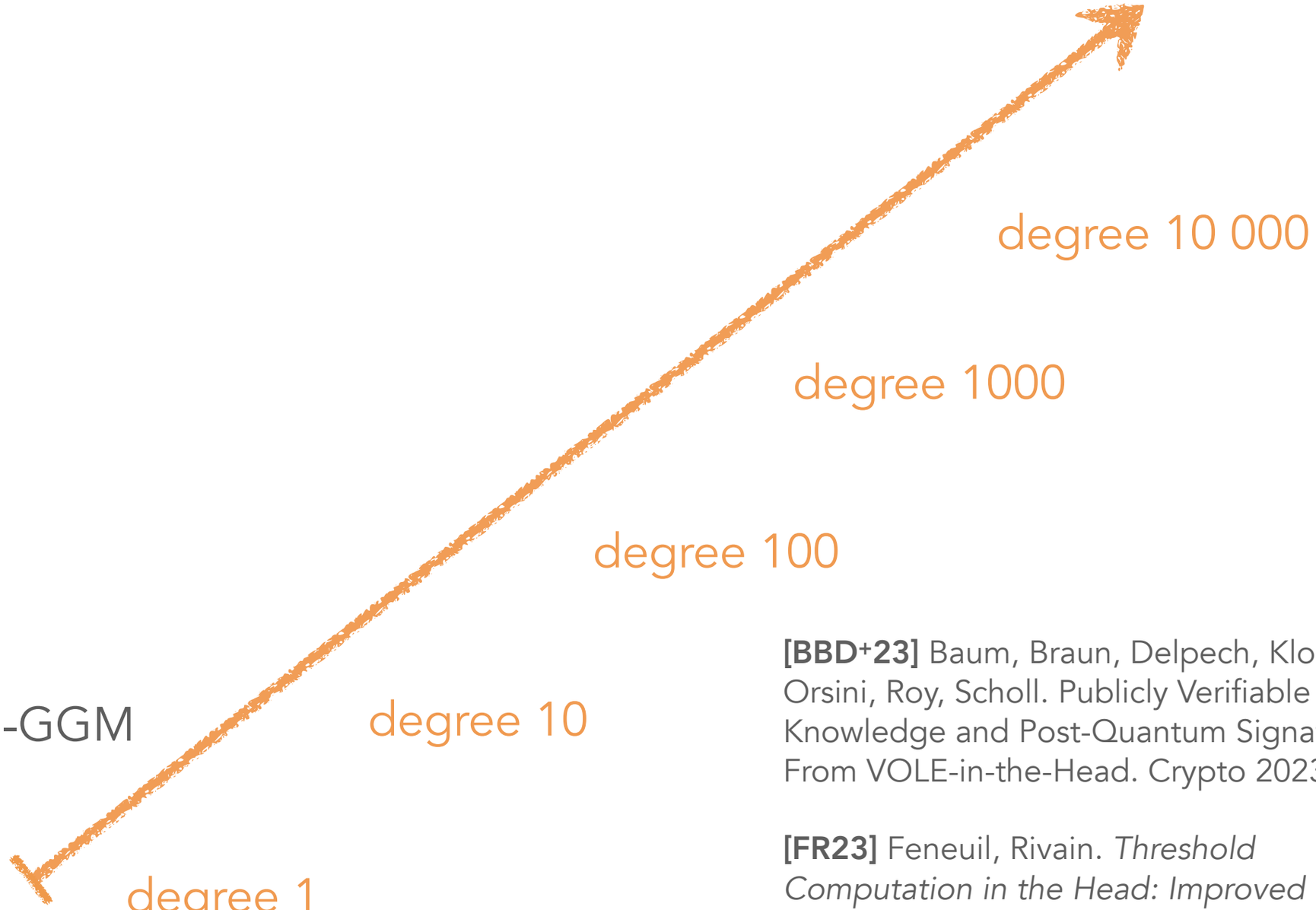
Values are indicative only and serve to illustrate the progression of the scale.



How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.



① VOLEitH / TCitH-GGM

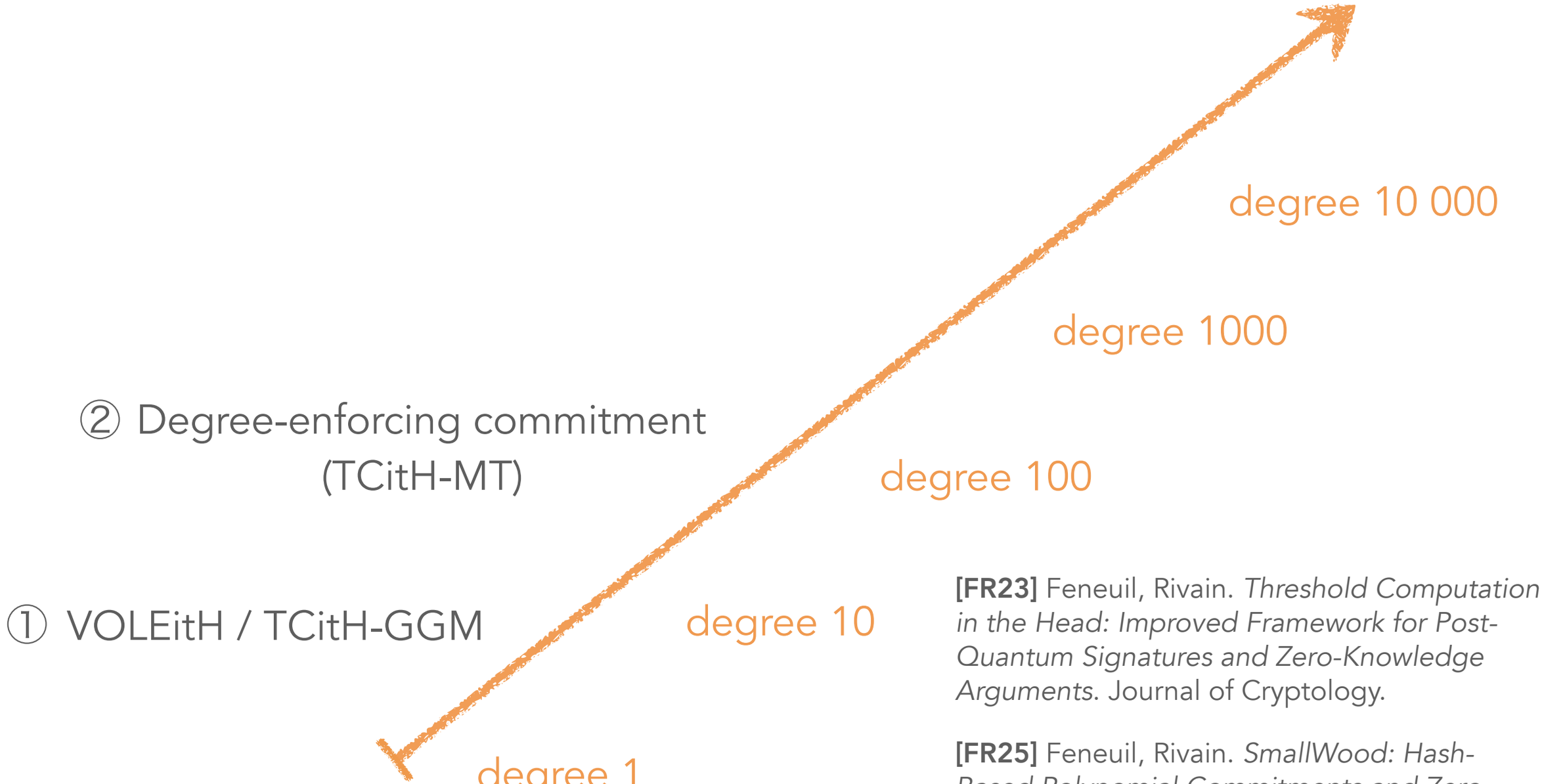
[BBD+23] Baum, Braun, Delpech, Klooß, Orsini, Roy, Scholl. Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures From VOLE-in-the-Head. Crypto 2023.

[FR23] Feneuil, Rivain. *Threshold Computation in the Head: Improved Framework for Post-Quantum Signatures and Zero-Knowledge Arguments*. To appear to Journal of Cryptology.

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.



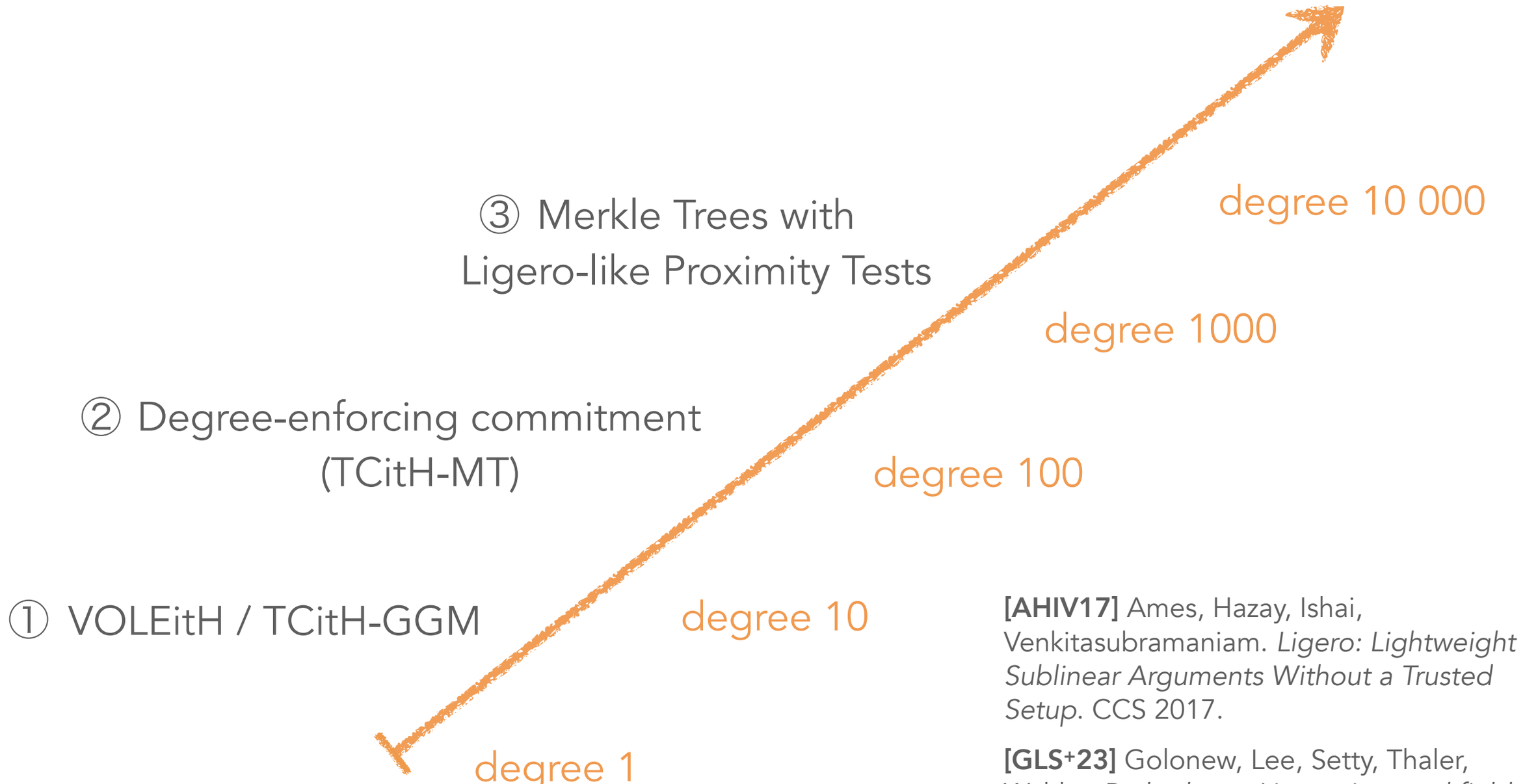
[FR23] Feneuil, Rivain. *Threshold Computation in the Head: Improved Framework for Post-Quantum Signatures and Zero-Knowledge Arguments*. Journal of Cryptology.

[FR25] Feneuil, Rivain. *SmallWood: Hash-Based Polynomial Commitments and Zero-Knowledge Arguments for Relatively Small Instances*. ePrint 2025/1085.

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.



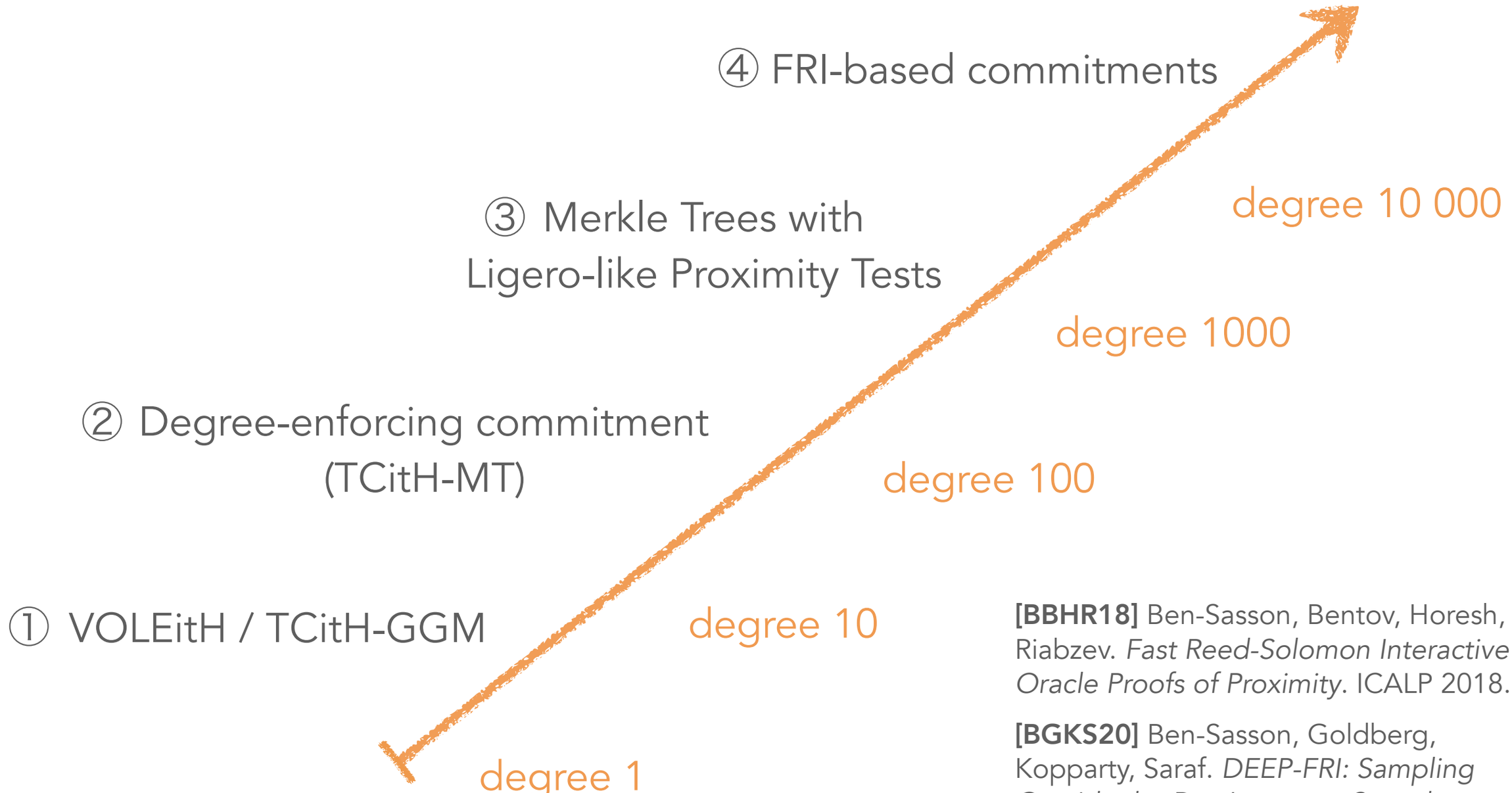
[AHIV17] Ames, Hazay, Ishai, Venkitasubramaniam. *Ligero: Lightweight Sublinear Arguments Without a Trusted Setup*. CCS 2017.

[GLS+23] Golonew, Lee, Setty, Thaler, Wahby. *Brakedown: Linear-time and field agnotic SNARKs for R1CS*. Crypto 2023.

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.



[BBHR18] Ben-Sasson, Bentov, Horesh, Riabzev. *Fast Reed-Solomon Interactive Oracle Proofs of Proximity*. ICALP 2018.

[BGKS20] Ben-Sasson, Goldberg, Kopparty, Saraf. *DEEP-FRI: Sampling Outside the Box Improves Soundness*. ITCS 2020.

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.

Merkle Tree

④ FRI-based commitments

③ Merkle Trees with
Ligero-like Proximity Tests

② Degree-enforcing commitment
(TCitH-MT)

① VOLEitH / TCitH-GGM

GGM Tree

degree 10 000

degree 1000

degree 100

degree 10

degree 1

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.

Merkle Tree

④ FRI-based commitments

③ Merkle Trees with
Ligero-like Proximity Tests

② Degree-enforcing commitment
(TCitH-MT)

① VOLEitH / TCitH-GGM

GGM Tree

degree 10 000

degree 1000

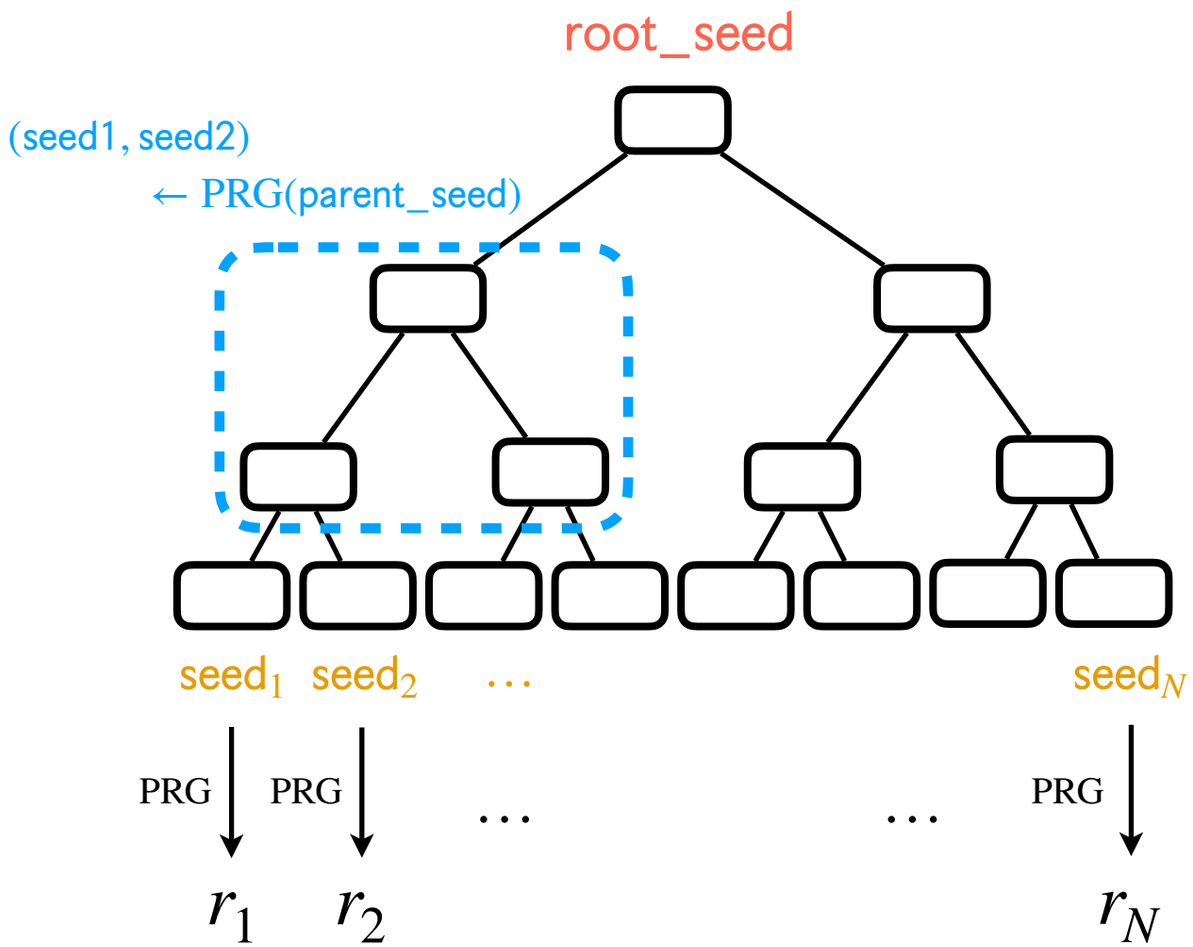
degree 100

degree 10

degree 1

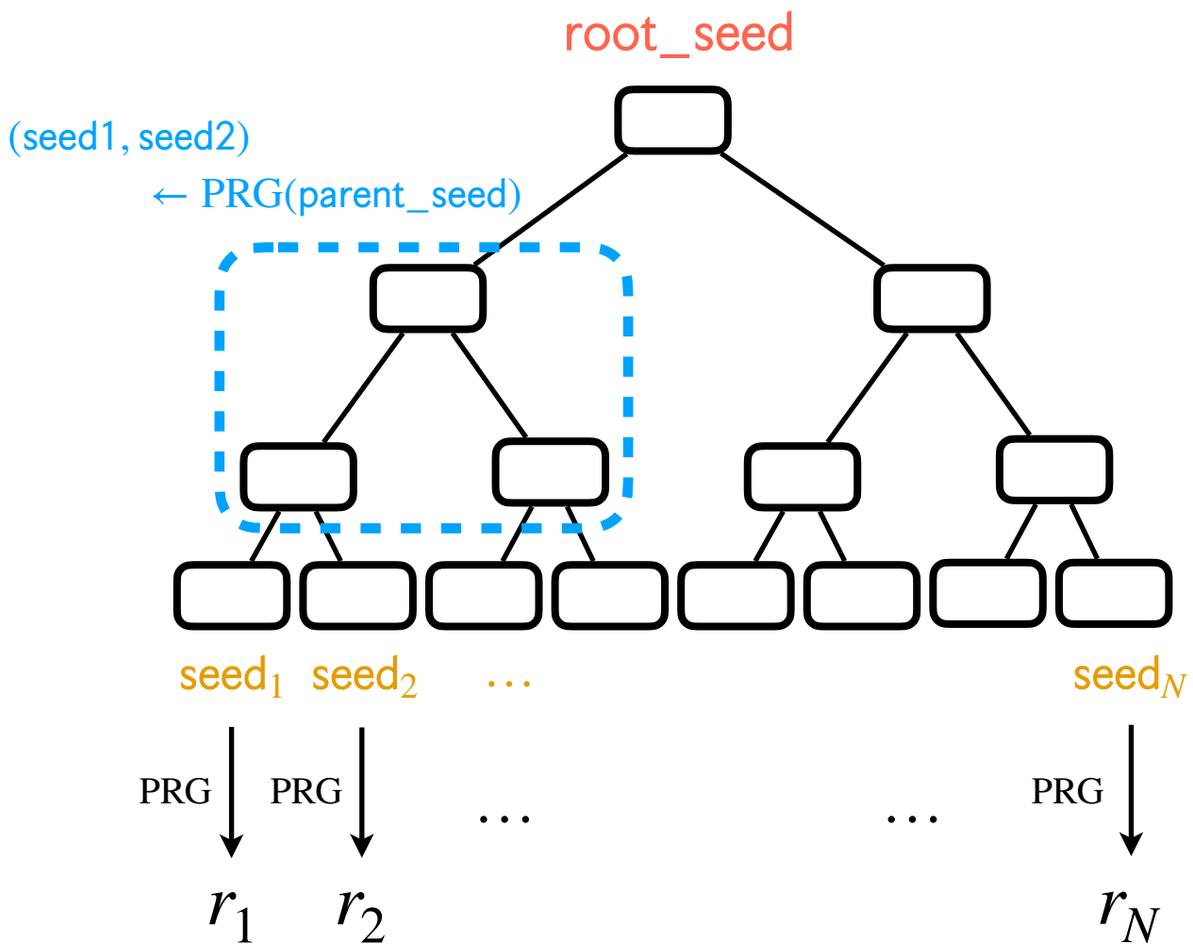
Option 1: Using a GGM tree (*ie.* a seed tree)

[GGM84] Goldreich, Goldwasser, Micali: "How to construct random functions (extended extract)" (FOCS 1984)



Option 1: Using a GGM tree (*ie.* a seed tree)

[GGM84] Goldreich, Goldwasser, Micali: "How to construct random functions (extended extract)" (FOCS 1984)



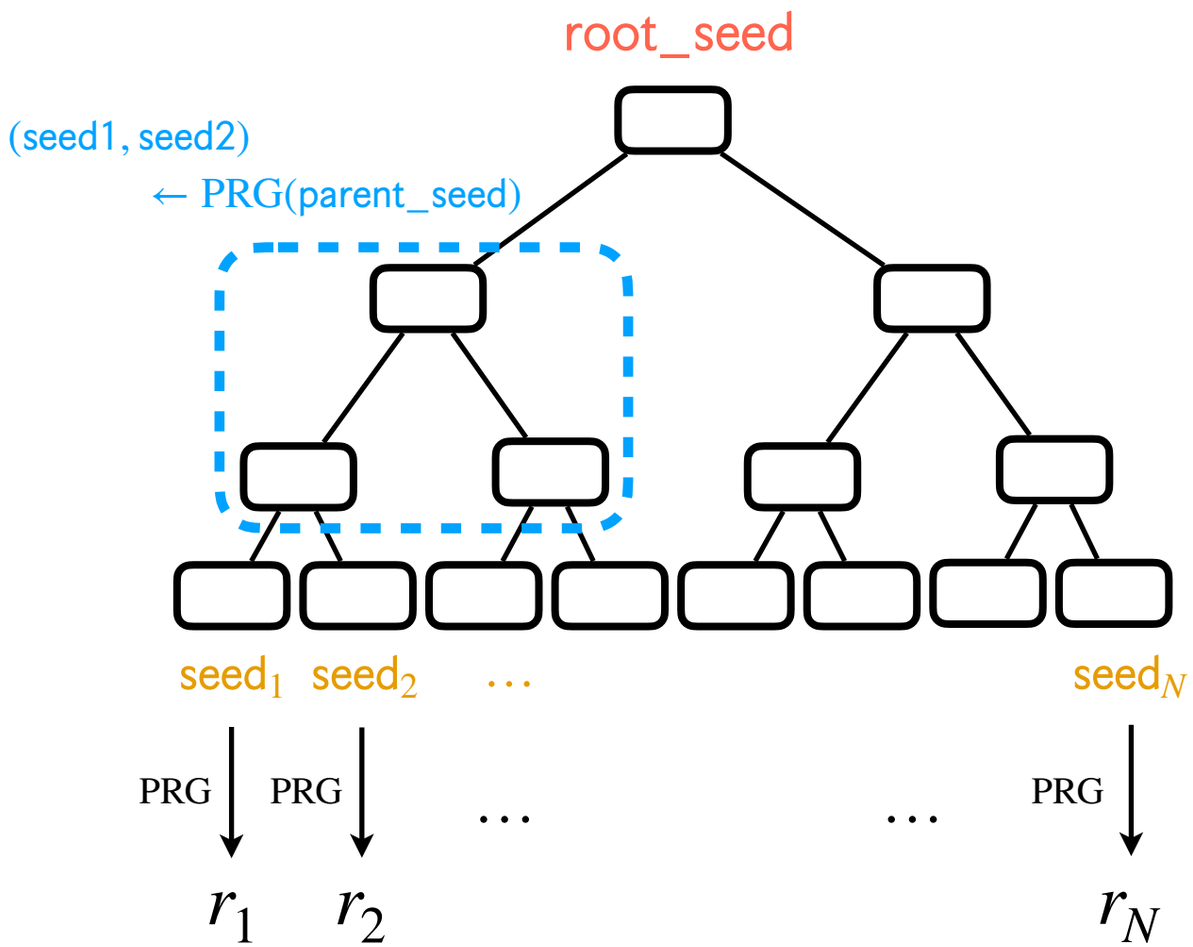
Build $\Delta P(X)$ as

$$\Delta P(X) := P(X) - \sum_{i=1}^{N} r_i \cdot (X - e_i)$$

(assuming $\deg P = 1$*)*

Option 1: Using a GGM tree (*ie.* a seed tree)

[GGM84] Goldreich, Goldwasser, Micali: "How to construct random functions (extended extract)" (FOCS 1984)



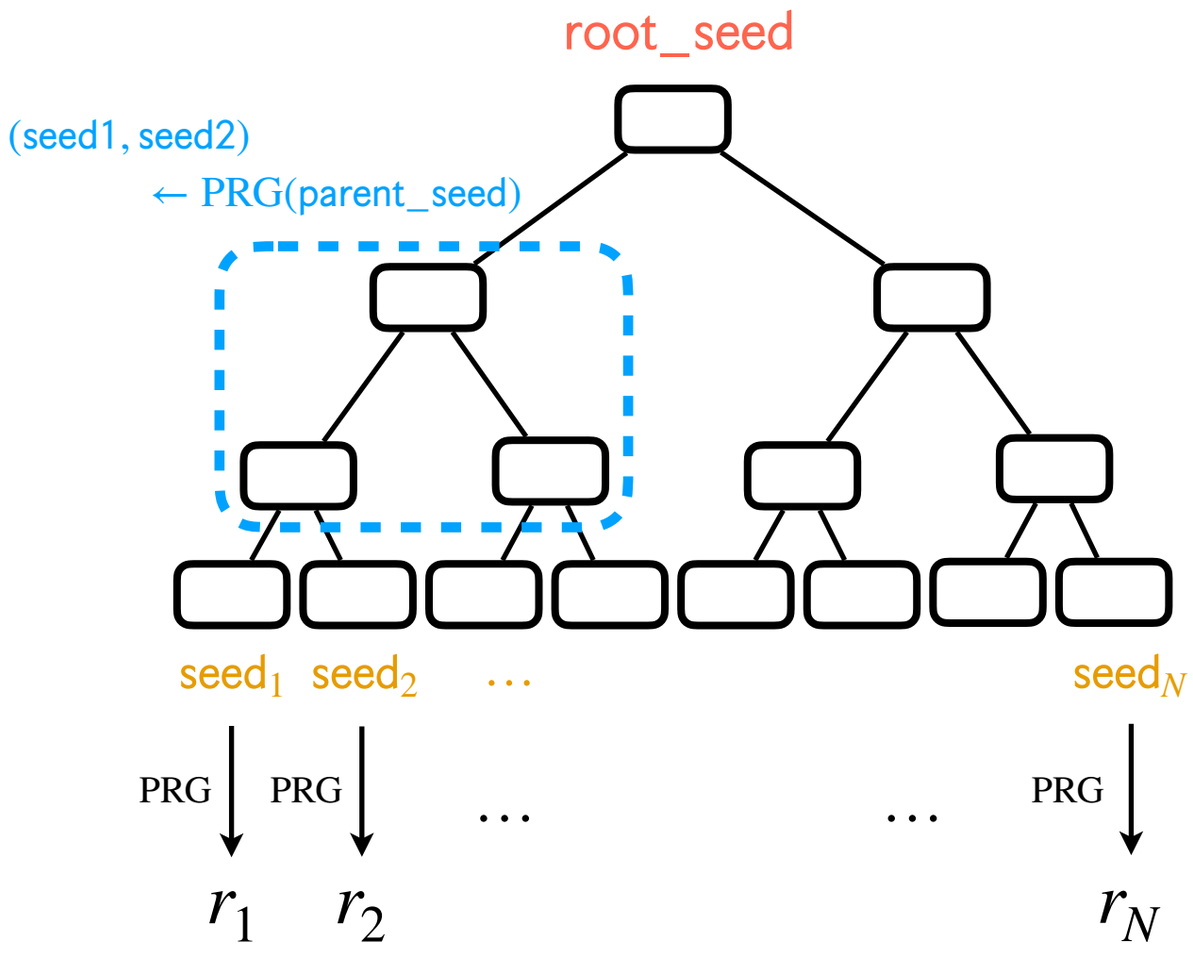
Build $\Delta P(X)$ as

$$\Delta P(X) := P(X) - \underbrace{\sum_{i=1}^{N} r_i \cdot (X - e_i)}_{\text{Mask}}$$

(assuming $\deg P = 1$*)*

Option 1: Using a GGM tree (*ie.* a seed tree)

[GGM84] Goldreich, Goldwasser, Micali: "How to construct random functions (extended extract)" (FOCS 1984)



Build $\Delta P(X)$ as

$$\Delta P(X) := P(X) - \underbrace{\sum_{i=1}^{N} r_i \cdot (X - e_i)}_{\textit{Mask}}$$

(assuming $\deg P = 1$*)*

Commitment:

- *Commit to each seed* ***independently***
- *Reveal the masked polynomial* $\Delta P(X)$

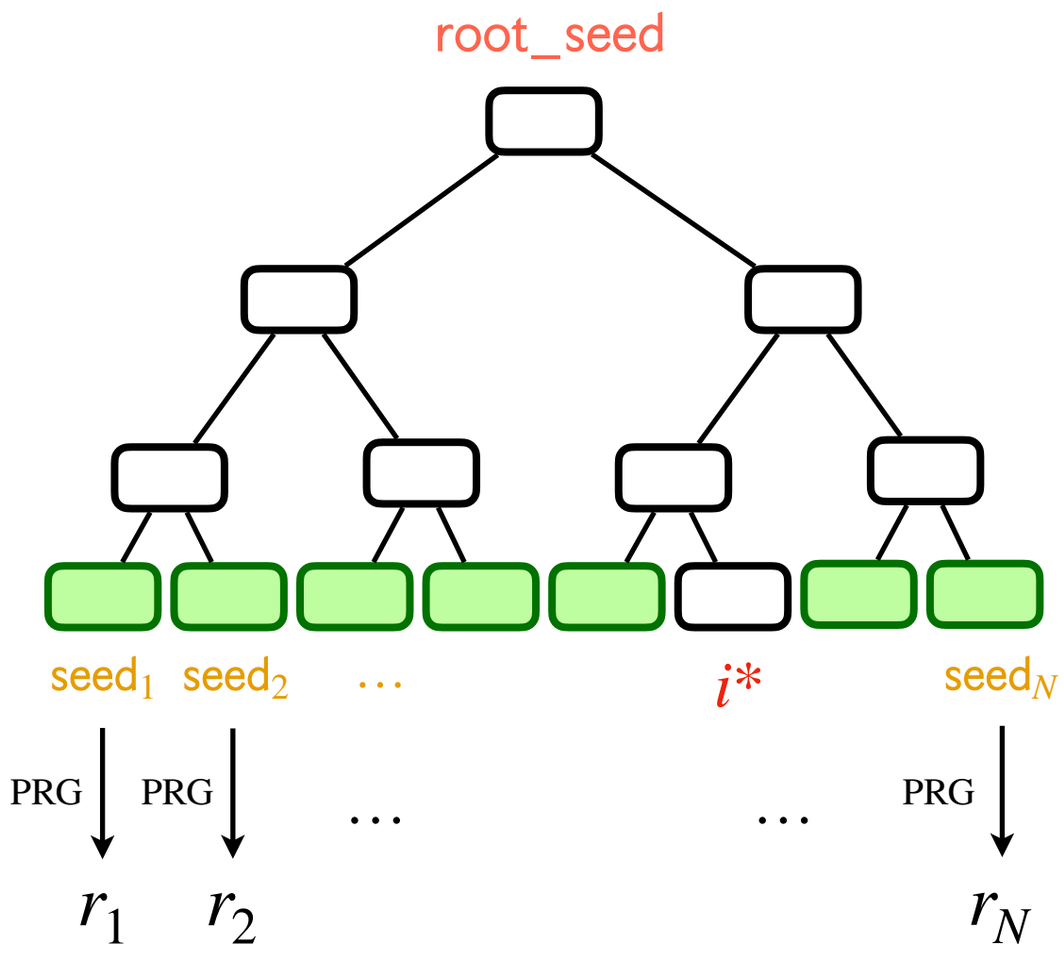
Open $P(e_{i^*})$*:*

Reveal all $\{r_i\}_{i \neq i^*}$ *since*

$$P(e_{i^*}) = \Delta P(e_{i^*}) + \sum_{i \neq i^*} r_i \cdot (e_{i^*} - e_i)$$

Option 1: Using a GGM tree (*ie.* a seed tree)

[GGM84] Goldreich, Goldwasser, Micali: "How to construct random functions (extended extract)" (FOCS 1984)



Build $\Delta P(X)$ as

$$\Delta P(X) := P(X) - \underbrace{\sum_{i=1}^{N} r_i \cdot (X - e_i)}_{\text{Mask}}$$

(assuming $\deg P = 1$*)*

Commitment:

- *Commit to each seed* ***independently***
- *Reveal the masked polynomial* $\Delta P(X)$

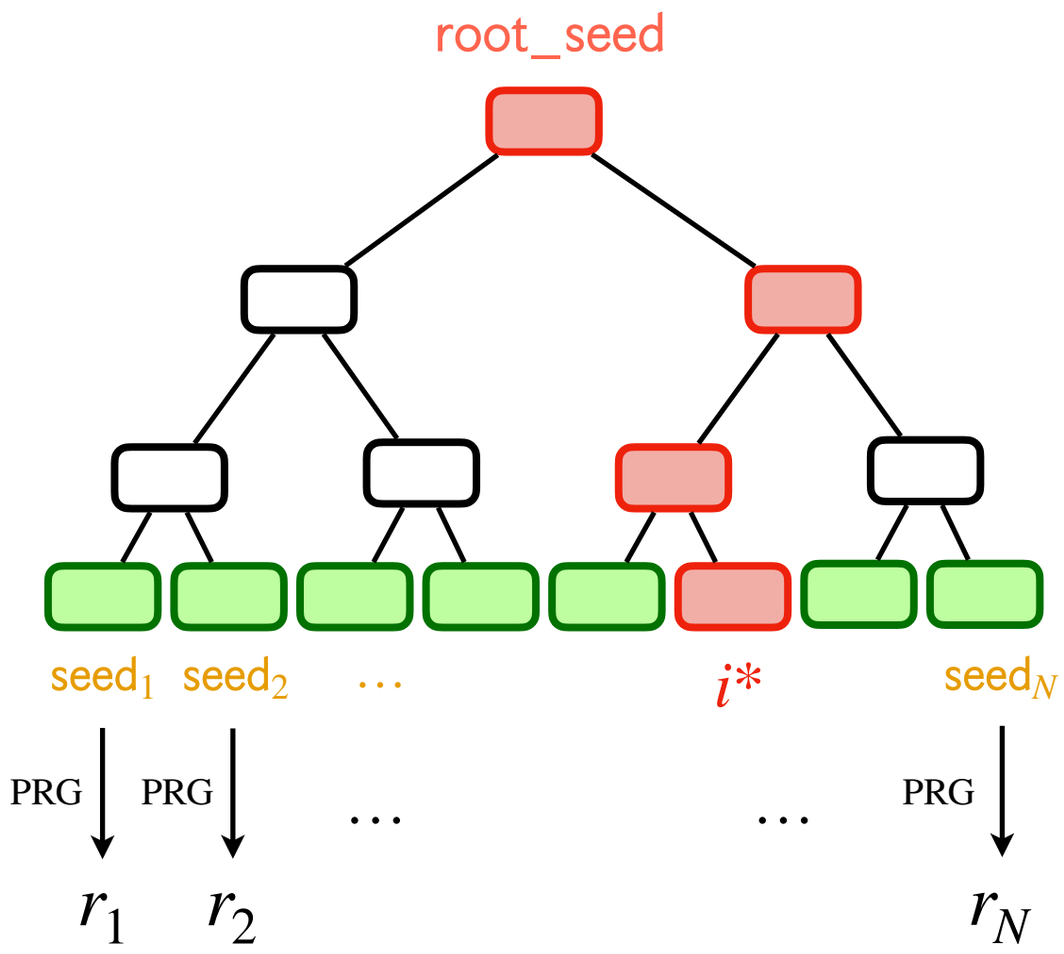
Open $P(e_{i^*})$*:*

Reveal all $\{r_i\}_{i \neq i^*}$ *since*

$$P(e_{i^*}) = \Delta P(e_{i^*}) + \sum_{i \neq i^*} r_i \cdot (e_{i^*} - e_i)$$

Option 1: Using a GGM tree (*ie.* a seed tree)

[GGM84] Goldreich, Goldwasser, Micali: "How to construct random functions (extended extract)" (FOCS 1984)



Commitment:

- *Commit to each seed* ***independently***
- *Reveal the masked polynomial* $\Delta P(X)$

Open $P(e_{i^*})$:

Reveal all $\{r_i\}_{i \neq i^*}$ *since*

$$P(e_{i^*}) = \Delta P(e_{i^*}) + \sum_{i \neq i^*} r_i \cdot (e_{i^*} - e_i)$$

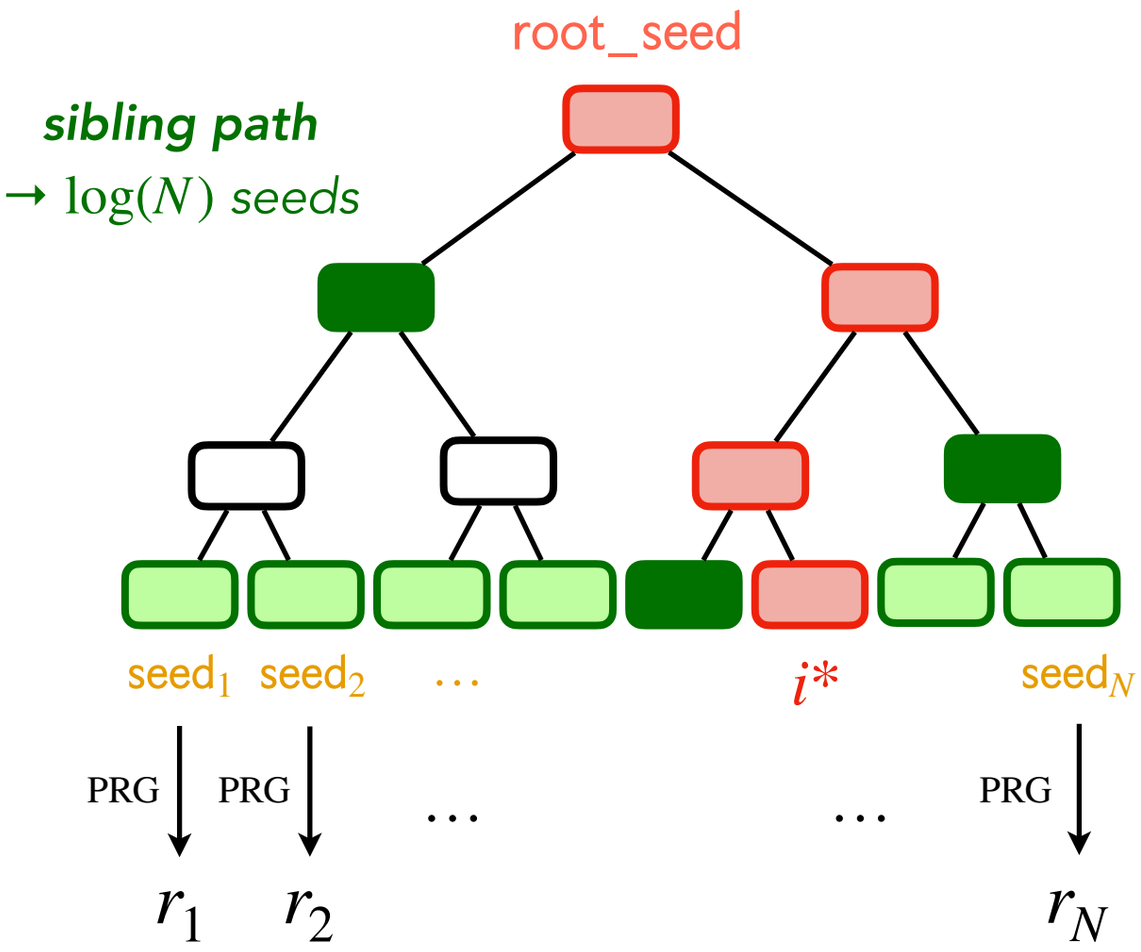
Build $\Delta P(X)$ as

$$\Delta P(X) := P(X) - \underbrace{\sum_{i=1}^{N} r_i \cdot (X - e_i)}_{\text{Mask}}$$

(assuming $\deg P = 1$*)*

Option 1: Using a GGM tree (*ie.* a seed tree)

[GGM84] Goldreich, Goldwasser, Micali: "How to construct random functions (extended extract)" (FOCS 1984)



Commitment:

- *Commit to each seed* ***independently***
- *Reveal the masked polynomial* $\Delta P(X)$

Open $P(e_{i^*})$:

Reveal all $\{r_i\}_{i \neq i^*}$ *since*

$$P(e_{i^*}) = \Delta P(e_{i^*}) + \sum_{i \neq i^*} r_i \cdot (e_{i^*} - e_i)$$

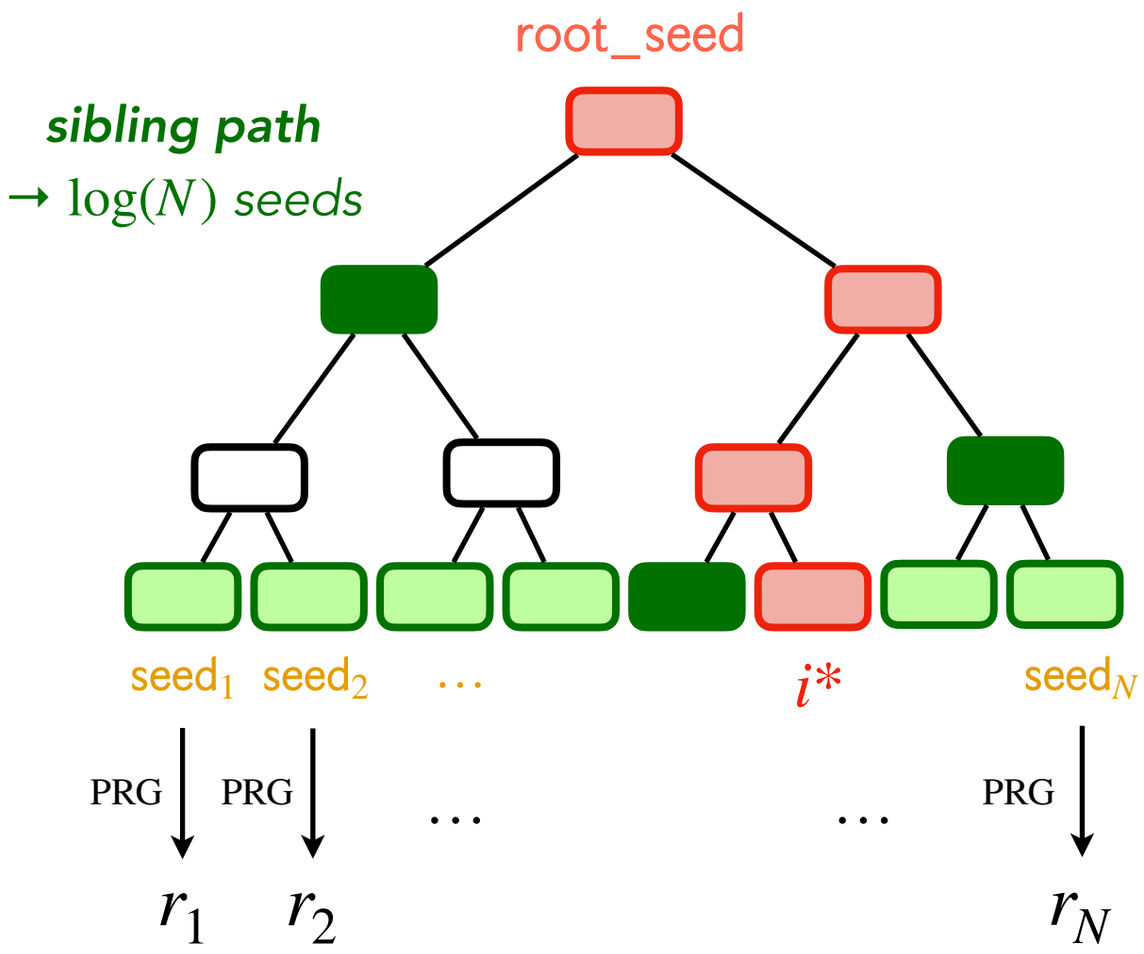
Build $\Delta P(X)$ as

$$\Delta P(X) := P(X) - \underbrace{\sum_{i=1}^{N} r_i \cdot (X - e_i)}_{\text{Mask}}$$

(assuming $\deg P = 1$*)*

Option 1: Using a GGM tree (*ie.* a seed tree)

[GGM84] Goldreich, Goldwasser, Micali: "How to construct random functions (extended extract)" (FOCS 1984)



Build $\Delta P(X)$ as

$$\Delta P(X) := P(X) - \underbrace{\sum_{i=1}^{N} r_i \cdot (X - e_i)}_{\text{Mask}}$$

(assuming $\deg P = 1$*)*

Commitment:

- *Commit to each seed* ***independently***
- *Reveal the masked polynomial* $\Delta P(X)$

Open $P(e_{i*})$*:*

Reveal all $\{r_i\}_{i \neq i^*}$ *since*

$$P(e_{i^*}) = \Delta P(e_{i^*}) + \sum_{i \neq i^*} r_i \cdot (e_{i^*} - e_i)$$

Properties:

- *Cost of sending a tree node:* λ *bits*
- *Verification complexity:* $O(N)$
- *Nodes contain sensitive information*
- *Commitment cost:* $O_\lambda(\#P \times \deg P)$
- *The committed polynomial* P *is naturally of the right degree*

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.

Merkle Tree

④ FRI-based commitments

③ Merkle Trees with
Ligero-like Proximity Tests

② Degree-enforcing commitment
(TCitH-MT)

① VOLEitH / TCitH-GGM

GGM Tree

degree 10 000

degree 1000

degree 100

degree 10

degree 1

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.

Merkle Tree

④ FRI-based commitments

③ Merkle Trees with
Ligero-like Proximity Tests

② Degree-enforcing commitment
(TCitH-MT)

① VOLEitH / TCitH-GGM

GGM Tree

degree 10 000

degree 1000

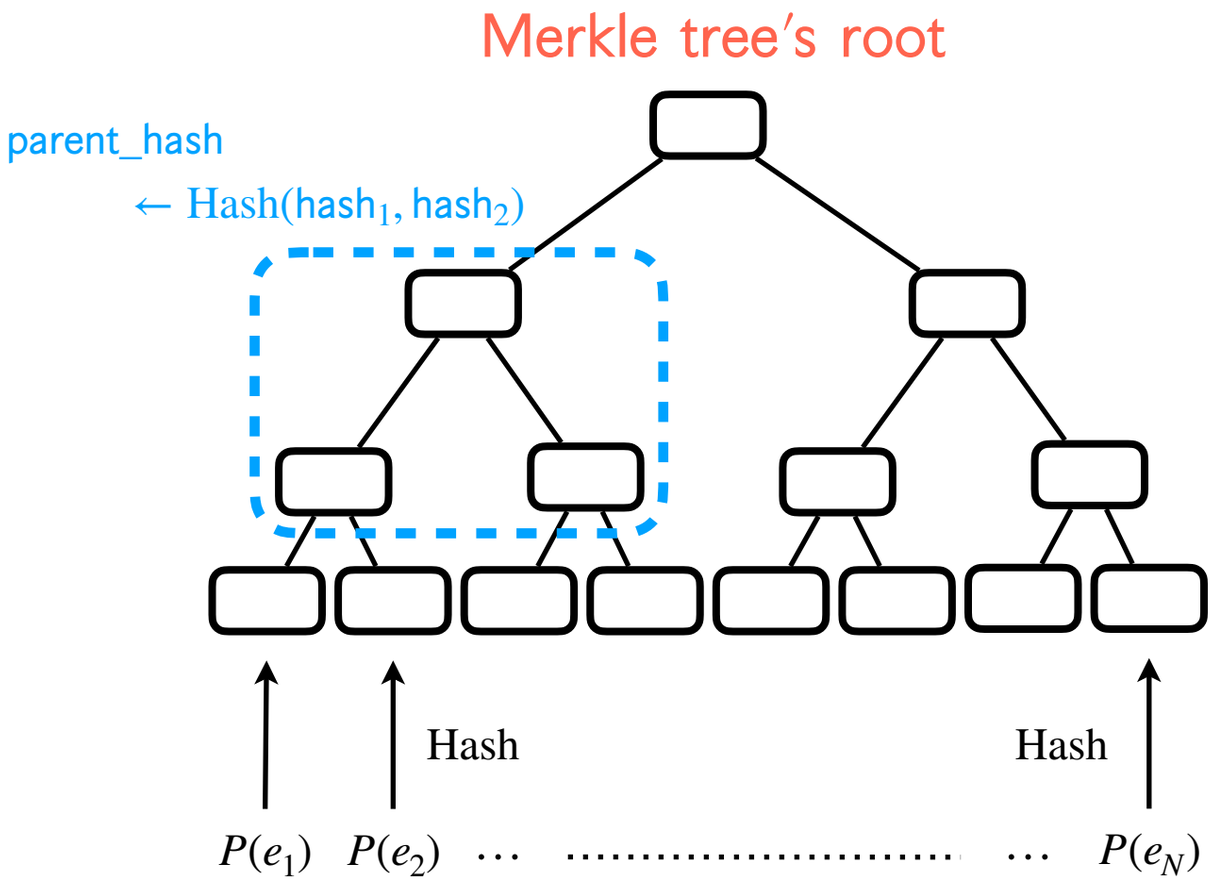
degree 100

degree 10

degree 1

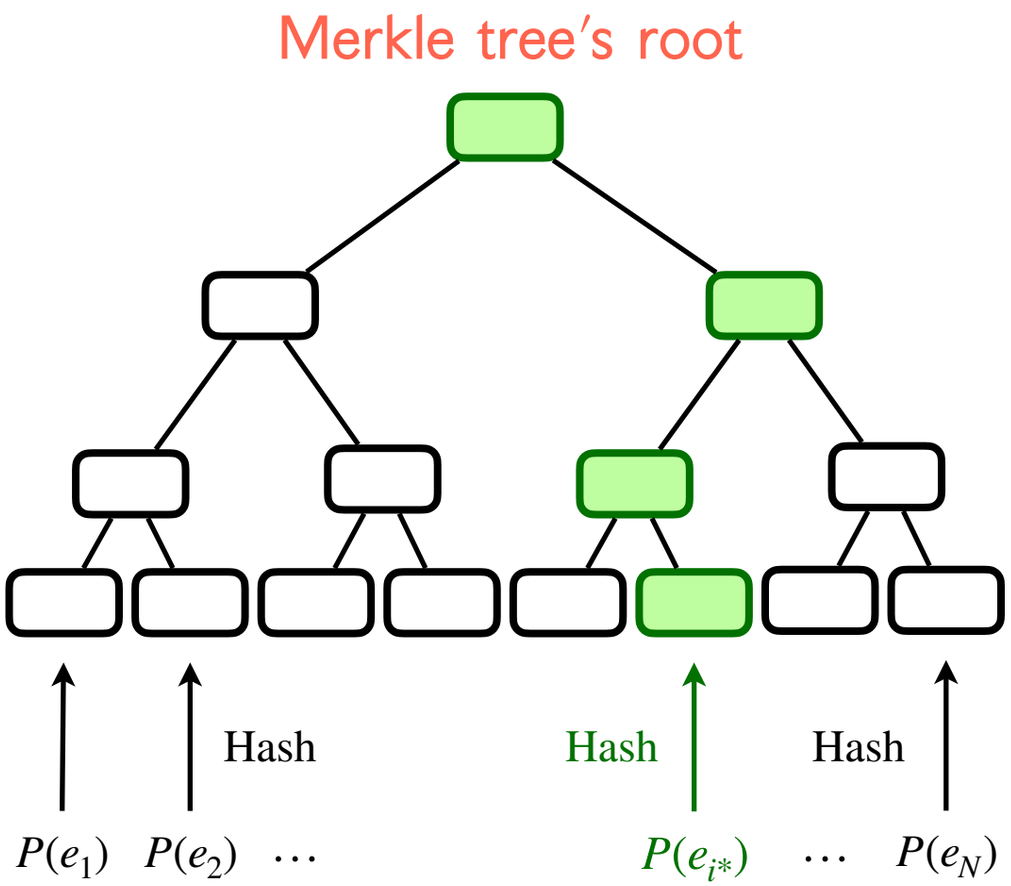
Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)



Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)

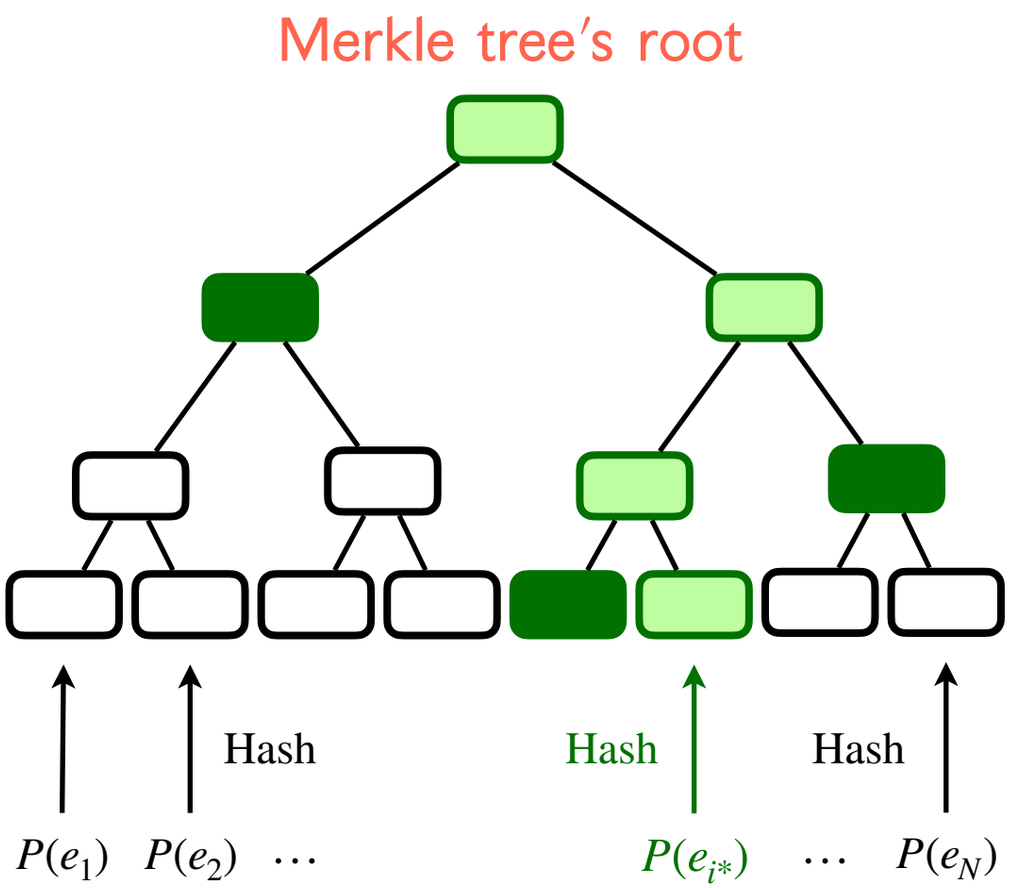


Open $P(e_{i*})$:

Reveal the authentication path of $P(e_{i*})$

Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)

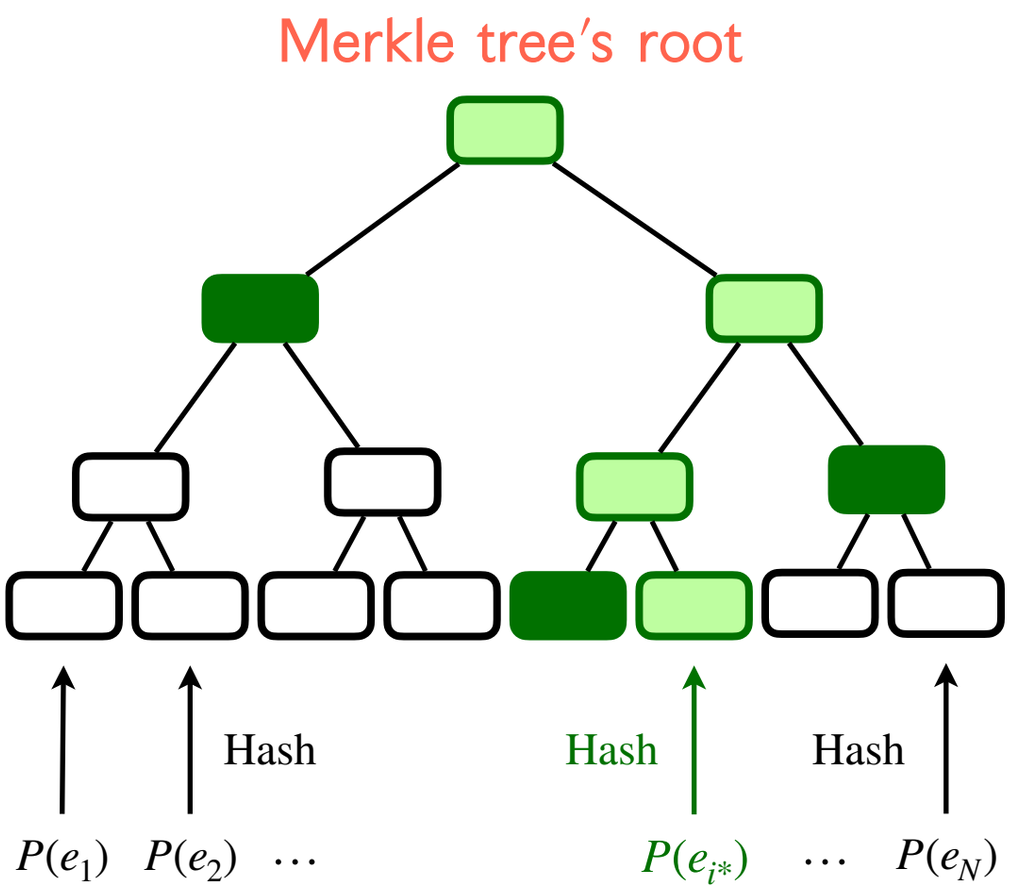


Open $P(e_{i*})$:

Reveal the authentication path of $P(e_{i*})$

Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)



Commitment:

- Reveal the Merkle root

- Use a mechanism to ensure that the committed polynomial of the right degree

Open $P(e_{i*})$:

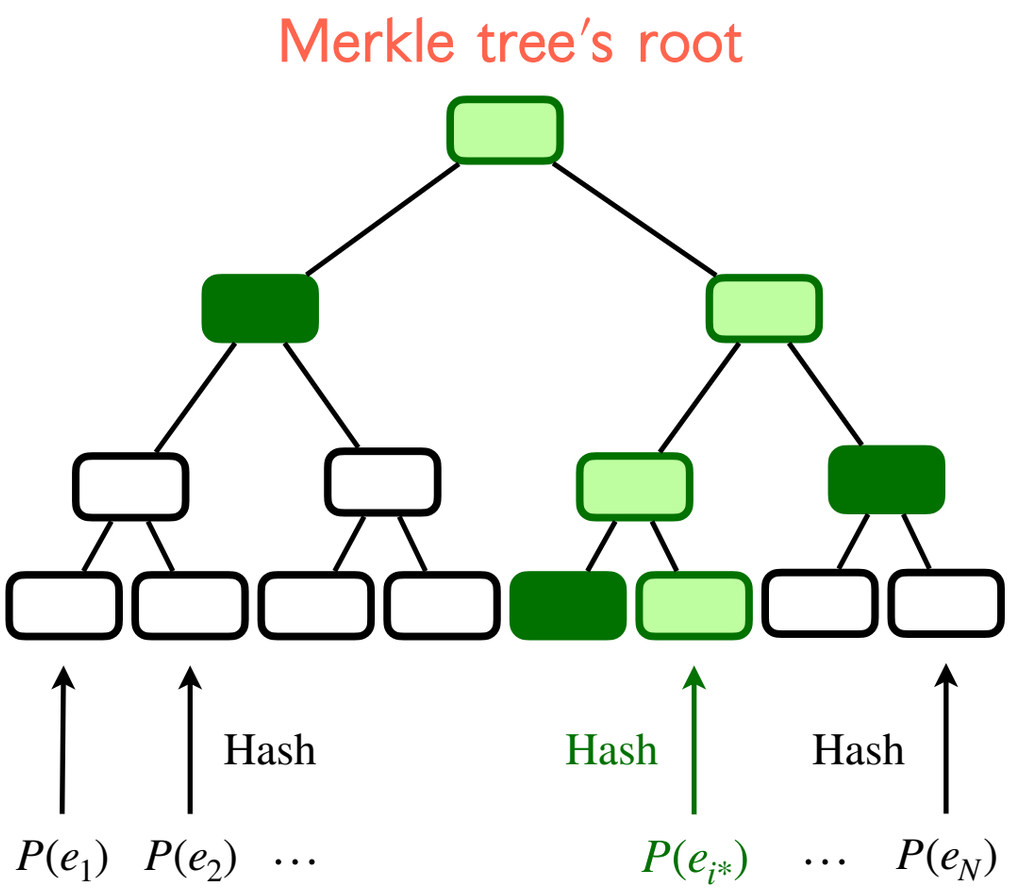
Reveal the authentication path of $P(e_{i*})$

⚠ Need to ensure that the committed evaluations correspond to a polynomial of the right degree:

Large polynomials: Proximity Test (Ligero-like or FRI)
Small polynomials: Degree-Enforcing Test (TCitH)

Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)



⚠ Need to ensure that the committed evaluations correspond to a polynomial of the right degree:

Large polynomials: Proximity Test (Ligero-like or FRI)
Small polynomials: Degree-Enforcing Test (TCitH)

Commitment:

- Reveal the Merkle root
- Use a mechanism to ensure that the committed polynomial of the right degree

Open $P(e_{i*})$:

Reveal the authentication path of $P(e_{i*})$

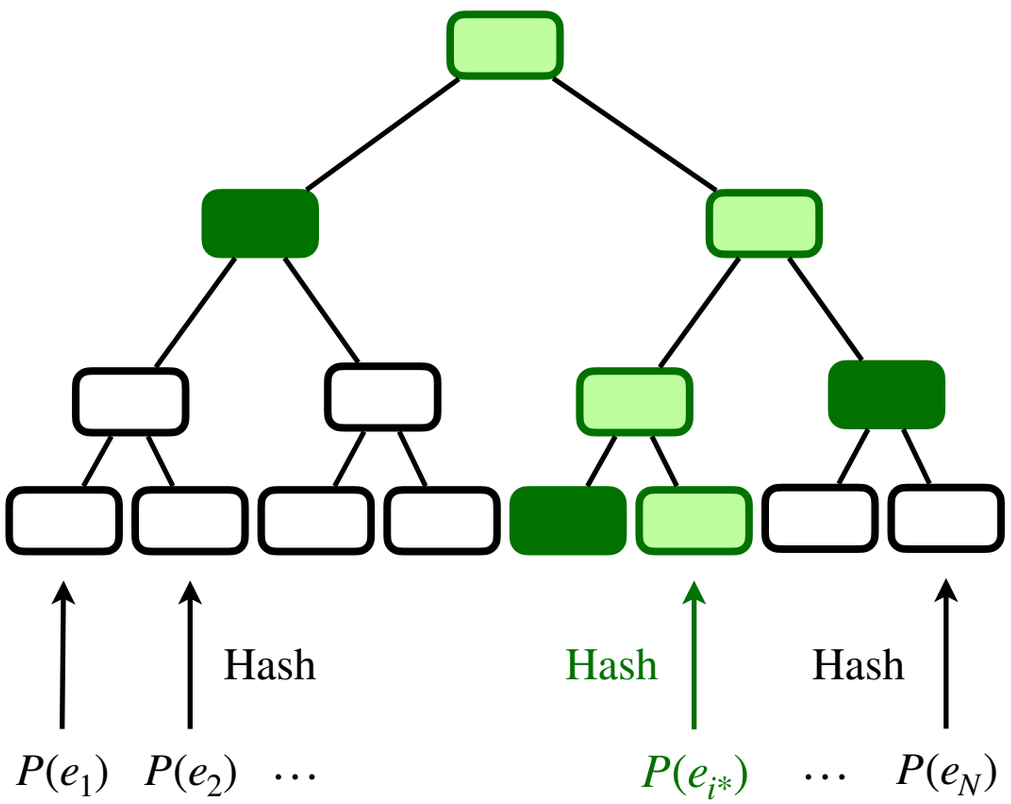
Properties:

- Cost of sending a tree node: 2λ bits
- Verification complexity: $O(\log_2 N)$
- Nodes contain non-sensitive information
- Commitment cost: $O_\lambda(\#P + \deg P)$
- Require a mechanism that provide some guarantee on the degree of the committed polynomial

Option 2: Using a Merkle tree (*ie.* a hash tree)

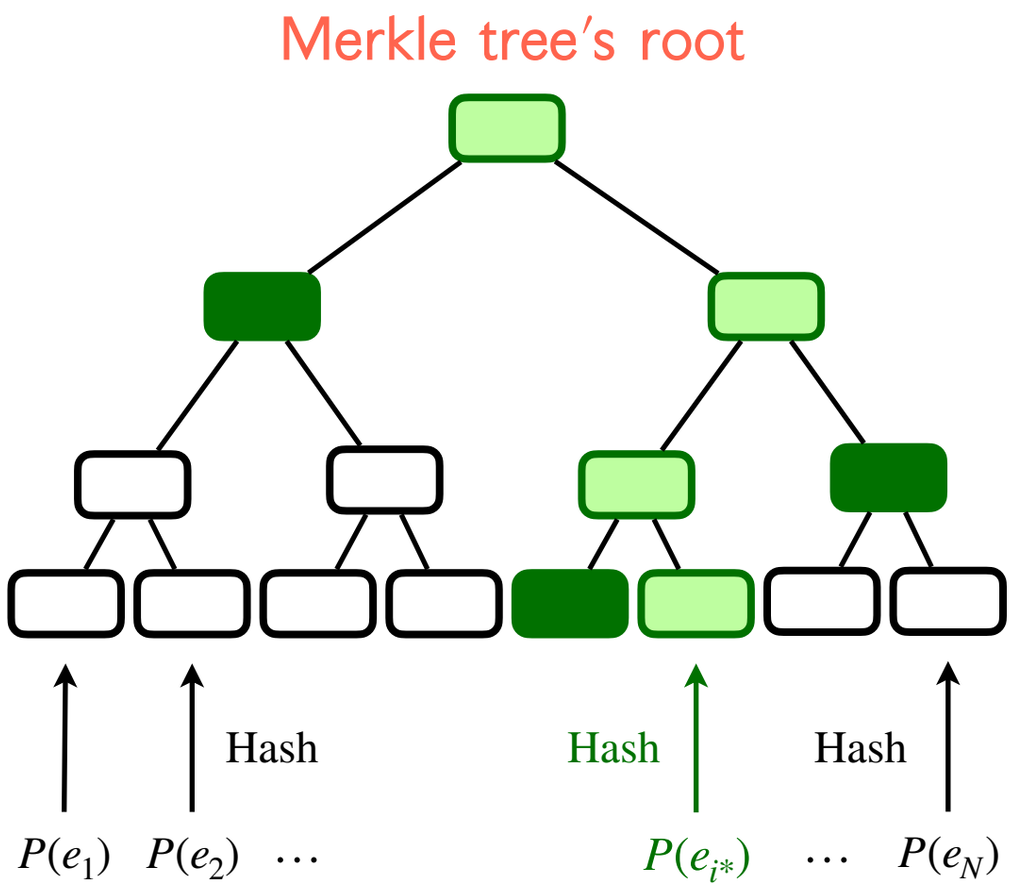
[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)

Using the degree-enforcing test



Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)



Using the degree-enforcing test

Commitment:

- *Reveal the Merkle root*
- *Reveal the degree-enforcing polynomial* R

Open $P(e_{i*})$*:*

Reveal the authentication path of $P(e_{i*})$

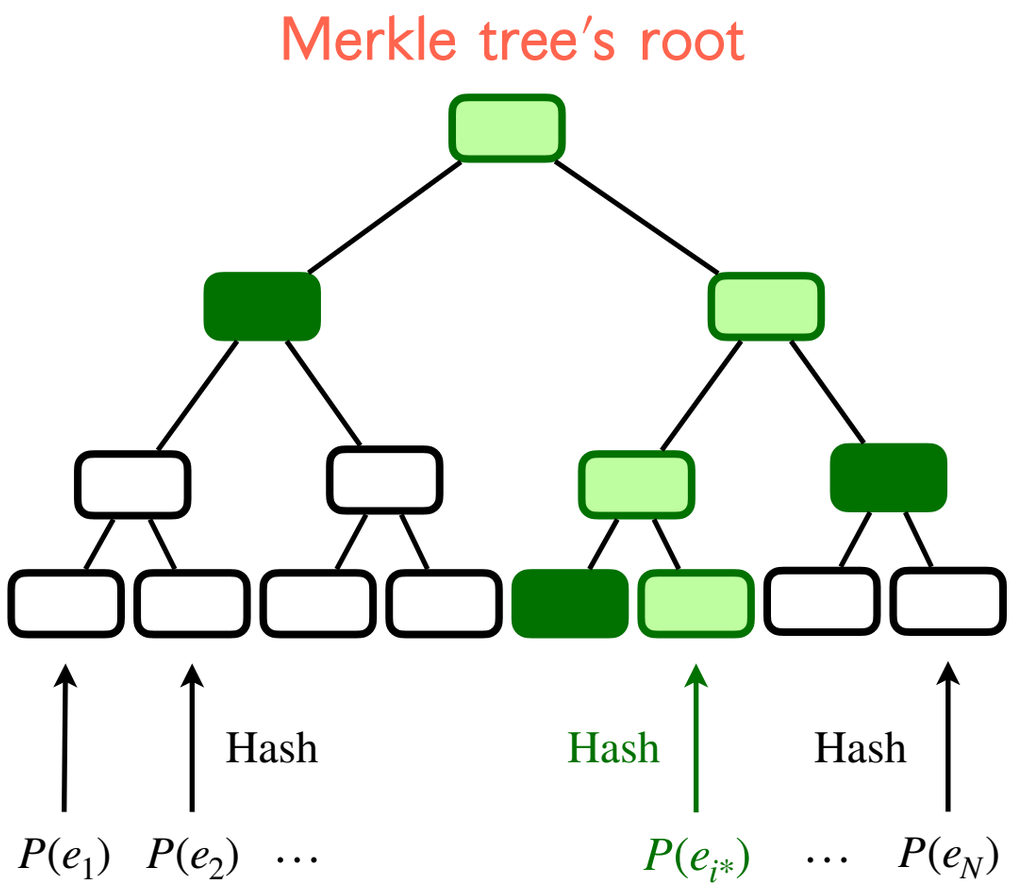
Build the degree-d polynomial R as

$$R := \sum_i \gamma_i \cdot P_i$$

where $\{\gamma_i\}_i$ *are random coefficients binded to the Merkle root.*

Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)



Build the degree-d polynomial R as

$$R := \sum_i \gamma_i \cdot P_i$$

where $\{\gamma_i\}_i$ are random coefficients binded to the Merkle root.

Using the degree-enforcing test

Commitment:

- *Reveal the Merkle root*
- *Reveal the degree-enforcing polynomial R*

Open $P(e_{i*})$:

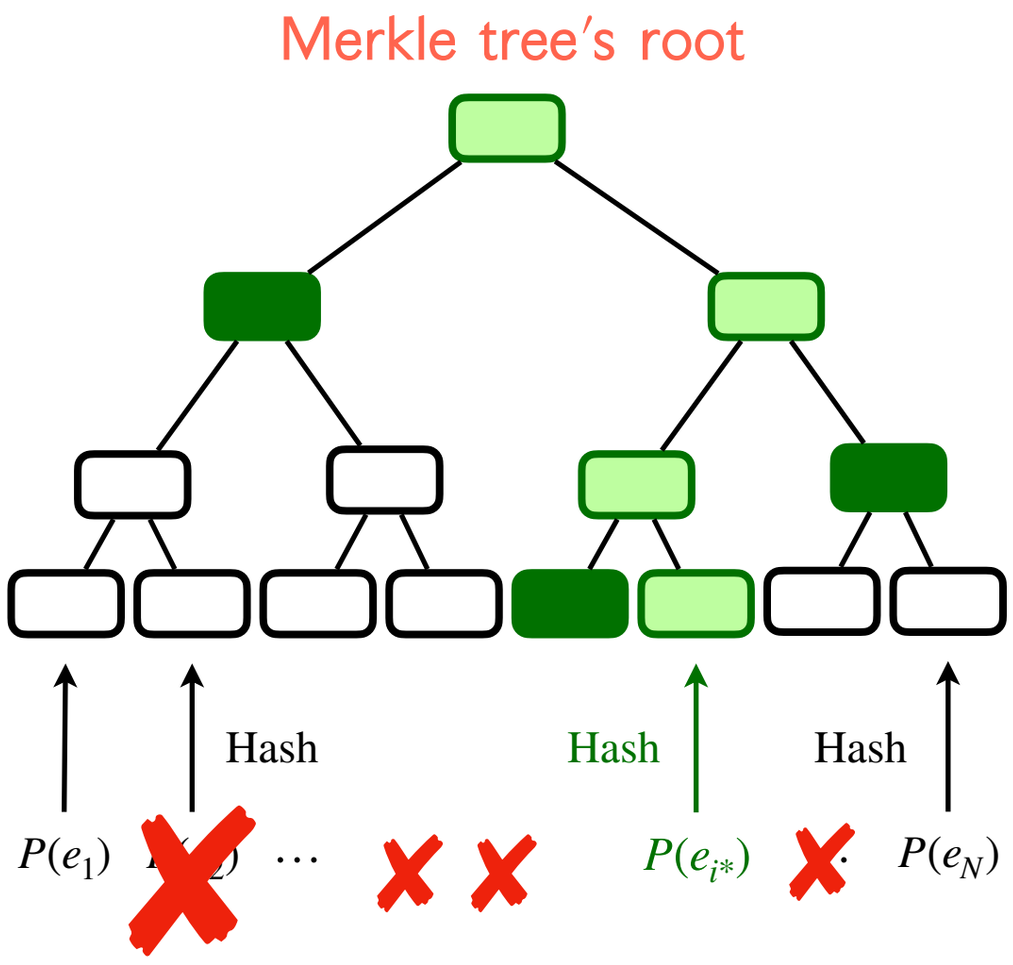
Reveal the authentication path of $P(e_{i*})$

Verification $P(e_{i*})$:

- *Check the authentication path*
- *Check that $R(e_{i*}) = \sum_i \gamma_i \cdot P_i(e_{i*})$*

Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)



Build the degree-d polynomial R as

$$R := \sum_i \gamma_i \cdot P_i$$

where $\{\gamma_i\}_i$ are random coefficients binded to the Merkle root.

Using the degree-enforcing test

Commitment:

- *Reveal the Merkle root*
- *Reveal the degree-enforcing polynomial R*

Open $P(e_{i*})$:

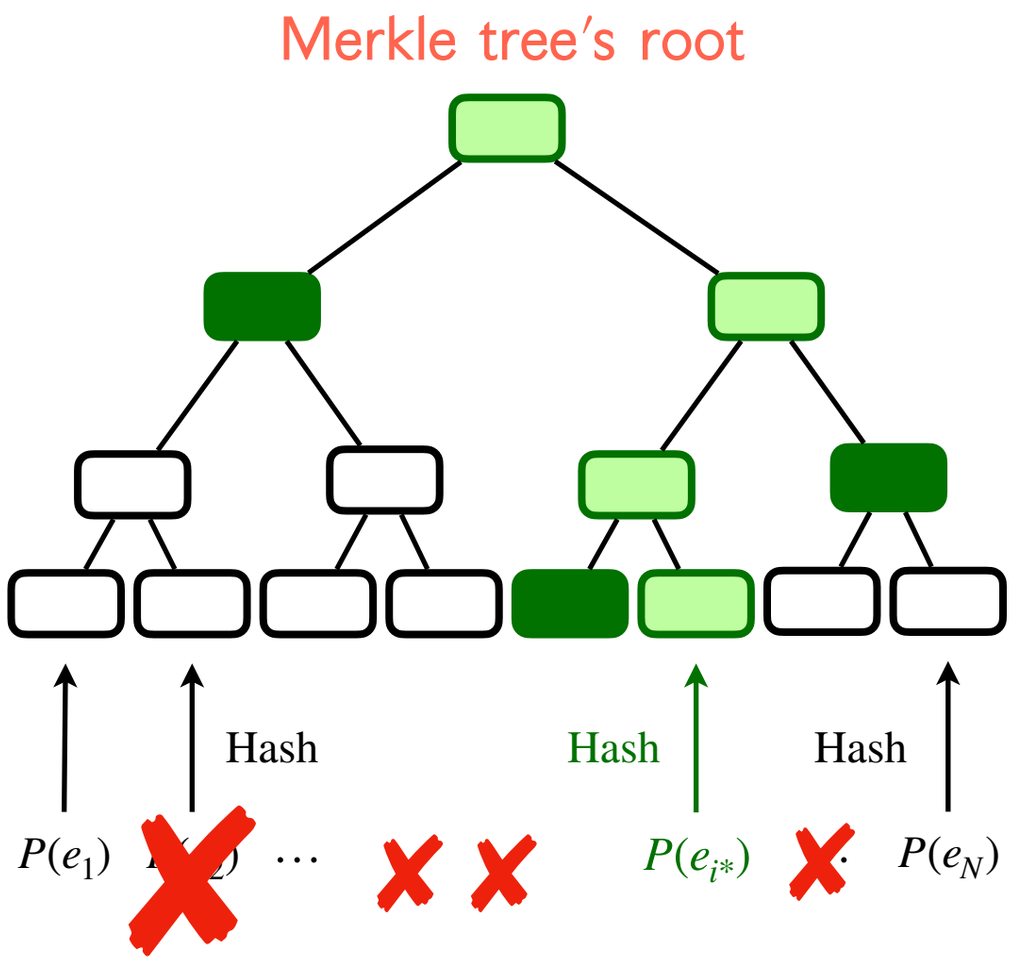
Reveal the authentication path of $P(e_{i*})$

Verification $P(e_{i*})$:

- *Check the authentication path*
- *Check that $R(e_{i*}) = \sum_i \gamma_i \cdot P_i(e_{i*})$*

Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)



Build the degree-d polynomial R as

$$R := \sum_i \gamma_i \cdot P_i$$

where $\{\gamma_i\}_i$ are random coefficients binded to the Merkle root.

Using the degree-enforcing test

Commitment:

- *Reveal the Merkle root*
- *Reveal the degree-enforcing polynomial R*

Open $P(e_{i*})$:

Reveal the authentication path of $P(e_{i*})$

Verification $P(e_{i*})$:

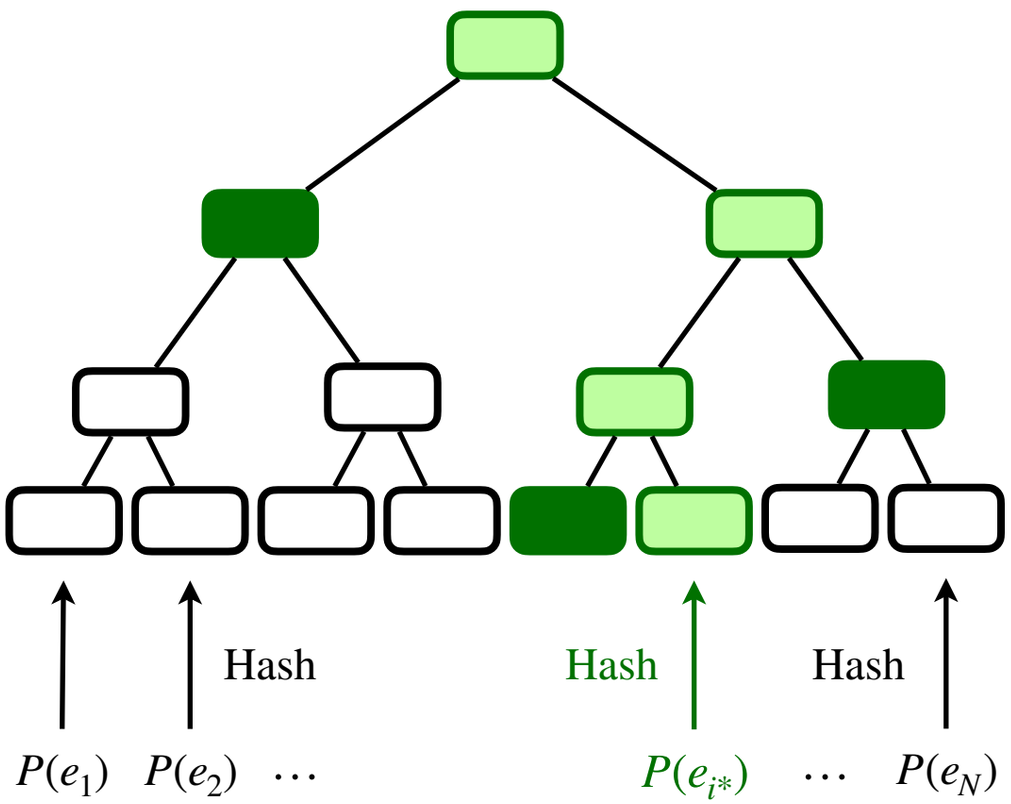
- *Check the authentication path*
- *Check that $R(e_{i*}) = \sum_i \gamma_i \cdot P_i(e_{i*})$*

Degree-Enforcing Property:
Except with negligible property, the only evaluations that can be still be opened are evaluations of a degree-d polynomial.

Option 2: Using a Merkle tree (*ie.* a hash tree)

[Mer79] Merkle: "Secrecy, authentication, and public key systems" (Ph.D. Thesis, 1979)

Using a proximity test



Proximity Tests:
Provide a guarantee that the committed polynomial is **somewhat close** to a polynomial of the right degree (**relaxed** property).

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.

Merkle Tree

④ FRI-based commitments

③ Merkle Trees with
Ligero-like Proximity Tests

② Degree-enforcing commitment
(TCitH-MT)

① VOLEitH / TCitH-GGM

GGM Tree

degree 10 000

degree 1000

degree 100

degree 10

degree 1

Comparison of the approaches

GGM Tree

- The nodes contain sensitive information.
- The complexity of the tree verification is in $O(N)$, where N is the number of leaves.
- A node is of λ bits

PCS from GGM Tree

- The committed is naturally of the right degrees.
- Commitment cost: $O_\lambda(\#P \cdot \deg P)$

Merkle Tree

- The nodes do not contain sensitive informations.
- The complexity of the tree verification is in $O(\log N)$, where N is the number of leaves.
- A node is of 2λ bits

PCS from Merkle Tree

- Need to add an additional mechanism to ensure the degree of the committed polynomials
- Commitment cost: $O_\lambda(\#P + \deg P)$

Minimizing the proof sizes

GGM Tree

- A node is of λ bits
- The committed is naturally of the right degrees.
- Commitment cost (PCS):

$$O_\lambda(\#P \cdot \deg P)$$

Merkle Tree

- A node is of 2λ bits
- Need to add an additional mechanism to ensure the degree of the committed polynomials
- Commitment cost (PCS):

$$O_\lambda(\#P + \deg P)$$

Minimizing the proof sizes

GGM Tree

- A node is of λ bits
- The committed is naturally of the right degrees.
- Commitment cost (PCS):

$$O_\lambda(\#P \cdot \deg P)$$

Merkle Tree

- A node is of 2λ bits
- Need to add an additional mechanism to ensure the degree of the committed polynomials
- Commitment cost (PCS):

$$O_\lambda(\#P + \deg P)$$

Scheme	*Using GGM Tree*	*Using Merkle Tree*
Signature schemes	2.5 - 6 KB	7 - 12 KB

Minimizing the proof sizes

GGM Tree

- A node is of λ bits
- The committed is naturally of the right degrees.
- Commitment cost (PCS):

$$O_\lambda(\#P \cdot \deg P)$$

Merkle Tree

- A node is of 2λ bits
- Need to add an additional mechanism to ensure the degree of the committed polynomials
- Commitment cost (PCS):

$$O_\lambda(\#P + \deg P)$$

Scheme	***Using GGM Tree***	***Using Merkle Tree***
Signature schemes	2.5 - 6 KB	7 - 12 KB
ZKPoK of Kyber512's secret key	≈ 12 KB	≈ 14 KB

Using VOLEitH *Using SmallWood*

Minimizing the proof sizes

GGM Tree

- A node is of λ bits
- The committed is naturally of the right degrees.
- Commitment cost (PCS):

$$O_\lambda(\#P \cdot \deg P)$$

Merkle Tree

- A node is of 2λ bits
- Need to add an additional mechanism to ensure the degree of the committed polynomials
- Commitment cost (PCS):

$$O_\lambda(\#P + \deg P)$$

Scheme	***Using GGM Tree***	***Using Merkle Tree***
Signature schemes	2.5 - 6 KB	7 - 12 KB
ZKPoK of Kyber512's secret key	≈ 12 KB	≈ 14 KB
ZKPoK of **four** Kyber512's secret keys	≈ 36 KB	≈ 21 KB
	Using VOLEitH	*Using SmallWood*

Minimizing the proof sizes

GGM Tree

- A node is of λ bits
- The committed is naturally of the right degrees.
- Commitment cost (PCS):

$$O_\lambda(\#P \cdot \deg P)$$

Merkle Tree

- A node is of 2λ bits
- Need to add an additional mechanism to ensure the degree of the committed polynomials
- Commitment cost (PCS):

$$O_\lambda(\#P + \deg P)$$

Scheme	***Using GGM Tree***	***Using Merkle Tree***
Signature schemes	2.5 - 6 KB	7 - 12 KB
ZKPoK of Kyber512's secret key	≈ 12 KB	≈ 14 KB
ZKPoK of **four** Kyber512's secret keys	≈ 36 KB	≈ 21 KB
ZKPoK of LWE (binary secret, $q \approx 2^{61}$, n=4096, m=1024)	≈ 102 KB	≈ 21 KB
	Using VOLEitH	*Using SmallWood*

Efficient Verification Algorithm

GGM Tree

- The complexity of the tree verification is in $O(N)$, where N is the number of leaves.

Merkle Tree

- The complexity of the tree verification is in $O(\log N)$, where N is the number of leaves.

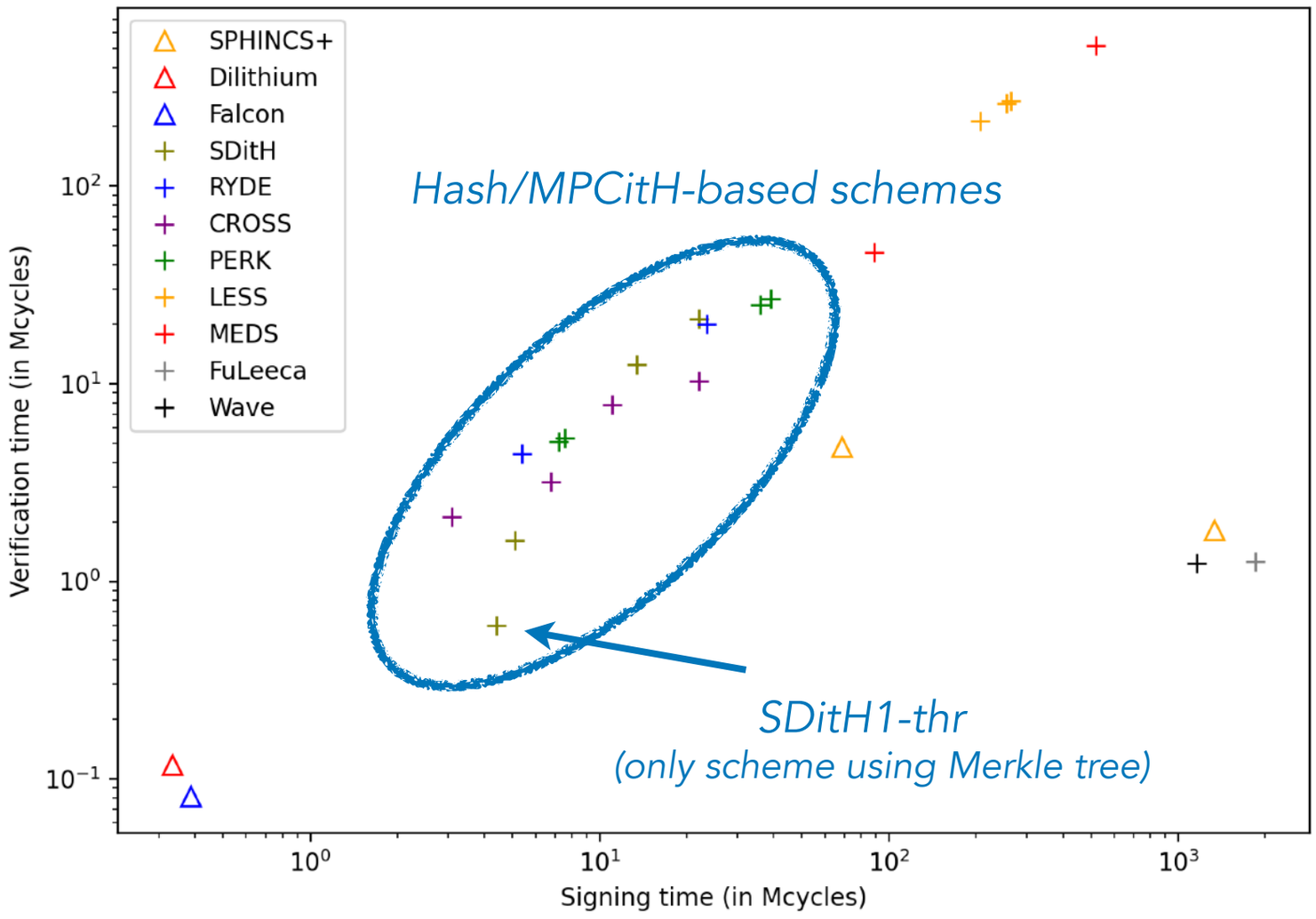
Efficient Verification Algorithm

GGM Tree

- The complexity of the tree verification is in $O(N)$, where N is the number of leaves.

Merkle Tree

- The complexity of the tree verification is in $O(\log N)$, where N is the number of leaves.



Round-1 code-based signature schemes

Efficient Verification Algorithm

GGM Tree

- The complexity of the tree verification is in $O(N)$, where N is the number of leaves.

Merkle Tree

- The complexity of the tree verification is in $O(\log N)$, where N is the number of leaves.

- Fast verification algorithm (for example, SDitH1-thr)

Efficient Verification Algorithm

GGM Tree

- The complexity of the tree verification is in $O(N)$, where N is the number of leaves.

Merkle Tree

- The complexity of the tree verification is in $O(\log N)$, where N is the number of leaves.

- Fast verification algorithm (for example, SDitH1-thr)
- The verification algorithm can be efficiently represented as an arithmetic circuit, *i.e.* leading to SNARK-friendly signatures.

For example,

[FR25] Feneuil, Rivain. CAPSS: A Framework for SNARK-Friendly Post-Quantum Signatures. ePrint 2025/061.

Signature Scheme	***Signature size***	***Nb R1CS Constraints***
VOLEitH-based signatures	2.5 - 5 KB	≥ 10 000 000
CAPSS-Anemoi (2^{256})	≈ 11 KB	≈ 19 000
CAPSS-RescuePrime (2^{256})	≈ 12 KB	≈ 36 000

Masking-Friendly Scheme

GGM Tree

- The nodes contain sensitive information.

Merkle Tree

- The nodes do not contain sensitive informations.

Masking-Friendly Scheme

GGM Tree	Merkle Tree
- The nodes contain sensitive information.	- The nodes do not contain sensitive informations.

- Since the nodes does not contain secret information, one does not need to mask Merkle trees, in a context where the secret values are *shared*.

For example, in the context of *side-channel attacks*:

> **[FRW25]** Feneuil, Rivain, Warmé-Janville. Masking-Friendly Post-Quantum Signatures in the Threshold-Computation-in-the-Head Framework. ePrint 2025/520.

Masking-Friendly Scheme

GGM Tree	Merkle Tree
- The nodes contain sensitive information.	- The nodes do not contain sensitive informations.

- Since the nodes does not contain secret information, one does not need to mask Merkle trees, in a context where the secret values are *shared*.

For example, in the context of *side-channel attacks*:

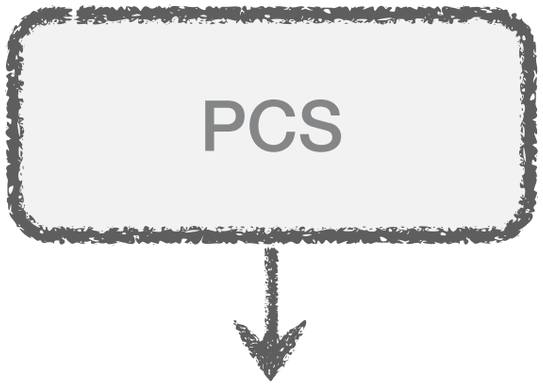
[FRW25] Feneuil, Rivain, Warmé-Janville. Masking-Friendly Post-Quantum Signatures in the Threshold-Computation-in-the-Head Framework. ePrint 2025/520.

For example, in the context of *threshold scheme*:

[FRVW25] Feneuil, Rivain, Vergnaud, Warmé-Janville. Threshold Signatures in the Head. ePrint 2026/1125.

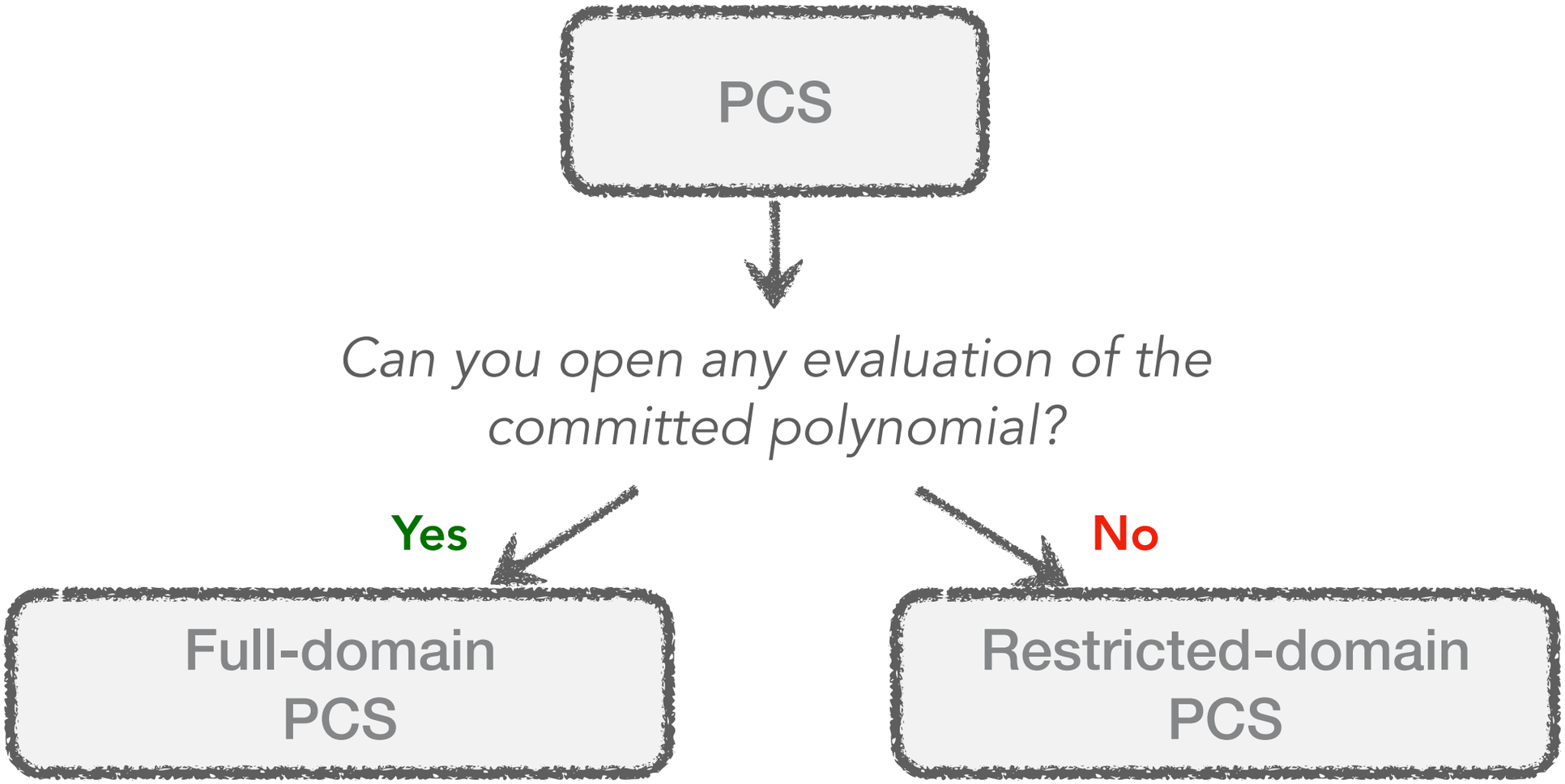
Extending to full-domain PCS

Polynomial Commitment Scheme

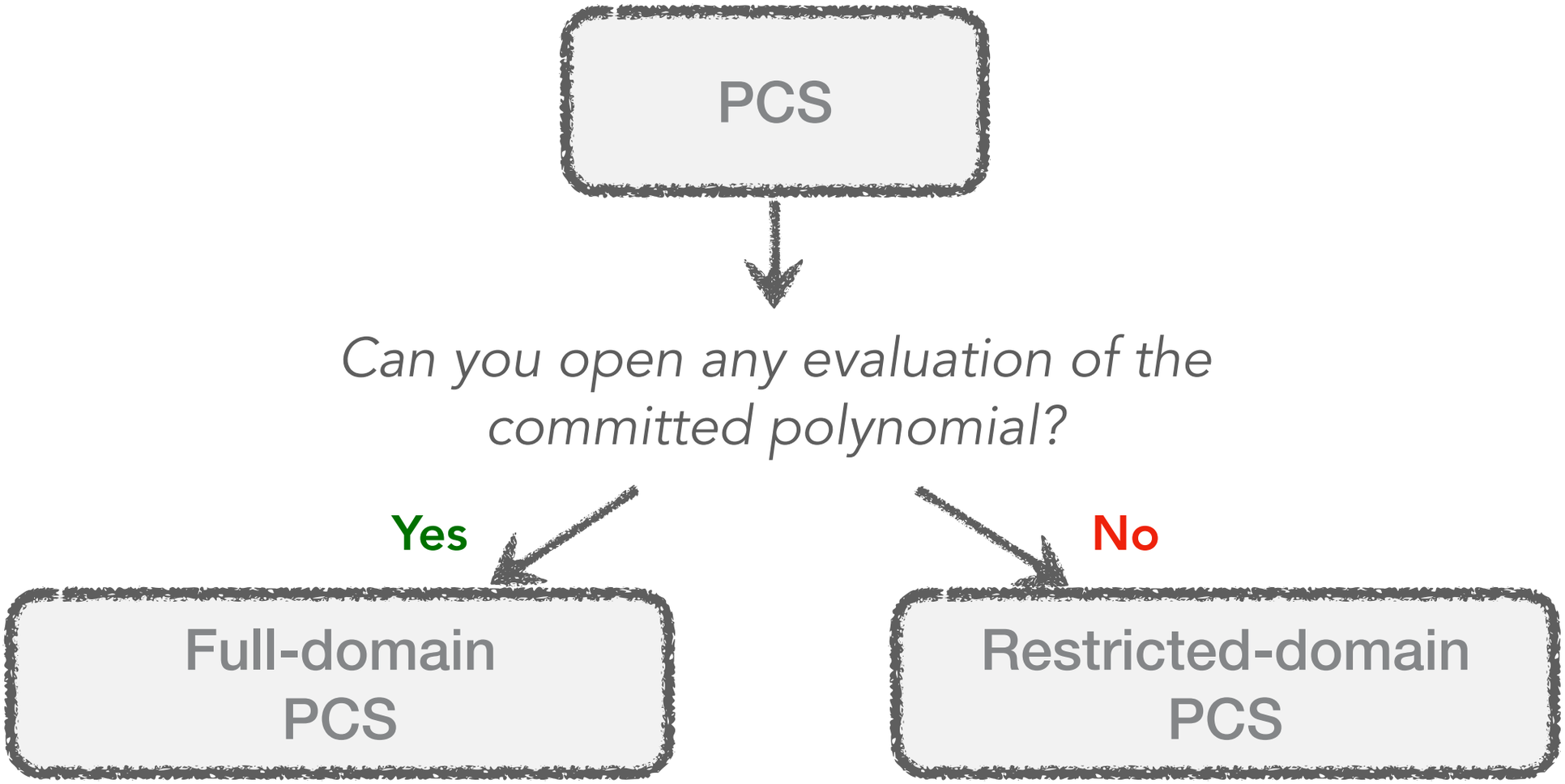


Can you open any evaluation of the committed polynomial?

Polynomial Commitment Scheme



Polynomial Commitment Scheme



For example, we commit to
$P(X) \in \mathbb{F}_q$, *with a large prime* q

We can open any evaluation $P(e)$ *for* $e \in \{0,\ldots,q-1\}$.

We can only open any evaluation $P(e)$ *for* $e \in \{0,\ldots,N-1\}$, *with* $N \ll q$.

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.

Merkle Tree

④ FRI-based commitments

③ Merkle Trees with
Ligero-like Proximity Tests

② Degree-enforcing commitment
(TCitH-MT)

① VOLEitH / TCitH-GGM

GGM Tree

degree 10 000

degree 1000

degree 100

degree 10

degree 1

*Natively, those techniques lead to **restricted-domain** polynomial commitment scheme*

Basic Proof System for Polynomial Constraints

I know $w_1, \ldots, w_n$ such that

$$\begin{cases} f_1(w_1, \ldots, w_n) & = 0 \\ & \vdots \\ f_m(w_1, \ldots, w_n) & = 0, \end{cases}$$

where $f_1, \ldots, f_m$ are public **degree-d polynomials**.

Prove it!

Prover

Verifier

Number of opened evaluations

$$\textbf{Soundness Error} = \frac{\binom{d \cdot \ell}{\ell}}{\binom{|\mathscr{C}|}{\ell}}$$

Probability that a malicious prover can convince the verifier.

Size of the challenge space that contains all the possible opened evaluations

Building a full-domain PCS from a restricted-domain PCS

- Restricted-domain PCS: $|\mathscr{C}| = N$

  where N is the size of the tree

- Full-domain PCS: $|\mathscr{C}| = |\mathbb{F}|$ or $|\mathscr{C}| = |\mathbb{K}|$

Building a full-domain PCS from a restricted-domain PCS

Out-of-sampling Technique

Rely on the equivalence:

$P(e) = z$ *iff there exists* $Q(X)$ *s.t.*

$$P(X) - z = (X - e) \cdot Q(X)$$

- Restricted-domain PCS: $|\mathscr{C}| = N$

  where N is the size of the tree

- Full-domain PCS: $|\mathscr{C}| = |\mathbb{F}|$ or $|\mathscr{C}| = |\mathbb{K}|$

[BGKS19] Ben-Sasson, Goldberg, Kopparty, Saraf. DEEP-FRI: Sampling outside the box improves soundness. ITCS 2020.

Building a full-domain PCS from a restricted-domain PCS

Out-of-sampling Technique

Rely on the equivalence:

$P(e) = z$ *iff there exists* $Q(X)$ *s.t.*

$$P(X) - z = (X - e) \cdot Q(X)$$

To commit to P

Commit to P *using the restricted-domain polynomial commitment scheme.*

To prove that $P(e) = z$

1. *Commit to* Q *using the restricted-domain polynomial commitment scheme.*
2. *Prove that* $P(X) - z = (X - e) \cdot Q(X)$*, by having the verifier check that the relation holds from randomly sampled points from* $\mathscr{C}$*.*

- Restricted-domain PCS: $|\mathscr{C}| = N$

  where N is the size of the tree

- Full-domain PCS: $|\mathscr{C}| = |\mathbb{F}|$ or $|\mathscr{C}| = |\mathbb{K}|$

[BGKS19] Ben-Sasson, Goldberg, Kopparty, Saraf. DEEP-FRI: Sampling outside the box improves soundness. ITCS 2020.

Building a full-domain PCS from a restricted-domain PCS

Out-of-sampling Technique

Using Tensor codes

Rely on the equivalence:

$P(e) = z$ *iff there exists* $Q(X)$ *s.t.*

$P(X) - z = (X - e) \cdot Q(X)$

- Restricted-domain PCS: $|\mathscr{C}| = N$

  where N is the size of the tree

- Full-domain PCS: $|\mathscr{C}| = |\mathbb{F}|$ or $|\mathscr{C}| = |\mathbb{K}|$

[BCG20] Bootle, Chiesa, Groth. Linear-time arguments with sublinear verification from tensor codes. TCC 2020.

[Lee21] Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. TCC 2021.

[GLS+23] Golovnev, Lee, Setty, Thalers, Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. Crypto 2023.

Building a full-domain PCS from a restricted-domain PCS

To commit to P

1. *Write* P *as* $P(X) := \sum_{i=0}^{d} p_i X^d$.
2. *Commit to the coefficients* $p_0, \ldots, p_d$.

To prove that $P(e) = z$

Using a restricted-domain polynomial commitment scheme, prove that

$$z = \sum_{i=0}^{d} p_i \cdot e^i,$$

which is a linear relation in the committed coefficients.

Using
Tensor codes

- Restricted-domain PCS: $|\mathscr{C}| = N$

  where N is the size of the tree

- Full-domain PCS: $|\mathscr{C}| = |\mathbb{F}|$ or $|\mathscr{C}| = |\mathbb{K}|$

[BCG20] Bootle, Chiesa, Groth. Linear-time arguments with sublinear verification from tensor codes. TCC 2020.

[Lee21] Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. TCC 2021.

[GLS+23] Golovnev, Lee, Setty, Thalers, Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. Crypto 2023.

Building a full-domain PCS from a restricted-domain PCS

Out-of-sampling Technique

Using Tensor codes

Rely on the equivalence:

$P(e) = z$ *iff* *there exists* $Q(X)$ *s.t.*

$P(X) - z = (X - e) \cdot Q(X)$

VOLE-in-the-Head Technique

Support only degree 1

Large-domain PCS

- Restricted-domain PCS: $|\mathscr{C}| = N$

  where N is the size of the tree

- Full-domain PCS: $|\mathscr{C}| = |\mathbb{F}|$ or $|\mathscr{C}| = |\mathbb{K}|$

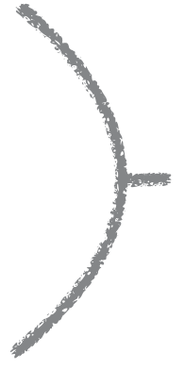
[BBD+23] Baum, Braun, Delpech, Klooß, Orsini, Roy, Scholl. Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures From VOLE-in-the-Head. Crypto 2023.

Building a full-domain PCS from a restricted-domain PCS

Commit to $P_1 := u \cdot X + v_1 \in \mathbb{F}[X]$
using a restricted-domain PCS

$\vdots$

Commit to $P_\tau := u \cdot X + v_\tau \in \mathbb{F}[X]$
using a restricted-domain PCS



Commit to
$P := u \cdot X + \Psi(v_1, \ldots, v_\tau) \in \mathbb{K}[X]$

where $\Psi : \mathbb{F}^\tau \to \mathbb{K}$, *with* $[\mathbb{K} : \mathbb{F}] = \tau$

VOLE-in-the-Head Technique

Support only degree 1
Large-domain PCS

- Restricted-domain PCS: $|\mathscr{C}| = N$
  where N is the size of the tree
- Full-domain PCS: $|\mathscr{C}| = |\mathbb{F}|$ or $|\mathscr{C}| = |\mathbb{K}|$

[BBD+23] Baum, Braun, Delpech, Klooß, Orsini, Roy, Scholl. Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures From VOLE-in-the-Head. Crypto 2023.

Building a full-domain PCS from a restricted-domain PCS

Out-of-sampling Technique

Using Tensor codes

VOLE-in-the-Head Technique

Midpoint Summary

Midpoint Summary

- Hash-based proof systems can
  - Either prove small statements
  - Or prove large statements
- The main difference in the design between those regimes is

how the polynomials are committed to

Midpoint Summary

- Hash-based proof systems can
  - Either prove small statements
  - Or prove large statements
- The main difference in the design between those regimes is

  how the polynomials are committed to

- Committing to polynomials using symmetric primitives can be done
  - Using ***GGM trees*** (a.k.a seed trees)
    - Shorter communication cost for very small polynomials
    - The committed polynomial is ensured to have the right degree

Midpoint Summary

- Hash-based proof systems can
  - Either prove small statements
  - Or prove large statements
- The main difference in the design between those regimes is

  how the polynomials are committed to

- Committing to polynomials using symmetric primitives can be done
  - Using ***GGM trees*** (a.k.a seed trees)
    - Shorter communication cost for very small polynomials
    - The committed polynomial is ensured to have the right degree
  - Using ***Merkle trees*** (a.k.a hash trees)
    - Asymptotically-better communication cost
    - Require a mechanism to ensure that the polynomial is of the right degree

Midpoint Summary

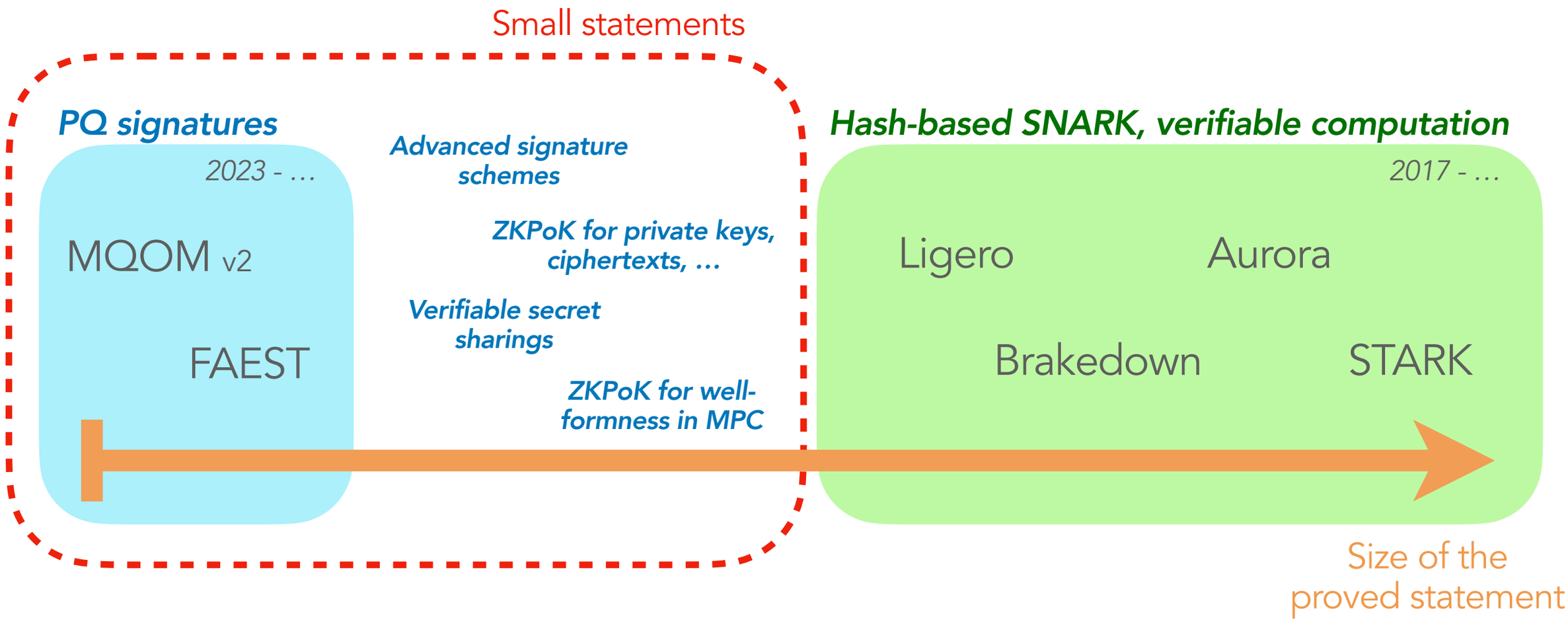
- Hash-based proof systems can
  - Either prove small statements
  - Or prove large statements
- The main difference in the design between those regimes is

  how the polynomials are committed to

- Committing to polynomials using symmetric primitives can be done
  - Using ***GGM trees*** (a.k.a seed trees)
    - Shorter communication cost for very small polynomials
    - The committed polynomial is ensured to have the right degree
  - Using ***Merkle trees*** (a.k.a hash trees)
    - Asymptotically-better communication cost
    - Require a mechanism to ensure that the polynomial is of the right degree
- While those techniques naturally leads to restricted-domain PCS, we can use them to build full-domain PCS.

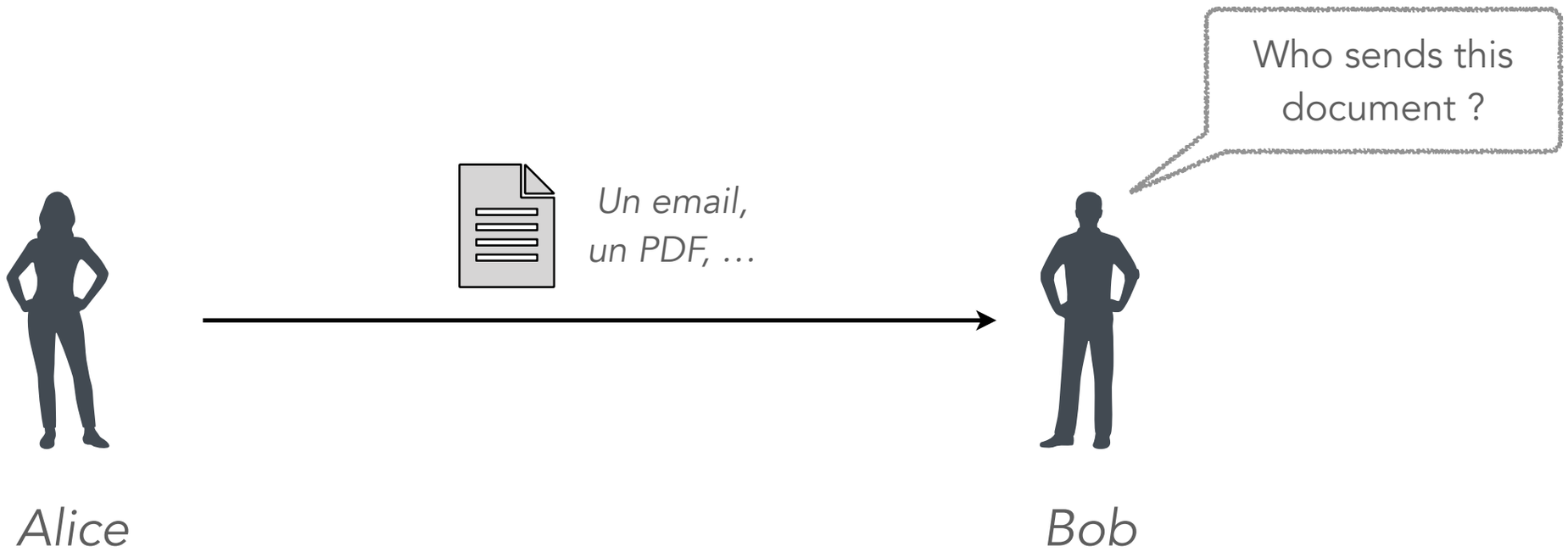
Building signatures

Blueprint of Hash-based Proof Systems



Zero-Knowledge is ***required***
Succinctness is ***optional***

Succinctness is ***required***
Zero-Knowledge is ***optional***

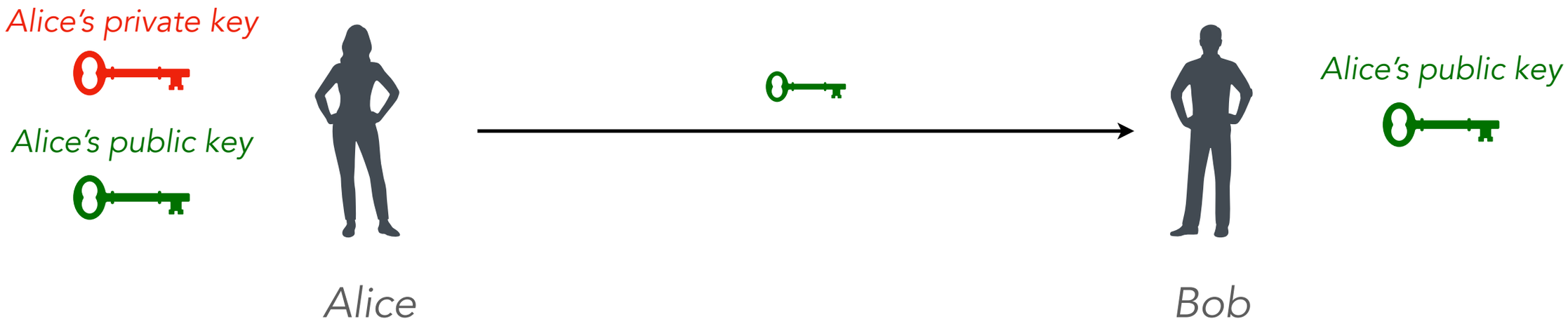

Who sends this document ?
Un email,
un PDF, …
Alice
Bob

Alice's private key

Alice's public key

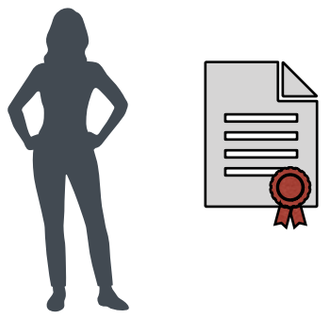
Alice

Bob


Alice's private key
Alice's public key
Alice's public key
Alice
Bob

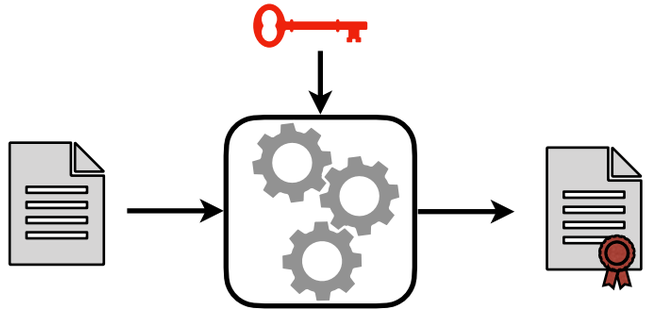
Alice's private key

Alice's public key



Alice

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

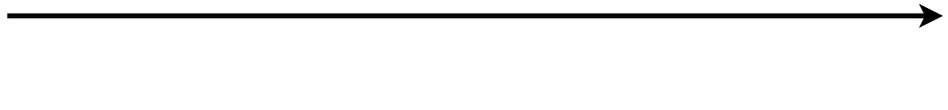


Alice's public key

Bob

Alice's private key

Alice's public key

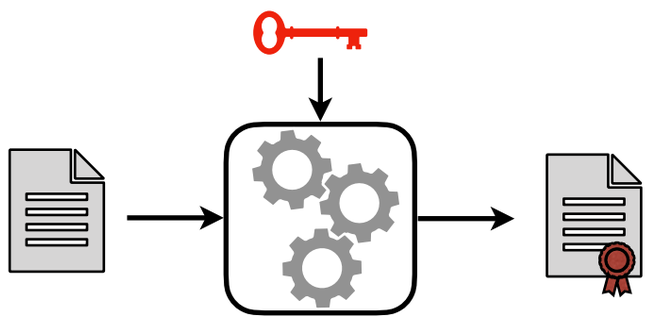


Alice's public key

Alice

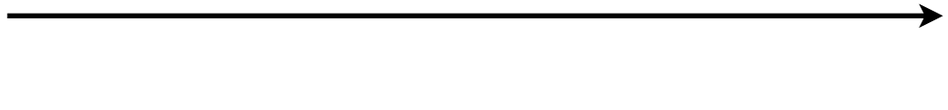
Bob

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



Alice's private key

Alice's public key



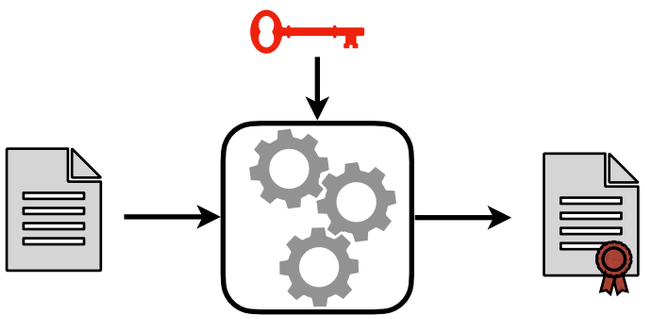
Alice's public key

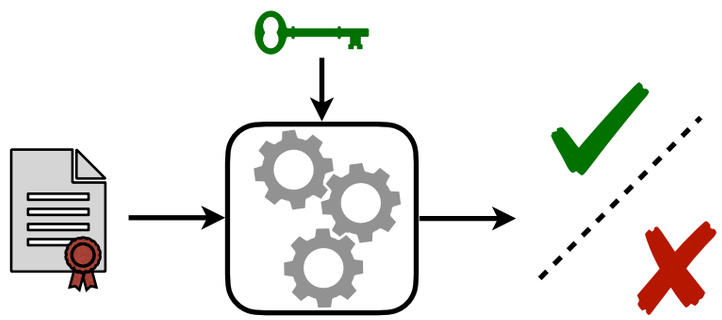
Alice

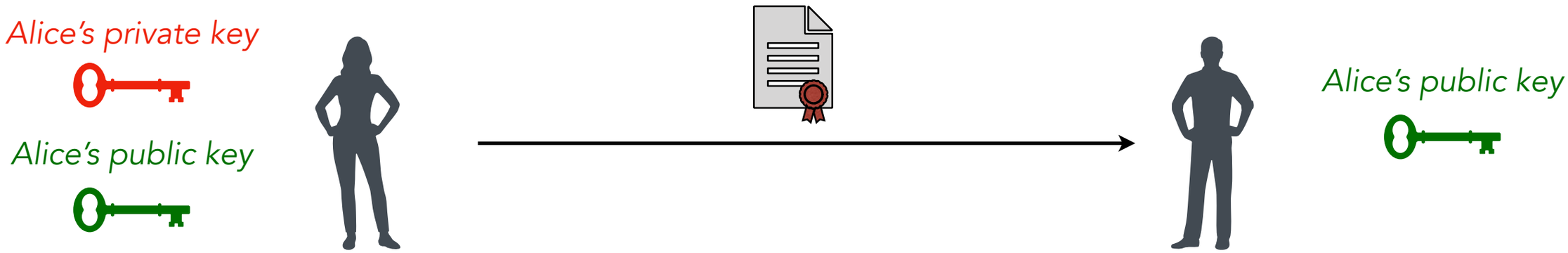
uses the private key
*to **sign** the digital document.*

Bob

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







Alice

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

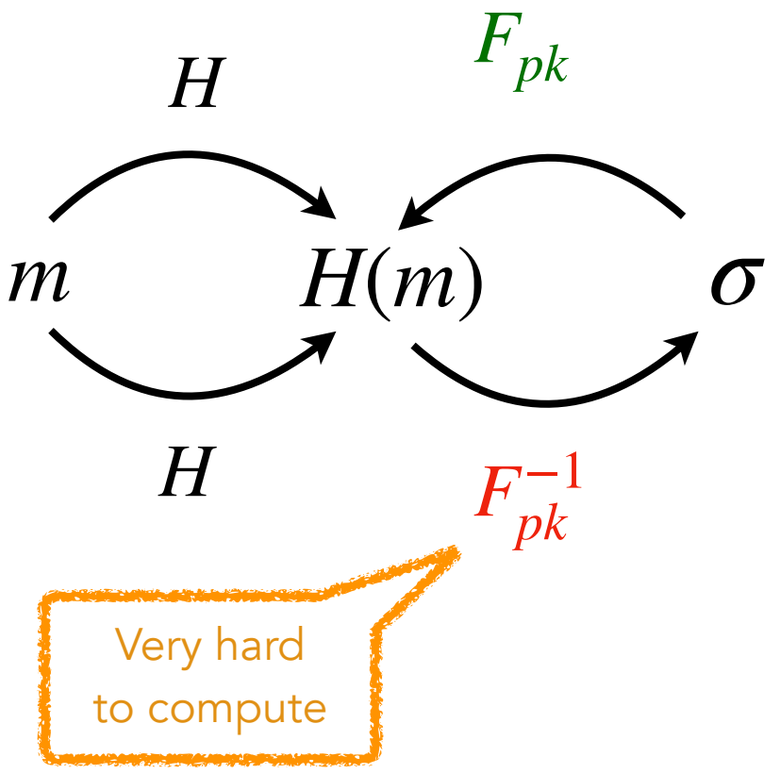
Bob

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

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

How to build signature schemes?

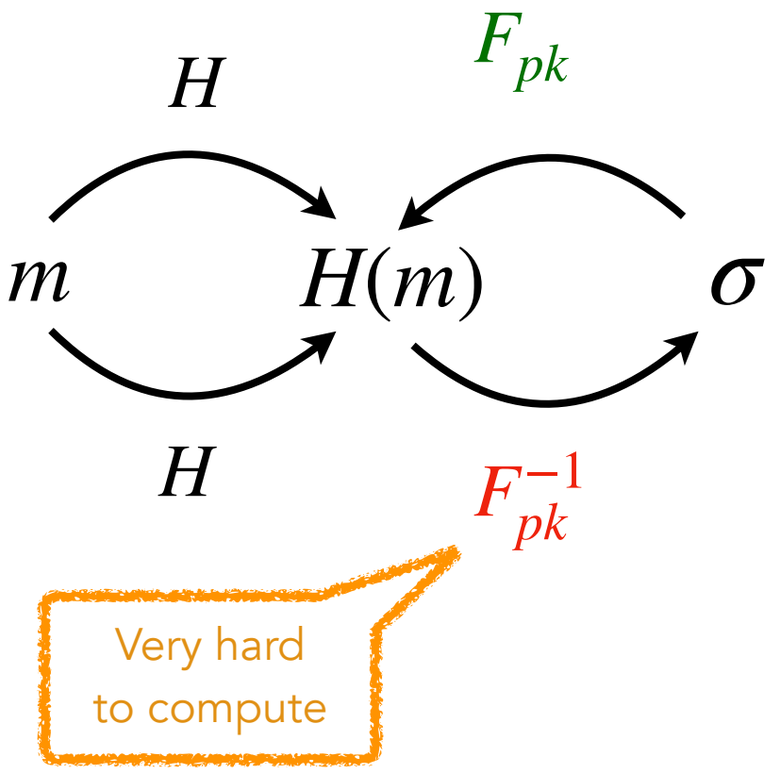
Hash & Sign



- Short signatures
- “Trapdoor” in the public key

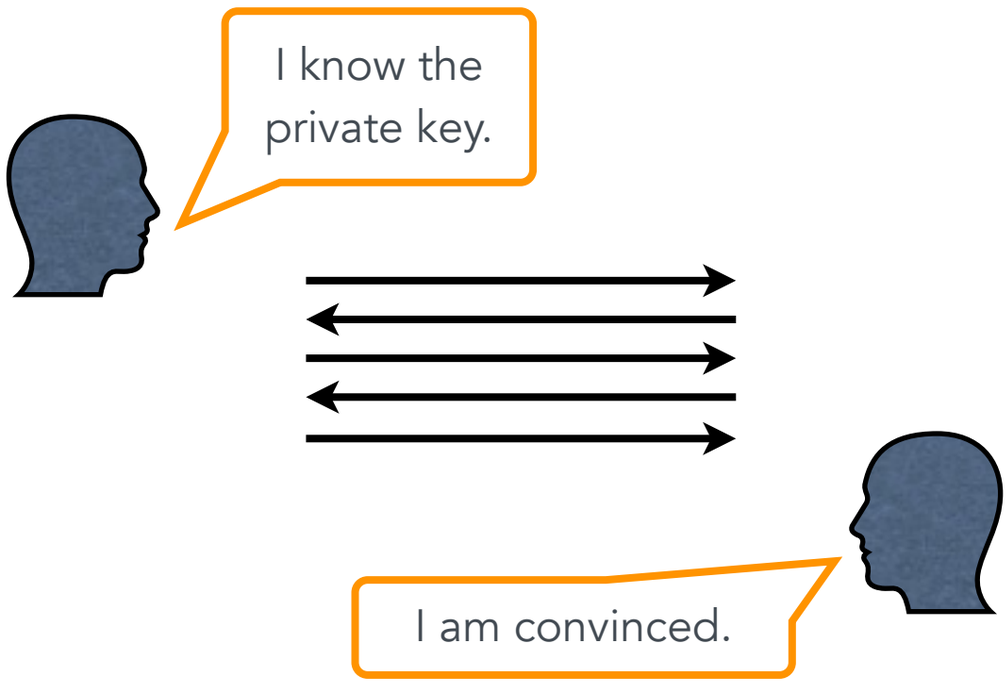
How to build signature schemes?

Hash & Sign



- Short signatures
- "Trapdoor" in the public key

From a zero-knowledge proof



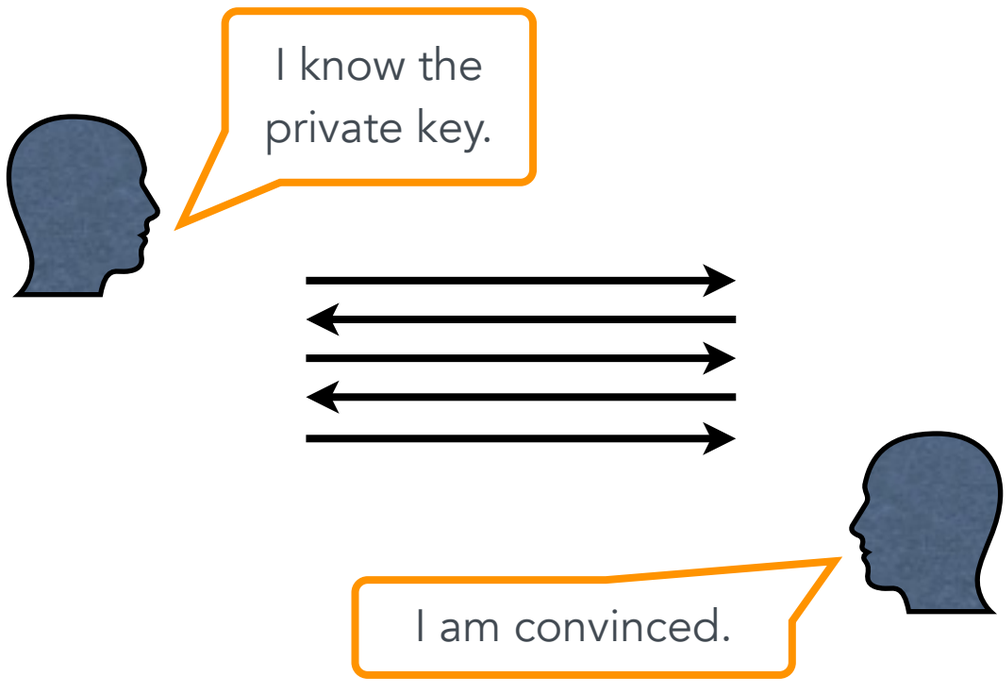
- Large(r) signatures
- Short public key

How to build signature schemes?

Hash & Sign

- Short signatures
- "Trapdoor" in the public key

From a zero-knowledge proof



- Large(r) signatures
- Short public key

How to build signature schemes?

Private key: solution of a degree-d multivariate system

How to build signature schemes?

Private key: solution of a degree-d multivariate system

PIOP
for polynomial constraints

Low-degree polynomials
Often degree-1 polynomials

How to build signature schemes?

Private key: solution of a degree-d multivariate system

PIOP
for polynomial constraints

Low-degree polynomials
Often degree-1 polynomials

PCS
from GGM trees

Soundness Error: $\frac{d}{N}$

with N the size of the tree

How to build signature schemes?

Private key: solution of a degree-d multivariate system

PIOP for polynomial constraints

Low-degree polynomials
Often degree-1 polynomials

PCS from GGM trees

Soundness Error: $\frac{d}{N}$

with N the size of the tree

Parallel repetitions

Soundness Error: $\left(\frac{d}{N}\right)^{\tau}$

Using VOLEitH technique

Soundness Error: $\frac{d}{N^{\tau}}$

How to build signature schemes?

Private key: solution of a degree-d multivariate system

PIOP for polynomial constraints

Low-degree polynomials
Often degree-1 polynomials

↓

PCS from GGM trees

Soundness Error: $\frac{d}{N}$

with N the size of the tree

↙ ↘

Parallel repetitions

Soundness Error: $\left(\frac{d}{N}\right)^{\tau}$

Using VOLEitH technique

Soundness Error: $\frac{d}{N^{\tau}}$

↘ ↙

Fiat-Shamir transformation

Arithmetization

Multivariate cryptography

Arithmetization

Multivariate cryptography

Public key: a multivariate quadratic (MQ) system $\mathcal{F}$

Private key: a solution of the MQ system

$x \in \mathbb{F}^n$ *such that* $\mathcal{F}(x) = 0$

Arithmetization

Multivariate cryptography

Public key: a multivariate quadratic (MQ) system $\mathscr{F}$

Private key: a solution of the MQ system

$$x \in \mathbb{F}^n \text{ such that } \mathscr{F}(x) = 0$$

👉 *We can directly apply the previous construction.*

Design of the MQOM scheme

Arithmetization

Rank-based cryptography

Arithmetization

Rank-based cryptography

Public key: a list of k matrices $M_1, \ldots, M_k \in \mathbb{F}^{m \times n}$

Private key: a vector $x \in \mathbb{F}^k$ such that

$$\sum_i x_i \cdot M_i \text{ is of rank at most } r$$

Not a polynomial constraint over the secret

Arithmetization

Rank-based cryptography

Public key: a list of k matrices $M_1, \ldots, M_k \in \mathbb{F}^{m \times n}$

Private key: a vector $x \in \mathbb{F}^k$ such that

$$\sum_i x_i \cdot M_i \text{ is of rank at most } r$$

Proved statement: there exists two matrices $L \in \mathbb{F}^{m \times r}$ and $R \in \mathbb{F}^{r \times n}$ such that

$$L \cdot R = \sum_i x_i \cdot M_i.$$

Degre-2 Polynomial Constraints

Design of the Mirath scheme

Arithmetization

Lattice-based cryptography

Public key: a matrix $A \in \mathbb{F}^{m \times n}$ *and a vector* $u \in \mathbb{F}^m$

Private key: a vector $x \in \mathbb{F}^k$ *such that*

$$u = Ax \quad \text{and} \quad \|x\|_\infty \leq \beta$$

Arithmetization

Lattice-based cryptography

Public key: a matrix $A \in \mathbb{F}^{m \times n}$ and a vector $u \in \mathbb{F}^m$

Private key: a vector $x \in \mathbb{F}^k$ such that

$$u = Ax \quad \text{and} \quad \|x\|_\infty \leq \beta$$

Proved statement: one can show that $u = Ax$ and for all i,

$$R(x_i) = 0$$

where $R := (X + \beta) \cdot (X + \beta - 1) \cdot \ldots \cdot (X - \beta)$.

Arithmetization

Code-based cryptography

Public key: a matrix $H \in \mathbb{F}^{(n-k)\times n}$ and a vector $y \in \mathbb{F}^{n-k}$

Private key: a vector $x \in \mathbb{F}^n$ such that

$$y = Hx \quad \text{and} \quad \mathrm{wt}_H(x) := \#\{i : x_i \neq 0\} \leq w$$

linear, easy to check

non-linear, hard to check

Arithmetization

Code-based cryptography

Public key: a matrix $H \in \mathbb{F}^{(n-k)\times n}$ and a vector $y \in \mathbb{F}^{n-k}$

Private key: a vector $x \in \mathbb{F}^n$ such that

$$y = Hx \quad \text{and} \quad \mathrm{wt}_H(x) := \#\{i : x_i \neq 0\} \leq w$$

Proved statement: x satisfies $y = Hx$ and

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

where

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

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

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

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

where

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

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

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

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

where

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

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

Then, one can deduce that

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

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

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

where

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

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

Then, one can deduce that

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

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

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

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

where

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

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

Then, one can deduce that

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

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

i.e.,

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

Arithmetization

Code-based cryptography

Public key: a matrix $H \in \mathbb{F}^{(n-k)\times n}$ and a vector $y \in \mathbb{F}^{n-k}$

Private key: a vector $x \in \mathbb{F}^n$ such that

$$y = Hx \quad \text{and} \quad \mathrm{wt}_H(x) := \#\{i : x_i \neq 0\} \leq w$$

Proved statement: x satisfies $y = Hx$ and

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

where

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

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

Design of the SDitH scheme (v1)

For the sake of diversity…

AES Key Recovery
AIM Key Recovery
LowMC Key Recovery
Rain Key Recovery

Anemoi Hash Preimage
Poseidon Hash Preimage
Griffin Hash Preimage
RescuePrime Hash Preimage

Legendre PRF
BHHG's PRF

LWE
SIS
Subset Sum

Discrete Logarithm
Integer Factorization
Double Discrete Logarithm

Syndrome Decoding
Regular Syndrome Decoding
Permuted Kernel
Linear Code Equivalence
Restricted Syndrome Decoding

MinRank
Rank Syndrome Decoding
Subfield Collision Problem
Matrix Subcode Equivalence

Multivariate Quadratic
PowAff2

For the sake of diversity…

AES Key Recovery
AIM Key Recovery
LowMC Key Recovery
Rain Key Recovery

Anemoi Hash Preimage
Poseidon Hash Preimage
Griffin Hash Preimage
RescuePrime Hash Preimage

LWE
SIS
Subset Sum

Syndrome Decoding
Regular Syndrome Decoding
Permuted Kernel
Linear Code Equivalence
Restricted Syndrome Decoding

MinRank
Rank Syndrome Decoding
Subfield Collision Problem
Matrix Subcode Equivalence

Multivariate Quadratic
PowAff2

Legendre PRF
BHHG's PRF

Discrete Logarithm
Integer Factorization
Double Discrete Logarithm

To use the PIOP-based frameworks,
one just needs to write those problems using polynomial constraints.

For the sake of diversity…

AES Key Recovery
AIM Key Recovery
LowMC Key Recovery
Rain Key Recovery

Anemoi Hash Preimage
Poseidon Hash Preimage
Griffin Hash Preimage
RescuePrime Hash Preimage

LWE
SIS
Subset Sum

Syndrome Decoding
Regular Syndrome Decoding
Permuted Kernel
Linear Code Equivalence
Restricted Syndrome Decoding

MinRank
Rank Syndrome Decoding
Subfield Collision Problem
Matrix Subcode Equivalence

Multivariate Quadratic
PowAff2

Legendre PRF
BHHG's PRF

Discrete Logarithm
Integer Factorization
Double Discrete Logarithm

We can build highly conservative schemes!

For the sake of diversity…

AES Key Recovery
AIM Key Recovery
LowMC Key Recovery
Rain Key Recovery

Anemoi Hash Preimage
Poseidon Hash Preimage
Griffin Hash Preimage
RescuePrime Hash Preimage

LWE
SIS
Subset Sum

Syndrome Decoding
Regular Syndrome Decoding
Permuted Kernel
Linear Code Equivalence
Restricted Syndrome Decoding

MinRank
Rank Syndrome Decoding
Subfield Collision Problem
Matrix Subcode Equivalence

Multivariate Quadratic
PowAff2

Legendre PRF
BHHG's PRF

Discrete Logarithm
Integer Factorization
Double Discrete Logarithm

We can build highly conservative schemes!

NIST Candidates

	NIST Submission		
Security Assumptions	***Candidate Name***	***Sig. Size***	***PK Size***
AES Block cipher	FAEST v2	3.9-4.5 KB	32 B
MinRank	Mirath	3.0-3.2 KB	57-73 B
Multivariate Quadratic	MQOM v2	2.8-3.2 KB	52-80 B
Permuted Kernel	PERK v2.1	3.5 KB	100 B
Rank Syndrome Decoding	RYDE v2	3.1 KB	69 B
Syndrome Decoding	SDitH v2	3.7 KB	70 B

Using seed trees of around 2048 leaves

NIST Candidates

	NIST Submission		
Security Assumptions	***Candidate Name***	***Sig. Size***	***PK Size***
AES Block cipher	FAEST v2	3.9-4.5 KB	32 B
MinRank	Mirath	3.0-3.2 KB	57-73 B
Multivariate Quadratic	MQOM v2	2.8-3.2 KB	52-80 B
Permuted Kernel	PERK v2.1	3.5 KB	100 B
Rank Syndrome Decoding	RYDE v2	3.1 KB	69 B
Syndrome Decoding	SDitH v2	3.7 KB	70 B

Using seed trees of around 2048 leaves

Due to historical reasons, this design approach is refered as a MPC-in-the-Head (MPCitH) framework.

MPC = Multi-Party Computation

NIST Candidates

	NIST Submission		
Security Assumptions	***Candidate Name***	***Sig. Size***	***PK Size***
AES Block cipher	FAEST v2	3.9-4.5 KB	32 B
MinRank	Mirath	3.0-3.2 KB	57-73 B
Multivariate Quadratic	MQOM v2	2.8-3.2 KB	52-80 B
Permuted Kernel	PERK v2.1	3.5 KB	100 B
Rank Syndrome Decoding	RYDE v2	3.1 KB	69 B
Syndrome Decoding	SDitH v2	3.7 KB	70 B

Using seed trees of around 2048 leaves

May 14th, 2026

*Announcement of the **3rd Round Onramp Candidates***

Comparison with the other families

Lattice-based schemes — *UOV-like schemes*

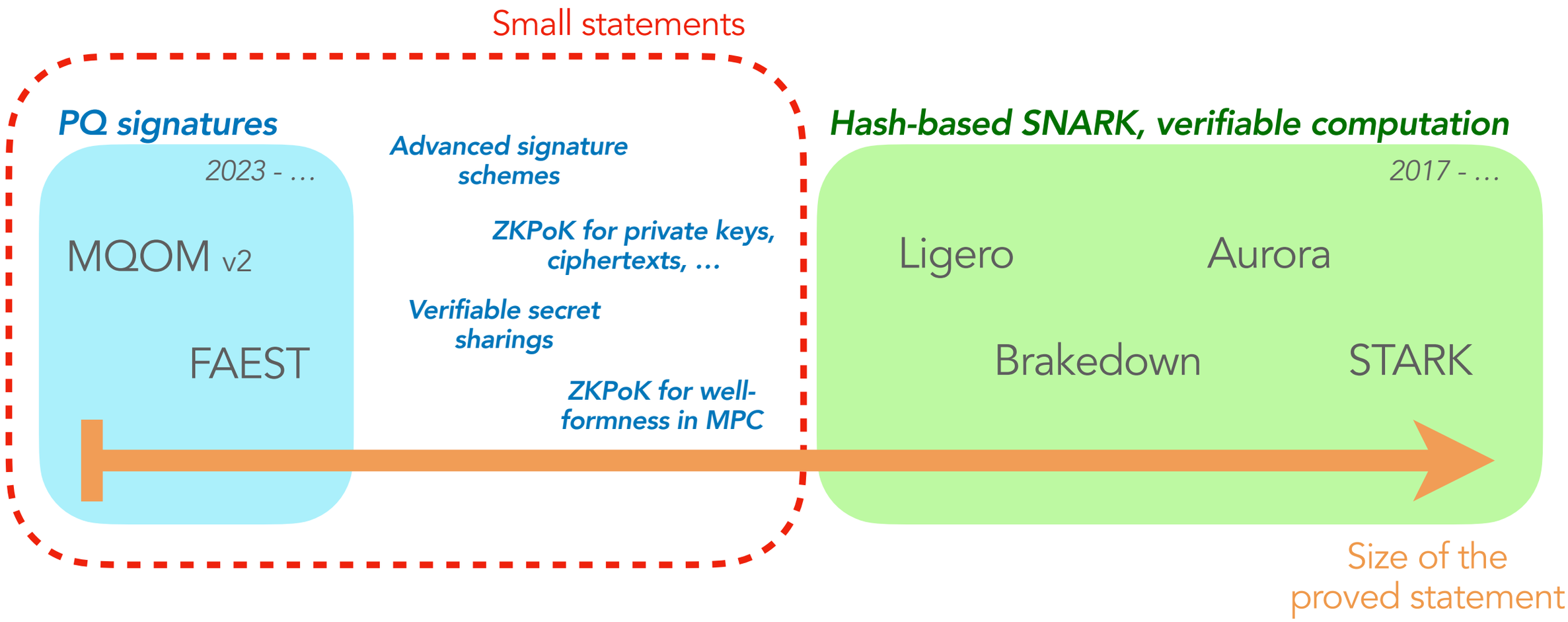
	MPCitH	***Dilithium ML-DSA***	***Falcon FN-DSA***	***SPHINCS⁺ SLH-DSA***	***UOV***	***Mayo***	***SQIsign***
Type	FS	FS	H&S	Hash-based	H&S	H&S	FS
|Sig|	2.5-4.5	2.4	0.7	7.8-17	0.1	0.2-0.5	0.1
|PK|	< 0.2	1.3	0.9	< 0.1	44-67	1.4-4.9	0.1
|Sig|+|PK|	2.5-4.6	3.7	1.6	7.9-17	44-67	1.9-5.1	0.2
Sign. Time	~	++	++	--	~	~	-
Verif. Time	~	++	++	~	++	++	~
Security	AES Unstructured SD Unstructured MQ …	Structured Lattice	Structured Lattice	Hash	UOV Trapdoor	New UOV-like Trapdoor	Isogeny

Sizes in kilobytes (KB)

FS: Fiat-Shamir transformation
H&S: Hash-and-sign scheme

Building Advanced Signatures

Blueprint of Hash-based Proof Systems



Zero-Knowledge is ***required***
Succinctness is ***optional***

Succinctness is ***required***
Zero-Knowledge is ***optional***

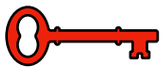
Building Advanced Signatures

In many privacy-sensitive applications, standard digital signatures may not be sufficient.

- What if the signer wishes to remain **anonymous**?
- What if the party signing the message is **distinct from the owner** of the signing key?
- What if the signing key is **shared** among multiple entities to eliminate a single point of failure?
- What if the user wishes to **reveal only part** of the signed message?
- What if signatures need to be issued **without revealing the message** to the signer?
- …

Ring Signature Schemes

The User



Signer's private key

Message to sign

The Verifier

Signer's public key

Ring Signature Schemes

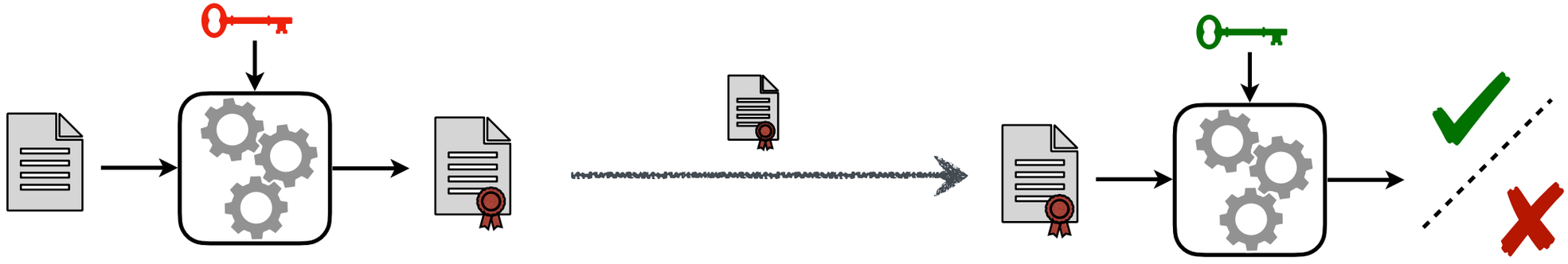
The User

Signer's private key

Message to sign

The Verifier

Signer's public key



Ring Signature Schemes

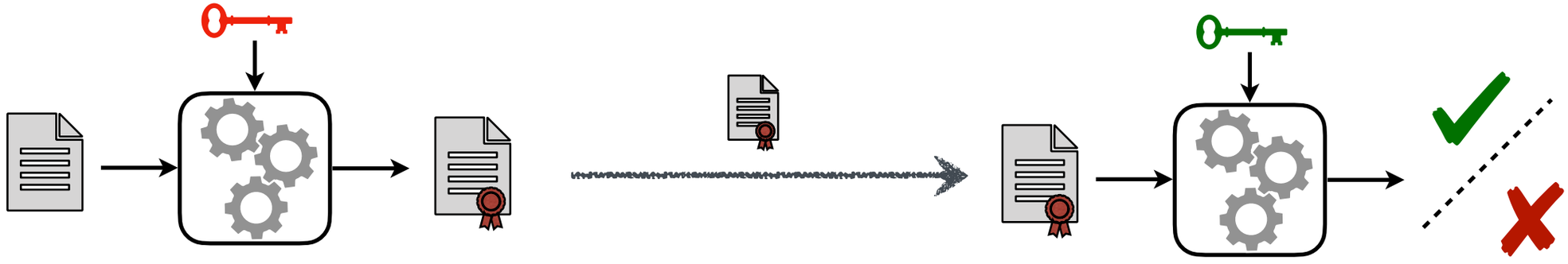
The User

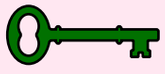
Signer's private key

Message to sign

The Verifier

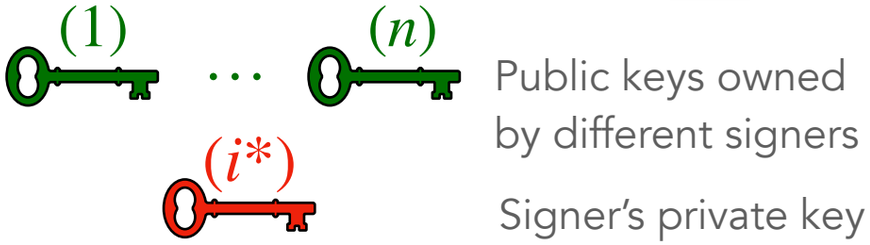
Signer's public key



The verifier knows that the owner of the secret key that corresponds to  has signed the message.

Ring Signature Schemes

The User



Public keys owned by different signers

Signer's private key

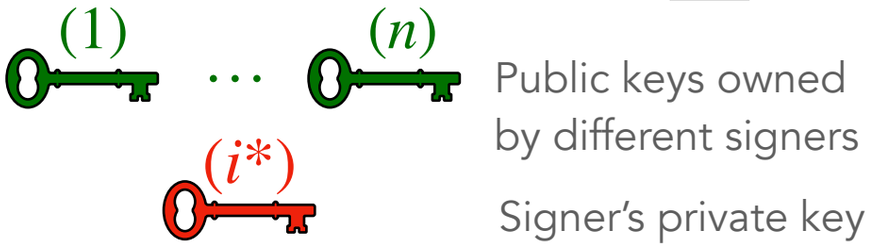
Message to sign

The Verifier

Public keys owned by different signers

Ring Signature Schemes

The User



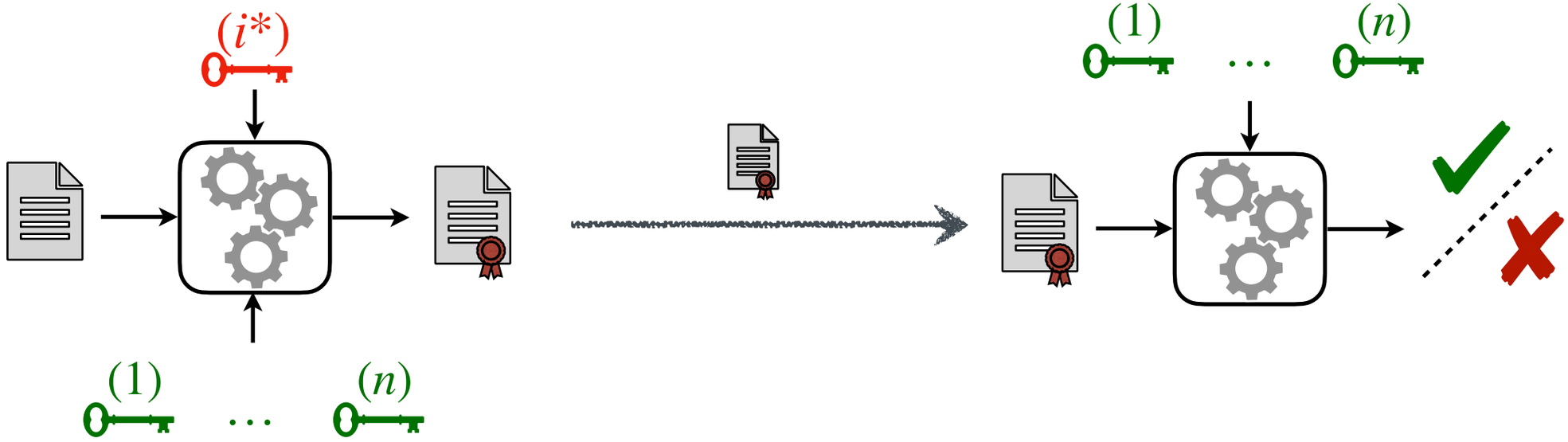
Public keys owned by different signers

Signer's private key

Message to sign

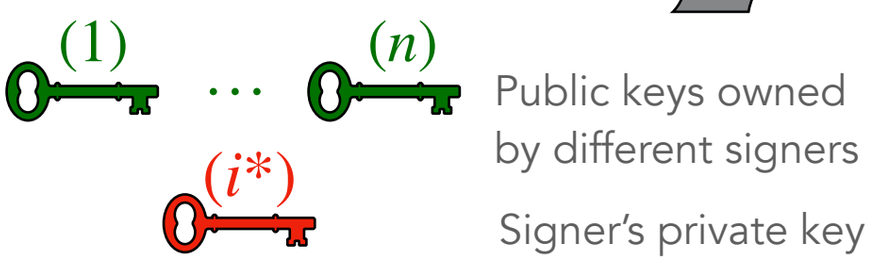
The Verifier

Public keys owned by different signers



Ring Signature Schemes

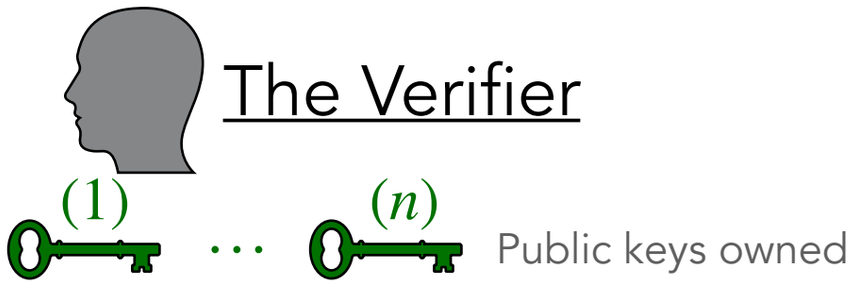
The User



Public keys owned by different signers

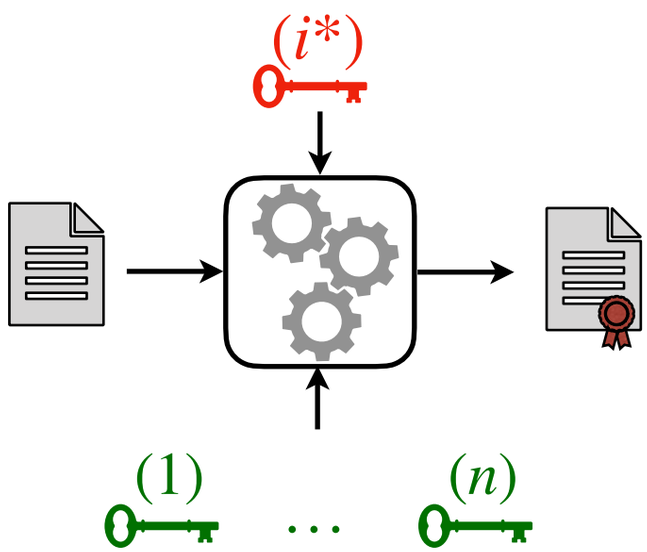
Signer's private key

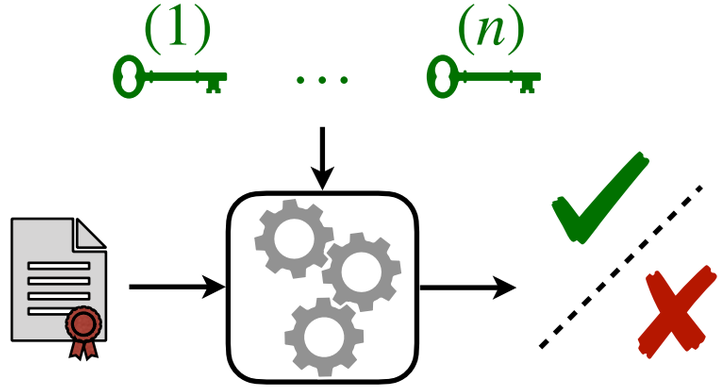
The Verifier



Public keys owned by different signers

Message to sign



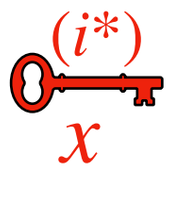


The verifier cannot determine which secret-key holder signed the message.

Ring Signature Schemes

Public keys owned by different signers

MQ systems $\mathcal{F}_1$ $\mathcal{F}_n$

Signer's private key 

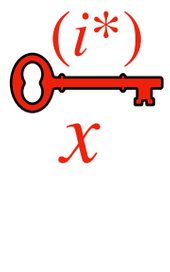
s.t. $\mathcal{F}_{i^*}(x) = 0$

Ring Signature Schemes

Public keys owned by different signers

MQ systems $\mathcal{F}_1$ $\mathcal{F}_n$

Signer's private key



s.t. $\mathcal{F}_{i^*}(x) = 0$

I know x *and a selector* $s \in \mathbb{F}^n$ *such that*

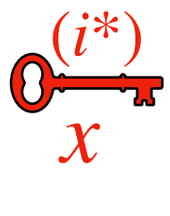
$$\sum_{i=1}^{n} s_i \cdot \mathcal{F}_i(x) = 0$$

where s is a vector with only *one non-zero coordinate*.

Ring Signature Schemes

Public keys owned by different signers

MQ systems $\mathcal{F}_1$ $\mathcal{F}_n$

Signer's private key  s.t. $\mathcal{F}_{i^*}(x) = 0$

I know x *and a selector* $s \in \mathbb{F}^n$ *such that*

$$\sum_{i=1}^{n} s_i \cdot \mathcal{F}_i(x) = 0$$

$$\begin{cases} \mathcal{F}_{i^*}(x) = 0 \\ \mathcal{F}_{i \neq i^*}(x) \neq 0 \end{cases}$$

where s is a vector with only *one non-zero coordinate.*

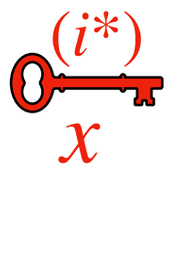
$$s_i := \begin{cases} 1 & \text{if } i = i^* \\ 0 & \text{otherwise.} \end{cases}$$

Ring Signature Schemes

Public keys owned by different signers

MQ systems $\mathcal{F}_1$ $\mathcal{F}_n$

Signer's private key



s.t. $\mathcal{F}_{i^*}(x) = 0$

I know x *and a selector* $s \in \mathbb{F}^n$ *such that*

$$\sum_{i=1}^{n} s_i \cdot \mathcal{F}_i(x) = 0$$

where s is a vector with only *one non-zero coordinate :*

- *We can prove its has at most one non-zero coordinate;*
- *By showing* $s_1 + \ldots + s_n = 1$*, we prove that it is not the zero vector.*

Ring Signature Schemes

Public keys owned by different signers

MQ systems $\mathcal{F}_1$ $\mathcal{F}_n$

Signer's private key x s.t. $\mathcal{F}_{i^*}(x) = 0$

I know x *and a selector* $s \in \mathbb{F}^n$ *such that*

$$\sum_{i=1}^{n} s_i \cdot \mathcal{F}_i(x) = 0$$

Degree-3 Polynomial Constraint

where s is a vector with only *one non-zero coordinate* :

Degree-2 Polynomial Constraint

- *We can prove its has at most one non-zero coordinate;*
- *By showing* $s_1 + \ldots + s_n = 1$ *to prove that it is not the zero vector.*

Linear Constraint

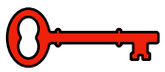
Ring Signature Schemes

	Size of the ring (i.e. nb of used public keys)		
Security Assumptions	***2^3***	***2^{10}***	***2^{20}***
Multivariate Quadratic	≈ 3.0 KB	≈ 3.2 KB	≈ 4.0 KB
Syndrome Decoding	≈ 4.0 KB	≈ 4.2 KB	≈ 5.0 KB
AES Block cipher	≈ 4.0 KB	≈ 4.2 KB	≈ 5.0 KB
Learning With Errors	≈ 7.0 KB	≈ 8.0 KB	≈ 12.0 KB

Estimated signature sizes

Blind Signature Schemes

The Signer



Signer's private key

The User

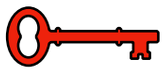
Message to sign

The Verifier

Signer's public key

Blind Signature Schemes

The Signer



Signer's private key

The User

Message to sign

The Verifier

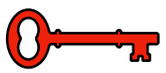
Signer's public key

Commitment of the message

① The User **commits** the message to sign.

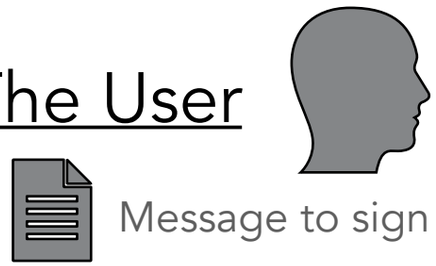
Blind Signature Schemes

The Signer



Signer's private key

The User



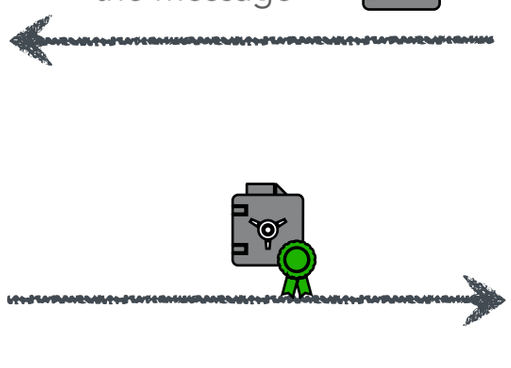
Message to sign

The Verifier

Signer's public key

① The User **commits** the message to sign.

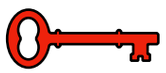
Commitment of the message



② Produce signing material using the private key

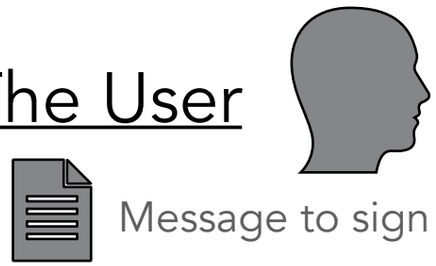
Blind Signature Schemes

The Signer



Signer's private key

The User



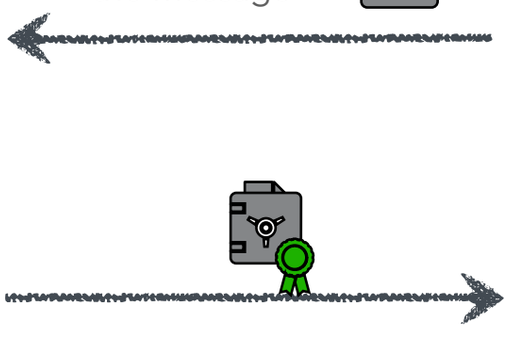
Message to sign

The Verifier

Signer's public key

① The User **commits** the message to sign.

Commitment of the message



② Produce signing material using the private key

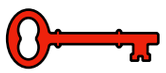
③ Deduce a **valid signature** for the message

Message **signed**

④ Verify the **signature** for the message

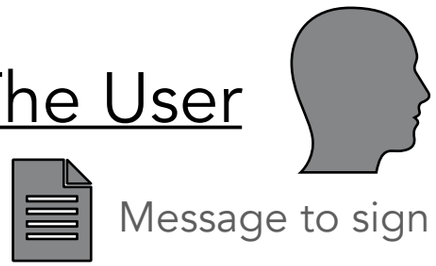
Blind Signature Schemes

The Signer



Signer's private key

The User



Message to sign

The Verifier

Signer's public key

① The User **commits** the message to sign.

Commitment of the message

② Produce signing material using the private key

③ Deduce a **valid signature** for the message

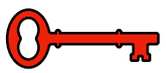
Message **signed**

④ Verify the **signature** for the message

Unforgeability: After interacting with the signer to obtain signatures on a set of messages, the user should remain unable to forge a valid signature on any new message by itself.

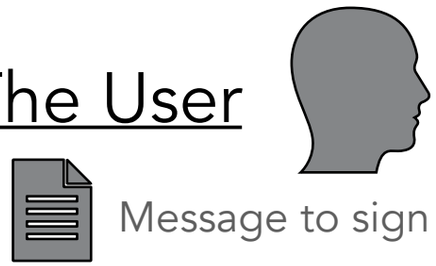
Blind Signature Schemes

The Signer



Signer's private key

The User



Message to sign

The Verifier

Signer's public key

① The User **commits** the message to sign.

Commitment of the message

② Produce signing material using the private key

③ Deduce a **valid signature** for the message

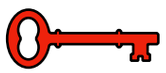
Message **signed**

④ Verify the **signature** for the message

Blindness: The signer must be unable to learn either the content of the message being signed or the resulting signature.

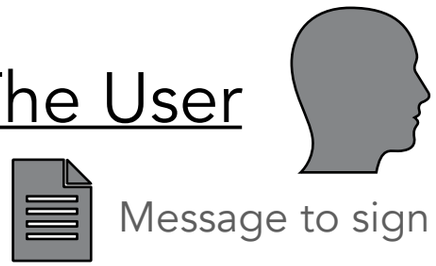
Blind Signature Schemes

The Signer



Signer's private key

The User



Message to sign

The Verifier

Signer's public key

① The User **commits** the message to sign.

Commitment of the message

② Produce signing material using the private key

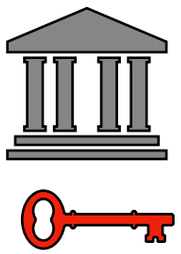
③ Deduce a **valid signature** for the message

Message **signed**

④ Verify the **signature** for the message

It has applications in many **privacy-enhancing technologies** (PETs), such as electronic voting systems.

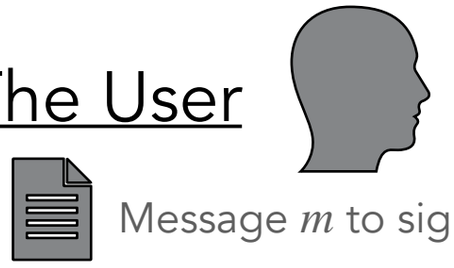
The Fischlin Framework (simplified)



The Signer

Signer's private key

The User



Message m to sign

The Verifier

Signer's public key

① com $\leftarrow \mathrm{Commit}(m; r)$

com

② $\sigma \leftarrow \mathrm{Sign}(\mathrm{sk}, \mathrm{com})$

σ

③ Generate a zero-knowledge proof π that he know σ and r such that

$\mathrm{Verif}(\mathrm{pk}, \sigma, \mathrm{Commit}(\mathrm{m}; \mathrm{r})) = 1.$

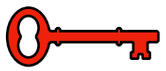
The final signature is π.

m, π

④ Verify the **proof** π

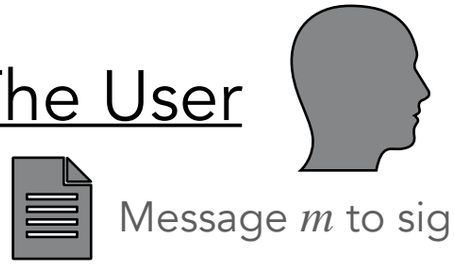
The Fischlin Framework (simplified)

The Signer



Signer's private key

The User



Message m to sign

The Verifier

Signer's public key

① $\text{com} \leftarrow \text{Commit}(m; r)$

com

② $\sigma \leftarrow \text{Sign}(\text{sk}, \text{com})$

σ

③ Generate a zero-knowledge proof π that he know σ and r such that

$\text{Verif}(\text{pk}, \sigma, \text{Commit}(\text{m}; \text{r})) = 1.$

The final signature is π.

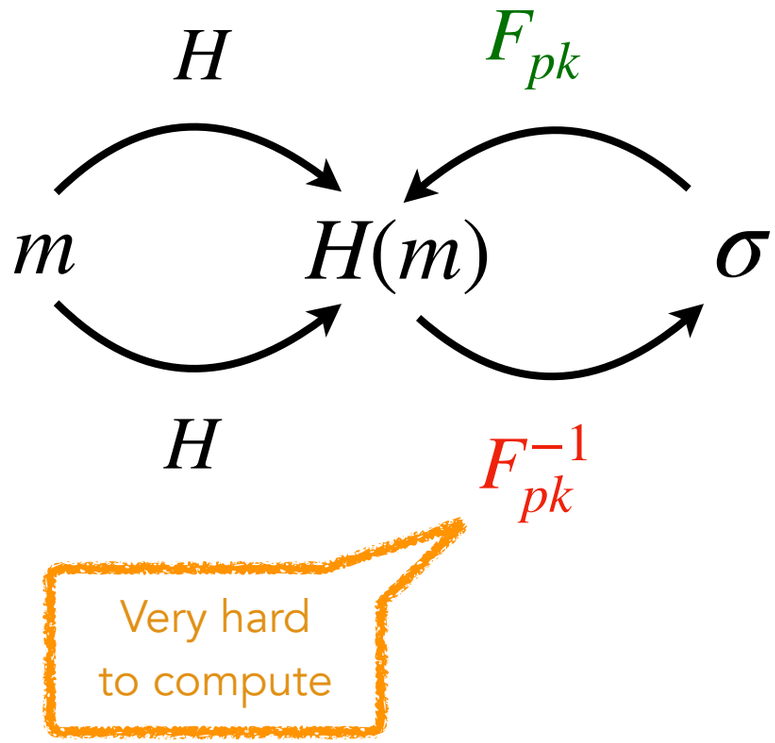
m, π

④ Verify the **proof** π

Require a proof system proving a « small » statement.

The overall performance (proof size, running times) will highly depend on how complex is the verification algorithm.

Post-Quantum Hash-and-sign signatures



Examples:

- Falcon / FN-DSA (lattice)
- Wave, Miranda (code)
- UOV, Mayo (multivariate)

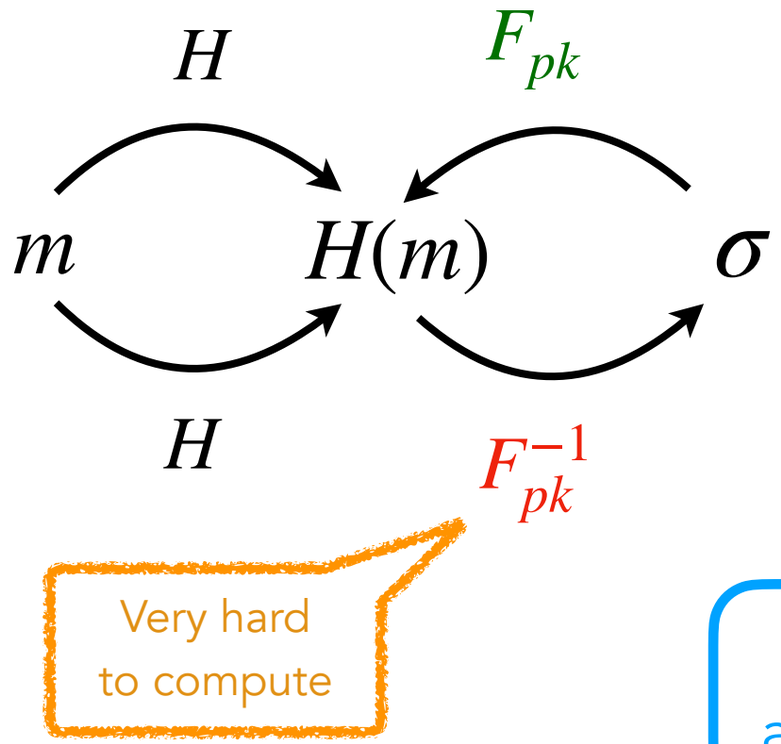
Proving knowledge of signature:

Denoting $h' := H(m)$,

The User proves that he knows σ and h' such that

$$h' = H(m) \quad \text{and} \quad h' = \mathcal{G}_{pk}(\sigma)$$

Post-Quantum Hash-and-sign signatures



Easy to prove,
as it is an algebraic relation

Examples:

- Falcon / FN-DSA (lattice)
- Wave, Miranda (code)
- UOV, Mayo (multivariate)

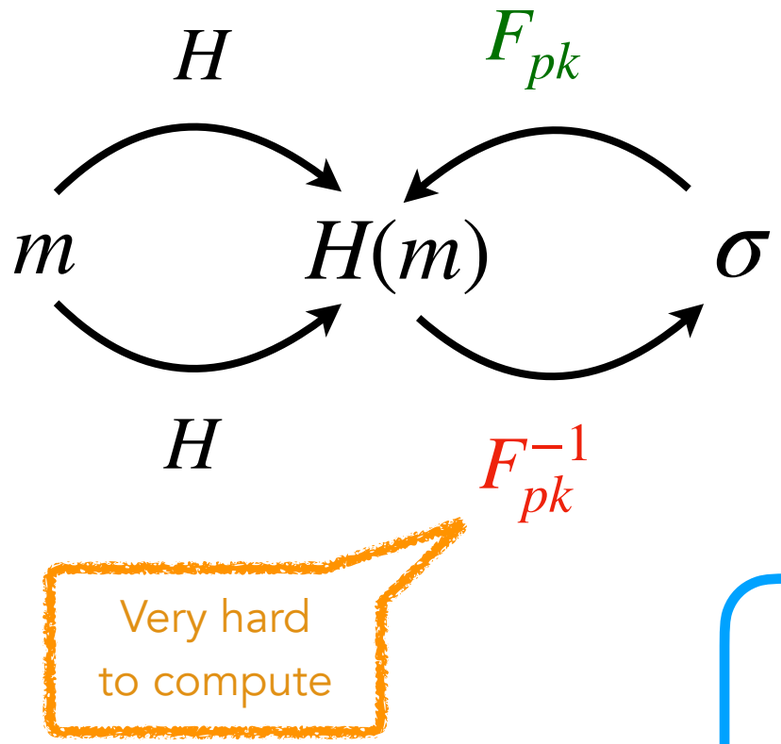
Proving knowledge of signature:

Denoting $h' := H(m)$,

The User proves that he knows σ and h' such that

$h' = H(m)$ and $h' = \mathcal{G}_{pk}(\sigma)$

Post-Quantum Hash-and-sign signatures



More ***complex/ expensive*** to prove

Examples:

- Falcon / FN-DSA (lattice)
- Wave, Miranda (code)
- UOV, Mayo (multivariate)

Proving knowledge of signature:

Denoting $h' := H(m)$,

The User proves that he knows σ and h' such that

$h' = H(m)$ and $h' = \mathcal{G}_{pk}(\sigma)$

Application - Multivariate Blind Signatures

Using UOV-like schemes, $h' = \mathscr{G}_{pk}(\sigma)$ is a quadratic relation.

	Same security as the original scheme	***Signature Size***	***Communication Size***
[BFMR+25a]	Yes	5-8 KB	Few MB
[BFMR+25b]	No	7 KB	< 1 KB
[BBBMR26] Conservative	Yes	24 KB	< 1 KB
[BBBMR26] Non-conservative	No	6-7 KB	< 1 KB

[BFMR+25a] Bouillaguet, Feneuil, Maire, Rivain, Sauvage, Vergnaud. Blinding Post-Quantum Hash-and-Sign Signatures. S&P 2026.

[BFMR+25b] Bouillaguet, Feneuil, Maire, Rivain, Sauvage, Vergnaud. Multivariate Commitment and Signatures with Efficient Protocols. ePrint 2025/2035.

[BBBMR26] Baum, Beckmann, Beullens, Mukherjee, Rechberger. Concretely Efficient Blind Signatures Based on VOLE-in-the-Head Proofs and the MAYO Trapdoor. ePrint 2026/109.

Application - Multivariate Blind Signatures

Using UOV-like schemes, $h' = \mathcal{G}_{pk}(\sigma)$ is a quadratic relation.

	Same security as the original scheme	*Signature Size*	*Communication Size*
[BFMR+25a]	Yes	5-8 KB	Few MB
[BFMR+25b]	No	7 KB	< 1 KB
[BBBMR26] Conservative	Yes	24 KB	< 1 KB
[BBBMR26] Non-conservative	No	6-7 KB	< 1 KB

Tweak the UOV scheme to get rid of the hash function, at the cost of degraded security.

Application - Multivariate Blind Signatures

Using UOV-like schemes, $h' = \mathscr{G}_{pk}(\sigma)$ is a quadratic relation.

	Same security as the original scheme	*Signature Size*	*Communication Size*
[BFMR+25a]	Yes	5-8 KB	Few MB
[BFMR+25b]	No	7 KB	< 1 KB
[BBBMR26] Conservative	Yes	24 KB	< 1 KB
[BBBMR26] Non-conservative	No	6-7 KB	< 1 KB

Tweak the blind-signature framework to get rid of the hash function, at the cost of degraded communication bewteen the user and signer.

Application - Multivariate Blind Signatures

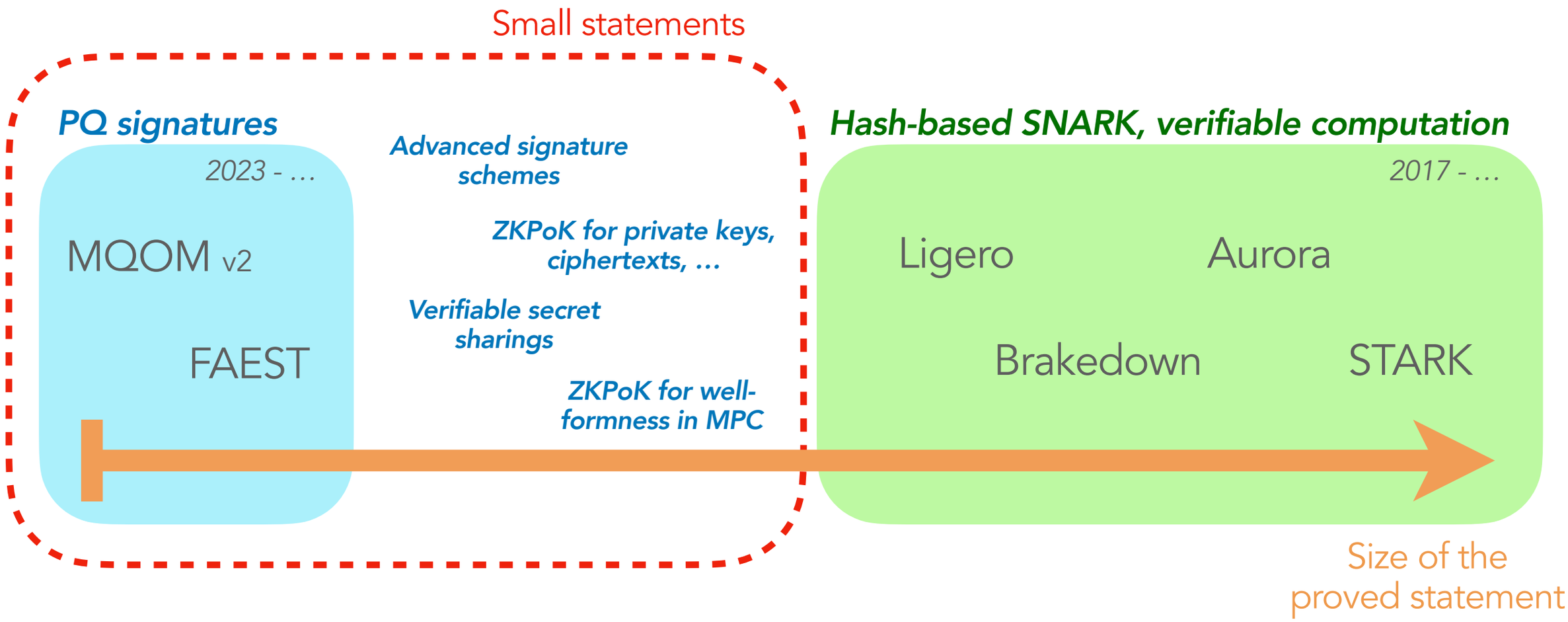
Using UOV-like schemes, $h' = \mathcal{G}_{pk}(\sigma)$ is a quadratic relation.

	Same security as the original scheme	*Signature Size*	*Communication Size*
[BFMR+25a]	Yes	5-8 KB	Few MB
[BFMR+25b]	No	7 KB	< 1 KB
[BBBMR26] Conservative	Yes	24 KB	< 1 KB
[BBBMR26] Non-conservative	No	6-7 KB	< 1 KB

Prove the hash operation, leading to larger signatures

Conclusion

Blueprint of Hash-based Proof Systems



Zero-Knowledge is ***required***
Succinctness is ***optional***

Succinctness is ***required***
Zero-Knowledge is ***optional***

How to commit to polynomials?

(using symmetric primitives)

Values are indicative only and serve to illustrate the progression of the scale.

Merkle Tree

④ FRI-based commitments

③ Merkle Trees with
Ligero-like Proximity Tests

② Degree-enforcing commitment
(TCitH-MT)

① VOLEitH / TCitH-GGM

GGM Tree

degree 10 000

degree 1000

degree 100

degree 10

degree 1

Conclusion

- Hash-based ***SNARK*** could be used to prove all the previous statements, but it would be not efficient.

Proved Statement	***Scheme***	***Used technique***	***Signature Size***	***Signing time***
AES block cipher	Preon	Hash-based SNARK	≈ 140 KB	> 1 second
	FAEST	Tailored Hash-based ZK	≈ 4 KB	Few ms
RescuePrime hash	[ADK24]	Hash-based SNARK	80-100 KB	Few tens of ms
	CAPSS	Tailored Hash-based ZK	≈ 14 KB	Few ms

[CCC+23] Preon: zk-SNARK based Signature Scheme. Chen, Chen, Cheng, Fu, Hong, Hsiang, Hu, Kuo, Lee, Liu, and Thaler. Preon: zk-SNARK based Signature Scheme. Round-1 NIST Candidate (2023).

[BBB+25] Baum, Beullens, Braun, Delpech, Klooß, Majenz, Mukherjee, Orsini, Ramacher, Rechberger, Roy, Scholl. FAEST v2: Algorithm Specifications. Round-2 NIST Candidate (2025).

[ADK24] Atapoor, Delpech, Kindi. *STARK-based signatures from the RPO permutation.* ePrint 2024/1553.

[FR25] Feneuil, Rivain. CAPSS: A Framework for SNARK-Friendly Post-Quantum Signatures. ePrint 2025/061.

Conclusion

- ***Polynomial commitment schemes*** (PCS) is the cornerstone of all the recent hash-based proof systems, including the ones used in signatures.
  - Before 2023, the MPCitH signatures were not following the PIOP+PCS paradigm.

Conclusion

- ***Polynomial commitment schemes*** (PCS) is the cornerstone of all the recent hash-based proof systems, including the ones used in signatures.
    - Before 2023, the MPCitH signatures were not following the PIOP+PCS paradigm.
- There are ***two main approches*** to commit to polynomials using only symmetric cryptography, each of them has its own advantage. ***Depending on the context***, one approach could be better than the other one.
    - Using ***GGM trees*** (a.k.a seed trees)
    - Using ***Merkle trees*** (a.k.a hash trees)

Conclusion

- ***Polynomial commitment schemes*** (PCS) is the cornerstone of all the recent hash-based proof systems, including the ones used in signatures.
  - Before 2023, the MPCitH signatures were not following the PIOP+PCS paradigm.
- There are ***two main approches*** to commit to polynomials using only symmetric cryptography, each of them has its own advantage. ***Depending on the context***, one approach could be better than the other one.
  - Using ***GGM trees*** (a.k.a seed trees)
  - Using ***Merkle trees*** (a.k.a hash trees)
- Hash-based proof systems can be applied to a large set of statements
  - To build **signature schemes**, by proving knowledge of the solution of a hard problem. It might require some effort to arithmetize the statement to prove.
    - **3 schemes** in the 3rd (current) round of the NIST process for standardizing additional post-quantum signatures.
  - To build **advanced** signature schemes

Current Research

- The design for hash-based zero-knowledge proofs of knowledge dedicated to standard signatures is ***stabilizing*** (thanks to the NIST standardization process), but the design for zero-knowledge proofs for small-to-medium statements is still improving.
    - Advanced signature schemes, anonymous credentials
    - ZKPoK for private keys, ciphertexts, signatures, …
    - ZKPoK for well-formness in MPC

Current Research

- The design for hash-based zero-knowledge proofs of knowledge dedicated to standard signatures is ***stabilizing*** (thanks to the NIST standardization process), but the design for zero-knowledge proofs for small-to-medium statements is still improving.
    - Advanced signature schemes, anonymous credentials
    - ZKPoK for private keys, ciphertexts, signatures, …
    - ZKPoK for well-formness in MPC
- While the current state of the art for proving small statements relying on univariate polynomials, the **multilinear polynomials** with the **sumcheck protocol** might be the future for proving small-to-medium statements.

[FS26] Frigo, Shelat. Anonymous credentials from ECDSA. CiC 2026.

Current Research

- The design for hash-based zero-knowledge proofs of knowledge dedicated to standard signatures is ***stabilizing*** (thanks to the NIST standardization process), but the design for zero-knowledge proofs for small-to-medium statements is still improving.
  - Advanced signature schemes, anonymous credentials
  - ZKPoK for private keys, ciphertexts, signatures, …
  - ZKPoK for well-formness in MPC
- While the current state of the art for proving small statements relying on univariate polynomials, the **multilinear polynomials** with the **sumcheck protocol** might be the future for proving small-to-medium statements.

[FS26] Frigo, Shelat. Anonymous credentials from ECDSA. CiC 2026.

Thank you for your attention.