Masking-Friendly Post-Quantum Signatures in the Threshold-Computation-in-the-Head Framework

Thibauld Feneuil, Matthieu Rivain, Auguste Warmé-Janville

CHES 2025

September 16, 2025 — Kuala Lumpur (Malaysia)

Introduction / Context

Context

- 2022: NIST announced the selected signatures for standardisation
    - ML-DSA (Dilithium): rely on structured lattices
    - FN-DSA (Falcon): rely on structured lattices
    - SLH-DSA (SPHINCS$^+$): rely on hash functions

👉 *Need to alternative signature schemes*

Context

- 2022: NIST announced the selected signatures for standardisation
    - ML-DSA (Dilithium): rely on structured lattices
    - FN-DSA (Falcon): rely on structured lattices
    - SLH-DSA (SPHINCS$^+$): rely on hash functions

        👉 *Need to alternative signature schemes*

- 2023: NIST Call for additional post-quantum signatures
    - In Round-2 (2025), six candidates rely on the MPC-in-the-Head paradigm:

        FAEST, Mirath, MQOM, PERK, RYDE, SDitH
    - There are relying on the recent **VOLEitH** and **TCitH** frameworks (2023).

This work

- In this work, we provide insights of how to tweak the TCitH framework to produce masking-friendly schemes, *i.e.* schemes for which the masking induces a reasonable computation cost.

  👉 Equivalent in lattice-based cryptography: Raccoon 

This work

- In this work, we provide insights of how to tweak the TCitH framework to produce masking-friendly schemes, *i.e.* schemes for which the masking induces a reasonable computation cost.

  👉 Equivalent in lattice-based cryptography: Raccoon 

- Goal:

  Given a parameter d, propose a new MPCitH-based signature scheme that can be **very efficiently masked** to achieve a **provable d-probing security**.

- d-probing security:

  A implementation is said **d-probing secure** when any combination of d intermediary variables does not leak secret information.

Introduction to masking

To achieve d-probing security, the most used technique is **masking**.

Instead of directly manipulating a sensitive variable x, we can use a sharing of it:

We denote $[\![x]\!] := ([\![x]\!]_0, \ldots, [\![x]\!]_d)$, where $[\![x]\!]_0, \ldots, [\![x]\!]_d$ are **random values** such that

$$x = [\![x]\!]_0 + [\![x]\!]_2 + \ldots + [\![x]\!]_d.$$

We can perform the computation over masked variables: for example

- $[\![x + y]\!]$ from $[\![x]\!]$ and $[\![y]\!]$: computational cost in $O(d)$
- $[\![x \cdot y]\!]$ from $[\![x]\!]$ and $[\![y]\!]$: computational cost in $O(d^2)$

While it is easy to obtain a d-secure implementation by simply applying masking over sensitive variables, the main issue is the **computational overhead**.

🤔 **Question**: How to tweak a MPCitH-based scheme to avoid a ***quadratic*** computational overhead when masking?

Tweaking the TCitH-based schemes

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.
Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.
Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

$$X \cdot Q(X) = X \cdot M(X) + f(P_1(X), \ldots, P_n(X))$$

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.
Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

$$X \cdot Q(X) = X \cdot M(X) + f(P_1(X), \ldots, P_n(X))$$

④ Choose evaluation points $E := \{e_1, \ldots e_\ell\}$ by hashing the commitment and the message

$$E = \mathrm{Hash}(\mathrm{msg}, h, Q) \subset \mathscr{C}$$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.
Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

$$X \cdot Q(X) = X \cdot M(X) + f(P_1(X), \ldots, P_n(X))$$

④ Choose evaluation points $E := \{e_1, \ldots e_\ell\}$ by hashing the commitment and the message

$$E = \mathrm{Hash}(\mathrm{msg}, h, Q) \subset \mathscr{C}$$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

⑥ Output the signature:

$$\sigma = \left(h, \pi, Q, \left\{\vec{v}^P_j, v^M_j\right\}_j\right) \quad \text{with} \quad \vec{v}^P_j := (v^P_{j,1}, \ldots, v^P_{j,n})$$

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.
Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

$$X \cdot Q(X) = X \cdot M(X) + f(P_1(X), \ldots, P_n(X))$$

④ Choose evaluation points $E := \{e_1, \ldots e_\ell\}$ by hashing the commitment and the message

$$E = \mathrm{Hash}(\mathrm{msg}, h, Q) \subset \mathscr{C}$$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

⑥ Output the signature:

$$\sigma = \left(h, \pi, Q, \left\{ \vec{v}^P_j, v^M_j \right\}_j \right) \quad \text{with} \quad \vec{v}^P_j := (v^P_{j,1}, \ldots, v^P_{j,n})$$

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$. Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

Sampling: computational overhead of $O(d)$

④ Choose ev… …age

$$E = \mathrm{Hash}(\mathrm{msg}, h, Q) \subset \mathscr{C}$$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

⑥ Output the signature:

$$\sigma = \left(h, \pi, Q, \left\{\vec{v}^P_j, v^M_j\right\}_j\right) \quad \text{with} \quad \vec{v}^P_j := (v^P_{j,1}, \ldots, v^P_{j,n})$$

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.
Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the poly

Linear operation: computational overhead of $O(d)$

④ Choose evaluation points $E := \{e_1, \ldots e_\ell\}$ by hashing the commitment and the message

$$E = \text{Hash}(\text{msg}, h, Q) \subset \mathscr{C}$$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

⑥ Output the signature:

$$\sigma = \left(h, \pi, Q, \left\{ \vec{v}^P_j, v^M_j \right\}_j \right) \quad \text{with} \quad \vec{v}^P_j := (v^P_{j,1}, \ldots, v^P_{j,n})$$

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f

Involve multiplications over $\mathbb{F}$ between sensitive variables: computational overhead of $O(d^2)$

① For all i, sample
Sample a random

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

$$X \cdot Q(X) = X \cdot M(X) + f(P_1(X), \ldots, P_n(X))$$

④ Choose evaluation points $E := \{e_1, \ldots e_\ell\}$ by hashing the commitment and the message

$$E = \mathrm{Hash}(\mathrm{msg}, h, Q) \subset \mathscr{C}$$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

⑥ Output the signature:

$$\sigma = \left(h, \pi, Q, \left\{ \vec{v}^P_j, v^M_j \right\}_j \right) \quad \text{with} \quad \vec{v}^P_j := (v^P_{j,1}, \ldots, v^P_{j,n})$$

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.
Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

$$X \cdot Q(X) = X \cdot M(X) + f(P_1(X), \ldots, P_n(X))$$

④ Choose evalu

Involve running symmetric primitives over sensitive variables: huge computational overhead of $O(d^2)$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

⑥ Output the signature:

$$\sigma = \left(h, \pi, Q, \left\{ \vec{v}^P_j, v^M_j \right\}_j \right) \quad \text{with} \quad \vec{v}^P_j := (v^P_{j,1}, \ldots, v^P_{j,n})$$

Introducing a slack

How much the polynomial P_i is sensitive ?

- P_i is a random degree-ℓ polynomial such that $P_i(0) = w_i$.
- ℓ evaluations of P_i are disclosed in the signature transcript.

Introducing a slack

How much the polynomial P_i is sensitive ?

- P_i is a random degree-ℓ polynomial such that $P_i(0) = w_i$.
- ℓ evaluations of P_i are disclosed in the signature transcript.

Leaking any other evaluation of P_i leaks w_i !

Introducing a slack

How much the polynomial P_i is sensitive ?

- P_i is a random degree-ℓ polynomial such that $P_i(0) = w_i$.
- ℓ evaluations of P_i are disclosed in the signature transcript.

Leaking any other evaluation of P_i leaks w_i !

Possible tweak: introduce a slack $\sigma > 0$

- P_i is a degree-ℓ polynomial such that $P_i(0) = w_i$.
- $\ell - \sigma$ evaluations of P_i are disclosed in the signature transcript.

Introducing a slack

How much the polynomial P_i is sensitive ?

- P_i is a random degree-ℓ polynomial such that $P_i(0) = w_i$.
- ℓ evaluations of P_i are disclosed in the signature transcript.

Leaking any other evaluation of P_i leaks w_i !

Possible tweak: introduce a slack $\sigma > 0$

- P_i is a degree-ℓ polynomial such that $P_i(0) = w_i$.
- $\ell - \sigma$ evaluations of P_i are disclosed in the signature transcript.

Leaking σ other evaluations of P_i leaks no informations about w_i !

Introducing a slack

How much the polynomial P_i is sensitive ?

- P_i is a random degree-ℓ polynomial such that $P_i(0) = w_i$.
- ℓ evaluations of P_i are disclosed in the signature transcript.

Leaking any other evaluation of P_i leaks w_i !

Possible tweak: introduce a slack $\sigma > 0$

- P_i is a degree-ℓ polynomial such that $P_i(0) = w_i$.
- $\ell - \sigma$ evaluations of P_i are disclosed in the signature transcript.

Leaking σ other evaluations of P_i leaks no informations about w_i !

Larger signatures

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f

Involve multiplications over $\mathbb{F}$ between sensitive variables: computational overhead of $O(d^2)$

① For all i, sample
Sample a random

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

$$X \cdot Q(X) = X \cdot M(X) + f(P_1(X), \ldots, P_n(X))$$

④ Choose evaluation points $E := \{e_1, \ldots e_\ell\}$ by hashing the commitment and the message

$$E = \mathrm{Hash}(\mathrm{msg}, h, Q) \subset \mathscr{C}$$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

⑥ Output the signature:

$$\sigma = \left(h, \pi, Q, \left\{ \vec{v}^P_j, v^M_j \right\}_j \right) \quad \text{with} \quad \vec{v}^P_j := (v^P_{j,1}, \ldots, v^P_{j,n})$$

TCitH-based Signing Algorithm* (Simplified)

* using the PIOP formalism

Secret key: a field vector $w \in \mathbb{F}^n$

Public key: a multivariate degree-t polynomial f such that $f(w) = 0$

① For all i, sample a random degree-ℓ polynomial $P_i(X)$ such that $P_i(0) = w_i$.
Sample a random degree-$(\ell \cdot t - 1)$ polynomial $M(X)$.

② Commit the polynomials $(P_1, \ldots, P_n)$ and M to obtain a commitment digest h.

③ Build the polynomial Q such that

$$X \cdot Q(X) = X \cdot M(X) + f(P_1(X), \ldots, P_n(X))$$

④ Choose evalu

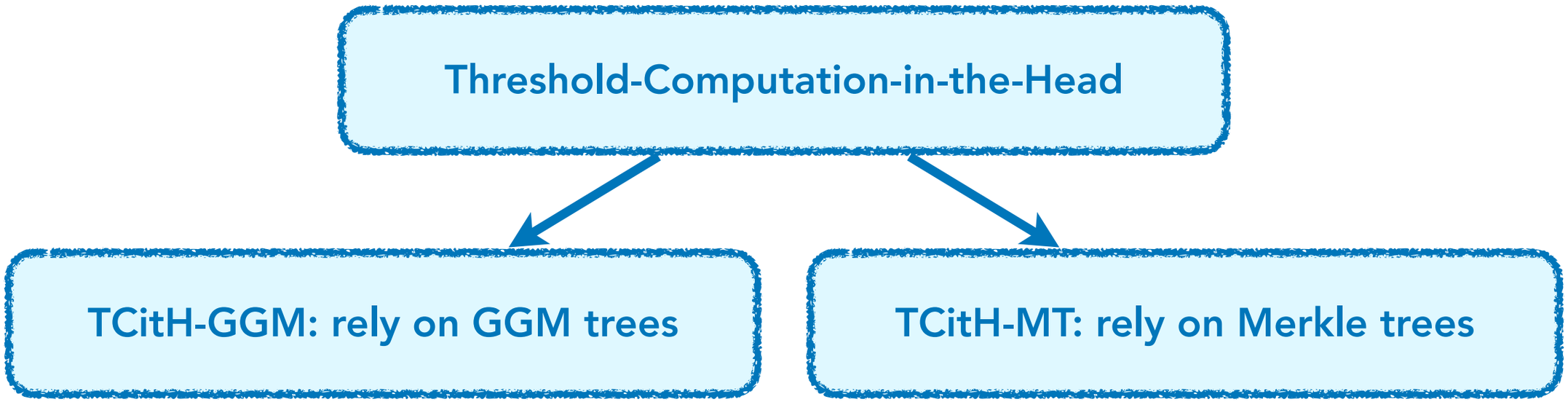
Involve running symmetric primitives over sensitive variables: huge computational overhead of $O(d^2)$

⑤ For all j, compute the evaluations $v^P_{j,i} := P_i(e_j)$ for all i, and $v^M_j := M(e_j)$, together with a opening proof π.

⑥ Output the signature:

$$\sigma = \left(h, \pi, Q, \left\{ \vec{v}^P_j, v^M_j \right\}_j \right) \quad \text{with} \quad \vec{v}^P_j := (v^P_{j,1}, \ldots, v^P_{j,n})$$

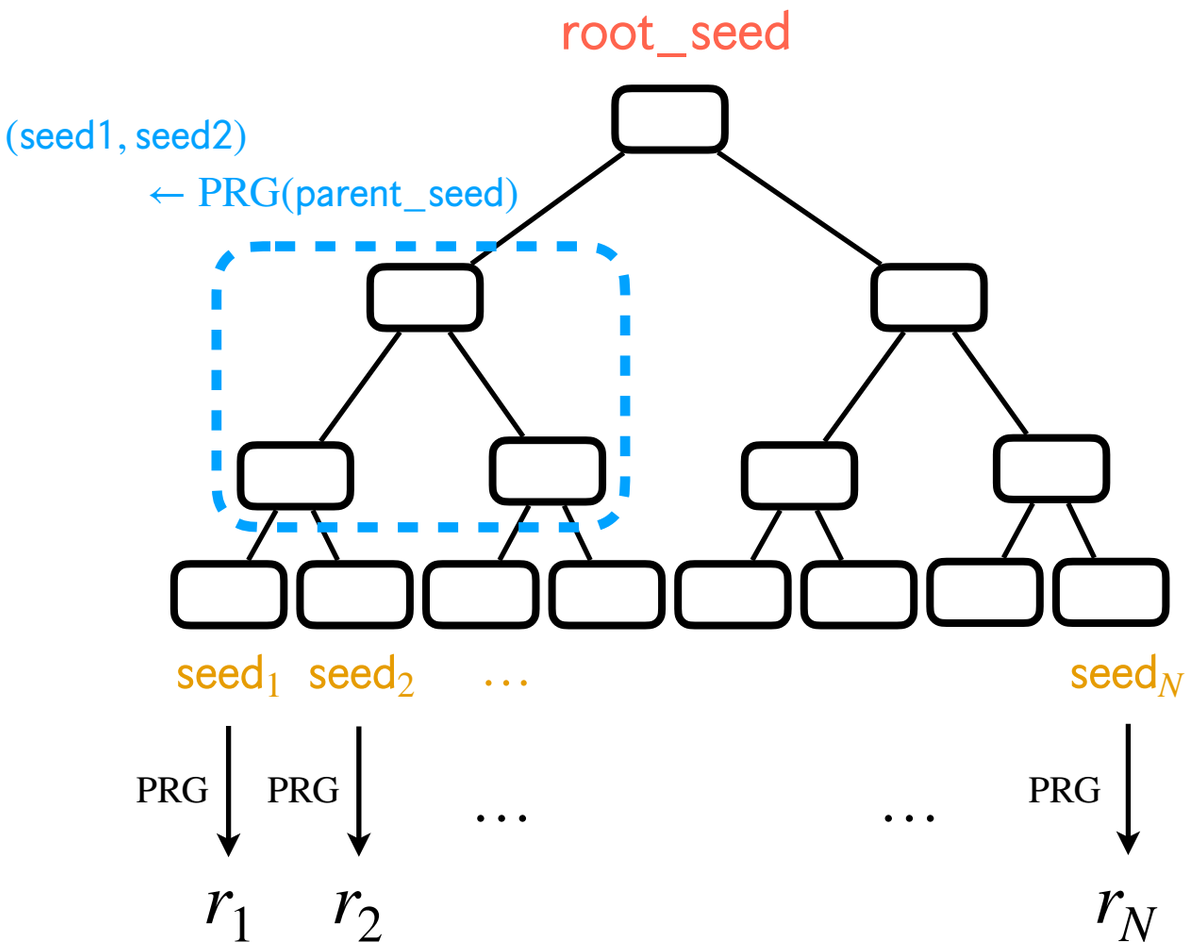
TCitH-based Signing Algorithm



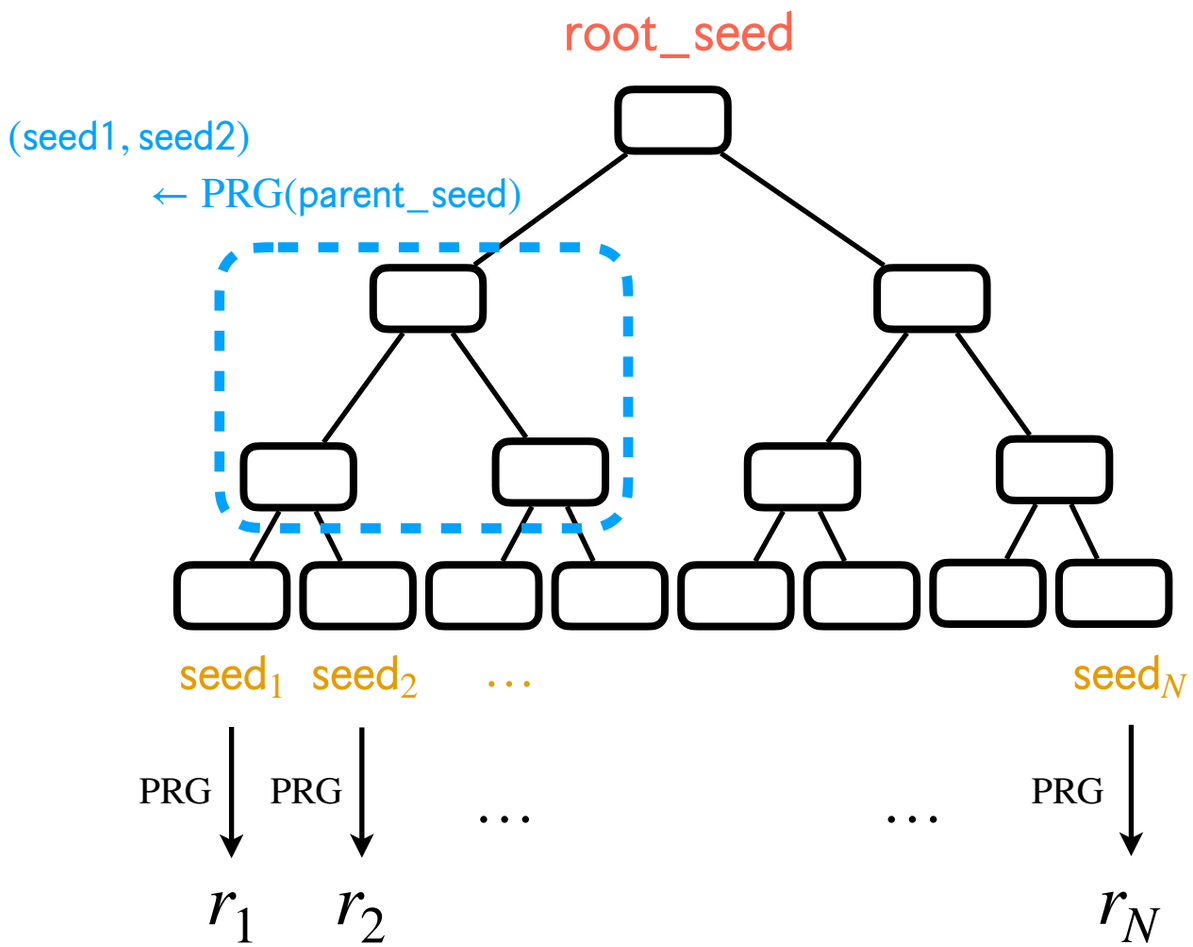
- Used in NIST candidates
- Produce to smaller signatures (around 2.5~5 KB)

- Used in some (small) ZKPoK
- Produce to larger signatures (around 5~10 KB)

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



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

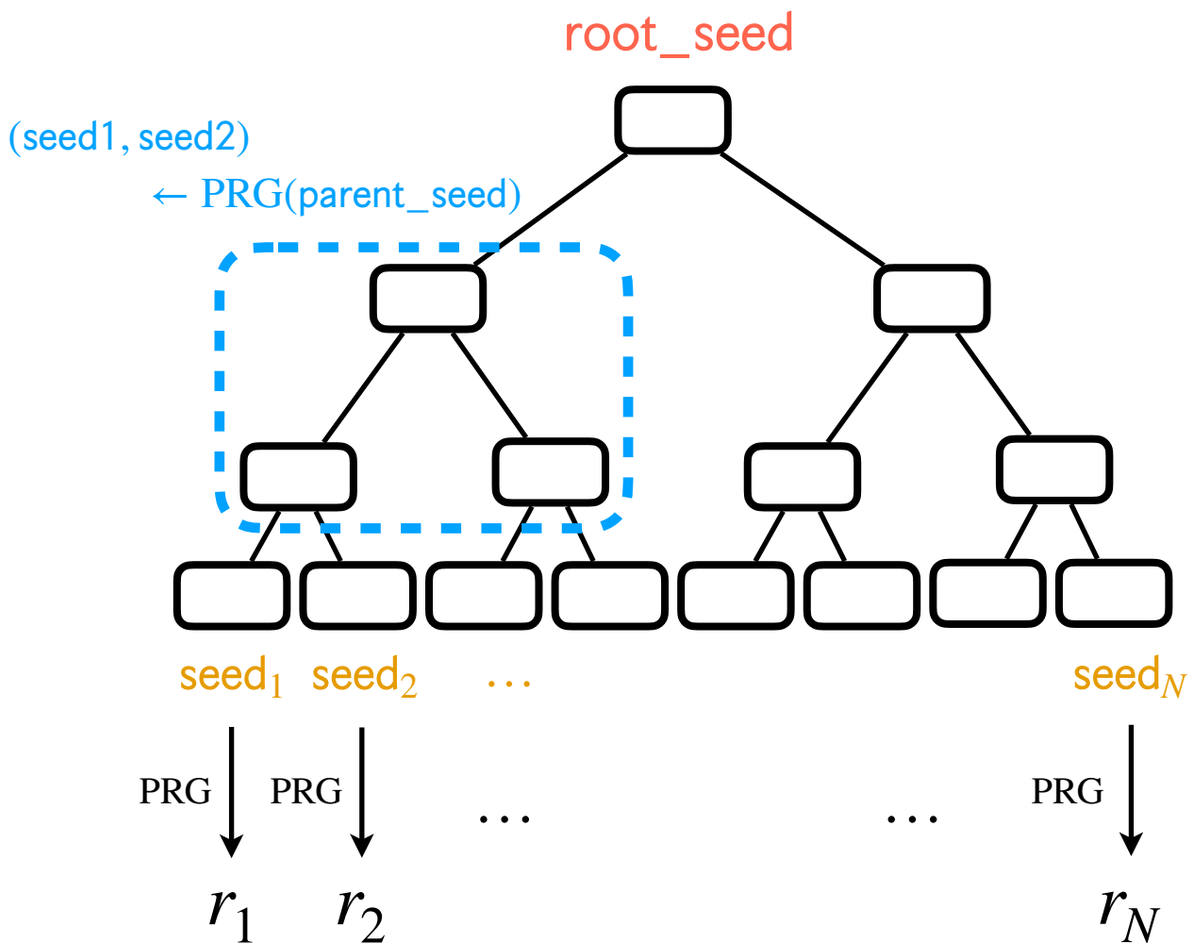


Build $\Delta P(X)$ as

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

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

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

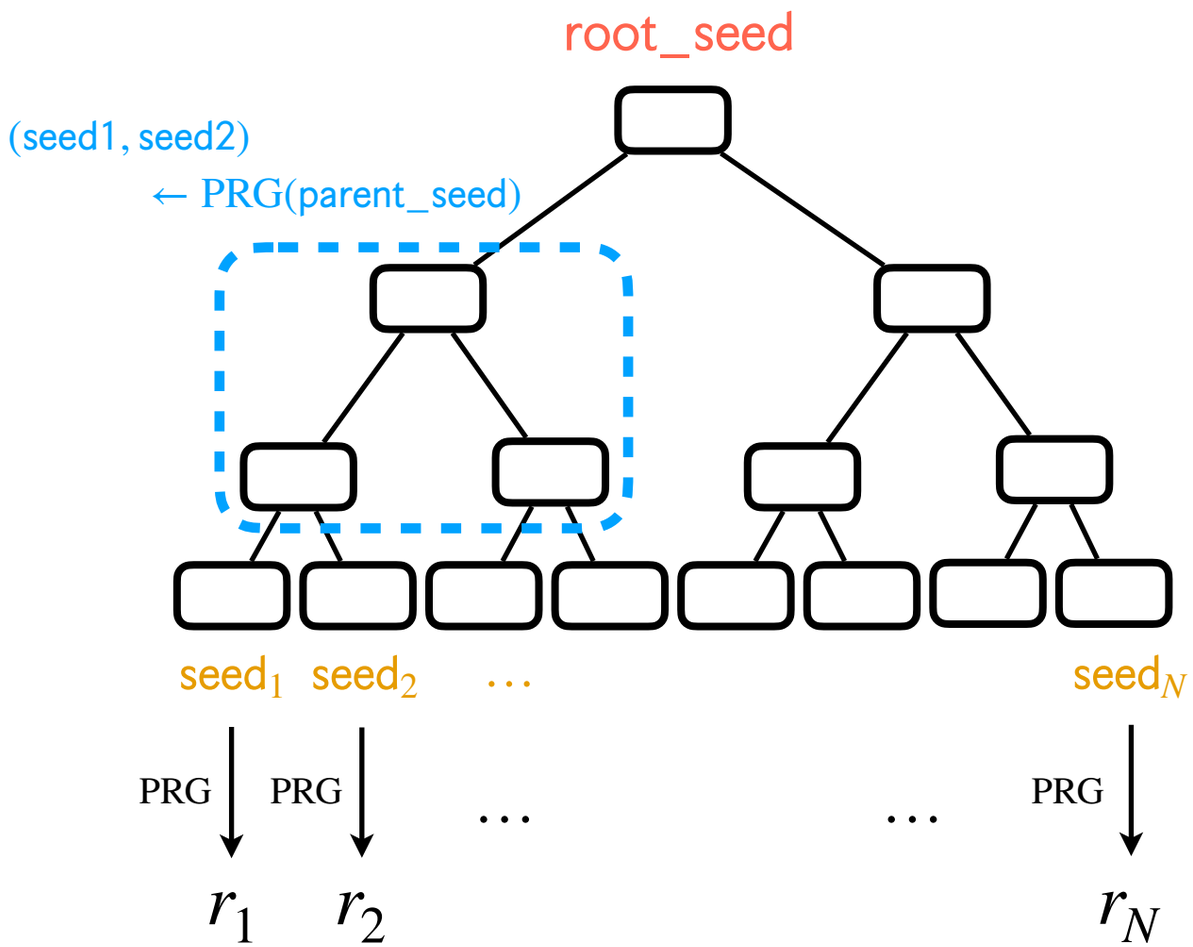


Build $\Delta P(X)$ as

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

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

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



Commitment:

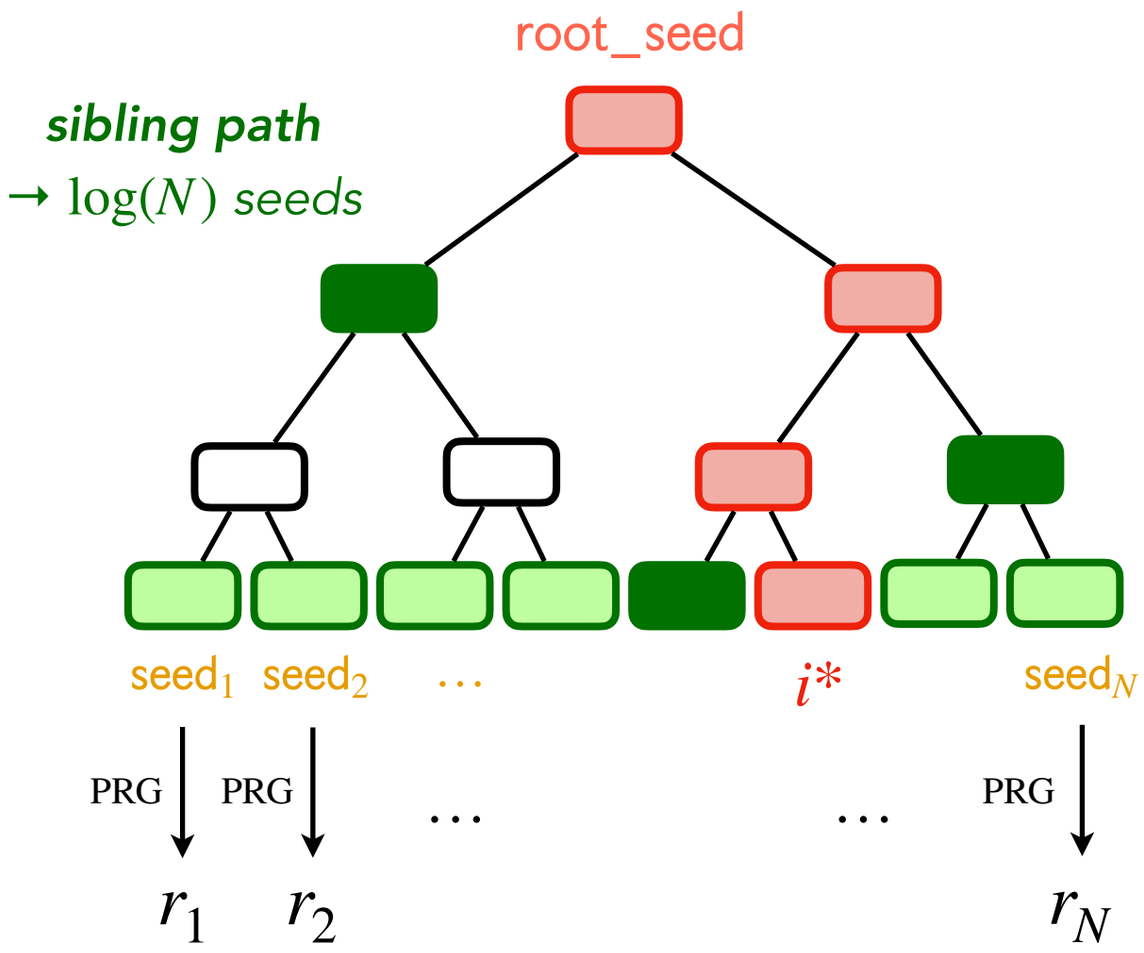
- *Commit to each seed* ***independently***
- *Reveal the masked polynomial* $\Delta P(X)$

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)



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:

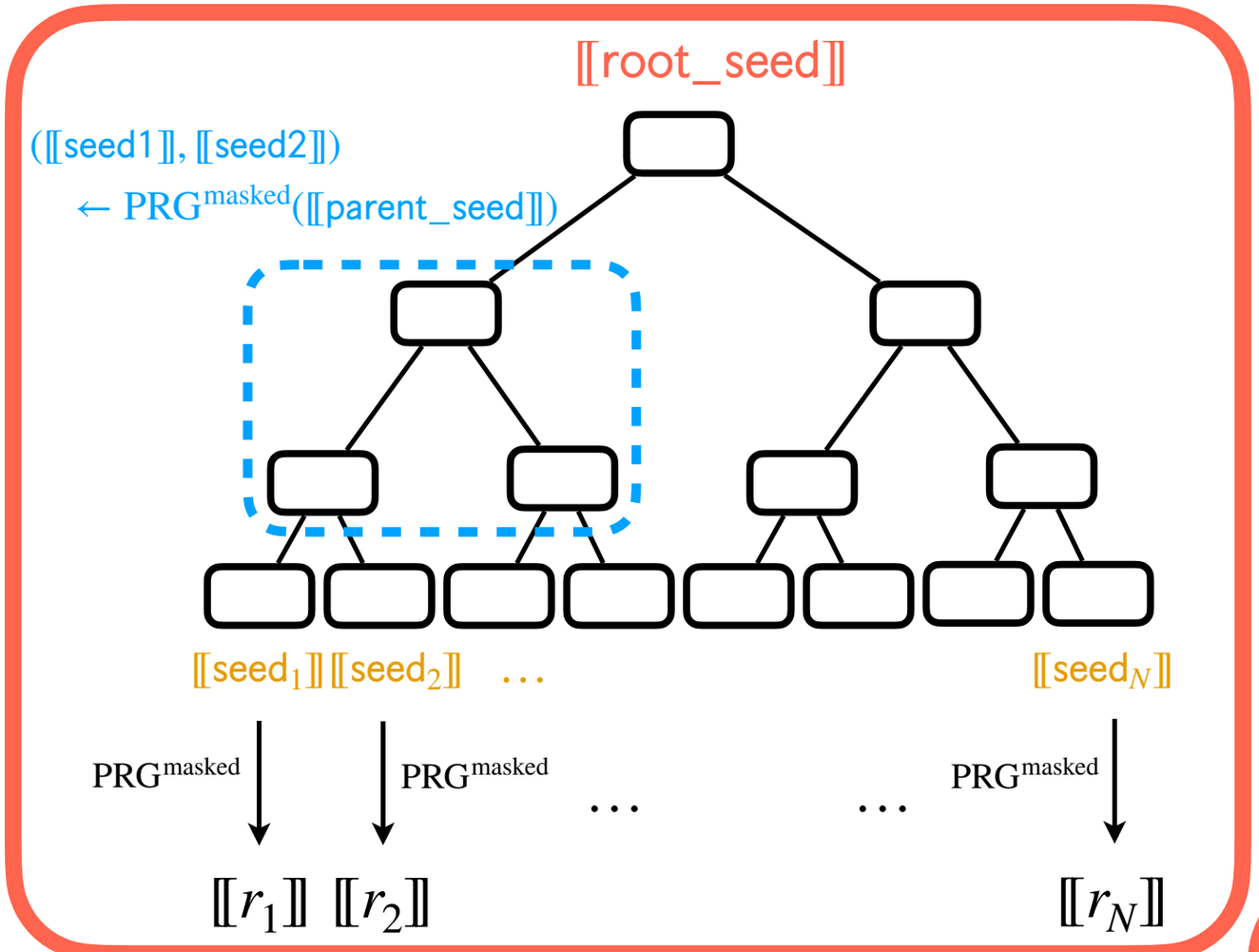
- *Commit to each seed* ***independently***
- *Reveal the masked polynomial* $\Delta P(X)$

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)



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

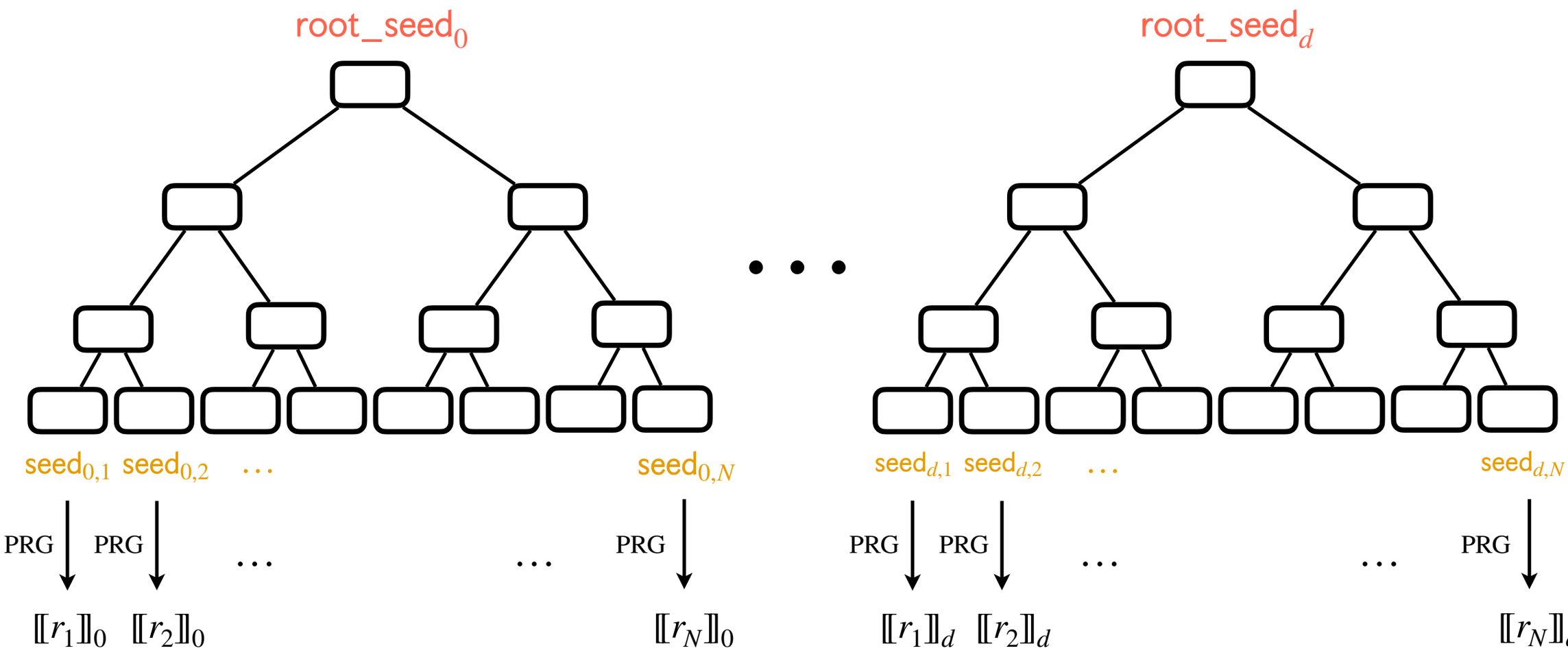
Reveal all $\{r_i\}_{i \neq i*}$ *since*

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

Need to protect around $\sim 2N \cdot \tau$ calls to PRG, usually a PRG based on SHAKE or on AES in counter mode: Extremely computationally costly!

Typically value for $2\tau N$: $2 \times 2048 \times 12 \approx 50000$

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

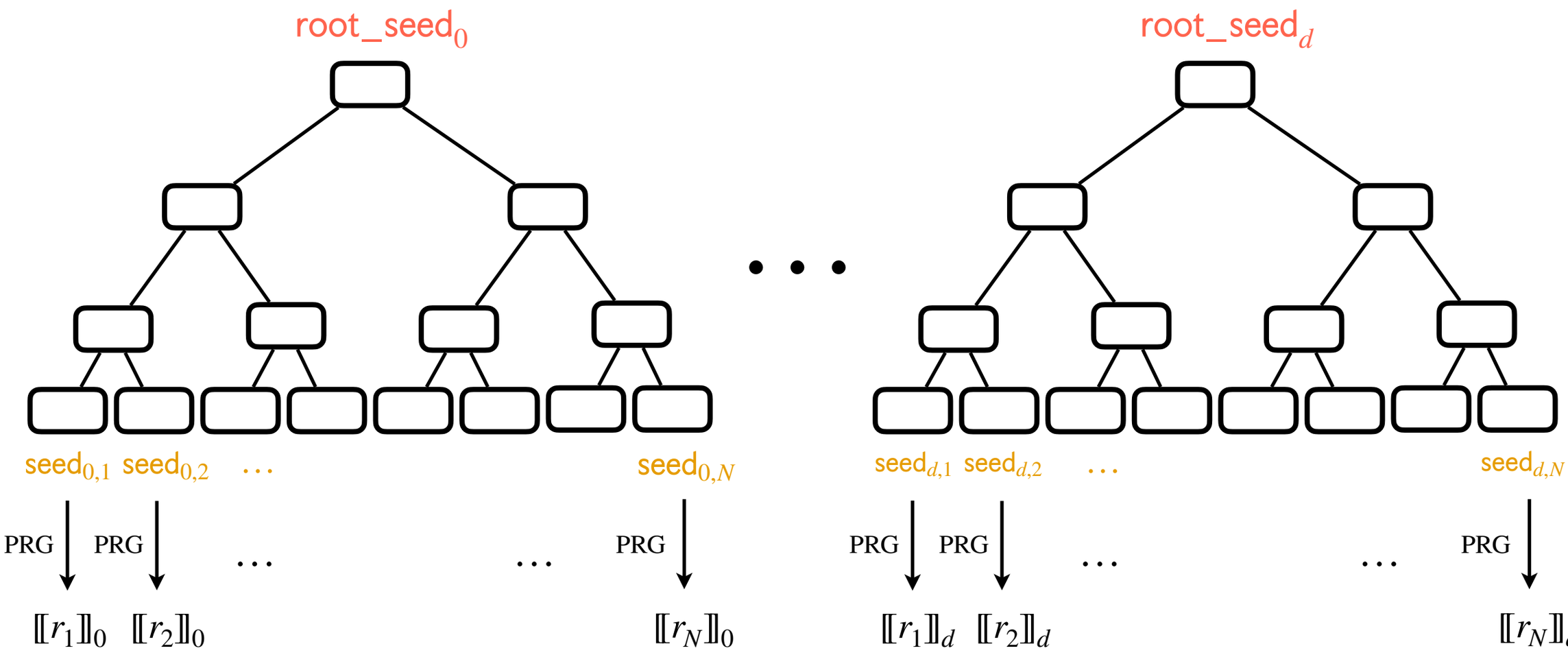


Build $\Delta P(X)$ as

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

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

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



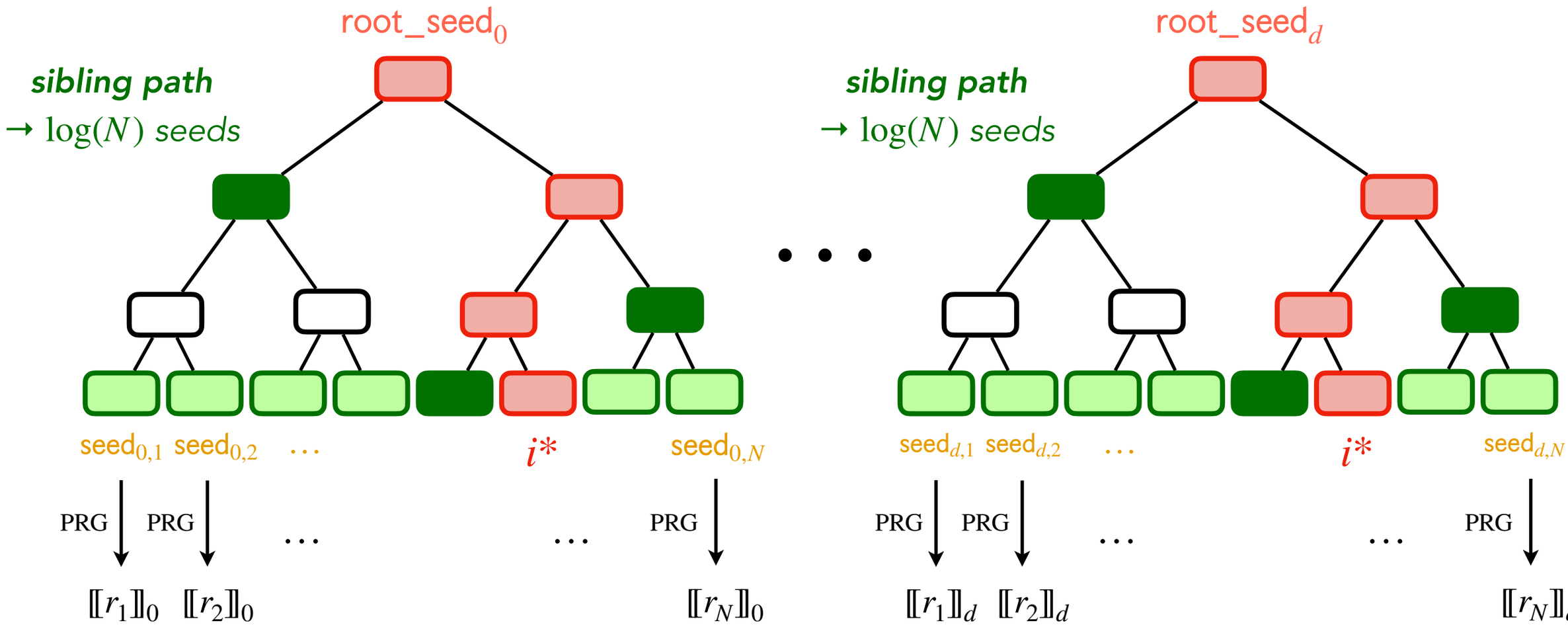
Build $\Delta P(X)$ as

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

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

Linear computation overhead, i.e. $O(d)$

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



Build $\Delta P(X)$ as

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

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

Linear computation overhead, i.e. $O(d)$

The number of revealed sibling paths is $d+1$ times larger.

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

	No Tweak (using masking)		*Parallel Trees*	
Nb shares	***Signing time***	***Sig. Size***	***Signing time***	***Sig. Size***
1	0,35 s	4.5 KB	0,35 s	4.5 KB
2	14 s	4.5 KB	0,93 s	6.7 KB
3	27 s	4.5 KB	1,43 s	8.9 KB
4	53 s	4.5 KB	2,03 s	11.0 KB
8	236 s	4.5 KB	4,94 s	19.7 KB
16	993 s	4.5 KB	14,0 s	37.1 KB
32	4519 s	4.5 KB	45,0 s	72.0 KB

Performance of a TCitH-GGM-based signature scheme for which the security relies on the hardness of solving the MQ problem. Running times estimated for a RISC-V platform assuming the presence of a hardware accelerator for plain Keccak and that the PRG is instantiated using SHAKE.

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

No masking, i.e. basic scheme

	No Tweak (using masking)		*Parallel Trees*	
Nb shares	*Signing time*	*Sig. Size*	*Signing time*	*Sig. Size*
1	0,35 s	4.5 KB	0,35 s	4.5 KB
2	14 s	4.5 KB	0,93 s	6.7 KB
3	27 s	4.5 KB	1,43 s	8.9 KB
4	53 s	4.5 KB	2,03 s	11.0 KB
8	236 s	4.5 KB	4,94 s	19.7 KB
16	993 s	4.5 KB	14,0 s	37.1 KB
32	4519 s	4.5 KB	45,0 s	72.0 KB

Performance of a TCitH-GGM-based signature scheme for which the security relies on the hardness of solving the MQ problem. Running times estimated for a RISC-V platform assuming the presence of a hardware accelerator for plain Keccak and that the PRG is instantiated using SHAKE.

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

	No Tweak (using masking)		*Parallel Trees*	
Nb shares	***Signing time***	***Sig. Size***	***Signing time***	***Sig. Size***
1	0,35 s	4.5 KB	0,35 s	4.5 KB
2	14 s	4.5 KB	0,93 s	6.7 KB
3	27 s	4.5 KB	1,43 s	8.9 KB
4	53 s	4.5 KB	2,03 s	11.0 KB
8	236 s	4.5 KB	4,94 s	19.7 KB
16	993 s	4.5 KB	14,0 s	37.1 KB
32	4519 s	4.5 KB	45,0 s	72.0 KB

Performance of a TCitH-GGM-based signature scheme for which the security relies on the hardness of solving the MQ problem. Running times estimated for a RISC-V platform

Signature size independent of the masking order

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

	No Tweak (using masking)		*Parallel Trees*	
Nb shares	***Signing time***	***Sig. Size***	***Signing time***	***Sig. Size***
1	0,35 s	4.5 KB	0,35 s	4.5 KB
2	14 s	4.5 KB	0,93 s	6.7 KB
3	27 s	4.5 KB	1,43 s	8.9 KB
4	53 s	4.5 KB	2,03 s	11.0 KB
8	236 s	4.5 KB	4,94 s	19.7 KB
16	993 s	4.5 KB	14,0 s	37.1 KB
32	4519 s	4.5 KB	45,0 s	72.0 KB

Big computation overhead

Performance of a TCitH-GGM-based signature scheme for
which the security relies on the hardness of solving the MQ
s estimated for a RISC-V platform
of a hardware accelerator for plain
RG is instantiated using SHAKE.

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

	No Tweak (using masking)		*Parallel Trees*	
Nb shares	***Signing time***	***Sig. Size***	***Signing time***	***Sig. Size***
1	0,35 s	4.5 KB	0,35 s	4.5 KB
2	14 s	4.5 KB	0,93 s	6.7 KB
3	27 s	4.5 KB	1,43 s	8.9 KB
4	53 s	4.5 KB	2,03 s	11.0 KB
8	236 s	4.5 KB	4,94 s	19.7 KB
16	993 s	4.5 KB	14,0 s	37.1 KB
32	4519 s	4.5 KB	45,0 s	72.0 KB

Performance of a TCitH-GGM-based signature scheme for which the security relies on the hardness of solving the MQ problem. Running times estimated for a RISC-V platform assuming the presence of a hardware accelerator for plain HAKE.

Much faster implementation

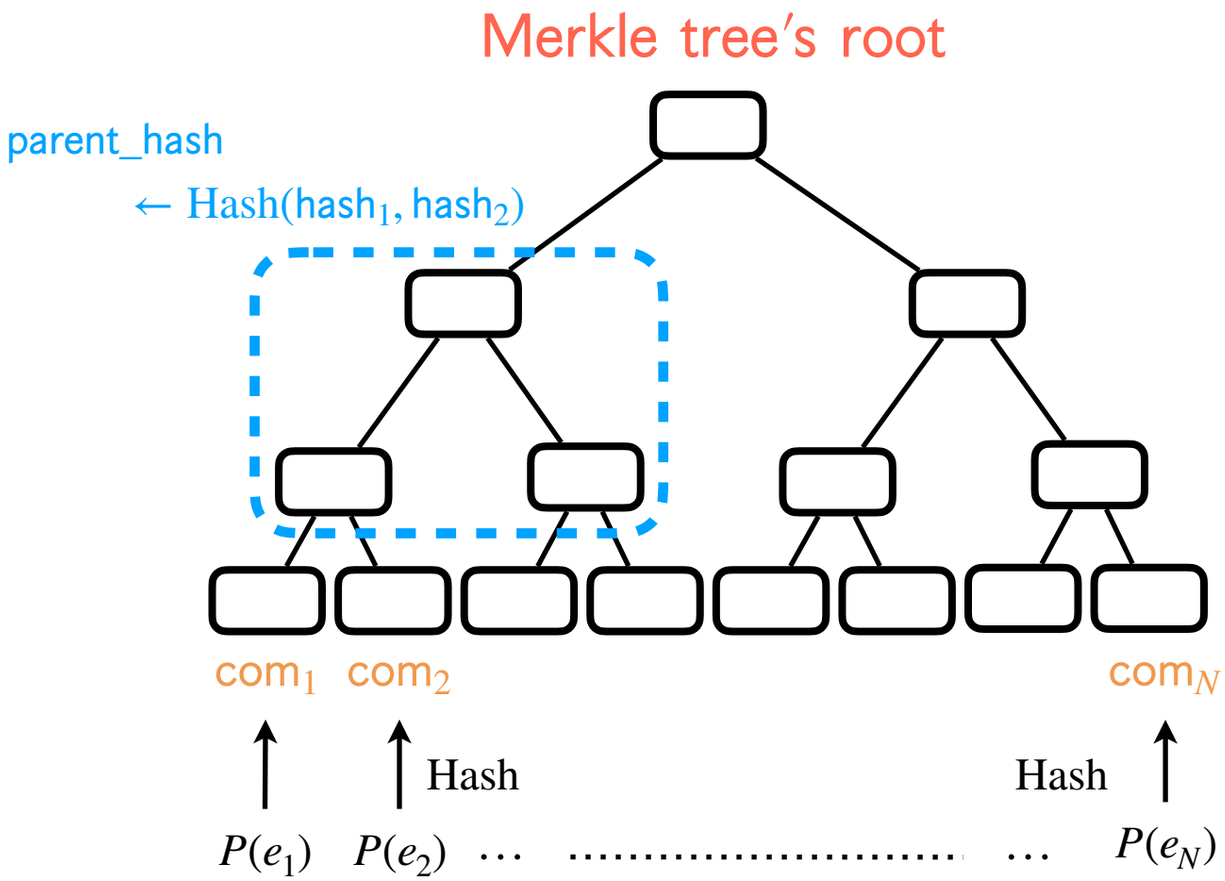
Option 1: Using a GGM tree (*ie.* a seed tree)

	No Tweak (using masking)		*Parallel Trees*	
Nb shares	*Signing time*	*Sig. Size*	*Signing time*	*Sig. Size*
1	0,35 s	4.5 KB	0,35 s	4.5 KB
2	14 s	4.5 KB	0,93 s	6.7 KB
3	27 s	4.5 KB	1,43 s	8.9 KB
4	53 s	4.5 KB	2,03 s	11.0 KB
8	236 s	4.5 KB	4,94 s	19.7 KB
16	993 s	4.5 KB	14,0 s	37.1 KB
32	4519 s	4.5 KB	45,0 s	72.0 KB

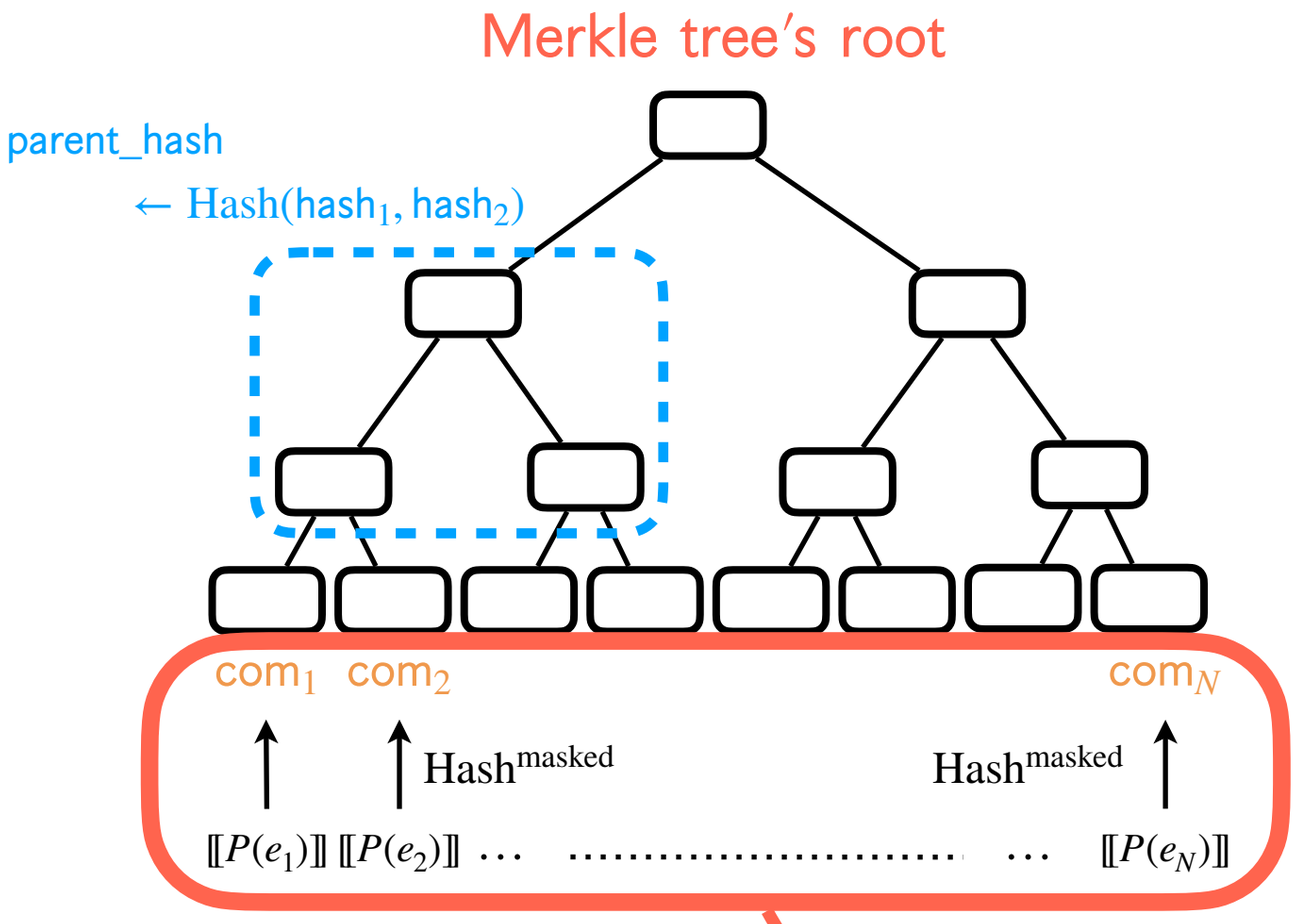
Performance of a TCitH-GGM-based signature scheme for the MQ
tform
or plain
Keccak and that the PRG is instantiated using SHAKE.

Big communication overhead

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

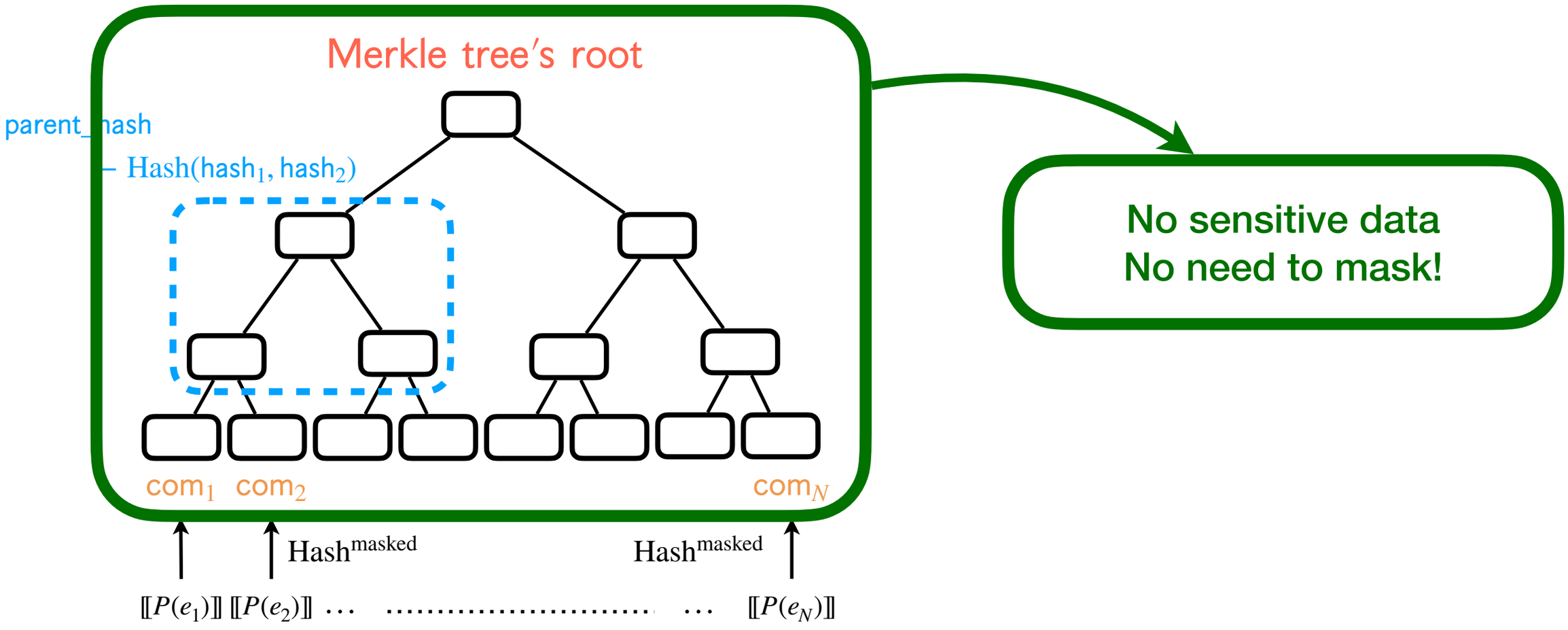


Option 2: Using a Merkle tree (*ie.* a hash tree)

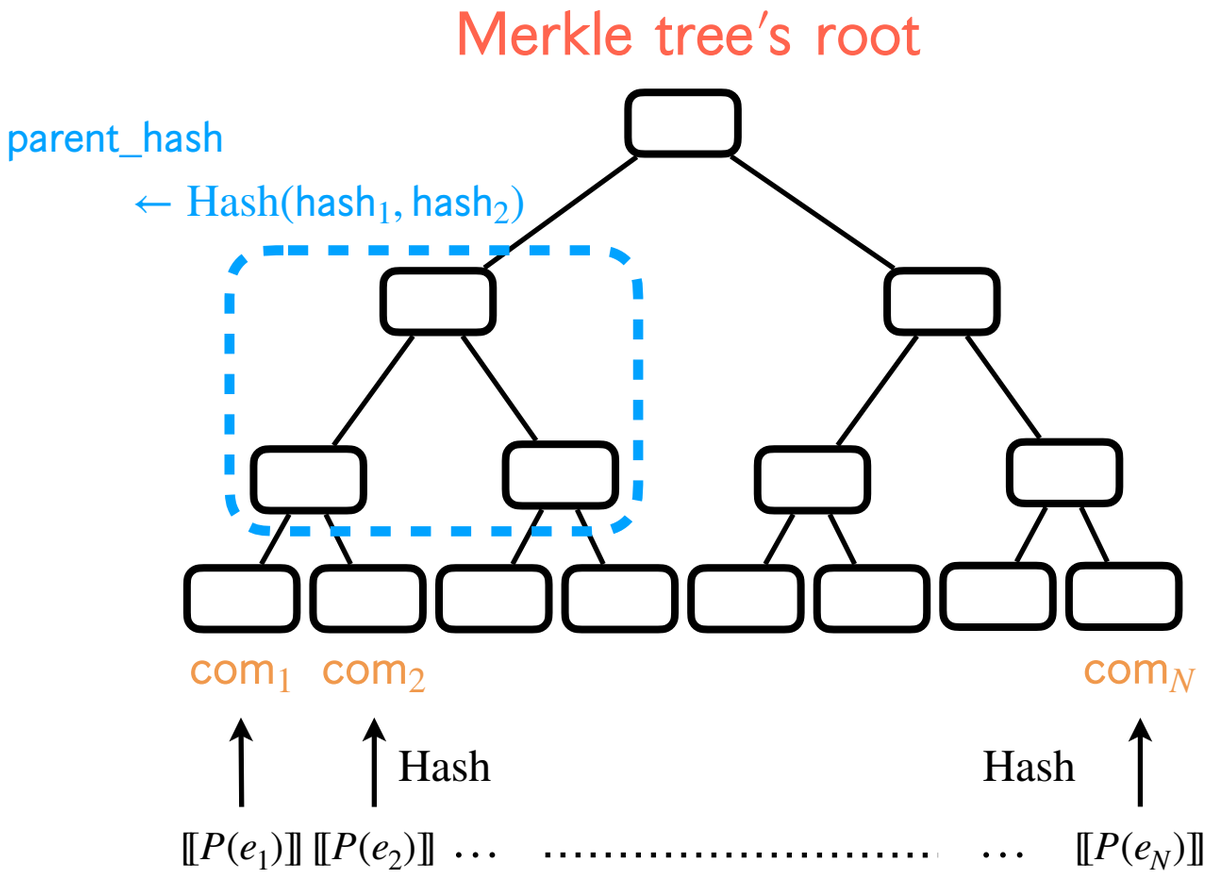


Need to protect around $\sim N \cdot \tau$ calls to hash function:
Computationally costly!

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



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



To commit to a value v (no tweaked):

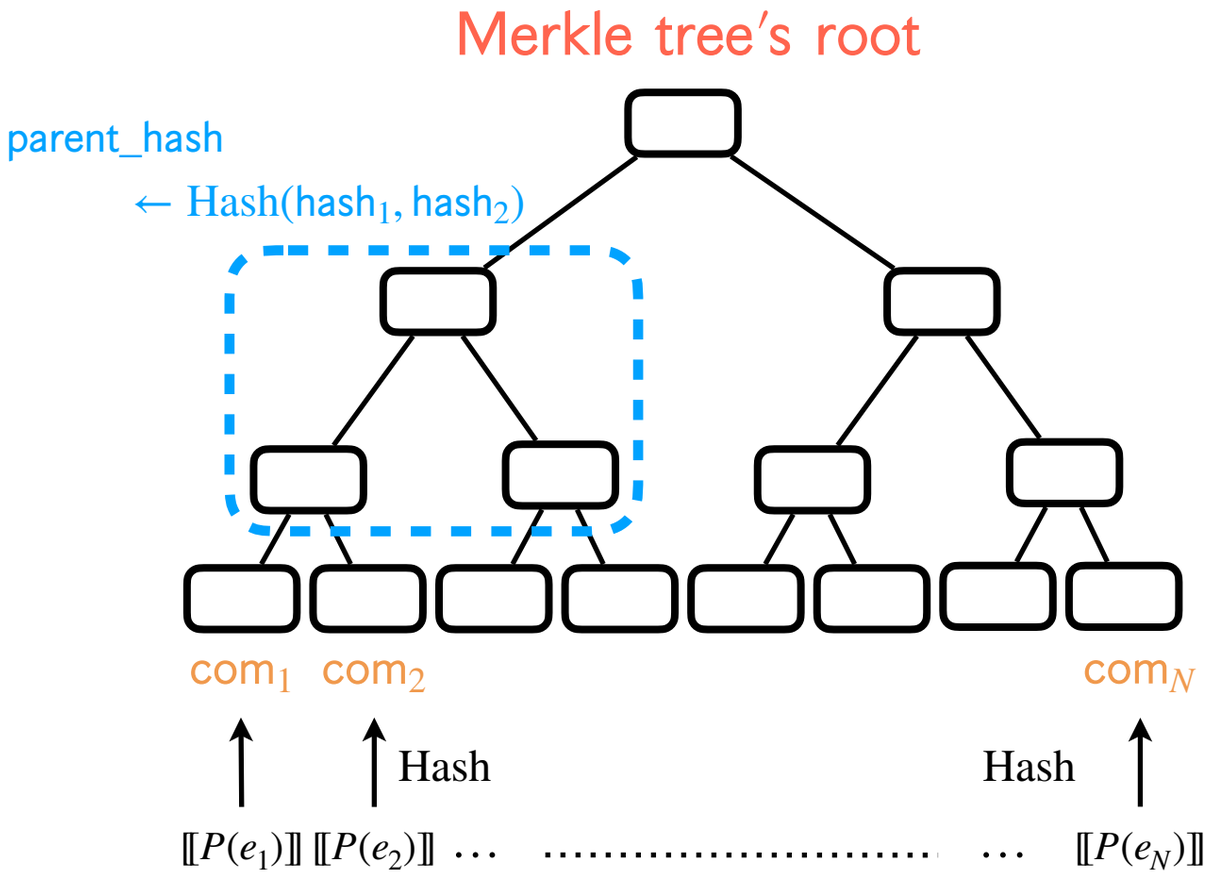
$\mathsf{com}_v := \mathrm{Hash}(v)$ from $[\![v]\!]$

Tweak Idea:

One can commit directly to the shares $[\![v]\!]_0, \ldots, [\![v]\!]_d$. Namely,

$\mathsf{com}_v := \mathrm{Hash}([\![v]\!]_0) \,\|\, \ldots \,\|\, \mathrm{Hash}([\![v]\!]_d)$.

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



To commit to a value v (no tweaked):

$\mathsf{com}_v := \mathrm{Hash}(v)$ from $[\![v]\!]$

Tweak Idea:

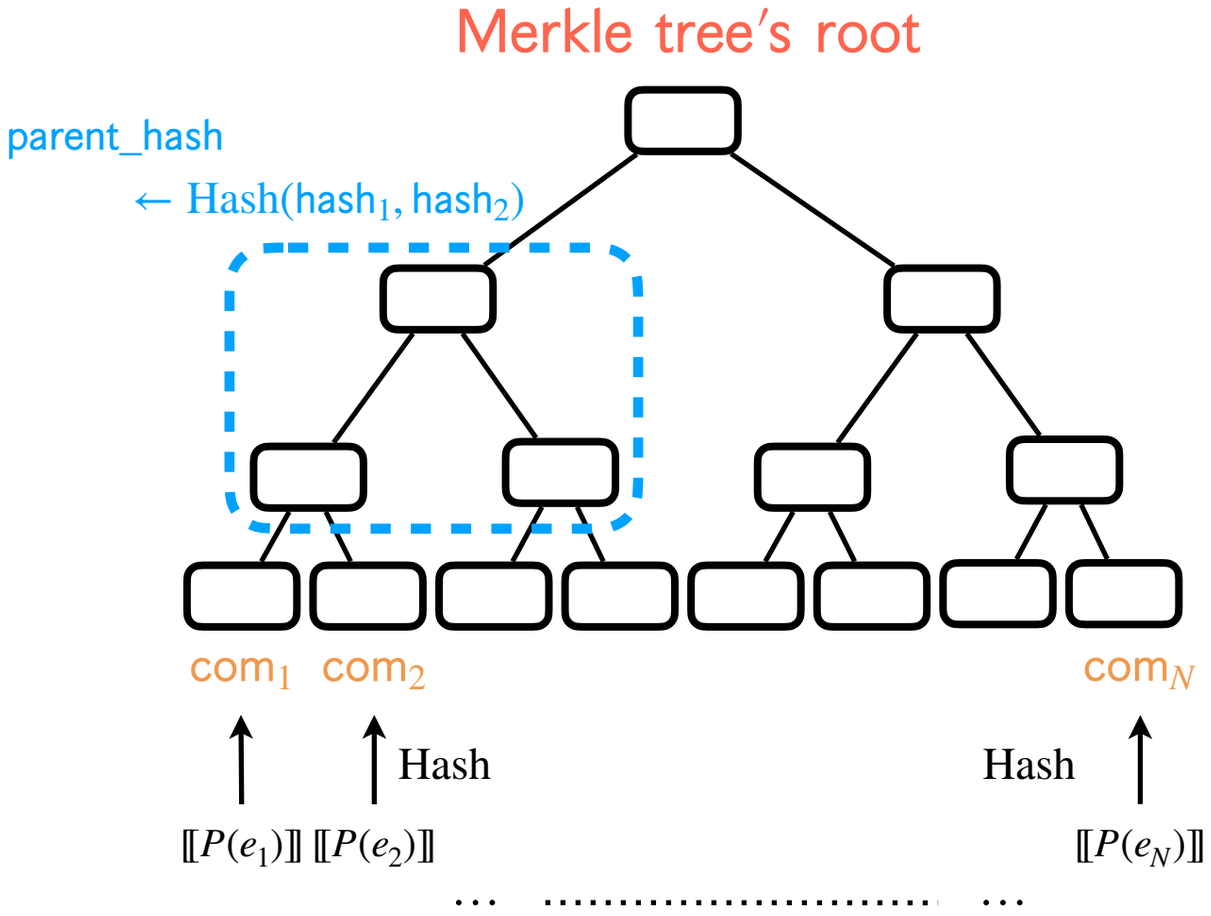
One can commit directly to the shares $[\![v]\!]_0, \ldots, [\![v]\!]_d$. Namely,

$\mathsf{com}_v := \mathrm{Hash}([\![v]\!]_0) \,\|\, \ldots \,\|\, \mathrm{Hash}([\![v]\!]_d)$.

Issue:

To let the verifier check the commitment, the signature transcript should contain the entire $[\![v]\!]$ (and not just v), such that the verifier can recompute the digest com_v.

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



To commit to a value v (no tweaked):

$\text{com}_v := \text{Hash}(v)$ from $[\![v]\!]$

Basic Tweak Idea:

One can commit directly to the shares $[\![v]\!]_0, \ldots, [\![v]\!]_d$. Namely,

$\text{com}_v := \text{Hash}([\![v]\!]_0) \,\|\, \ldots \,\|\, \text{Hash}([\![v]\!]_d)$.

Improved Tweak Idea:

Commit to the shares $[\![v]\!]_0, \ldots, [\![v]\!]_d$ after compressing them into pseudo-random shares:

$$(v_0, \text{seed}_1, \ldots, \text{seed}_d) \leftarrow \text{MaskCompress}([\![v]\!])$$
$$\text{com}_i \leftarrow \text{Hash}(v_0) \,\|\, \text{Hash}(\text{seed}_1) \,\|\, \ldots \,\|\, \text{Hash}(\text{seed}_d)$$

where $v = v_0 + v_1 + \ldots + v_d$, with $v_i := \text{PRG}(\text{seed}_i)$ for all $i \geq 1$.

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

	No Tweak (using masking)		*Com. of PR sharings*	
Nb shares	***Signing time***	***Sig. Size***	***Signing time***	***Sig. Size***
1	0,35 s	6.5 KB	0,35 s	6.5 KB
2	5.2 s	6.5 KB	1.1 s	6.8 KB
3	9.8 s	6.5 KB	1.9 s	7.1 KB
4	19 s	6.5 KB	2.8 s	7.4 KB
8	81 s	6.5 KB	7.3 s	8.6 KB
16	339 s	6.5 KB	20 s	11.1 KB
32	1534 s	6.5 KB	60 s	15.9 KB

Performance of a TCitH-MT-based signature scheme for which the security relies on the hardness of solving the MQ problem. Running times estimated for a RISC-V platform assuming the presence of a hardware accelerator for plain Keccak.

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

No masking, i.e. basic scheme

	No Tweak (using masking)		*Com. of PR sharings*	
Nb shares	*Signing time*	*Sig. Size*	*Signing time*	*Sig. Size*
1	0,35 s	6.5 KB	0,35 s	6.5 KB
2	5.2 s	6.5 KB	1.1 s	6.8 KB
3	9.8 s	6.5 KB	1.9 s	7.1 KB
4	19 s	6.5 KB	2.8 s	7.4 KB
8	81 s	6.5 KB	7.3 s	8.6 KB
16	339 s	6.5 KB	20 s	11.1 KB
32	1534 s	6.5 KB	60 s	15.9 KB

Performance of a TCitH-MT-based signature scheme for which the security relies on the hardness of solving the MQ problem. Running times estimated for a RISC-V platform assuming the presence of a hardware accelerator for plain Keccak.

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

	No Tweak (using masking)		*Com. of PR sharings*	
Nb shares	***Signing time***	***Sig. Size***	***Signing time***	***Sig. Size***
1	0,35 s	6.5 KB	0,35 s	6.5 KB
2	5.2 s	6.5 KB	1.1 s	6.8 KB
3	9.8 s	6.5 KB	1.9 s	7.1 KB
4	19 s	6.5 KB	2.8 s	7.4 KB
8	81 s	6.5 KB	7.3 s	8.6 KB
16	339 s	6.5 KB	20 s	11.1 KB
32	1534 s	6.5 KB	60 s	15.9 KB

Performance of a TCitH-MT-based signature scheme for which the security relies on the hardness of solving the MQ problem.

Signature size independent of the masking order

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

	No Tweak (using masking)		*Com. of PR sharings*	
Nb shares	***Signing time***	***Sig. Size***	***Signing time***	***Sig. Size***
1	0,35 s	6.5 KB	0,35 s	6.5 KB
2	5.2 s	6.5 KB	1.1 s	6.8 KB
3	9.8 s	6.5 KB	1.9 s	7.1 KB
4	19 s	6.5 KB	2.8 s	7.4 KB
8	81 s	6.5 KB	7.3 s	8.6 KB
16	339 s	6.5 KB	20 s	11.1 KB
32	1534 s	6.5 KB	60 s	15.9 KB

Big computation overhead

Performance of a TCitH-MT-based signature scheme for which the security relies on the hardness of solving the MQ problem. ed for a RISC-V platform assuming the accelerator for plain Keccak.

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

	No Tweak (using masking)		*Com. of PR sharings*	
Nb shares	***Signing time***	***Sig. Size***	***Signing time***	***Sig. Size***
1	0,35 s	6.5 KB	0,35 s	6.5 KB
2	5.2 s	6.5 KB	1.1 s	6.8 KB
3	9.8 s	6.5 KB	1.9 s	7.1 KB
4	19 s	6.5 KB	2.8 s	7.4 KB
8	81 s	6.5 KB	7.3 s	8.6 KB
16	339 s	6.5 KB	20 s	11.1 KB
32	1534 s	6.5 KB	60 s	15.9 KB

Performance of a TCitH-MT-based signature scheme for which the security relies on the hardness of solving the MQ problem. Running times estimated for a RISC-V platform assuming the presence of a hardware accelerator for plain Keccak.

Much faster implementation

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

GGM with parallel trees

	No Tweak (using masking)		*Com. of PR sharings*		
Nb shares	*Signing time*	*Sig. Size*	*Signing time*	*Sig. Size*	
1	0,35 s	6.5 KB	0,35 s	6.5 KB	*4.5 KB*
2	5.2 s	6.5 KB	1.1 s	6.8 KB	*6.7 KB*
3	9.8 s	6.5 KB	1.9 s	7.1 KB	*8.9 KB*
4	19 s	6.5 KB	2.8 s	7.4 KB	*11.0 KB*
8	81 s	6.5 KB	7.3 s	8.6 KB	*19.7 KB*
16	339 s	6.5 KB	20 s	11.1 KB	*37.1 KB*
32	1534 s	6.5 KB	60 s	15.9 KB	*72.0 KB*

Small communication overhead

Performance of a TCitH-MT-based signature scheme for which Q problem. uming the ak.

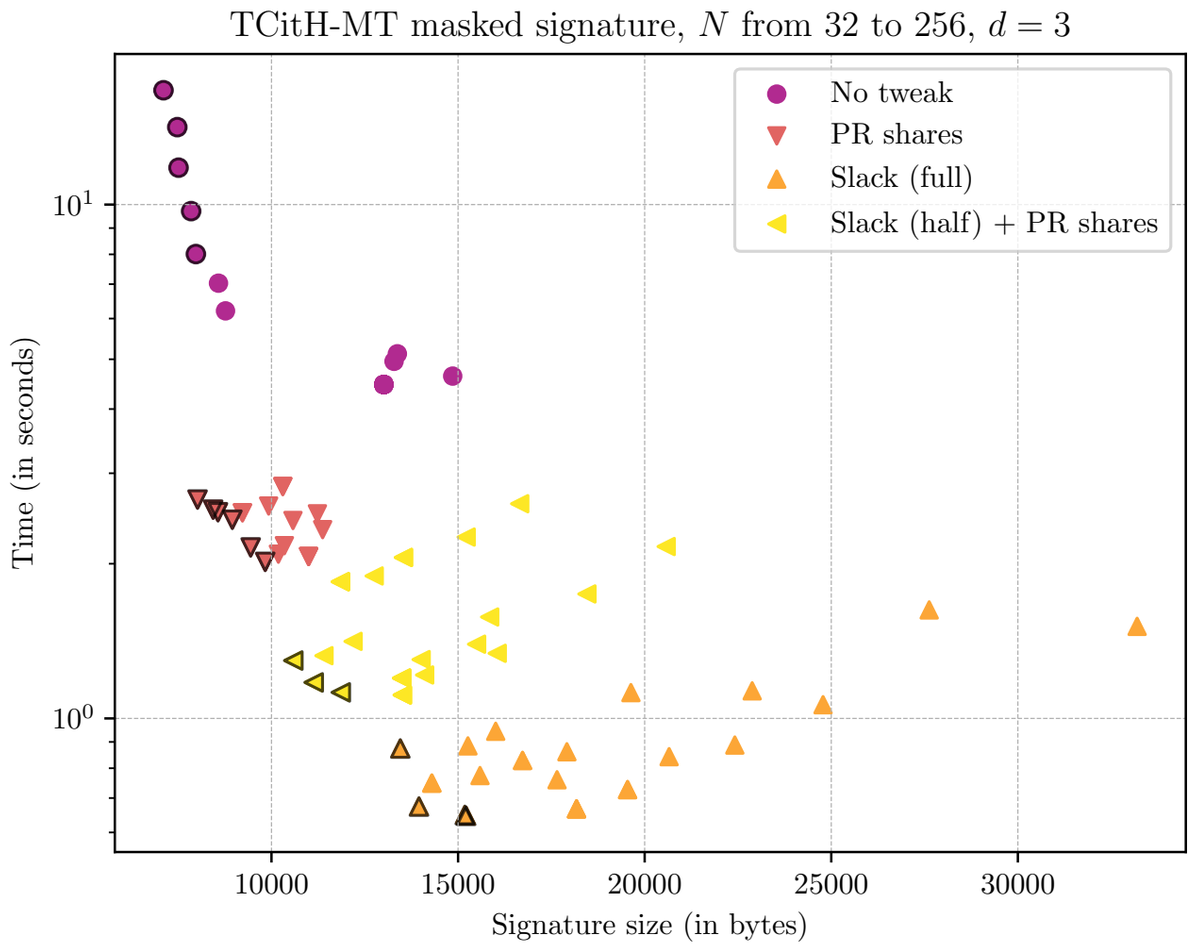
Conclusion

Conclusion

- Masking TCitH-based schemes (without tweak) is computational expensive.
- This issue can be mitigated by tweaking those schemes:
  - one can introduce a slack between the polynomial degree and the number of opened evaluations
  - when using GGM trees, one can use parallel trees.
  - when using Merkle trees, one can commit to pseudo-random sharings instead of committing to clear values.
- All those tweaks drastically reduce the running times, but introduce some variability in the signature size: the signature size depends on the masking order.

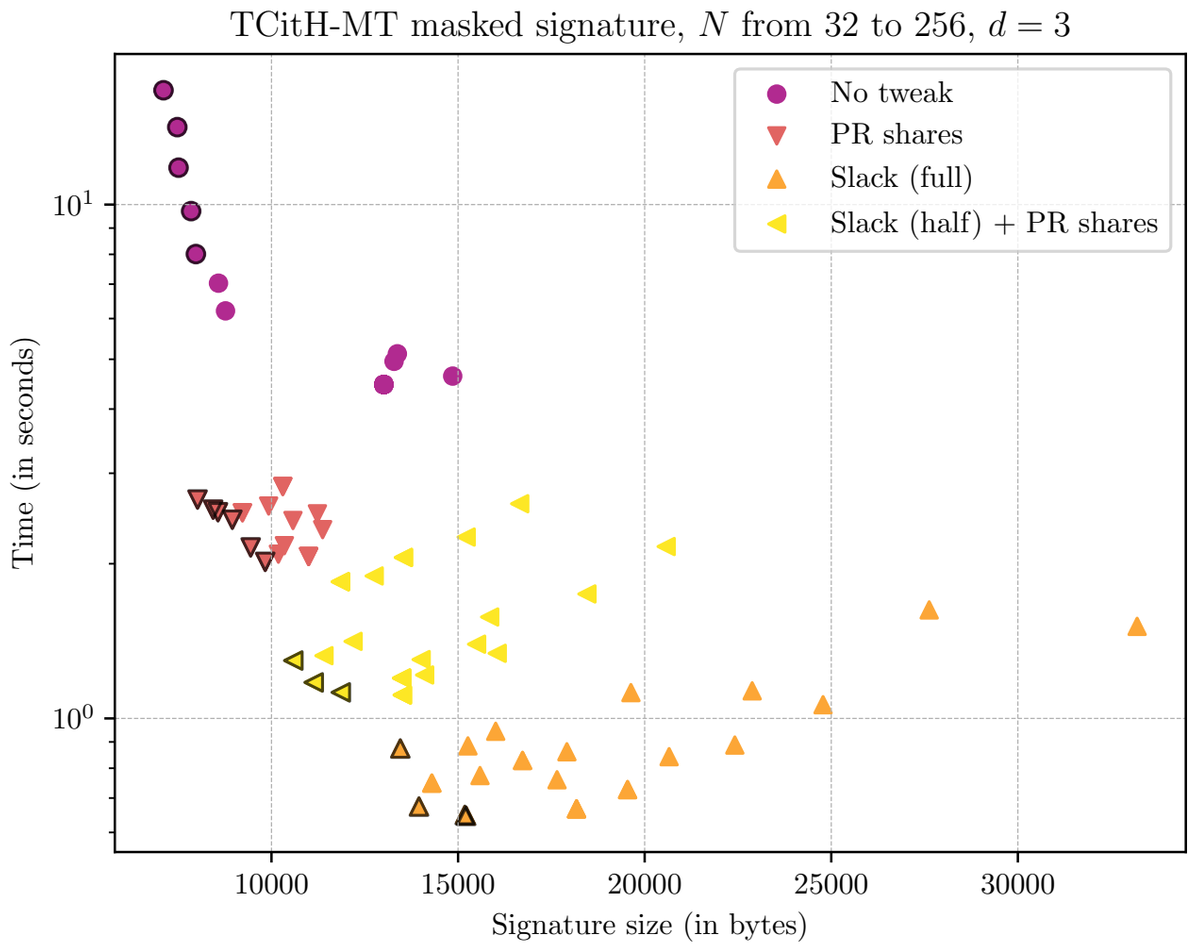
Conclusion

- One can combine slack with the other masking-friendly tweaks.



Conclusion

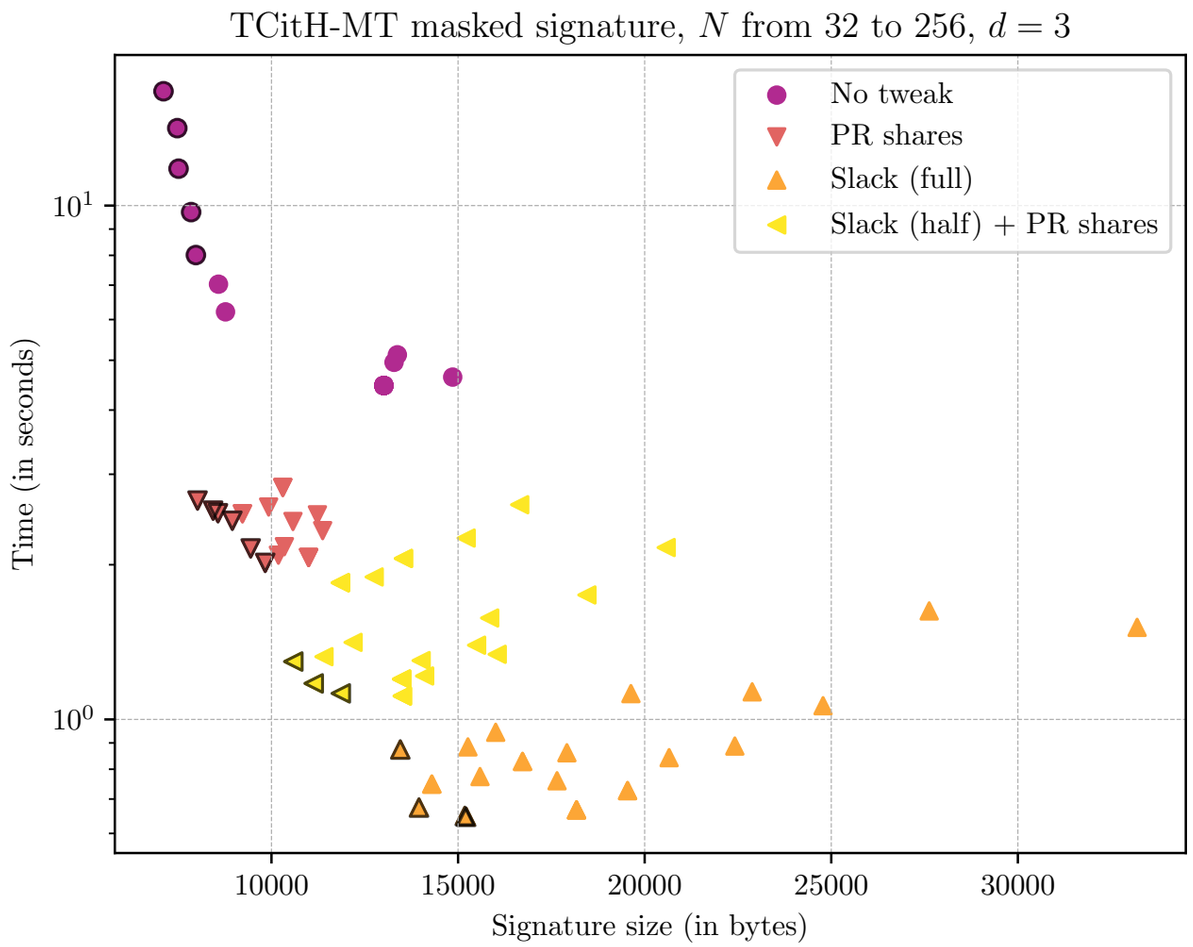
- One can combine slack with the other masking-friendly tweaks.



- Using Merkle trees leads to better trade-offs than using GGM trees, because the Merkle trees do not contain sensitive data.

Conclusion

- One can combine slack with the other masking-friendly tweaks.



- Using Merkle trees leads to better trade-offs than using GGM trees, because the Merkle trees do not contain sensitive data.

Thank you for your attention.