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

IACR eprint 2025/520

Thibauld Feneuil Matthieu Rivain **Auguste Warmé-Janville**

CryptoExperts
WE INNOVATE TO SECURE YOUR BUSINESS

JC2 – April 2nd, 2025

Intro - SCA

Side channel attacks exploit physical properties the machine. Attack model, d-probing model :

- Attacker is allowed to probe d different variables during the execution.
- Algorithm is d-probing secure $\Leftrightarrow$ any set of d **intermediate variables** during the execution is **independent** from the **secret**.

Intro - SCA

Side channel attacks exploit physical properties the machine. Attack model, d-probing model :

- Attacker is allowed to probe d different variables during the execution.
- Algorithm is d-probing secure $\Leftrightarrow$ any set of d **intermediate variables** during the execution is **independent** from the **secret**.

Protect variables with **masking** : encode a sensitive variable x into a random tuple

$$(x_0, \ldots, x_d)$$

of $d+1$ **masking shares**, s.t. $x = \sum_{j=0}^{d}$. We call d the **masking order** of $(x_0, \ldots, x_d)$.

Intro - SCA

Side channel attacks exploit physical properties the machine. Attack model, d-probing model :

- Attacker is allowed to probe d different variables during the execution.
- Algorithm is d-probing secure $\Leftrightarrow$ any set of d **intermediate variables** during the execution is **independent** from the **secret**.

Protect variables with **masking** : encode a sensitive variable x into a random tuple

$$(x_0, \ldots, x_d)$$

of $d+1$ **masking shares**, s.t. $x = \sum_{j=0}^{d}$. We call d the **masking order** of $(x_0, \ldots, x_d)$.

Gadgets are masked functions $\overline{F}$ that manipulate masked variables:

- If $y = F(x)$ then the corresponding gadget $\overline{F}$ satisfies:

$$y = \sum_{j=0}^{d} y_j, \text{ where } (y_0, \ldots, y_d) = \overline{F}(x_0, \ldots, x_d) \text{ and } x = \sum_{j=0}^{d} x_j \ .$$

- Linear operations: $O(d)$,
- Non-linear operations: $O(d^2)$.

Intro - TCitH

NIST call for PQ signatures (2023):

- Many new designs.
- Little work on these new schemes, especially against **side-channel attacks**.

Intro - TCitH

NIST call for PQ signatures (2023):

- Many new designs.
- Little work on these new schemes, especially against **side-channel attacks**.

MPC-in-the-Head is a method introduced in [IKOS07] to build signatures. Recent frameworks for MPCitH [FR23], [BBD+23].

- New techniques for better performance and flexibility...
- ... which apply to some 2023 NIST PQ candidates,
- Previous side-channel analysis does not fully apply to these new frameworks,
- $\Rightarrow$ The paper presents **SCA protections** for the **TCitH signature algorithms** from [FR23].

Signature scheme

We use the following notations:

- w: secret key,
- $F(X)$ s.t. $F(w) = 0$: public key
  - $F = (f_1, \ldots, f_m)$ is actually a system of multivariate quadratic $\mathcal{MQ}$ equations.

To make everything simpler, we used a different notation than in the article:

- $P_w(X) \Leftrightarrow$ underlying polynomial of the sharing $[\![w]\!]$,
- $P_w(e_i) \Leftrightarrow [\![w]\!]_i$.

Contributions

Original signature scheme:

- Theoretical leakage analysis of TCitH,
- Straightforward masked (unefficient) version of TCitH.

Contributions

Original signature scheme:

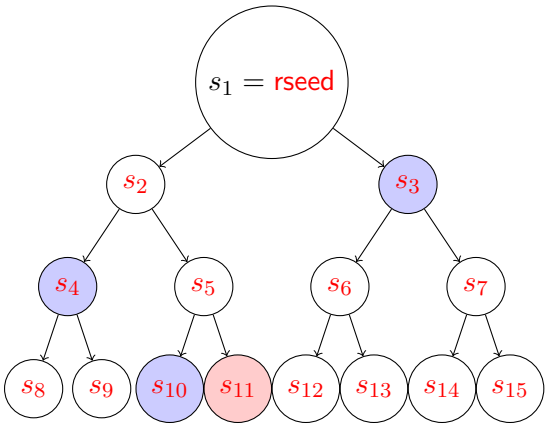
- Theoretical leakage analysis of TCitH,
- Straightforward masked (unefficient) version of TCitH.

Modified signature scheme, 3 tweaks:

- Parallel trees for the TCitH-GGM variant,
- Pseudorandom shares for the TCitH-MT variant,
- Add slack $\Rightarrow$ tweak the signature scheme parameters to increase its **native** side-channel resistance.
  - Benefits from the **secret sharing** used within the signature algorithm.

TCitH-GGM parallel trees

Colored functions and variables: need to be masked.

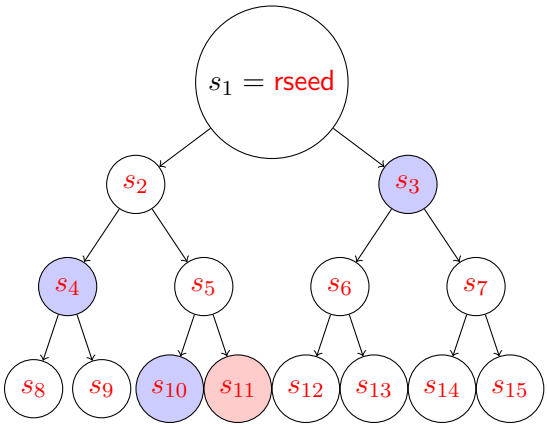


$\forall i \in [1, 2^{D-1}], (s_{2i} \mid s_{2i+1}) = \mathsf{PRG}(s_{2i})$

- GGM trees: widely used in MPCitH (short signatures, NIST 2^{nd} round).

TCitH-GGM parallel trees

Colored functions and variables: need to be masked.

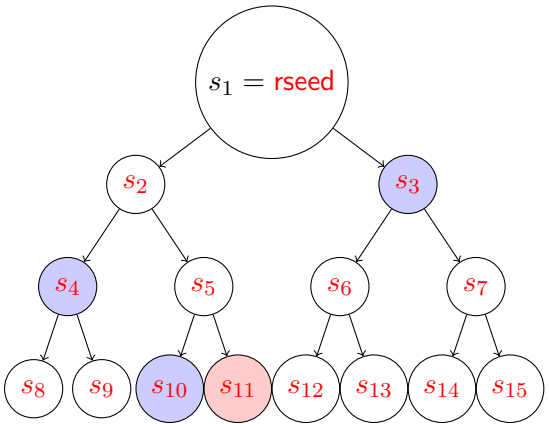


$\forall i \in [1, 2^{D-1}], (s_{2i} \mid s_{2i+1}) = \mathsf{PRG}(s_{2i})$

- GGM trees: widely used in MPCitH (short signatures, NIST 2^{nd} round).
- Untweaked: masked tree $\Rightarrow$ HUGE computation overhead.

TCitH-GGM parallel trees

Colored functions and variables: need to be masked.



$\forall i \in [1, 2^{D-1}], (s_{2i} \mid s_{2i+1}) = \text{PRG}(s_{2i})$

- GGM trees: widely used in MPCitH (short signatures, NIST 2^{nd} round).
- Untweaked: masked tree $\Rightarrow$ HUGE computation overhead.
- Tweak:
  - Compute $d + 1$ **unmasked** trees.
  - Include $d + 1$ sibling paths in sig.
- Size for a **masking order** $d = 7$:
  - Masked, untweaked: 4.2 kB,
  - Tweked: 21 kB.
- Conclusion:
  - GGM trees are not well suited for masking.

TCitH-MT signing algorithm (simplified)

Colored functions and variables: need to be masked.

Inputs: sk: w, pk: $F = \{f_i\}_i$.

1. Expand randomness as: $\{r_i\}_i = \mathsf{PRG}(\mathsf{rseed})$,
2. Using $\{r_i\}_i$, compute the random polynomials $M(X)$ and $P_w(X)$ s.t. $P_w(0) = w$.
3. Commit, for all $i \in [1 : N]$: $\mathsf{com}_i = \mathsf{Hash}(P_w(e_i), M(e_i))$.
4. Hash the commitments : $h_1 := \mathsf{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$.
5. Expand $\{\gamma_i\}_i$ from h_1.
6. Compute the polynomial $Q(X) = M(X) + \sum_i \gamma_i \cdot f_i(P_w(X))$.
7. Compute $h_2 := \mathsf{Hash}(m, h_1, Q(X))$.
8. Expand the open evaluation points $I \subset [N]$ from h_2 s.t. $|I| = \ell$.
9. Output the signature:

$$\mathsf{sig} := h_1 \mid h_2 \mid \big((P_w(e_i), M(e_i))_{i \in I}, (\mathsf{com}_j)_{j \in [N]}, Q(X)\big)$$

TCitH-MT signing algorithm (simplified)

Colored functions and variables: need to be masked.

Inputs: sk: w, pk: $F = \{f_i\}_i$.

1. Expand randomness as: $\{r_i\}_i = \mathsf{PRG}(\mathsf{rseed})$, can be made $\Rightarrow O(d)$ simply
2. Using $\{r_i\}_i$, compute the random polynomials $M(X)$ and $P_w(X)$ s.t. $P_w(0) = w$.
3. Commit, for all $i \in [1 : N]$: $\mathsf{com}_i = \mathsf{Hash}(P_w(e_i), M(e_i))$.
4. Hash the commitments : $h_1 := \mathsf{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$.
5. Expand $\{\gamma_i\}_i$ from h_1.
6. Compute the polynomial $Q(X) = M(X) + \sum_i \gamma_i \cdot f_i(P_w(X))$.
7. Compute $h_2 := \mathsf{Hash}(m, h_1, Q(X))$.
8. Expand the open evaluation points $I \subset [N]$ from h_2 s.t. $|I| = \ell$.
9. Output the signature:

$$\mathsf{sig} := h_1 \mid h_2 \mid \big((P_w(e_i), M(e_i))_{i \in I}, (\mathsf{com}_j)_{j \in [N]}, Q(X)\big)$$

TCitH-MT signing algorithm (simplified)

Colored functions and variables: need to be masked.

Inputs: sk: w, pk: $F = \{f_i\}_i$.

1. Expand randomness as: $\{r_i\}_i = \mathsf{PRG}(\mathsf{rseed})$,
2. Using $\{r_i\}_i$, compute the random polynomials $M(X)$ and $P_w(X)$ s.t. $P_w(0) = w$.
3. Commit, for all $i \in [1:N]$: $\mathsf{com}_i = \mathsf{Hash}(P_w(e_i), M(e_i))$.
4. Hash the commitments : $h_1 := \mathsf{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$.
5. Expand $\{\gamma_i\}_i$ from h_1.
6. Compute the polynomial $Q(X) = M(X) + \sum_i \gamma_i \cdot f_i(P_w(X))$. $\Rightarrow O(d^2)$ using $\mathcal{MQ}$ equations
7. Compute $h_2 := \mathsf{Hash}(m, h_1, Q(X))$.
8. Expand the open evaluation points $I \subset [N]$ from h_2 s.t. $|I| = \ell$.
9. Output the signature:

$$\mathsf{sig} := h_1 \mid h_2 \mid \big((P_w(e_i), M(e_i))_{i \in I}, (\mathsf{com}_j)_{j \in [N]}, Q(X)\big)$$

TCitH-MT signing algorithm (simplified)

Colored functions and variables: need to be masked.

Inputs: sk: w, pk: $F = \{f_i\}_i$.

1. Expand randomness as: $\{r_i\}_i = \mathsf{PRG}(\mathsf{rseed})$,
2. Using $\{r_i\}_i$, compute the random polynomials $M(X)$ and $P_w(X)$ s.t. $P_w(0) = w$.
3. Commit, for all $i \in [1 : N]$: $\mathsf{com}_i = \mathsf{Hash}(P_w(e_i), M(e_i))$. $\Rightarrow O(d^2)$: huge overhead
4. Hash the commitments : $h_1 := \mathsf{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$.
5. Expand $\{\gamma_i\}_i$ from h_1.
6. Compute the polynomial $Q(X) = M(X) + \sum_i \gamma_i \cdot f_i(P_w(X))$.
7. Compute $h_2 := \mathsf{Hash}(m, h_1, Q(X))$.
8. Expand the open evaluation points $I \subset [N]$ from h_2 s.t. $|I| = \ell$.
9. Output the signature:

$$\mathsf{sig} := h_1 \mid h_2 \mid \big((P_w(e_i), M(e_i))_{i \in I}, (\mathsf{com}_j)_{j \in [N]}, Q(X)\big)$$

TCitH-MT tweak: pseudorandom shares

Let's focus on $\mathsf{com}_i = \mathsf{Hash}(P_w(e_i), M(e_i))$. How to cut the complexity to $O(d)$?

Let $x_i := (P_w(e_i), M(e_i))$ and $(x_i^0, \ldots, x_i^d)$ denote the **masked** variable x_i to be committed.

Our solution, **tweak** the signature scheme and compute:

$$\begin{cases} \mathsf{com}_i^0 = \mathsf{Hash}(x_i^0) \\ \mathsf{com}_i^1 = \mathsf{Hash}(x_i^1) \\ \ldots \\ \mathsf{com}_i^d = \mathsf{Hash}(x_i^d) \ , \end{cases}$$

TCitH-MT tweak: pseudorandom shares

Let's focus on $\mathsf{com}_i = \mathsf{Hash}(P_w(e_i), M(e_i))$. How to cut the complexity to $O(d)$?

Let $x_i := (P_w(e_i), M(e_i))$ and $(x_i^0, \ldots, x_i^d)$ denote the **masked** variable x_i to be committed.

Our solution, **tweak** the signature scheme and compute:

$$\begin{cases} \mathsf{com}_i^0 = \mathsf{Hash}(x_i^0) \\ \mathsf{com}_i^1 = \mathsf{Hash}(x_i^1) \\ \ldots \\ \mathsf{com}_i^d = \mathsf{Hash}(x_i^d) \ , \end{cases}$$

The final commitment is given by:

$$\mathsf{com}_i := \mathsf{Hash}(\mathsf{com}_i^0, \ldots, \mathsf{com}_i^d) \ .$$

$\Rightarrow O(d)$ **complexity.**

TCitH-MT tweak: Impact on signature verification

Recall that $x_i := (P_w(e_i), M(e_i))$ and $(x_i^0, \ldots, x_i^d)$ denote the masked variable x_i to be committed.

To verify the signature:

- The algorithm needs to recompute $\{\mathsf{com}_i\}_{i \in I}$ given $(P_w(e_i), M(e_i))_{i \in I}$.

TCitH-MT tweak: Impact on signature verification

Recall that $x_i := (P_w(e_i), M(e_i))$ and $(x_i^0, \ldots, x_i^d)$ denote the masked variable x_i to be committed.

To verify the signature:

- The algorithm needs to recompute $\{\mathsf{com}_i\}_{i \in I}$ given $(P_w(e_i), M(e_i))_{i \in I}$.
- **Untweaked:** masked hash function, the verifier can simply use the usual verification algorithm.

TCitH-MT tweak: Impact on signature verification

Recall that $x_i := (P_w(e_i), M(e_i))$ and $(x_i^0, \ldots, x_i^d)$ denote the masked variable x_i to be committed.

To verify the signature:

- The algorithm needs to recompute $\{\mathsf{com}_i\}_{i \in I}$ given $(P_w(e_i), M(e_i))_{i \in I}$.
- **Untweaked:** masked hash function, the verifier can simply use the usual verification algorithm.
- **Tweaked:** the verifier needs to know the **complete encoding** $(x_i^0, \ldots, x_i^d)$ to recompute the commitment.
  - Indeed, we have $\mathsf{com}_i := \mathsf{Hash}(\mathsf{com}_i^0, \ldots, \mathsf{com}_i^d)$ where $\forall j \in [0 : d]$, $\mathsf{com}_i^j = \mathsf{Hash}(x_i^j)$.

TCitH-MT tweak: mask compression

Recall that $x_i := (P_w(e_i), M(e_i))$ and $(x_i^0, \ldots, x_i^d)$ denote the masked variable x_i to be committed.

$(x_i^0, \ldots, x_i^d)$ is too large to be included in the signature. Using the **mask compression** technique from [SR23] it is then converted as:

$$(y_i^0, \ldots, y_i^d) \text{ , with } y_i^0 = x_i^0 + \sum_{j=1}^{d} (x_i^j - y_i^j) \text{ ,}$$

where $y_i^j = \mathsf{PRG}(\mathsf{seed}_i^j)$ for all $j \in [1, d]$.

TCitH-MT tweak: mask compression

Recall that $x_i := (P_w(e_i), M(e_i))$ and $(x_i^0, \ldots, x_i^d)$ denote the masked variable x_i to be committed.

$(x_i^0, \ldots, x_i^d)$ is too large to be included in the signature. Using the **mask compression** technique from [SR23] it is then converted as:

$$(y_i^0, \ldots, y_i^d) \text{ , with } y_i^0 = x_i^0 + \sum_{j=1}^{d} (x_i^j - y_i^j) \text{ ,}$$

where $y_i^j = \mathsf{PRG}(\mathsf{seed}_i^j)$ for all $j \in [1, d]$.
We have that $(y_i^0, \ldots, y_i^d)$ is a valid encoding of x. Then, the compressed encoding:

$$(y_i^0, \mathsf{seed}_i^1, \ldots, \mathsf{seed}_i^d)$$

is included in the signature $\Rightarrow$ **$d \cdot \lambda$ bits added to the signature per evaluation point.**

Results

Benchmarks on a RISC-V platform with hardware Keccak (SHA3) acceleration.

	$d = 3$		$d = 7$	
	Signature Size	Signing Time	Signature Size	Signing Time
Masked (no tweak)	11.2 kB	4.7s	11.2 kB	16.4s

Results

Benchmarks on a RISC-V platform with hardware Keccak (SHA3) acceleration.

	$d = 3$		$d = 7$	
	Signature Size	Signing Time	Signature Size	Signing Time
Masked (no tweak)	11.2 kB	4.7s	11.2 kB	16.4s
Tweaked (same size)	11.1 kB	1.18s	11.1 kB	5.56s
Tweaked (short)	8 kB	2.66s	9.2 kB	6.9s
Tweaked (fast)	14 kB	0.67s	17.5 kB	1.75s
Speedup		4-7x		2-9x

Table: Masked TCitH-MT signature performance for masking orders $d \in \{3, 7\}$. Standard (untweaked) instance parameters are chosen to have the fastest signing time. Tweaked instances use different techniques detailed in the article to reduce the computation overhead.

Thank you!

Thank you for your attention !! Any questions ?

Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl.
Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head.
In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023.

Thibauld Feneuil and Matthieu Rivain.
Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments.
Cryptology ePrint Archive, Report 2023/1573, 2023.

Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai.
Zero-knowledge from secure multiparty computation.
In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

Markku-Juhani O. Saarinen and Mélissa Rossi.
Mask compression: High-order masking on memory-constrained devices.
Cryptology ePrint Archive, Paper 2023/1117, 2023.

https://eprint.iacr.org/2023/1117.