So Long, and Thanks for All the Seeds: Attacking GGM-trees in Post-quantum signatures

Gustavo Banegas[1] and Damya Bouizegarene[2]

[1] Inria, LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France
gustavo@cryptme.in
[2] Télécom Paris, Palaiseau, France
bouizegarene.damya@gmail.com

Abstract. Post-quantum signatures built from Fiat–Shamir transforms of zero-knowledge identification protocols—including LESS, CROSS, and MEDS—use GGM-tree seed compression to shrink signatures, revealing only the seeds of public rounds while hiding the challenge-dependent ones. This mechanism introduces a fault-attack surface: faulting the seed-publication can expose hidden seeds alongside their zero-knowledge responses, enabling recovery of secret information. We introduce the Generic ZK Seed Tree (GZKST), a unified abstraction of GGM-tree generation, challenge partitioning, and seed publication across these schemes, and formalize its correctness and seed-hiding invariants. We show that prior attacks on LESS-v1 and LESS-v2 violate the same invariant despite targeting different implementation layers and tree constructions, derive generic key-recovery algorithms from this view, and bound the number of effective faulted signatures needed for full recovery—only a few successful queries suffice for every MEDS parameter set. We further demonstrate the attacks in practice through clock-glitch fault injection against the MEDS reference implementation on an ARM Cortex-M4 (ChipWhisperer-Lite), identifying multiple exploitable surfaces in tree traversal and path construction that enable complete tree disclosure, partial subtree recovery, or leakage of hidden leaves.

Keywords: GGM tree · Zero-knowledge signatures · Fault attacks · Post-quantum cryptography · Seed compression

1 Introduction

Digital signatures form the backbone of many aspects of modern digital life, ranging from firmware updates and secure boot mechanisms to messaging applications and public-key infrastructures. Their primary purpose is to guarantee

* Author list in alphabetical order; see https://ams.org/profession/leaders/CultureStatement04.pdf. This work was supported by the HYPERFORM consortium, funded by France through Bpifrance, and by the France 2030 program under grant agreement ANR-22-PETQ-0008 PQ-TLS. Date of this document: 2026-06-24.

authenticity, integrity, and non-repudiation: only the holder of the secret signing key should be able to produce a valid signature for a given message, thereby ensuring that the message has not been modified. Due to their widespread deployment in security-critical systems, the compromise of a digital signature scheme may have severe consequences, including malicious software updates, device impersonation, and unauthorized access to sensitive information.

The security of classical signature schemes, such as RSA-based signatures and elliptic-curve-based signatures rely on the prime integer factorization and Elliptic Curve Discrete Logarithm Problem (ECDLP), would be compromised by a large quantum computer through the use of Shor's algorithm. This threat has motivated the development of post-quantum cryptography, whose objective is to design cryptographic schemes resistant to both classical and quantum adversaries. Among the most promising candidates are zero-knowledge-based signature schemes obtained via the Fiat–Shamir transformation [9,10]. Several recent constructions, including LESS [2], CROSS [3], and MEDS [8,7], rely on the Fiat–Shamir transform to construct digital signatures. However, these schemes suffer from large signature sizes. To reduce this issue, they employ GGM-tree-like seed-compression techniques, which reduce the signature size by compacting it to large collections of ephemeral randomness.

During signing, these schemes reveal only the seeds associated with public rounds while keeping challenge-dependent seeds hidden. Although this mechanism is essential for efficiency, recent works have shown that it also introduces a significant fault-attack surface: faults affecting the seed-publication logic may leak hidden seeds together with their corresponding zero-knowledge responses, enabling recovery of secret information and, in the end, full key recovery or forgery attacks [14,13].

1.1 Related Work

Several recent works study fault attacks on GGM- or seed-tree-based post-quantum signatures—LESS [2], CROSS [3], MEDS [8], and MPC-in-the-Head schemes [6,4]. ZKFault [13] faults the seed-publication of code-based ZK signatures (LESS-v1, extended to CROSS), disclosing hidden ephemeral seeds and recovering secret information. Fault-to-Forge [14] revisits LESS-v2, whose incomplete GGM-tree breaks that strategy, finding new injection points in the incomplete tree-generation and seed-publication and showing that secret-equivalent information suffices for universal forgery. More recently, [12] avoids the tree entirely, exploiting iteration-skip and loop-abort faults in the digest-preprocessing step before tree generation for high-probability full key recovery. In a different target, [15] attacks PERK [5] using fault attacks and attacking the tree generation.

Our work unifies these attacks under a single Generic ZK Seed Tree (GZKST) abstraction, formalizing the correctness and seed-hiding invariants they implicitly exploit, and additionally provides practical fault attacks against MEDS.

Our Contributions. This work provides a unified analysis of fault attacks against post-quantum signature schemes relying on seed-tree compression. Our main contributions are as follows:

- **Generic Seed-Tree Abstraction.** We introduce the *Generic ZK Seed Tree* (GZKST), a unified model capturing the seed-tree generation mechanisms used in schemes such as LESS-v2, MEDS, and CROSS.
- **Unified View of Existing Attacks.** We show that previously proposed fault attacks against LESS-v1 and LESS-v2 can be interpreted as violations of a common seed-hiding invariant.
- **Query Complexity Analysis.** We derive bounds on the number of faulted signatures required for secret-key recovery and show that only a small number of successful queries is needed for all MEDS parameter sets.
- **Practical Attacks on MEDS.** We show fault attacks against the MEDS reference implementation on an ARM Cortex-M4 using a ChipWhisperer-Lite, revealing several fault surfaces that enable complete or partial seed-tree disclosure.
- **Countermeasures for MEDS.** We propose and evaluate practical countermeasures against fault attacks.

2 Background

2.1 GGM trees

Pseudorandom functions. A central problem in cryptography is generating randomness that appears truly random while remaining reproducible from a secret key. A **pseudorandom function** (PRF) addresses this problem: it is a keyed function that, to any efficient adversary without the key, is indistinguishable from a truly random function. Intuitively, a PRF expands a short secret key into an exponentially large table of "random-looking" outputs without storing the table explicitly.

PRFs are fundamental primitives in symmetric cryptography and appear in the construction of message authentication codes (MACs), stream ciphers, and key-derivation functions. A standard construction of PRFs from one-way functions is the **Goldreich–Goldwasser–Micali (GGM) tree** [11], which builds a PRF by iteratively applying a pseudorandom generator along the branches of a binary tree indexed by the input bits. GGM trees are the main PRF construction studied in this work.

Definition 1 (GGM Tree). *Let* $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *be a length-doubling pseudorandom generator, decomposed as* $\mathsf{G}(x) = (\mathsf{G}_0(x), \mathsf{G}_1(x))$*, where* G_0 *and* G_1 *each output* λ *bits. Given a root seed* $\mathsf{seed}_r \in \{0,1\}^\lambda$ *and a depth* d*, the* GGM tree *is a complete binary tree of depth* d *in which:*

- *the root holds* seed_r*;*

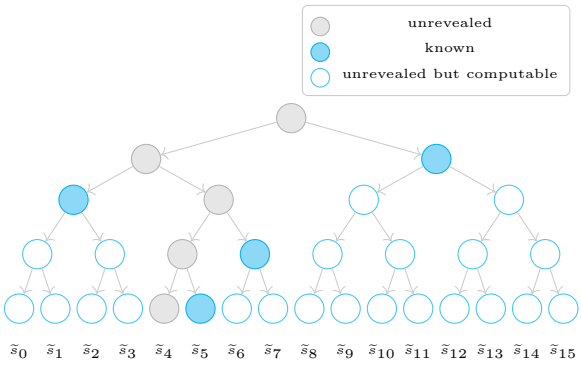


Fig. 1: Receiver's GGM tree of depth 4. Given only the blue nodes and the PRNG function, the receiver can retrieve all leaves except x_4.

- *each internal node v storing seed s generates its left child as $\mathsf{G}_0(s)$ and its right child as $\mathsf{G}_1(s)$;*
- *the 2^d leaves are the G outputs at depth d.*

The crucial property exploited by ZK signature schemes is the *punctured-PRF property*: given all leaf seeds *except* leaf i, it is computationally infeasible to recover the seed at leaf i, provided G is a secure pseudorandom generator. This makes the GGM tree suitable for hiding one or more seeds while revealing the rest, as shown in Figure 1.

Incomplete GGM *trees.* When the number of required leaves t is not a power of 2, a scheme may use an *incomplete* tree: leaves appear at multiple levels, and internal nodes near the bottom may have fewer than two children or act as leaves themselves. This reduces memory and computation, but the additional index-tracking logic introduces new fault surfaces. The scheme may also use the first t leaves of the tree.

2.2 Attack target

This section provides an overview of the signature scheme to which we apply our generalized model and against which we carry out our attacks.

The MEDS Signature Scheme. The Matrix Equivalence Digital Signature (MEDS) scheme [8] is a code-based post-quantum primitive whose security relies on the hardness of the Matrix Code Equivalence (MCE) Problem. Given a public matrix pair (G_0, G_1), the MCEP asks an adversary to find a secret tuple $(A, B) \in \mathrm{GL}_k(q) \times \mathrm{GL}_n(q)$ such that $G_1 = A \cdot G_0 \cdot B$.

To transform this structural identification problem into a non-interactive digital signature, MEDS applies the classical Fiat–Shamir paradigm to an underlying 3-pass identification protocol across t parallel execution rounds. In each round, the

signer expands an ephemeral leaf seed into a randomized matrix pair $(\tilde{A}_i, \tilde{B}_i)$, computing a commitment hash based on their product. To minimize the signature size, all t ephemeral seeds are hierarchically derived from a single master seed using a binary GGM seed tree.

During the execution of the Fiat–Shamir transform, the message and the accumulated commitments are hashed to generate a challenge vector of weight w. This vector splits the execution rounds into two validation paths: for w active rounds, the signer publishes a masked variant of the secret matrices while keeping the corresponding leaf seeds strictly hidden. For the remaining $t - w$ rounds, the signer discloses the internal leaf seeds, allowing the verifier to recompute the commitments.

Since Section 4 demonstrates a practical attack on the reference implementation of MEDS, we describe the algorithms of key generation, signing and verification in more detail below. Table 1 shows the parameters for MEDS [8].

Table 1: Cryptographic parameters for MEDS [8]. ($\star$: MEDS uses $n{=}m{=}k$.)

Scheme	Parameter set	t	w	s	Sec.	Extra
MEDS*	MEDS-9923	1152	14	4	I	n=14, q=4093
	MEDS-13220	192	20	5	I	n=14, q=4093
	MEDS-41711	608	26	4	III	n=22, q=4093
	MEDS-69497	160	36	5	III	n=22, q=4093
	MEDS-134180	192	52	5	V	n=30, q=2039
	MEDS-167717	112	66	6	V	n=30, q=2039

Key Generation. As detailed in Algorithm 1, key generation initializes a master secret seed δ to derive a public matrix seed σ_{G_0}, which expands into a baseline systematic matrix $G_0 \in \mathbb{F}_q^{k \times mn}$, and a private seed σ. For each of the $s-1$ rounds, a private randomized matrix $T_i \in \mathrm{GL}_k(q)$ is generated to multiply G_0, yielding an intermediate matrix G_0'. The Solve function then parses G_0' into a linear system over $\mathbb{F}_q$ bounded by an explicit scalar constraint $a_{m-1,m-1}$, which is uniquely resolved by the secret equivalence pair (A_i, B_i^{-1}). Finally, the target public matrix G_i is computed via the equivalence transformation $\pi_{A_i,B_i}(G_0)$ and evaluated by the SF operator. The resulting public key comprises σ_{G_0} alongside the compressed representations of $(G_1, \ldots, G_{s-1})$, while the secret key retains δ, σ_{G_0}, and the matrix inverses (A_i^{-1}, B_i^{-1}).

Signing. Algorithm 2 draws a fresh master seed δ, expands it into a tree-root seed ρ and salt α, and materialises all t leaf seeds via $\mathsf{SeedTree}_t(\rho, \alpha)$. Each σ_i yields an ephemeral pair $(\tilde{A}_i, \tilde{B}_i)$ and commitment $\tilde{G}_i = \mathsf{SF}(\pi_{\tilde{A}_i,\tilde{B}_i}(G_0))$; hashing their non-systematic parts with the message gives the digest d, parsed into the fixed-weight challenge $(h_0, \ldots, h_{t-1})$. For each non-zero h_i the signer releases

Algorithm 1: MEDS Key Generation (simplified)

Input: —
Output: Public key pk, secret key sk

$\delta \xleftarrow{\$} \mathcal{B}^{\ell_{\text{sec\_seed}}}$
$\sigma_{G_0}, \sigma \leftarrow \mathsf{XOF}(\delta, \ell_{\text{pub\_seed}}, \ell_{\text{sec\_seed}})$
$G_0 \leftarrow \mathsf{ExpandSystMat}(\sigma_{G_0})$
for $i = 1$ **to** $s-1$ **do**
  $\sigma_a, \sigma_{T_i}, \sigma \leftarrow \mathsf{XOF}(\sigma, \ell_{\text{sec\_seed}}, \ell_{\text{sec\_seed}}, \ell_{\text{sec\_seed}})$
  $T_i \leftarrow \mathsf{ExpandInvMat}(\sigma_{T_i}, k)$
  $a_{m-1,m-1} \leftarrow \mathsf{ExpandFqs}(\sigma_a, 1)$
  $G'_0 \leftarrow T_i G_0$
  $(A_i, B_i^{-1}) \leftarrow \mathsf{Solve}(G'_0, a_{m-1,m-1})$ // Fail ⇒ retry ln. 5
  $G_i \leftarrow \mathsf{SF}(\pi_{A_i,B_i}(G_0))$ // ⊥⇒ retry ln. 5
$\mathsf{pk} \leftarrow (\sigma_{G_0} \mid \mathsf{CompressG}(G_1) \mid \cdots \mid \mathsf{CompressG}(G_{s-1}))$
$\mathsf{sk} \leftarrow (\delta \mid \sigma_{G_0} \mid \mathsf{Compress}(A_1^{-1}) \mid \cdots \mid \mathsf{Compress}(A_{s-1}^{-1}) \mid \mathsf{Compress}(B_1^{-1}) \mid \cdots \mid \mathsf{Compress}(B_{s-1}^{-1}))$
return pk, sk

Algorithm 2: MEDS Signing (simplified)

Input: Secret key sk, message m
Output: Signed message m_s

Parse sk to recover σ_{G_0}, A_i^{-1}, B_i^{-1}, $i = 1, \ldots, s-1$
$G_0 \leftarrow \mathsf{ExpandSystMat}(\sigma_{G_0})$
$\delta \xleftarrow{\$} \mathcal{B}^{\ell_{\text{sec\_seed}}}$
$\rho, \alpha \leftarrow \mathsf{XOF}(\delta, \ell_{\text{tree\_seed}}, \ell_{\text{salt}})$
$(\sigma_0, \ldots, \sigma_{t-1}) \leftarrow \mathsf{SeedTree}_t(\rho, \alpha)$
for $i = 0$ **to** $t-1$ **do**
  $\sigma'_i \leftarrow (\alpha \mid \sigma_i \mid \mathsf{ToBytes}(2^{1+\lceil \log_2 t \rceil} + i, 4))$
  $\sigma_{\tilde{A}_i}, \sigma_{\tilde{B}_i} \leftarrow \mathsf{XOF}(\sigma'_i, \ell_{\text{pub\_seed}}, \ell_{\text{pub\_seed}})$
  $\tilde{A}_i \leftarrow \mathsf{ExpandInvMat}(\sigma_{\tilde{A}_i}, m)$; $\tilde{B}_i \leftarrow \mathsf{ExpandInvMat}(\sigma_{\tilde{B}_i}, n)$
  $\tilde{G}_i \leftarrow \mathsf{SF}(\pi_{\tilde{A}_i,\tilde{B}_i}(G_0))$ // ⊥⇒ resample
  $d \leftarrow H(\mathsf{Compress}(\tilde{G}_0[;k, mn-1]) \mid \cdots \mid \mathsf{Compress}(\tilde{G}_{t-1}[;k, mn-1]) \mid m)$
$(h_0, \ldots, h_{t-1}) \leftarrow \mathsf{ParseHash}_{s,t,w}(d)$
for $i = 0$ **to** $t-1$ ***with*** $h_i > 0$ **do**
  $\mu_i \leftarrow \tilde{A}_i A_{h_i}^{-1}$; $\nu_i \leftarrow B_{h_i}^{-1} \tilde{B}_i$
  Append $(\mathsf{Compress}(\mu_i) \mid \mathsf{Compress}(\nu_i))$ to v
$p \leftarrow \mathsf{SeedTreeToPath}_t(h_0, \ldots, h_{t-1}, \rho, \alpha)$
return $m_s \leftarrow (v \mid p \mid d \mid \alpha \mid m)$

the coset representative $(\mu_i, \nu_i) = (\tilde{A}_i A_{h_i}^{-1}, B_{h_i}^{-1} \tilde{B}_i)$, and the remaining seeds are revealed compactly in the seed-tree path p.

Verification. Algorithm 3 reconstructs $G_0, \ldots, G_{s-1}$ from the public key, recovers the path p, digest d, salt α, and the w response pairs from m_s, recomputes the challenge from d, and runs $\mathsf{PathToSeedTree}_t$ to recover the $t-w$ disclosed seeds. For each round, if $h_i > 0$ it applies the published (μ_i, ν_i) to G_{h_i} (checking invertibility); if $h_i = 0$ it re-expands $(\hat{A}_i, \hat{B}_i)$ from the recovered seed. The signature is accepted iff the recomputed digest d' equals d.

3 Abstract Model

We now define a unified abstraction that captures the common structure of the GGM-tree mechanism across the schemes discussed in Section 2.

Definition 2 (GZKST). *A* Generic ZK Seed Tree *(*GZKST*) is a tuple* $\Pi =$ (Setup, Expand, Chal, Decide, Publish, Recon) *parametrised by:*

- λ*: the security parameter;*

Algorithm 3: MEDS Verification (simplified)

Input: Public key pk, signed message m_s
Output: Message m or $\perp$

$G_0 \leftarrow \mathsf{ExpandSystMat}(\mathsf{pk}[0, \ell_{\mathrm{pub\_seed}}-1])$; $G_1, \ldots, G_{s-1} \leftarrow \mathsf{DecompressG}(\mathsf{pk})$
Parse m_s: pairs $((\mu_i, \nu_i))_{h_i>0}$, path p, digest d, salt α, message m
$(h_0, \ldots, h_{t-1}) \leftarrow \mathsf{ParseHash}_{s,t,w}(d)$
$(\sigma_0, \ldots, \sigma_{t-1}) \leftarrow \mathsf{PathToSeedTree}_t(h_0, \ldots, h_{t-1}, p, \alpha)$
for $i = 0$ **to** $t-1$ **do**
  if $h_i > 0$
    $(\mu_i, \nu_i) \leftarrow$ next response pair from m_s
    $\hat{G}_i \leftarrow \mathsf{SF}(\pi_{\mu_i,\nu_i}(G_{h_i}))$
    if $\mu_i \notin \mathrm{GL}_m(q)$ ***or*** $\nu_i \notin \mathrm{GL}_n(q)$ ***or*** $\hat{G}_i = \perp$ **return** $\perp$
  else
    $\sigma'_i \leftarrow (\alpha \mid \sigma_i \mid \mathsf{ToBytes}(2^{1+\lceil \log_2 t \rceil} + i, 4))$
    $\hat{A}_i \leftarrow \mathsf{ExpandInvMat}(\mathsf{XOF}(\sigma'_i)_1, m)$; $\hat{B}_i \leftarrow \mathsf{ExpandInvMat}(\mathsf{XOF}(\sigma'_i)_2, n)$
    $\hat{G}_i \leftarrow \mathsf{SF}(\pi_{\hat{A}_i,\hat{B}_i}(G_0))$
$d' \leftarrow H\big(\mathsf{Compress}(\hat{G}_0[;k, mn-1]) \mid \cdots \mid \mathsf{Compress}(\hat{G}_{t-1}[;k, mn-1]) \mid m\big)$
if $d = d'$ **return** m
return $\perp$

- $t \in \mathbb{N}$*: the number of parallel ZK repetitions;*
- $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$*: a secure pseudorandom generator;*
- $\mathcal{S}$*: the* secret type *(e.g., permutations for LESS, matrix pairs for MEDS, party shares for MPCitH).*

The six algorithms are:

1. $\mathsf{Setup}(1^\lambda) \to (\mathsf{seed}_r, \mathsf{salt})$*: samples a master seed* $\mathsf{seed}_r \xleftarrow{\$} \{0,1\}^\lambda$ *and a public salt.*
2. $\mathsf{Expand}(\mathsf{seed}_r, \mathsf{salt}) \to (\mathcal{T}, (\widetilde{s}_0, \ldots, \widetilde{s}_{t-1}))$*: builds a binary tree* $\mathcal{T}$ *by top-down* G*-expansion from* seed_r*. Each leaf* $i \in \{0, \ldots, t-1\}$ *holds a seed* $\widetilde{s}_i$*. Tree topology may be complete or incomplete.*
3. $\mathsf{Chal}(\mathsf{cmt}, \mathsf{msg}) \to \mathbf{c} = (c_0, \ldots, c_{t-1})$*: derives a Fiat–Shamir challenge from the commitment and message.*
4. $\mathsf{Decide}(\mathbf{c}) \to (\mathcal{H}, \mathcal{R})$*: partitions* $\{0, \ldots, t-1\}$ *into the hidden set* $\mathcal{H} = \{i : c_i \neq 0\}$ *and the revealed set* $\mathcal{R} = \{i : c_i = 0\}$*.*
5. $\mathsf{Publish}(\mathcal{T}, \mathcal{H}) \to \mathsf{path}$*: returns the minimal set of nodes from which every* $\widetilde{s}_i$ *with* $i \in \mathcal{R}$ *is reconstructible while no* $\widetilde{s}_i$ *with* $i \in \mathcal{H}$ *is derivable.*
6. $\mathsf{Recon}(\mathsf{path}, \mathbf{c}, \mathsf{salt}) \to \{\widetilde{s}_i : i \in \mathcal{R}\}$*: verifier-side reconstruction of revealed leaf seeds from* path*.*

The Flag Structure. The flag structure $\mathcal{F}$ produced internally by $\mathsf{Publish}$ is the key intermediate object. It is a labelling $\mathcal{F} : V(\mathcal{T}) \to \{0,1\}$ where:

- $\mathcal{F}(v) = 0$: node v is *publishable*—either directly included in path or derivable from a published ancestor;

– $\mathcal{F}(v) = 1$: node v is *hidden*—it must not appear in path and must not be derivable from published nodes.

The flag structure satisfies two invariants:

Invariant 1 (Correctness). For every $i \in \mathcal{R}$ there exists a node v on the path from the root to leaf $\widetilde{s}_i$ such that $\mathcal{F}(v) = 0$ and $\mathcal{F}(\mathsf{parent}(v)) = 1$.

Invariant 2 (ZK hiding). For every $i \in \mathcal{H}$, no ancestor of leaf $\widetilde{s}_i$ satisfies $\mathcal{F}(v) = 0$.

The ZK property of the signature scheme is contingent on Invariant 2 holding for *all* $i \in \mathcal{H}$. A fault that changes $\mathcal{F}(v)$ from 1 to 0 for any v that is an ancestor of some $\widetilde{s}_i$ with $i \in \mathcal{H}$ *breaks Invariant 2* and may allow an attacker to reconstruct $\widetilde{s}_i$. We provide an example of the GGM tree, together with the concepts of the hidden tree and flag structure, in Figure 2.

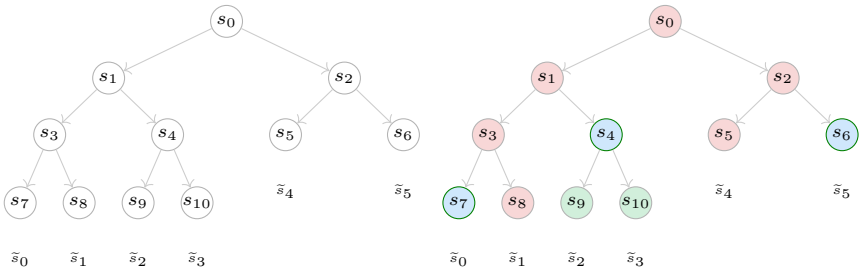
3.1 Mapping Schemes to the GZKST Model

We now establish that the MEDS signature scheme introduced in Section 2, as well as two other zero-knowledge-base postquantum signature schemes, LESS-V2 [2] and CROSS [2], are valid GZKST instances in the sense of Definition 2. We exhibit the concrete instantiation of the six algorithms and verify that Invariants 1 and 2 hold by construction.

Proposition 1 (MEDS is a GZKST instance). *MEDS instantiates Definition 2 with secret type* $\mathcal{S} = \{(A_j^{-1}, B_j^{-1})\}_{j=1}^{s-1}$, *$t$ parallel repetitions, and a* GGM *tree of height* $\lceil \log_2 t \rceil$ *(complete when t is a power of two, incomplete otherwise; only the first t of the* $2^{\lceil \log_2 t \rceil}$ *leaves are used). Invariants 1 and 2 hold by the* $\mathsf{SeedTreeToPath}$ *construction regardless of whether the tree is complete or incomplete; see Appendix A.1.*

The six algorithms instantiate as follows. Setup derives $(\mathsf{seed}_r, \mathsf{salt})$ from a single XOF call, and Expand builds the height-$\lceil \log_2 t \rceil$ tree by hashing each parent with the salt and node index, returning the first t leaves (the rightmost $2^{\lceil \log_2 t \rceil} - t$ are unused when t is not a power of two). Chal expands each leaf into ephemeral matrices, hashes the compressed generators with m, and parses the digest via $\mathsf{ParseHash}_{s,t,w}$ into $\mathbf{h} \in \{0, \ldots, s-1\}^t$ of weight w; Decide then sets $\mathcal{H} = \{i : h_i \neq 0\}$, $\mathcal{R} = \{i : h_i = 0\}$. Finally, $\mathsf{Publish} = \mathsf{SeedTreeToPath}_t$ applies the bottom-up rule $\mathcal{F}(v) = \mathcal{F}(\ell) \vee \mathcal{F}(r)$ and serialises the boundary nodes (Invariant 1) in depth-first order, while $\mathsf{Recon} = \mathsf{PathToSeedTree}_t$ re-expands them downward, leaving the positions $i \in \mathcal{H}$ empty.

Remark 1. This $\mathsf{Publish}$ description follows the MEDS submission [7]; the reference implementation uses a different algorithm, detailed in Section 4.



(a) Expanded reference GGM seed tree.

(b) After Chal, Decide, Publish; $\mathbf{b} = \{0, 1, 0, 0, 1, 0\}$, $\mathsf{path} = \{s_4, s_6, s_7\}$.

Fig. 2: GGM seed tree ($t = 6$): structure (a) and flag state (b). In (b): pink nodes ($\mathcal{F}$=1, hidden subtree); green nodes ($\mathcal{F}$=0, fully revealed subtree); blue-bordered nodes ($\mathcal{F}$=0, in path).

The same instantiation extends to LESS-v2 and CROSS, yielding Propositions 2 and 3; the proofs follow the MEDS argument (Appendix A.1).

Proposition 2 (LESS-v2 is a GZKST instance). *LESS-v2[2] instantiates Definition 2 with secret type $\mathcal{S} = \{\pi_1, \ldots, \pi_{s-1}\}$ (the $s-1$ secret monomial permutations), t parallel repetitions, and an* incomplete GGM *tree of height $\lceil \log_2 t \rceil$. Invariants 1 and 2 hold by construction of the three-phase flag tree.*

Remark 2 (Invariants under incomplete trees). When the tree is incomplete, some leaves appear at depth $\lceil \log_2 t \rceil - 1$. The LESS-v2 index-tracking arrays keep LabelLeaves and ComputeSeeds on the correct nodes regardless of level, so both invariants hold for any tree shape the expansion produces.

Proposition 3 (CROSS is a GZKST instance). *CROSS [3] instantiates Definition 2 with secret type $\mathcal{S} = \{\mathbf{e}\}$ (the R-SDP solution), t parallel repetitions, and a complete* GGM *tree. The seed-publishing component satisfies Invariants 1 and 2 by the same argument as MEDS (Proposition 1). The additional Merkle proof over $\{\mathsf{cmt}_0[i]\}_{i \in \mathcal{H}}$ is a CROSS-specific commitment-authenticity mechanism orthogonal to the* GZKST *seed-hiding invariants.*

3.2 Oracle Abstraction

To model fault attacks, we extend the GZKSTabstraction with a faulted signing oracle and a secret-extraction procedure.

Definition 3 (Faulted Signing Oracle). *Let Π = (Setup, Expand, Chal, Decide, Publish, Recon) be a* GZKST*instance and let* (sk, pk) *be a fixed key pair. For a node $v^* \in V(\mathcal{T})$, the oracle $\mathcal{O}^{v^*}(\mathsf{sk}, \mathsf{pk})$ signs a message* msg *as usual except that, during* Publish$(\mathcal{T}, \mathcal{H})$*, it forces $\mathcal{F}(v^*) \leftarrow 0$ before the normal upward propagation step. The oracle outputs the resulting (possibly faulted) signature* sig^*.

A fault is *effective* if v^* originally satisfies $\mathcal{F}(v^*) = 1$ and is an ancestor of some hidden leaf $\widetilde{s}_i$ with $i \in \mathcal{H}$. In this case, the published path reveals a node from which $\widetilde{s}_i$ is derivable, violating Invariant 2. Otherwise, the fault is *ineffective*.

Remark 3. Definition 3 captures several practical fault mechanisms, including instruction skips, stuck-at-zero faults, bit-flips, and clock glitches [13,14].

Remark 4. We assume access to an algorithm `SolverS` that efficiently recovers secrets of type $\mathcal{S}$ once sufficient information is revealed. For LESS, for example, this follows from Lemma 1 of [13].

For every hidden round $i \in \mathcal{H}$, a valid signature reveals the response resp_i while keeping $\widetilde{s}_i$ hidden. An effective fault breaks this separation: the attacker obtains both resp_i and $\widetilde{s}_i$. We call this event *dual revelation*; it is the basic primitive underlying all secret-recovery attacks considered in this work.

Secret extraction. We extend the GZKST tuple with a seventh, scheme-specific procedure:

Definition 4 (Extract). $\mathsf{Extract}(\widetilde{s}_i, \mathsf{resp}_i, \mathsf{pk}) \to \mathsf{sk}'$*: given the leaked seed* $\widetilde{s}_i$ *and the ZK response* resp_i *for the same round* $i \in \mathcal{H}$*, recover the (partial) secret* $\mathsf{sk}' \subseteq \mathcal{S}$*.*

For the schemes studied in this work, Extract can be construct as:

- **MEDS**. For a hidden round i with $h_i = j$, the signer publishes $(\mu_i, \nu_i) = (\widetilde{A}_i A_j^{-1},\ B_j^{-1}\widetilde{B}_i)$ (consistent with the background description in §2.2). Expanding $\widetilde{s}_i$ yields $(\widetilde{A}_i, \widetilde{B}_i)$, so

$$A_j^{-1} \;=\; \widetilde{A}_i^{-1} \cdot \mu_i, \qquad B_j^{-1} \;=\; \nu_i \cdot \widetilde{B}_i^{-1}.$$

  Both operations are direct matrix multiplications and inversions over $\mathbb{F}_q$; no column-reconstruction step is required.
- **LESS-v2**. The response is a coset representative $\mathsf{CosetRep}(\pi_{b[i]} \circ \tilde{\pi}_i^*)$. Expanding $\widetilde{s}_i$ gives the partial monomial matrix $\tilde{\pi}_i^*$, and the secret permutation $\pi_{b[i]}$ is recovered via the column-extraction argument of Lemma 1 in [14].
- **CROSS**. The response encodes $\mathbf{v}_i = \mathbf{e} \star (\mathbf{e}'_i)^{-1}$. Expanding $\widetilde{s}_i$ gives $\mathbf{e}'_i$, so $\mathbf{e} = \mathbf{v}_i \star \mathbf{e}'_i$ directly.

Full key recovery. When $|\mathcal{S}| = s - 1 > 1$ (as in MEDS with $s \geq 3$, and LESS with $s \geq 3$), a single effective fault recovers only the secret indexed by h_i. Full recovery requires observing each index $j \in \{1, \ldots, s-1\}$ at least once.

Proposition 4 (Correctness of Algorithm 4). *Let* Π *be a* GZKST *instance, suppose* $v^* = \text{root}$ *(so the fault exposes all* w *hidden leaves, making* $L = w$ *deterministic), and let* $p_{\min} = 1 - \left(\frac{s-2}{s-1}\right)^w$*. Algorithm 4 terminates and outputs*

Algorithm 4: GZKST key recovery from faulted oracle

Input : Faulted oracle $\mathcal{O}^{v^*}$, public key pk
Output : Recovered secret key $\widehat{\mathsf{sk}}$

$\mathsf{Collected} \leftarrow \emptyset$
while $|\mathsf{Collected}| < |\mathcal{S}|$ **do**
  $\mathsf{sig}^* \leftarrow \mathcal{O}^{v^*}(\mathsf{msg})$ for an arbitrary msg
  if sig^* *is* effective
    for *each* $i \in \mathsf{leaked}(\mathsf{sig}^*)$ **do**
      $\mathsf{sk}' \leftarrow \mathsf{Extract}(\widetilde{s}_i, \mathsf{resp}_i, \mathsf{pk})$
      $\mathsf{Collected} \leftarrow \mathsf{Collected} \cup \{(j, \mathsf{sk}')\}$ where $j = c_i$
return $\widehat{\mathsf{sk}} \leftarrow \mathsf{Collect}$

$\widehat{\mathsf{sk}} = \mathsf{sk}$ *with probability at least* $1-\delta$ *after at most* $\left\lceil \frac{s-1}{p_{\min}} \cdot \ln\frac{1}{\delta} \right\rceil$ *effective oracle queries; see Appendix A.2 for the full proof.*

Remark 5 (General fault location). For an effective fault at an arbitrary node v^* with ℓ leaf descendants, the number $L = |\mathcal{H} \cap \mathrm{leaves}(v^*)|$ of hidden leaves in the subtree is a hypergeometric random variable whose realisation may be smaller than its mean $\bar{w} = w\ell/t$, making the per-query success probability $p_r(L) = 1-\big((s-1-r)/(s-1)\big)^L$ potentially smaller than the root-fault $p_{\min}$. A valid bound for any effective fault is obtained by taking the worst case $L = 1$, which gives $p_r \geq r/(s-1)$ for all $r \geq 1$ and therefore $\mathbb{E}[Q] \leq (s-1)\, H_{s-1}$, where $H_{s-1} = \sum_{r=1}^{s-1} 1/r$ is the $(s-1)$-th harmonic number. For $s \leq 6$ (all schemes in Table 1), this gives $\mathbb{E}[Q] \leq 5H_5 \approx 11.4$, still a small constant.

Detecting whether a received signature is effective is feasible for the adversary: given the fault location v^* and the challenge $\mathbf{c}$ (recoverable from the digest d in the signature), the adversary recomputes the honest flag tree, checks whether $\mathcal{F}(v^*) = 1$, and verifies that at least one hidden leaf falls in the subtree of v^*. This detection is detailed for LESS in [13] and applies unchanged to MEDS and CROSS.

Expected query count. With $p_{\min}$ as in Proposition 4, the root fault admits a clean expectation bound.

Proposition 5. *Let* $v^* = \mathrm{root}$*, so* $L = w$ *exactly. Then* $p_r \geq p_{\min}$ *for all* $r \geq 1$*, and the expected total number of effective queries satisfies* $\mathbb{E}[Q] \leq (s-1)/p_{\min}$*. See Appendix A.3 for the derivation.*

Remark 6 (Role of Jensen's inequality). The per-query gain $E_{\mathrm{dist}} = (s-1)\, p_{\min}$ is the expected number of distinct secret indices revealed when $L = w$ is deterministic at the root. For $v^* \neq \mathrm{root}$, convexity of α^x makes E_{dist} an *upper* bound

on per-query gain, hence only a *lower* bound on $\mathbb{E}[Q]$; the bound in Proposition 5 relies solely on the direct inequality $p_r \geq p_{\min}$, not on the Jensen argument.

Table 2 instantiates the root-fault bound ($p_{\min} = 1 - ((s-2)/(s-1))^w$, $L = w$ exactly) for every MEDS parameter set: since $\ell = t$ gives $\bar{w} = w \gg 1$, all sets have $p_{\min} > 0.99$ and $\mathbb{E}[Q] \leq s - 1 \leq 5$.

Table 2: Expected effective queries $\mathbb{E}[Q]$ for MEDS (fault at root).

Parameter set	s	w	$s-1$	$p_{\min}$	$\mathbb{E}[Q] \leq$
MEDS-9923	4	14	3	0.9966	3.01
MEDS-13220	5	20	4	0.9968	4.01
MEDS-41711	4	26	3	1.0000	3.00
MEDS-69497	5	36	4	1.0000	4.00
MEDS-134180	5	52	4	1.0000	4.00
MEDS-167717	6	66	5	1.0000	5.00

From key recovery to forgery. Once Algorithm 4 terminates, the adversary holds a complete signing key $\widehat{\mathsf{sk}}$ and can produce valid signatures on any message msg^* — including messages never submitted to any oracle. This constitutes a full break of EUF-CMA:

Corollary 1. *Let Π be a* GZKST*-based signature scheme and let the fault model be as in Definition 3 (a stuck-at-zero perturbation of a single flag bit during* Publish*). If an efficient adversary can induce this fault at a fixed node $v^* \in V(\mathcal{T})$ on the victim's signing device, then Π is* not *EUF-CMA-secure in the fault model: the adversary recovers* sk *via Algorithm 4 and forges signatures on arbitrary messages without further oracle access.*

4 Fault Injection attacks on MEDS

This section details a fault injection (FI) attack against the C reference implementation of MEDS. We focus on exploiting the tree traversal logic used in the path construction to leak hidden seeds.

Target Analysis. The primary target of our analysis is the SeedTreePath function, specifically targeting the logic represented in the signing phase of the MEDS algorithm (instruction 16 of algorithm 2). In the reference implementation[3], both SeedTreePath (path generation) and SeedTree (tree reconstruction) are handled by a unified function, `stree_to_path_to_stree`, which operates in two modes defined by a `mode` parameter.

[3] Reference implementation available in https://github.com/MEDSpqc/meds.

Listing 1.1: minimal description of `stree_to_path_to_stree` implementation.

```
1  void stree_to_path_to_stree(uint8_t *stree, uint8_t *h_digest, uint8_t *path, uint8_t *salt
       , int mode) {
2      ...
3      while (1==1) {
4          int index_leaf = 0;
5          while ((indices[id] » (MEDS_seed_tree_height-h)) == i)
6          {// Traverse down toward the secret leaf
7              h += 1;
8              i <<= 1;
9              if (h > MEDS_seed_tree_height) {
10                 h -= 1;
11                 i >>= 1;
12                 if (id + 1 < MEDS_w) id++;
13                 index_leaf = 1; // Flag node as hidden
14                 break;
15             }
16         }
17         ...
18     }
19 }
```

The algorithm identifies secret leaves (where $h[i] > 0$) and performs a Depth-First Search (DFS). The exploration halts as soon as it reaches a node that is no longer an ancestor of the hidden leaf. This node is then appended to the signature path. If the exploration reaches a target hidden leaf, the `index_leaf` flag is set to ensure the leaf seed remains unpublished and we move on to the next hidden leaf.

Objective. The main goal for these attacks is to manipulate the control flow via fault injection to leak unpublished seeds (internal nodes or leaves) into the public signature path, ultimately compromising the security of the signature scheme.

Experimental Setup. Our experimental platform consists of a **CW303-STM32F3** target board, featuring an ARM Cortex-M4 STM32F303RCT7 processor. Clock glitching is performed using a **ChipWhisperer-Lite**. As a proof-of-concept, the target board is flashed exclusively with the `stree_to_path_to_stree` function, using fixed values for the reference tree, salt, and challenge.

Fault Models. We propose three fault models targeting different stages of the tree traversal.

Model 1: Full Tree Recovery (Root Leakage). This attack aims to reveal the root seed, which effectively compromises the entire tree. By injecting a fault that causes the skip of the first iteration of `while` loop at line 5 of listing 1.1, the DFS halts immediately at the root ($h = 0, i = 0$). The root is incorrectly identified as a node ready for publication.

Detection and recovery. The signature path contains only a single non-zero element. Verification is performed by expanding this single seed into a full tree and checking if the resulting signature is accepted.

Model 2: Subtree Recovery. By glitching the conditional `if (id + 1 < MEDS_w) id++;` at line 12 of listing 1.1, the algorithm fails to update the target index `id`. The traversal logic becomes desynchronized, resulting in the algorithm treating subsequent branches as public. All leaves to the right of the faulted `id` become computable. *Detection and recovery.* This fault is hard to detect without a reference signature. However, one can simulate tree reconstructions for all possible faulted `id` values to verify which leads to a valid signature. (up to w verifications)

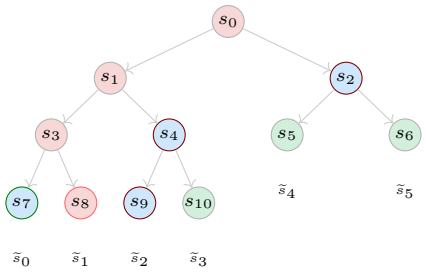


Fig. 3: Effect of fault model 2 on the example presented in figure 2, if the glitch occured while `id==1`. The returned path is $\{s_7, s_9, s_4, s_2\}$ and all leaves are computable except $\widetilde{s}_1$.

Model 3: Hidden Leaf Recovery. This model targets the specific instruction `index_leaf = 1;` at line 13 of listing 1.1. By skipping this assignment, a hidden leaf is no longer marked as private. The execution continues normally, but the secret leaf seed is copied into the `path` buffer.

Detection and recovery. The path length will be $w + k$ (where $k > 0$ is the number of iterations of the target instruction that were skipped). Detection involves testing subsets of the path to find a valid signature, which may be computationally expensive ($\binom{w+k}{w}$ verifications).

4.1 Results and Observations

We successfully executed the fault models on the Chipwhisperer board. This section detailed the compiler and glitcher configurations that were required to obtain the desired faults.

Step-by-Step attack. For each fault model we first define a target output state signifying a successful injection, then bracket the target assembly instructions with a `trigger_high()`/`trigger_low()` pair so the ChipWhisperer counts the right clock-cycle window; we verify the disassembly to confirm compiler optimizations did not move or drop the triggers. The `repeat` parameter sets the glitch duration in clock cycles, calibrated from the ARM nominal cycle counts for the target sequence.

We sweep three parameters to map the fault space: `width`, the glitch pulse width (XOR of the system clock, 0%–50% of the clock period); `offset`, the phase shift relative to the system clock (−50%–50%); and `ext_offset`, the number of clock cycles between the trigger and the glitch.

Model 1: Full tree recovery. This attack is successful if the returned path contains a single node which is the root node.

Compilation. This attack is carried out on a code compiled with the default `-Os` optimization.

Trigger position. We directly target the evaluation of the conditional and the branching instruction corresponding to line 5 of Listing 1.1. The compiled result is in Listing 1.2. We fault for 3 clock cycles.

Listing 1.2: Compiled trigger points for fault model 1.

```
0800062c <stree_to_path_to_stree_glitch>:
...
    // trigger_high();
 8000710:   f001 fa12   bl  8001b38 <trigger_high>
    // while ((indices[id] >> (MEDS_seed_tree_height - h)) == i)
 8000714:   ab06        add r3, sp, #24
 8000716:   eb03 0488   add.w   r4, r3, r8, lsl #2
 800071a:   e00d        b.n 8000738 <stree_to_path_to_stree_glitch+0x66>
    // trigger_low();
 800071c:   f001 fa13   bl  8001b46 <trigger_low>
...
```

Parameters. Figure 4 shows that the attack has a high success rate and lower reset rates for an `ext_offset` ranging from 1.5 to 2.5, a small `offset` of −1 to 1 and rather wide glitches (`width` over 20).

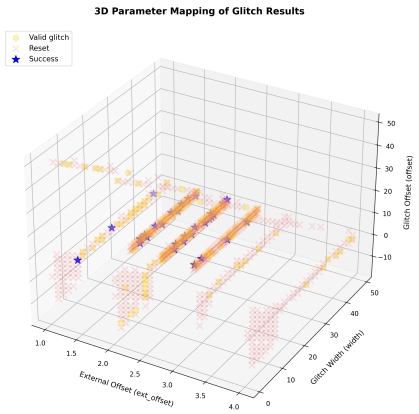


Fig. 4: Glitch success frequency by glitch width and offset for fault model 1, with 10 tries per parameter set. Valid glitches are runs that did not crash and output paths that were neither correct nor the target faulted path.

Model 2: Subtree recovery. We consider this attack successful if we obtain the faulted path pattern described by Figure 3.

Compilation. This is carried out on a code compiled with the default `-Os` optimization.

Trigger position. We set the trigger to frame line 12 of Listing 1.1. The compiled result is in Listing 1.3.

Listing 1.3: Compiled trigger points for fault model 2.

```
080006d2 <stree_to_path_to_stree_glitch>:
...
 8000726:   f001 fa09    bl  8001b3c <trigger_high> // trigger_high();
 800072a:   f108 0401    add.w   r4, r8, #1
 800072e:   2c05         cmp r4, #5                  //if (id+1 < MEDS_w)
 8000730:   bf88         it  hi
 8000732:   4644         movhi   r4, r8              // id++ ;
 8000734:   f001 fa09    bl  8001b4a <trigger_low>  // trigger_low();
...
```

Parameters. The evaluation of the condition and the incrementation of `id` require 3 clock cycles. We therefore set `repeat` to 3. Figure 5 shows the success in terms of glitch offset and width.

Fig. 5: Glitch success frequency by glitch width and offset for fault model 2, with 10 tries per parameter pair.

We observe a distinct, bounded region of system resets delimited by successful attacks and other valid glitches. The paths of many of these valid glitches contain a large set of intermediate nodes which provide sufficient coverage to compute all of the leaf values.

Model 3: Single hidden leaf recovery. In this model, we recover only a hidden leaf.

Compilation. Higher compiler optimization levels made it difficult to precisely target the flags. The optimization flags `-O1`, `-O2`, `-O3`, and `-Os` resulted in compiled assembly code that omitted the explicit assignment instruction entirely, relying instead on complex control-flow manipulations to achieve the same logical outcome. To resolve this issue, we enforced the `-O0` optimization level on the targeted function, as applying `-O0` to the entire project increased the binary size beyond the storage capacity of the target board.

Trigger position. For the unoptimized version, we position the glitch high trigger right before the `index_leaf` assignment and the low trigger right after. This results in the compiled code in listing 1.4.

Listing 1.4: Compiled trigger points for fault model 3.

```
0800062c <stree_to_path_to_stree_glitch>:
...
 80006be:   f001 fd3f   bl  8002140 <trigger_high> // trigger_high();
 80006c2:   2301        movs    r3, #1             // index_leaf = 1;
 80006c4:   62bb        str r3, [r7, #40]   @ 0x28
 80006c6:   f001 fd43   bl  8002150 <trigger_low>  // trigger_low();
...
```

Parameters. The disassembled code shows that the assignement is done in two instructions which are both 1 clock cycle long. We set the `repeat` parameter of the glitch controller to 2. Figure 6 shows the success rate and reset for the glitch offset and width.

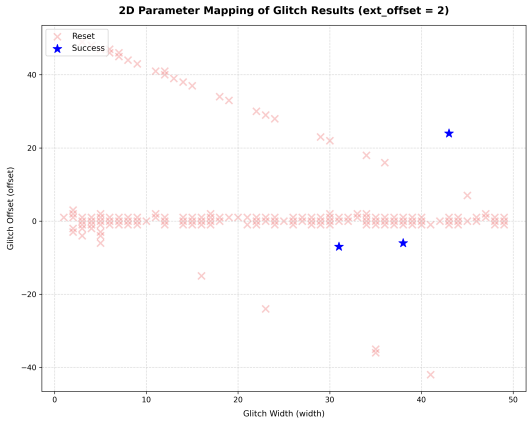


Fig. 6: Glitch success frequency by glitch width and offset for fault model 3, with 10 tries per parameter pair.

Results. The experiments confirm that the attack is reproducible. Across all parameter sweeps, attempts strictly resulted in either normal execution, a device reset, or the expected exploit. We observed a complete absence of other valid

glitches. This is due to our single-instruction targeting. Unlike fault models that disrupt larger execution windows and introduce non-deterministic errors, this injection is less likely to bleed into surrounding cycles or to trigger a control-flow collapse.

These experiments demonstrate that the root reveal fault model, which is the most efficient approach requiring the fewest queries for full secret key recovery, is highly reproducible. Furthermore, our results confirm the viability of two alternative fault models, although they prove more challenging to replicate under experimental conditions. Despite their lower reproducibility rates, fault models 2 and 3 remain relevant for threat modeling; as detailed in Section 5, they incur a higher computational or architectural cost to detect during signature generation.

5 Countermeasures

In this section, we present several countermeasures aimed at improving the resilience of the signature scheme against the fault attacks described above.

We group the countermeasures into three categories. The first category consists of *post-publication code-path checks*, which preserve both the signature size and the original tree-construction procedure. The second category comprises *modifications to the path-construction algorithm*, which maintain the original signature size while altering the path-generation process. Finally, the third category considers *tree-less GGM signatures*, which eliminate the tree structure entirely at the cost of a larger signature size.

5.1 Post-publication path check

This countermeasure runs checks on the output of SeedTreeToPath.

A simple size check on the path is enough to be secure against fault models 1 and 3, since they result in path lengths of 1 and golden path size$+1$ respectively. The expected path size can be computed in linear time from $\mathcal{H}$. A more thorough check would be to rebuild the tree using PathToSeedTree on the obtained path and to compare the obtained leaves to $\mathcal{H}$ and the reference tree.

Remark 7. In order to pass this check, it would be sufficient to induce the same fault as in the SeedTreeToPath on PathToSeedTree, seeing as they both call the same function in the reference implementation.

5.2 Path construction algorithm modification

Although the unified implementation of SeedTreeToPath and PathToSeedTree is efficient, its structural coupling of the flag tree construction and path extraction phases creates a vulnerability. Decoupling these operations into independent processing stages, and introducing validation checks after each phase, adds a redundancy that ensures that any fault injected during the initial flag tree generation will cause a detectable mismatch during the subsequent path construction phase.

We implement this countermeasure using the following multi-stage verification procedure:

1. Construct a binary flag tree $\mathcal{F}$ derived from $\mathcal{H}$.
2. Verify that for every leaf position i where $\mathcal{F}[i] = 1$, no ancestral node in its path to the root is flagged as 0.
3. Reconstruct the node path using the validated flag tree $\mathcal{F}$ and the reference seed tree.
4. Assert that no leaf node explicitly flagged as 1 is included in the final extracted path.

Complexity. All four verification steps scale linearly with the size of the tree, maintaining a time complexity of $\mathcal{O}(t)$.

Impact on performances. We implement all three countermeasures and compare the execution times of the signature.

The results are summarized in table 3. We refer to the path size check countermeasure by $\mathbf{C}_{size}$, the full tree check by $\mathbf{C}_{tree}$ and the flag tree implementation by $\mathbf{C}_{flagtree}$.

Experimental Setup. We evaluated MEDS under RIOT OS on two platforms. The embedded target is a bare-metal Nordic nRF52840 DK (32-bit ARM Cortex-M4 at 64 MHz, 256 KB RAM, 1 MB Flash) running the Toy parameter set; the host baseline runs all parameter sets (MEDS9923–MEDS167717) natively on an Intel Core i7 desktop under Ubuntu 24.04 LTS. Code was compiled with `arm-none-eabi-gcc` (Cortex-M4) and `gcc` 13.3.0 (host).

Table 3: MEDS signature time (microseconds) by parameter set. All values are the median value of 100 runs.

Parameter set	**base**	C_{size}	C_{tree}	$C_{flagtree}$
Toy	6845	6793	6794	6835
MEDS9923	1609626	1582806	1590051.5	1585796
MEDS13220	299950	275909	283484	283477.5
MEDS41711	4847694.5	5200203	4962401.5	5360679.5
MEDS55604	1280193	1381107	1321436	1307889
MEDS134180	5009655.5	5074072.5	5140707.5	5735290
MEDS167717	2950702	2951835.5	3076849.5	3051602.5
Toy (Cortex M4)	753566	737162	738843	737739

The benchmarks reveal that for parameter sets corresponding to NIST Security Level I (such as *MEDS13220* and *MEDS55604*), the proposed countermeasures do not introduce overhead. Because these configurations utilize shallow tree structures, the overall execution time is heavily dominated by base matrix arith-

metic, rendering the validation loops virtually cost-free. However, in the larger parameter sets with deeper trees (such as *MEDS41711* and *MEDS134180*), we do observe an overhead in the overall signature time (up to +8% for C_{size}, +5% for C_{tree}, and +14% for $C_{flagtree}$).

5.3 GGM-tree-less signature

Alternative architectural designs could consider abandoning GGM tree structures entirely. A potential solution, adapting the fault attack countermeasure for LESS proposed by [13] would require the response vector v to explicitly contain the precomputed matrices $\tilde{A}_i$ and $\tilde{B}_i$ that are sampled from the published leaves σ_i.

$$v[i] = \begin{cases} (\tilde{A}_i.A_{h_i}^{-1}, B_{h_i}^{-1}.\tilde{B}_i), & \text{if } h_i > 0 \\ (\tilde{A}_i,\ \tilde{B}_i), & \text{if } h_i = 0 \end{cases} \tag{1}$$

While this design ensures that the secret matrices $(\tilde{A}_i, \tilde{B}_i)$ remain computationally infeasible to derive for any index i, it introduces a disadvantage by significantly inflating the overall signature size to the size `sig_cm` shown in table 4.

Table 4: Comparison of Signature Sizes and Overhead across MEDS Parameter Sets. `sig_cm` is given by $\ell_{\text{digest}} + t(\ell_{\mathbb{F}_q^{m\times m}} + \ell_{\mathbb{F}_q^{n\times n}}) + \ell_{\text{salt}}$

Parameter Set	NIST Level	`sig` (Bytes)	`sig_cm` (Bytes)	**Size Overhead**
MEDS-9923	I	9,896	677,440	6745.59%
MEDS-13220	I	12,976	112,960	770.53%
MEDS-41711	III	41,080	882,880	2049.17%
MEDS-69497	III	54,736	232,384	324.55%
MEDS-134180	V	132,424	475,072	258.75%
MEDS-167717	V	165,332	277,152	67.63%

6 Conclusion

We presented a unified study of fault attacks against post-quantum signature schemes based on seed-tree compression. Through the GZKST abstraction, we identified a common seed-hiding invariant underlying schemes such as LESS-v2, CROSS, and MEDS, and showed that previously proposed attacks can be viewed as violations of this invariant.

Building on this framework, we derived generic key-recovery procedures and analyzed their query complexity. We also demonstrated practical clock-glitch attacks

against the MEDS reference implementation on an ARM Cortex-M4 microcontroller, revealing multiple fault surfaces that enable hidden-seed disclosure and secret-key recovery. Finally, we proposed and evaluated several countermeasures, showing that effective protections can be achieved with small overhead. Overall, our results highlight the importance of securing seed-publication mechanisms against implementation-level fault attacks.

References

1. Emmanuelle Anceaume, Yann Busnel, Ernst Schulte-Geers, and Bruno Sericola. Optimization results for a generalized coupon collector problem. *J. Appl. Probab.*, 53(2):622–629, 2016.
2. M. Baldi, A. Barenghi, L. Beckwith, J.-F. Biasse, T. Chou, A. Esser, K. Gaj, P. Karl, K. Mohajerani, G. Pelosi, E. Persichetti, M.-J. O. Saarinen, P. Santini, R. Wallace, and F. Zweydinger. LESS: Linear code equivalence signature scheme — round 2 specification. NIST PQC Round 2, 2025. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/less-spec-round2-web.pdf.
3. Marco Baldi, Alessandro Barenghi, Michele Battagliola, Sebastian Bitzer, Marco Gianvecchio, Patrick Karl, Felice Manganiello, Alessio Pavoni, Gerardo Pelosi, Paolo Santini, Jonas Schupp, Edoardo Signorini, Freeman Slaughter, Antonia Wachter-Zeh, and Violetta Weger. CROSS: Codes and restricted objects signature scheme — specification document, version 2.0. NIST PQC Round 2 submission, 2025.
4. Carsten Baum, Ward Beullens, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST v2: Algorithm Specifications. https://faest.info/faest-spec-v2.0.pdf, February 2025. Accessed: 2026-05-13.
5. Slim Bettaieb, Loïc Bidoux, Victor Dyseryn, Andre Esser, Philippe Gaborit, Mukul Kulkarni, and Marco Palumbi. PERK: compact signature scheme based on a new variant of the permuted kernel problem. *Des. Codes Cryptogr.*, 92(8):2131–2157, 2024.
6. Loïc Bidoux, Jesús-Javier Chi-Domínguez, Thibauld Feneuil, Philippe Gaborit, Antoine Joux, Matthieu Rivain, and Adrien Vinçotte. RYDE: a digital signature scheme based on rank syndrome decoding problem with MPC-in-the-Head paradigm. *Designs, Codes and Cryptography*, 93(5):1451–1486, May 2025.
7. Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Lars Ran, Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. MEDS post-quantum cryptography. https://www.meds-pqc.org/, 2023.
8. Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. Take your MEDS: digital signatures from matrix code equivalence. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *Progress in Cryptology - AFRICACRYPT 2023 - 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19-21, 2023, Proceedings*, Lecture Notes in Computer Science, pages 28–52. Springer, 2023.
9. Özgür Dagdelen, Marc Fischlin, and Tommaso Gagliardoni. The fiat-shamir transformation in a quantum world. In Kazue Sako and Palash Sarkar, editors, *Advances*

in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II, Lecture Notes in Computer Science, pages 62–81. Springer, 2013.
10. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, Lecture Notes in Computer Science, pages 186–194. Springer, 1986.
11. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.
12. Xiao Huang, Zhuo Huang, Yituo He, Quan Yuan, Chao Sun, Mehdi Tibouchi, and Yu Yu. Faultless key recovery: Iteration-skip and loop-abort fault attacks on LESS. *IACR Cryptol. ePrint Arch.*, 2026:117, 2026.
13. Puja Mondal, Supriya Adhikary, Suparna Kundu, and Angshuman Karmakar. Zkfault: Fault attack analysis on zero-knowledge based post-quantum digital signature schemes. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VIII*, Lecture Notes in Computer Science, pages 132–167. Springer, 2024.
14. Puja Mondal, Suparna Kundu, Hikaru Nishiyama, Supriya Adhikary, Daisuke Fujimoto, Yuichi Hayashi, and Angshuman Karmakar. Fault to forge: Fault assisted forging attacks on LESS signature scheme. *IACR Cryptol. ePrint Arch.*, 2025:1838, 2025.
15. Tanguy Stekke, Durba Chatterjee, and Lejla Batina. Practical end-to-end fault attacks on PERK. Cryptology ePrint Archive, Paper 2026/1266, 2026.
16. Weiyu Xu and Ao Kevin Tang. A generalized coupon collector problem. *CoRR*, abs/1010.5608, 2010.

A Full Proofs

Throughout this appendix, $\mathcal{T}$ denotes a GGM seed tree with leaf set $\{0,\dots,t-1\}$ and $\mathcal{F} : V(\mathcal{T}) \to \{0,1\}$ its flag labelling, as produced by the Publish phase of the relevant GZKST instance. We use the convention $\mathcal{F}(v) = 1$ to mean that v's subtree contains at least one hidden leaf ($i \in \mathcal{H}$), and $\mathcal{F}(v) = 0$ to mean that all leaf descendants of v are revealed ($i \in \mathcal{R}$).

A.1 Proof of Proposition 1 (MEDS is a GZKST instance)

Let $\mathcal{T}$ be the MEDS seed tree with $\lceil \log_2 t \rceil$ levels. The SeedTreeToPath algorithm implements the following erasure: for each $i \in \mathcal{H}$ the algorithm (i) removes the label of leaf_i, and (ii) propagates the removal upward—a node v loses its label as soon as *at least one* leaf in its subtree has been removed.

Define flag values on the nodes of $\mathcal{T}$ by

$$\mathcal{F}(\text{leaf}_i) = \mathbf{1}[i \in \mathcal{H}], \quad i \in \{0,\dots,t-1\}, \tag{2}$$

$$\mathcal{F}(v) = \mathcal{F}(\ell) \vee \mathcal{F}(r), \quad \text{for every internal node } v \text{ with children } \ell, r. \tag{3}$$

Equations (2)–(3) jointly imply:

$$\mathcal{F}(v) = 1 \iff \exists\, i \in \mathcal{H} : \text{leaf}_i \text{ is a descendant of } v. \tag{4}$$

Claim. After the erasure pass, a node v retains its label if and only if $\mathcal{F}(v) = 0$, i.e., *every* leaf descendant of v is in $\mathcal{R}$ (no hidden leaf in v's subtree).

Proof. ($\Rightarrow$) Suppose v is retained. By definition of the upward propagation, no ancestor of any hidden leaf is retained. Hence v's subtree contains no hidden leaf, i.e., $\mathcal{F}(v) = 0$.

($\Leftarrow$) Suppose $\mathcal{F}(v) = 0$, i.e., every leaf in v's subtree is in $\mathcal{R}$. The erasure step removes only hidden leaves and their ancestors. Since v's subtree contains no hidden leaf, no removal is triggered in v's subtree, so v retains its label.

The published path path consists of the highest surviving nodes in the tree, serialised in depth-first order. Equivalently,

$$\mathsf{path} = \{\, v : v \text{ retains its label and parent}(v) \text{ was erased} \,\}.$$

By Claim A.1, this equals $\{v : \mathcal{F}(v) = 0,\ \mathcal{F}(\text{parent}(v)) = 1\}$ (with the convention that $\mathcal{F}(\text{parent}(\text{root})) = 1$).

Proof (Proof of Invariant 1 (Correctness)). Let $i \in \mathcal{R}$. Consider the root-to-leaf path $\pi_i = (v_0 = \text{root}, v_1, \ldots, v_d = \text{leaf}_i)$.

Since $|\mathcal{H}| \geq 1$ (the challenge weight satisfies $w \geq 1$), at least one leaf has flag 1. By (4), $\mathcal{F}(v_0) = \mathcal{F}(\text{root}) = 1$.

Since $i \in \mathcal{R}$, equation (2) gives $\mathcal{F}(v_d) = 0$.

Because $\mathcal{F}(v_0) = 1$ and $\mathcal{F}(v_d) = 0$, the path π_i contains at least two nodes with different flags. Define $k = \min\{j \in \{0, \ldots, d\} : \mathcal{F}(v_j) = 0\}$. Then $k \geq 1$ (since $\mathcal{F}(v_0) = 1$), $\mathcal{F}(v_k) = 0$, and $\mathcal{F}(v_{k-1}) = 1$. Hence $v_k \in \mathsf{path}$, and v_k lies on the root-to-leaf path π_i, establishing Invariant 1.

Proof (Proof of Invariant 2 (ZK hiding)). Let $i \in \mathcal{H}$ and let v be any proper ancestor of leaf_i (including the root). Since leaf_i is a descendant of v and $i \in \mathcal{H}$, equation (4) gives $\mathcal{F}(v) = 1$. Hence no ancestor of leaf_i can satisfy $\mathcal{F}(v) = 0$, so no ancestor of leaf_i lies in path. Invariant 2 holds.

Proof (Extension to incomplete trees (Remark 2)). When t is not a power of 2, the MEDS tree is incomplete: some leaves appear at depth $\lceil \log_2 t \rceil - 1$ rather than $\lceil \log_2 t \rceil$. The index-tracking arrays ensure that $\mathsf{SeedTreeToPath}$ correctly identifies which nodes are leaves, assigns flags according to (2), and performs the bottom-up OR pass (3) over the actual tree structure.

The proofs of both invariants above are purely graph-theoretic and depend only on properties (2)–(4), which hold regardless of whether leaves appear at a uniform depth. Hence both invariants hold for any incomplete tree shape produced by MEDS.

A.2 Proof of Proposition 4 (Correctness of Algorithm 4)

Let $m = |\mathcal{S}| = s-1$. The process of recovering the secret components is modeled as a *generalized coupon-collector problem* with group drawings [16,1], where each query draws a batch of w coupons out of m distinct types.

Correctness of Extract. For each scheme, Definition 4 specifies $\mathsf{Extract}$ as a closed-form algebraic procedure: matrix inversion over $\mathbb{F}_q$ (MEDS), column extraction (LESS), or a group operation (CROSS). Each procedure is deterministic and correct given the dual revelation $(\widetilde{s}_i, \mathsf{resp}_i)$: the algebraic relationship between the response and the hidden seed is an identity (not an approximation), so $\mathsf{Extract}$ returns exactly $\mathsf{sk}' =$ the secret component indexed by $j = c_i$.

We work under the assumption of Proposition 4: $v^* = \text{root}$, so the subtree of v^* is the entire tree and $L = |\mathcal{H} \cap \text{leaves(root)}| = |\mathcal{H}| = w$ exactly—a deterministic quantity, not a random variable. Define

$$p_{\min} \;=\; 1 - \left(\frac{s-2}{s-1}\right)^{w} \;=\; 1 - \left(1 - \frac{1}{s-1}\right)^{w}.$$

For any $r \geq 1$ uncollected secrets, the probability that at least one of the w hidden leaves reveals a new secret index is

$$p_r \;=\; 1 - \left(1 - \frac{r}{s-1}\right)^{w} \;\geq\; 1 - \left(1 - \frac{1}{s-1}\right)^{w} \;=\; p_{\min} \;>\; 0,$$

where the inequality holds because $1 - r/(s-1)$ is decreasing in r, so $r = 1$ gives the weakest bound. Hence

$$\Pr\big[\mathsf{new}(q) \geq 1 \;\big|\; \mathsf{Collected}\big] \;\geq\; p_{\min} \;>\; 0 \qquad \text{for all } s \geq 2,\ w \geq 1.$$

Let T_j be the number of effective queries to go from $|\mathsf{Collected}| = j-1$ to $|\mathsf{Collected}| = j$. Conditional on $|\mathsf{Collected}| = j-1$, each query succeeds with probability at least $p_{\min}$, so T_j is stochastically dominated by a $\mathrm{Geom}(p_{\min})$ random variable, which is finite almost surely. Hence $Q = \sum_{j=1}^{m} T_j$ is finite almost surely, and Algorithm 4 terminates with probability 1.

Expected query count and tail bound. Let R_k be the number of uncollected secret indices after k effective queries, and let $r = R_{k-1} > 0$. Each of the w hidden leaves (recall $v^* = \text{root}$, so all w hidden leaves are exposed) independently draws a uniform index from $\{1, \ldots, s-1\}$, so the probability that *none* of the w draws falls in the remaining set of size r is $\big((s-1-r)/(s-1)\big)^{w}$. Hence the probability of revealing at least one new index is

$$p_r \;=\; 1 - \left(\frac{s-1-r}{s-1}\right)^{w} \;=\; 1 - \left(1 - \frac{r}{s-1}\right)^{w}. \tag{5}$$

Since $1 - r/(s-1)$ is decreasing in r, we have $p_r \geq p_{\min} = 1 - (1 - 1/(s-1))^{w}$ for all $r \geq 1$.

The expected total number of effective queries is

$$\mathbb{E}[Q] \;=\; \sum_{r=1}^{m} \frac{1}{p_r} \;\le\; \frac{m}{p_{\min}} \;=\; \frac{s-1}{p_{\min}}, \tag{6}$$

where the inequality uses $p_r \ge p_{\min}$ for all $r \ge 1$.

Remark 8. The bound in (6) should be seen as an upper-bound, as it treats the recovery process as just learning one secret per time. In practice, a query may reveal multiple unseen indices simultaneously, especially during the early stages of the algorithm when r is large. As a consequence the actual expected number of queries can be smaller than $(s-1)/p_{\min}$.

For the tail bound, each $T_j \le k_0$ with probability at least $1-(1-p_{\min})^{k_0}$ (geometric tail), so by a union bound over $m = s-1$ stages,

$$\Pr[Q > k] \;\le\; (s-1)\cdot(1-p_{\min})^{\lfloor k/(s-1)\rfloor}.$$

Setting $(s-1)(1-p_{\min})^{\lfloor k/(s-1)\rfloor} \le \delta$ and solving gives $k \le \left\lceil \frac{s-1}{p_{\min}} \cdot \ln \frac{s-1}{\delta} \right\rceil$, establishing the bound in Proposition 4.

Output correctness. When the loop terminates, $|\mathsf{Collected}| = m$, and for each $j \in \{1,\dots,s-1\}$ there is exactly one entry $(j, \mathsf{sk}'_j) \in \mathsf{Collected}$ with $\mathsf{sk}'_j = A_j^{-1}$ (or the corresponding secret for each scheme). The $\mathsf{Collect}$ step assembles these into $\widehat{\mathsf{sk}}$, which equals sk by correctness of $\mathsf{Extract}$.

A.3 Derivation of Proposition 5 (Expected query count)

Setup. We work in the root-fault setting of Proposition 5: $v^* = \text{root}$, so $\ell = t$ and $L = w$ exactly (deterministic). By the Fiat–Shamir uniformity assumption, the challenge index c_i is uniform in $\{1,\dots,s-1\}$ independently for each $i \in \mathcal{H}$.

Distinct secrets per query. Conditioning on $L = w$, the number of *distinct* secret indices revealed in one effective query has expectation

$$\mathbb{E}[\#\text{distinct} \mid L = w] = (s-1)\left(1 - \left(1 - \frac{1}{s-1}\right)^{w}\right) \;=\; E_{\text{dist}}, \tag{7}$$

by linearity of expectation and symmetry of the uniform distribution.

Remark 9 (General fault location). When $v^* \ne \text{root}$ and L is a hypergeometric random variable with mean $\bar{w} = w\ell/t$, taking expectation over L gives

$$\mathbb{E}[\#\text{distinct}] = (s-1)\left(1 - \mathbb{E}\left[\left(1 - \tfrac{1}{s-1}\right)^{L}\right]\right).$$

Since $f(x) = \alpha^x$ with $\alpha = 1 - 1/(s-1) \in (0,1)$ is convex, Jensen's inequality yields $\mathbb{E}[f(L)] \geq f(\bar{w}) = \alpha^{\bar{w}}$; hence

$$\mathbb{E}[\#\text{distinct}] \;\leq\; (s-1)\left(1 - \left(1 - \frac{1}{s-1}\right)^{\bar{w}}\right) \;=:\; E_{\text{dist}}^{\bar{w}}. \tag{8}$$

$E_{\text{dist}}^{\bar{w}}$ is an *upper* bound on the expected per-query gain, which implies a *lower* bound on $\mathbb{E}[Q]$. It cannot be used to derive an upper bound on $\mathbb{E}[Q]$; the upper bound in Proposition 5 follows solely from the direct inequality $p_r \geq p_{\min}$ established in the root-fault setting.

Total queries (root fault). From equation (5) (see page 24), $p_r = 1 - \big(1 - r/(s-1)\big)^w$ (with $L = w$ deterministic), and since $p_r \geq p_{\min}$ for all $r \geq 1$,

$$\mathbb{E}[Q] = \sum_{r=1}^{m} \frac{1}{p_r} \leq \frac{m}{p_{\min}} = \frac{s-1}{p_{\min}},$$

which for $p_{\min} \approx 1$ (all MEDS parameter sets, cf. Table 2) gives $\mathbb{E}[Q] \lesssim s - 1$.