

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Amortizing Randomness Cost: Efficient Masked Implementation of SDitH Signatures with Common Shares

Guowei Liu, Weijia Wang, Lixuan Wu, Chaoran Wang, Yanhong Fan, Jinliang Wang, and Meiqin Wang, *Shandong University*

https://www.usenix.org/conference/usenixsecurity26/presentation/liu-guowei

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

Amortizing Randomness Cost: Efficient Masked Implementation of SDitH Signatures with Common Shares

Guowei Liu[1,2,3], Weijia Wang[1,2,3], Lixuan Wu[1,2,3], Chaoran Wang[1,2,3], Yanhong Fan[1,2,3], Jinliang Wang[1,2,3], and Meiqin Wang[1,2,3,*]
[1]*School of Cyber Science and Technology, Shandong University*
[2]*State Key Laboratory of Cryptography and Digital Economy Security, Shandong University*
[3]*Key Laboratory of Cryptologic Technology and Information Security, Shandong University*

Abstract

MPC-in-the-Head (MPCitH) signatures are attractive for post-quantum deployment, but they are structurally vulnerable to side-channel leakage: verification opens almost all simulated views, so any leakage correlated with the remaining unopened view(s) can be amplified and exploited to recover the witness. Among the MPCitH-based signature candidates considered in NIST's additional-signature round, Syndrome-Decoding-in-the-Head (SDitH)-Hypercube is particularly susceptible to this concern, since its signing process combines: (i) offline last-party completion involving nonlinear arithmetic computations, (ii) online Baum–Nof (BN) arithmetic checking, and (iii) Keccak sponge calls that generate or absorb secret-bearing values.

To the best of our knowledge, this is the first protocol-compatible masked implementation-level protection architecture for SDitH-Hypercube on Cortex-M4. Our implementation combines three techniques, each protecting one attack-critical hotspot: (i) Common Shares to amortize the masking randomness required by masked multiplications in the offline phase, (ii) a Sec/Pub typing discipline that reconstructs transcript-public and verifier-reconstructible values early to avoid costly Sec×Sec gadgets in the online BN kernels, and (iii) selective masked Keccak backends applied only to secret-bearing sponge call sites.

On the L1 parameter set, Common Shares reduces arithmetic masking randomness from 8.37 KiB/signature of the Strong Non-Interference (SNI) profile to 16 B/signature, while the Sec/Pub-typed optimization removes 3,264 B/signature from the online BN kernels. When Keccak is masked, masking randomness is dominated by the sponge: 41,118.75 KiB/signature with DOM-Keccak and 82,237.50 KiB/signature with SNI-Keccak, motivating selective masked hashing and tunable backends. We implement a low-stack reference signer and all protected signers on Cortex-M4. Across protected profiles, signing takes 913.7–1,542.0 Mcycles (1.84×–3.10× over the reference), and fits within 155.6 KiB flash, 136.7 KiB static RAM, and 7.96 KiB signing stack. A 1M-trace first-order fixed-vs-random Test Vector Leakage Assessment (TVLA) experiment on Cortex-M4 no longer detects the previously observed leakage peaks after protection.

1 Introduction

Post-Quantum Cryptography (PQC) is moving from standardization to deployment, making implementation security as important as algorithmic security. This is especially true on embedded and resource-constrained platforms, where power consumption, EM emanations, and micro-architectural effects can expose secret-dependent intermediates and enable practical key-recovery attacks [22]. For digital signatures, the risk is amplified by repeated signing: the signer processes large amounts of secret-dependent data under tight cycle, memory, and entropy budgets.

This pressure is now visible for MPCitH-based signatures. Recent work has shown that such signatures can be made practical on Cortex-M-class devices by rethinking stack usage, memory layout, streaming computation, and recomputation strategies [7, 9]. These developments make embedded deployment realistic, but they also make side-channel protection a deployment requirement. A signature that fits on a microcontroller must also resist the leakage patterns induced by that platform. The Arm Cortex-M4 has become a standard target for embedded PQC benchmarking, as illustrated by the pqm4 framework [21].

SDitH-Hypercube is a syndrome-decoding-based signature in NIST's additional digital-signature process [12, 23]. We target the v1 Hypercube construction, which follows the MPCitH approach [2].[1] In this proof system, the verification opens almost all simulated information, and security depends on the remaining hidden information staying hidden from the attacker.

*Corresponding Author.

[1]The newer SDitH v2 specification uses a VOLE-itH proof system [1]; when we use "MPCitH-based," we mean the v1 Commit-and-Prove proof system.

This Commit-and-Prove structure creates a structural side-channel fragility. Leakage correlated with the unopened or hidden information can be combined with public transcript data and amplified across signatures. Prior work has demonstrated this risk on MPCitH-based signatures, from DPA-style attacks on Picnic [14] to single-trace attacks on arithmetic MPCitH/TCitH-style computations [15].

Masking is a widely used countermeasure for secret-dependent computation: sensitive values are split into shares and processed through dedicated gadgets [5, 18]. For MPCitH-based signers, however, applying composable masking naively is costly: party simulations, refreshes, and symmetric-primitive calls can make both runtime and masking randomness dominate [3, 25, 26]. This motivates randomness-amortization techniques such as PRG-based masking and Common Shares [10,27]. The central question is therefore not whether the signer can be masked instruction by instruction, but how to identify the computations that actually need protection and how to amortize the required masking randomness safely.

We answer this question with a leakage-driven protection architecture. We first isolate the signer computations whose leakage can be combined with the public transcript to recover witness information, and we protect those computations with mechanisms matched to their algebraic structure. The design is protocol-compatible: it preserves the SDitH parameters, transcript format, Fiat–Shamir flow, openings, and verifier interface. Only the computation of sensitive signer intermediates is changed.

We use LS1–LS3 to denote the three attack-critical hotspots. LS1 covers offline nonlinear preprocessing and last-party share completion, where the signer performs secret-dependent finite-field multiplications. We protect this hotspot with Common Shares [27], reusing structured masking randomness across RMul/RAdd gadgets under the MDS conditions. LS2 covers the online BN kernels [6]. Inspired by the boundary-reconstruction idea of Aranha et al. [3], we introduce a Sec/Pub type discipline that reconstructs transcript-public and verifier-reconstructible values early, turning unnecessary Sec×Sec gadgets into randomness-free share-wise Sec×Pub affine maps. LS3 covers secret-bearing XOF and Com_N sponge calls. Secret-bearing Keccak calls require special care: unprotected Keccak has known single-trace leakage risks, and masked Keccak backends have been studied in several forms [5, 8, 17, 20].

We implement a low-stack reference signer and all protected signers on Cortex-M4. Across protected profiles, signing takes 913.7–1,542.0 Mcycles (1.84×–3.10× over the reference), and fits within 155.6 KiB flash, 136.7 KiB static RAM, and 7.96 KiB signing stack. The fastest protected profile is SNI+opt with IND-Keccak, while RNI+opt provides the low-masking randomness Common Shares option. With the stronger SNI-Keccak backend, RNI+opt signs in 1,526.2 Mcycles. We further validate the protected hotspots with first-order fixed-vs-random TVLA on Cortex-M4; after protection, the previously observed leakage peaks are no longer detected under our measurement setup.

1.1 Contributions

This paper makes the following contributions.

- **Leakage-driven hotspot isolation for SDitH-Hypercube.** We identify three attack-critical signing hotspots: last-party completion in Fill_N, online BN arithmetic in $\mathsf{MPC}_{\mathrm{sec}}$, and secret-bearing XOF+Com_N sponge calls. This decomposition explains where transcript-amplified leakage must be blocked.
- **Implementation-level protection without protocol changes.** We design a signer-side protection architecture that preserves the SDitH parameters, transcript format, Fiat–Shamir flow, openings, and verifier interface. The architecture combines Common Shares for arithmetic, Sec/Pub typing for BN kernels, and selective masked Keccak for secret-bearing sponge calls.
- **Masking randomness reduction for arithmetic hotspots.** At L1, Common Shares reduces arithmetic masking randomness from 8.37 KiB/signature to 16 B/signature, and the Sec/Pub optimization removes 3,264 B/signature from the online BN kernels. The reduction concerns masking randomness only; SDitH seeds, salt, Fiat–Shamir challenges, and protocol tapes are unchanged.
- **Cortex-M4 implementation and leakage validation.** We implement a low-stack reference signer and all protected signers on Cortex-M4. Protected signing takes 913.7–1,542.0 Mcycles and fits within 155.6 KiB flash, 136.7 KiB static RAM, and 7.96 KiB stack. First-order TVLA no longer detects the previously observed leakage peaks after protection.

1.2 Related Work

Recent work on embedded MPCitH-based signatures shows that these schemes are no longer only high-end-server targets. PERK has been optimized for resource-constrained devices and implemented on Arm Cortex-M4-class platforms, while MQOM has also been revisited for embedded microcontrollers [7, 9]. These works focus on feasibility and performance. Our work addresses the complementary problem: side-channel protection once such deployment becomes realistic.

Side-channel attacks against MPCitH-based signatures motivate the need for such protection. Prior work demonstrated practical leakage attacks on Picnic and single-trace attacks on arithmetic MPCitH/TCitH-style computations [14, 15]. These attacks exploit the same structural feature: most simulated

information is eventually opened, so leakage on the hidden part can be combined with the public transcript.

One mitigation direction keeps the signature scheme and verifier interface unchanged and hardens the implementation. This is the line taken by side-channel protections for Picnic and FAEST [3,4]. Our work follows the same design point for SDitH-Hypercube: the proof system, transcript format, openings, and verifier remain unchanged, while the signer implementation is protected at carefully selected secret-dependent boundaries.

A different direction changes the proof system or framework-level hiding structure. SNI-in-the-head builds side-channel protection into the MPC-in-the-Head construction itself [26]. Masking-friendly TCitH signatures adjust the TCitH framework to reduce masking cost [13]. HYPERSHIELD hardens Hypercube MPCitH against differential probing adversaries without masking [25]. These works are complementary to ours: they harden the protocol or framework, whereas we preserve the SDitH signer/verifier interface and apply implementation-level masking.

Finally, our optimizations have different degrees of transferability. Hotspot isolation, Sec/Pub typing, and selective masking of secret-bearing sponge calls are broadly applicable to MPCitH-based signatures whenever public or verifier-reconstructible values can be separated from secret-dependent values. Common Shares is more specialized: it targets finite-field arithmetic pipelines expressible as RMul/RAdd gadgets satisfying the MDS conditions.

2 Preliminaries and Threat Model

This section fixes notation and recalls the SDitH-Hypercube signing workflow under the MPCitH-based Commit-and-Prove abstraction. We then formalize the side-channel threat model and the concrete security target of this work. Finally, we summarize the masking notions and the leakage-detection methodology used in our evaluation.

2.1 Notation and Two Layers of Sharing

For integers $a \le b$, let $[a{:}b] := \{a, a+1, \dots, b\}$. All arithmetic takes place over the finite field $\mathbb{F}$ specified by the SDitH parameter set. We denote vectors by bold letters, e.g., $\mathbf{x} \in \mathbb{F}^n$.

SDitH signing runs τ independent repetitions indexed by $e \in [1{:}\tau]$. Each repetition simulates N virtual parties indexed by $i \in [1{:}N]$. We write Commit for the per-view commitment and Hash for the Fiat–Shamir hashing interface.

We use two different sharing layers, and we explicitly separate their notations:

- **MPC shares (protocol layer).** For a value $x \in \mathbb{F}$ shared among the N virtual parties, we write $[x] := ([x]_1, \dots, [x]_N)$. For additive sharing, reconstruction is $x = \sum_{i=1}^{N} [x]_i$ over $\mathbb{F}$. In particular, the witness-dependent secret state used by the signer is represented at this layer, i.e., the signer operates on protocol-layer sharings of sensitive values.
- **Masking shares (implementation/SCA layer).** To protect a sensitive value $y \in \mathbb{F}$ against side-channel leakage, we encode it into $d+1$ masking shares: $\langle y \rangle := (y^{(0)}, \dots, y^{(d)}), y = \sum_{j=0}^{d} y^{(j)}$. The index $j \in [0{:}d]$ refers only to the masking layer.

In the protected implementation, some protocol-layer values are stored and processed in masked form. Concretely, an MPC share $[x]_i$ may be represented as $\langle [x]_i \rangle := ([x]_i^{(0)}, \dots, [x]_i^{(d)}), [x]_i = \sum_{j=0}^{d} [x]_i^{(j)}$. Accordingly, *opening* views refers to revealing protocol-layer information (seeds, protocol shares, and view contents), whereas *reconstructing* refers to combining masking shares inside the implementation.

2.2 SDitH Signing Overview

SDitH follows the Commit-and-Prove structure of MPCitH and applies Fiat–Shamir to obtain a signature. Figure 1 summarizes the signing architecture we target. In each repetition e, the signer (i) derives party randomness from seeds, (ii) prepares MPC-shared inputs and preprocessing material, (iii) commits to each party view, (iv) derives MPC challenges from a first hash h_1, (v) simulates the MPC computation and records the public transcript, and (vi) derives an opening set I from a second hash h_2 and outputs the corresponding openings.

Verification opens most views, so security hinges on the secrecy of the unopened or hidden information given the opened views and the public transcript. This makes the implementation sensitive to leakage correlated with witness-dependent states and arithmetic checks. Our protections therefore focus on attack-critical regions that manipulate protocol-layer secret sharings of the witness and values derived from them. We leave unmasked only protocol-public logic and non-critical glue code: loop and index handling, serialization, tree navigation, and hashing over fully public inputs. In addition, seed-only TreePRG and seed-only commitments are kept outside the masked surface only under the bounded Seed-leakage boundary made explicit in Section 4.1. This seed-only material is not treated as Pub: once seed-derived bytes are interpreted as protocol shares, or once a computation absorbs witness-derived correction data such as aux_e, the value is promoted to Sec and protected.

2.3 Threat Model and Security Target

We consider a passive t-probing physical adversary with access to the signing device. The attacker can request signatures on chosen messages, observe the resulting signatures, and record side-channel traces during signing, such as power

Signature (SDitH-Hypercube):

1. Generate shared states and preprocessing.

For each repetition $e \in [1{:}\tau]$, derive party randomness from rseed_e and $\mathsf{seed}_{e,i}$, and sample secret-shared inputs $([\mathsf{wit}] := [x_A], [P], [Q])$ and Beaver-triple shares $([a], [b], [c])$.

$$\left([\mathsf{wit}]_{e,i},\ [a]_{e,i},\ [b]_{e,i},\ [c]_{e,i}\right) \leftarrow \mathsf{XOF}(\mathsf{salt}, \mathsf{seed}_{e,i})\ (i \neq N), \quad \left([a]_{e,N},\ [b]_{e,N}\right) \leftarrow \mathsf{XOF}(\mathsf{salt}, \mathsf{seed}_{e,N}).$$

The last party is computed as a correction term to satisfy global consistency, where $c_e = a_e \times b_e$.

2. Compute the last-party auxiliary correction.

Construct an auxiliary value aux_e that packs the missing-share corrections so that the shares reconstruct plaintext witness and Beaver triples:

$$\mathsf{aux}_e \ := \ \left(x_{\mathsf{plain}} - \textstyle\sum_{i \neq N} [x]_{e,i}\right)_{x \in \mathcal{S}_e}, \text{ where } \mathcal{S}_e \text{ contains } x_A, P, Q, \text{ and } c.$$

3. Commit to party views.

For each party $i \in [1{:}N]$, compute a commitment to the party's view/state:

$$\mathsf{state}_{e,i} \xrightarrow{\mathsf{Commit}} com_{e,i}, \text{ where } \mathsf{state}_{e,i} := \mathsf{seed}_{e,i}\ (i \neq N),\ \mathsf{state}_{e,N} := (\mathsf{seed}_{e,N} \parallel \mathsf{aux}_e).$$

4. Derive the first Fiat–Shamir challenge. Hash all commitments to obtain h_1 and derive MPC challenge material:

$$com_{e,1}, \ldots, com_{e,N} \xrightarrow{\mathsf{Hash}} h_1 \longrightarrow (r, \varepsilon).$$

5. Simulate the MPC computation and record broadcasts.

Using shared inputs and (r, ε), simulate the SDitH arithmetic MPC computation and produce the public transcript (broadcast values):

$$[\alpha] = \varepsilon \cdot [Q(r)] + [a],\ [\beta] = [S(r)] + [b],\ [\nu] = \varepsilon \cdot [F \cdot P(r)] - [c] + \alpha \cdot [b] + \beta \cdot [a],$$

where $[S(r)], [Q(r)]$ and $[F \cdot P(r)]$ can be locally computed.

6. Derive the second Fiat–Shamir challenge and open views.

Hash h_1 together with the broadcast transcript to obtain h_2 and derive an index set I of parties/views to open:

$$h_1,\ ([\alpha], [\beta], [\nu]) \xrightarrow{\mathsf{Hash}} h_2 \longrightarrow I.$$

Output (h_1, h_2) together with the openings for $i \in I$ (seeds/shares and auxiliary data), and the remaining commitments/transcript fragments for $i \notin I$.

Figure 1: High-level signing workflow with the schematic variables that are central to the leakage analysis in Section 3.

or EM traces. This captures multi-trace statistical attacks against long-term witness-dependent intermediates. Active faults, clock or voltage glitching, and invasive probing are outside this model.

The notation is stated for a generic masking order d, because the underlying gadgets are naturally parameterized by the number of masking shares. The concrete implementation and empirical evaluation in this paper target first-order protection. In the protected regions, sensitive values are processed as masked encodings $\langle \cdot \rangle$ and are reconstructed only at protocol-public or verifier-reconstructible boundaries.

We do not claim that every instruction executed during signing is masked. Rather, we provide a disciplined protection architecture that minimizes the protected surface while covering the computations whose leakage can be combined with the public transcript to recover witness information. All leakage experiments reported in Section 5 are first-order fixed-vs-random TVLA experiments.

Bounded seed-only leakage. For the selective profiles, we distinguish leakage from one-time seed material from leakage of long-term witness-dependent values. A Seed value is fresh for each signature and domain separated; it may remain unmasked only while it is used for seed-tree expansion, seed-only commitments, or as an input to a protected share-generation XOF. The claim is bounded: up to t probes of such one-time seed material are treated as a bounded loss in seed hiding, but complete single-trace recovery of a hidden seed or seed-tree state is outside our claim, because it could regenerate the corresponding hidden view. This exception does not apply after the seed output becomes a protocol share or after a sponge absorbs aux_e; those computations are Sec.

2.4 Masking Notions and Interfaces

We rely on two composable masking views, chosen to match the two dominant implementation "shapes" in SDitH signing: (i) local operations and small nonlinearities, and (ii) multiplication-dominated arithmetic kernels.

For local masked computation blocks, we follow the standard SNI-style discipline: linear operations are performed share-wise, while nonlinear operations are implemented using dedicated gadgets. This provides a clean module boundary for composing protected blocks under the first-order target.

For arithmetic kernels dominated by repeated masked multiplications, we use Common Shares to amortize the cost of randomness and intermediates across many nonlinear gadgets, while retaining a probing-style security guarantee within its framework. Common Shares comes with its own composability notion (Randomness Reusable Non-Inference, RNI) that is designed to support controlled reuse.

Common Shares is an arithmetic masking framework defined over finite fields. Moreover, its temporary sharings used to enable reuse are not plain additive sharings; instead, they follow an inner-product style encoding to avoid insecure cross-gadget leakage relations. We therefore restrict the application of Common Shares to the arithmetic subroutines of SDitH that naturally live in $\mathbb{F}$ and keep the rest under the SNI-style discipline above. (We defer the concrete boundary/conversion design to the countermeasure section.)

A key practical feature of Common Shares is that a large part of the masked computation can be shifted to a precomputation phase that depends only on randomness, reducing the online cost of repeated arithmetic kernels. We exploit this structure in the SDitH setting, where signing repeatedly invokes the same witness-dependent arithmetic patterns across τ repetitions.

2.5 Leakage Detection Methodology

We use the standard TVLA methodology based on Welch's t-test [11, 16]. Two trace sets are collected under identical measurement conditions according to a fixed-vs-random experiment. The fixed-vs-random split may vary either the protected input or the signing key, depending on the experiments; Section 5 states the class definition for each experiment.

Let $X_b[j]$ denote the sample at trace point j in class $b \in \{0,1\}$, and let n_b be the number of traces in that class. Welch's statistic at sample point j is $t(j) = \frac{\mu_0(j)-\mu_1(j)}{\sqrt{\frac{s_0^2(j)}{n_0}+\frac{s_1^2(j)}{n_1}}}$. Here $\mu_b(j)$ and $s_b^2(j)$ are the sample mean and sample variance of class b at sample point j. A sustained excursion of $|t(j)|$ beyond the experiment-specific threshold indicates statistically significant first-order leakage. We select the concrete thresholds in Section 5 following Ding et al. [11], based on the number of tested sample points and the trace-count sweep.

3 Leakage Analysis on Cortex-M4

This section characterizes the side-channel leakage surface of the SDitH-Hypercube signer (Algorithm 10 in [2]) on an ARM Cortex-M4. Our objective is leakage localization—to identify witness-related computations that materialize protocol-critical intermediates and to derive protection boundaries for Section 4—rather than to present an optimized key-recovery attack. We identify hotspots from the signing flow in Fig. 1 and validate them with fixed-vs-random TVLA [11, 16].

3.1 Hotspots and TVLA Methodology

To keep the presentation consistent across figures and evaluation tables, we use four short labels for recurring attack-critical hotspots in the signer:

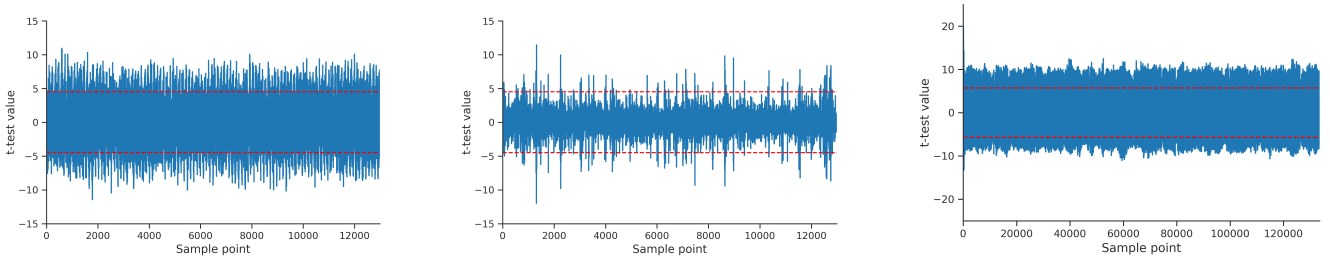
- Fill_N: The offline block in Algorithm 10 that (i) samples the first $N-1$ shares of witness-dependent inputs and multiplication preprocessing, (ii) evaluates nonlinear relations (notably $a \cdot b \mapsto c$), and (iii) computes the last-party correction packed into aux_e.
- $\mathsf{MPC}_{\mathsf{sec}}$: The online MPC evaluation driven by Fiat–Shamir challenges that processes witness-dependent MPC shares to produce transcript broadcasts.
- XOF: Keccak(SHAKE) sponge calls whose outputs are secret shares (e.g., $[\mathsf{wit}]_{e,i}$ and preprocessing shares such as $[a]_{e,i}, [b]_{e,i}, [c]_{e,i}$ for $i \neq N$), hence creating witness-correlated intermediates in the sponge state.
- Com_N: The commitment implementation for the distinguished last party $i = N$, whose committed state includes aux_e (witness-derived corrections). We treat this as a Keccak(SHA3) absorption of witness-related bytes.

For each hotspot, we run fixed-vs-random experiments that vary the secret-bearing input relevant to that hotspot (e.g., witness-dependent shares or aux_e-dependent bytes), while keeping public inputs constant. We align traces to the hotspot region and compute Welch's t-test point-wise [16]. We use TVLA here as a leakage screening tool (not as a security proof).

Figure 2 summarizes the three leakage sources discussed below. Each subfigure shows the TVLA t-statistic over time for traces aligned to the corresponding hotspot.

3.2 Leakage Source 1 (LS1): Fill_N.

This hotspot occurs in the offline stage of Fig. 1, namely Steps 1–2. In each repetition e, the signer samples secret-shared witness-related inputs ($[\mathsf{wit}] := [x_A], [P], [Q]$) and Beaver triples ($[a], [b], [c]$) using an XOF, and then computes the auxiliary correction value aux_e that completes the last



(a) Unprotected LS1 Case. (b) Unprotected LS2 Case. (c) Unprotected LS3 Case.

Figure 2: TVLA leakage localization for the unprotected Cortex-M4 signer. (a) targets Fill_N, where the sensitive values are Beaver completion values (a,b,c) with $c = a \cdot b$ and the last-party correction aux_e. (b) targets $\mathsf{MPC}_{\mathrm{sec}}$, where the sensitive operations are BN arithmetic terms on secret protocol shares, especially $\alpha \cdot [b]_i$ and $\beta \cdot [a]_i$. (c) targets the secret-bearing Keccak implementations: XOF share generation and Com_N absorption of aux_e.

party so that the shares reconstruct the plaintext values. Crucially, the offline preprocessing enforces the nonlinear relation $c_e = a_e \times b_e$, which is implemented with field multiplications on the target microcontroller.

Field multiplication is nonlinear and is a common leakage-sensitive operation on embedded targets. If an attacker obtains sufficient leakage from the implementation of $c_e = a_e \times b_e$ to recover the plaintext triple components (a_e, b_e, c_e) (or, equivalently, enough information to derive the missing shares for one party), then the MPCitH "Commit-and-Prove" structure turns this leakage into protocol-level information: for each repetition e, verification reveals (opens) the views of $N-1$ parties and keeps only one party index $i^\star(e)$ unopened. Hence, knowing a_e and the opened shares $\{[a]_{e,i}\}_{i \neq i^\star(e)}$ immediately yields the missing share $[a]_{e,i^\star(e)} = a_e - \sum_{i \neq i^\star(e)} [a]_{e,i}$, and analogously for $[b]_{e,i^\star(e)}$ and $[c]_{e,i^\star(e)}$ (with $c_e = a_e \times b_e$). Therefore, leakage of the offline nonlinearity can be turned into full recovery of the unopened party's Beaver-triple shares.

Once the unopened party's Beaver shares are known, the BN-protocol computations in Step 5 of Fig. 1 allow an attacker to derive additional unopened-party shares of witness-dependent intermediates. Indeed, Step 5 computes $[\alpha] = \varepsilon \cdot [Q(r)] + [a]$, $[\beta] = [S(r)] + [b]$, where ε and r are public Fiat–Shamir challenges and $[\alpha], [\beta]$ contribute to the public transcript. Rearranging share-wise for the unopened party gives $[Q(r)]_{i^\star(e)} = \varepsilon^{-1} \cdot \left([\alpha]_{i^\star(e)} - [a]_{i^\star(e)}\right)$, $[S(r)]_{i^\star(e)} = [\beta]_{i^\star(e)} - [b]_{i^\star(e)}$.

Similarly, the check value $[\nu]$ satisfies $[\nu] = \varepsilon \cdot [F \cdot P(r)] - [c] + \alpha \cdot [b] + \beta \cdot [a]$, so $[F \cdot P(r)]_{i^\star(e)}$ can be solved share-wise from $[\nu]_{i^\star(e)}$ once $[a]_{i^\star(e)}, [b]_{i^\star(e)}, [c]_{i^\star(e)}$ are known. Consequently, leakage from Fill_N can be propagated to unopened-party shares of $[Q(r)]$, $[P(r)]$-related terms, and other witness-dependent BN intermediates.

In SDitH, the secret key material is encoded in the sharings of witness-related objects (here abstracted as $[x_A], [P], [Q]$ in Step 1). Since the signature already provides all opened views (and hence $\{[x_A]_{e,i}, [P]_{e,i}, [Q]_{e,i}\}_{i \neq i^\star(e)}$), recovering the missing-party shares for these objects (directly via aux_e completion, or indirectly via the relations above at enough challenge points) enables reconstruction of the corresponding plaintext witness representation.

To localize this leakage source empirically, we run fixed-vs-random TVLA experiments aligned to the Fill_N execution (Steps 1–2), choosing the fixed/random classes so that the witness-related inputs (and thus the derived Beaver triples and aux_e) vary between classes while public inputs are held constant. Figure 2a shows representative leakage localized to Fill_N.

3.3 Leakage Source 2 (LS2): $\mathsf{MPC}_{\mathrm{sec}}$.

LS2 appears in the online BN arithmetic of Step 5, after the first Fiat–Shamir hash fixes the public challenges (r, ε). Although (r, ε) and the reconstructed transcript values are public, the party-local computations still manipulate secret protocol shares derived from the witness and preprocessing material. Since verification opens only $N-1$ views per repetition, leakage correlated with the hidden view remains security-critical.

The online computation is organized around the BN consistency checks: $[\alpha] = \varepsilon \cdot [Q(r)] + [a]$, $[\beta] = [S(r)] + [b]$, $[\nu] = \varepsilon \cdot [F \cdot P(r)] - [c] + \alpha \cdot [b] + \beta \cdot [a]$. The leakage-relevant operations are the public-scalar products on secret protocol shares, especially $\alpha \cdot [b]_i$ and $\beta \cdot [a]_i$ for party-local shares. These are implemented as field multiplications and dominate the side-channel surface on embedded targets.

LS2 is algebraically coupled with LS1 through the same BN relations. As discussed in LS1, once Beaver-triple shares for the unopened party are recovered (e.g., via leakage in the offline nonlinearity), the above relations allow an attacker to propagate this knowledge to unopened-party shares of $[Q(r)]$, $[S(r)]$, and $[F \cdot P(r)]$, and ultimately to the witness-related sharings. Therefore, in this section we focus on localizing leakage in $\mathsf{MPC}_{\mathrm{sec}}$ rather than repeating the full recovery strategy.

We collect the EM traces during the execution of the BN

online kernels and perform fixed-vs-random TVLA experiments with classes that vary the witness-dependent operands while keeping public inputs and challenges random. Figure 2b shows representative leakage localized to $\mathsf{MPC}_{\mathrm{sec}}$, motivating that the online arithmetic kernels must be protected by the same masking discipline as the offline completion.

3.4 Leakage Source 3 (LS3): XOF and Com_N.

LS3 corresponds to the Keccak-based sponge invocations in Fig. 1: the XOF expansions in Step 1 that generate protocol shares from per-party seeds, and the commitment computations in Step 3, especially the commitment for the distinguished last party whose committed state includes aux_e.

Keccak is a sponge construction built around a nonlinear permutation. Whenever the sponge squeezes secret-bearing outputs or absorbs secret-bearing inputs, the intermediate sponge states become nonlinear functions of secret material [20]. On embedded targets, such nonlinear state updates are common first-order leakage candidates when the sponge state carries secret-bearing data.

In SDitH-Hypercube signing (Step 1), the XOF output is not merely "randomness": it directly instantiates protocol-layer shares, including witness-related inputs ($[\mathsf{wit}] := [x_A], [P], [Q]$) and Beaver-triple shares that feed into the BN checks. Therefore, leakage in XOF is immediately informative about secret shares. Moreover, as shown in LS1, recovering even partial information about shares that participate in the BN arithmetic can be amplified by the MPCitH open-view structure (only one party remains hidden per repetition).

In Step 3, the last party's committed state is $\mathsf{state}_{e,N} = (\mathsf{seed}_{e,N} \parallel \mathsf{aux}_e)$. The auxiliary value aux_e packs the last-party correction shares needed to reconstruct plaintext witness/Beaver-triple shares from the $N-1$ seed-derived parties (Step 2). Consequently, the commitment sponge absorbs witness-derived bytes via aux_e. Since verification opens $N-1$ views, any additional leakage correlated with the remaining hidden view—in particular leakage from the sponge state while absorbing aux_e—directly reduces the uncertainty of the missing shares and can enable witness recovery when combined with the opened data (cf. LS1).

For TVLA, we collect the EM traces during the execution of Keccak, which corresponds to the XOF and Com_N computations in the implementation. The fixed and random classes vary the secret-bearing sponge I/O, including XOF outputs for share sampling and commitment inputs involving aux_e. Figure 2c shows representative leakage localized to these Keccak implementations, motivating that secret-bearing sponge calls must be covered by the protection strategy.

4 Countermeasures: A Global Masking Strategy

This section presents a protocol-compatible protection strategy for the three attack-critical hotspots identified in Section 3: offline nonlinear preprocessing and last-party completion (LS1), online BN arithmetic (LS2), and secret-bearing Keccak share generation and commitment (LS3). The countermeasures do not change the SDitH parameters, transcript format, Fiat–Shamir flow, openings, or verifier interface. They only change how the signer computes sensitive intermediates, by defining explicit protection boundaries and reconstruction points aligned with protocol-public outputs.

Design principles. Our strategy follows three principles: (i) localize and isolate only attack-critical computations, (ii) minimize fresh masking randomness by amortizing expensive nonlinear masking where possible, and (iii) reconstruct early whenever values are public by protocol design or verifier-reconstructible, so that we avoid promoting public scalars into Sec values.

4.1 Global Strategy Overview and Domain Discipline

Figure 3 summarizes the strategy for one SDitH repetition. The key idea is to draw an explicit security boundary around computations that (directly or indirectly) determine the missing party view, while keeping ephemeral-seed-only and transcript-public computations outside.

We use a type-and-representation discipline to describe boundary crossings. The type records whether a value is secret-dependent, while the representation records whether it is stored in the clear or as a masked encoding. These are separate notions. A Sec value may be reconstructed once it becomes protocol-public or verifier-reconstructible. Conversely, a Pub value may be encoded as $\langle x \rangle$ when a protected backend expects masked inputs, but its semantic type remains Pub.

- **Sec values.** A value is Sec if it is witness-dependent, hidden-view-dependent, or an intermediate whose leakage can be combined with the public transcript, as shown in Section 3. This includes (i) LS1 offline completion and nonlinear preprocessing, (ii) LS2 BN-kernel computations on witness-dependent protocol shares, and (iii) LS3 Keccak implementations that generate protocol shares or absorb aux_e for the last-party commitment. Inside protected computations, Sec values are represented as order-d encodings $\langle \cdot \rangle$ and processed using the selected masked backend.

- **Pub values.** A value is Pub if it is public by protocol design or verifier-reconstructible from the transcript and openings. This includes commitments, Fiat–Shamir challenges, reconstructed transcript values, and public correction terms. Pub values may be kept in the clear. If a masked backend requires a masked input interface, a Pub value may be lifted

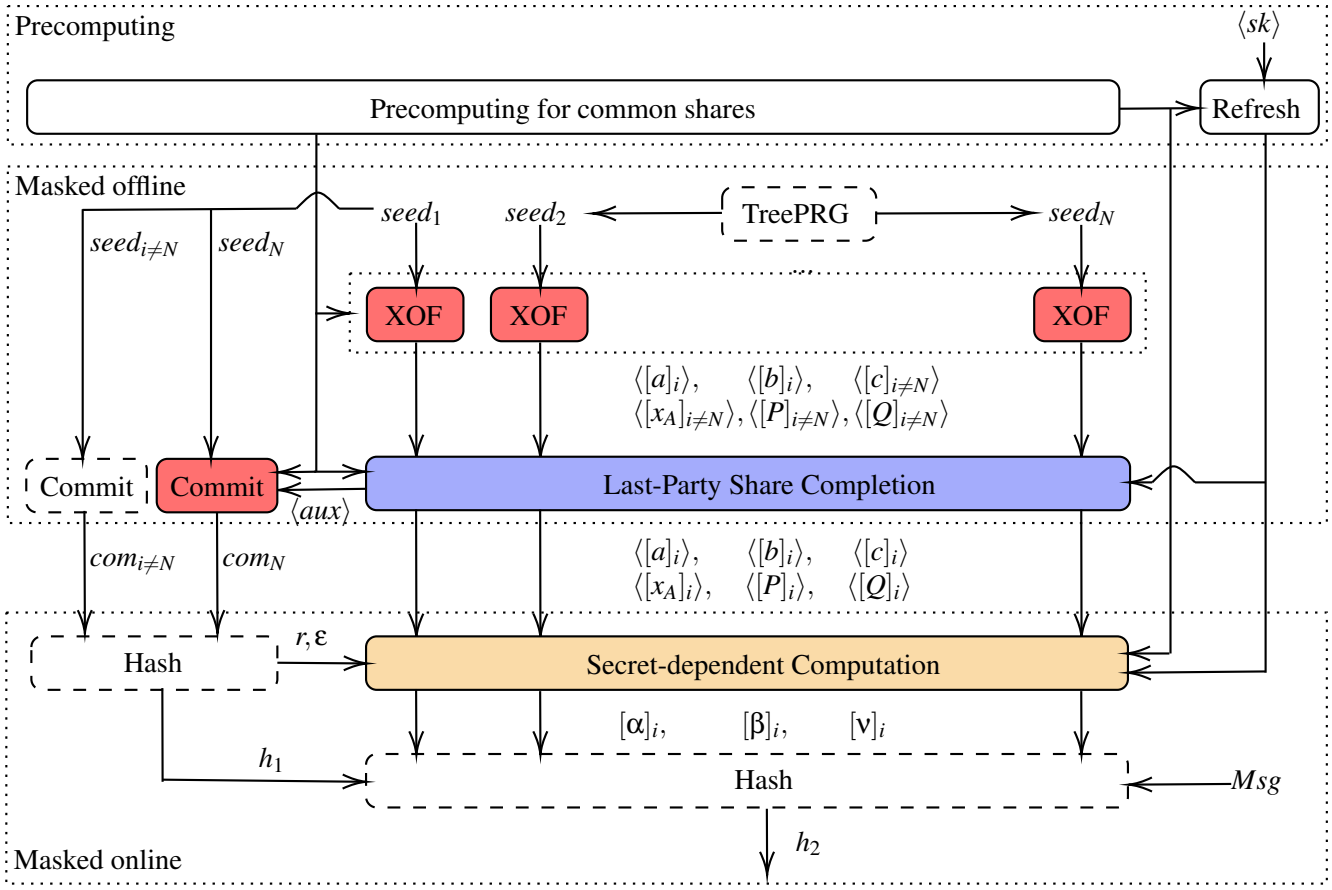


Figure 3: Framework of our protection scheme for a single instance among the τ MPC instances. Each rectangular block denotes an operation: white solid blocks denote precomputations that enable later countermeasures (common-share precomputation and state refresh); red solid blocks implement protected primitives (XOF expansion and commitment computation); blue solid block protects the offline Last-party share completion, where the auxiliary value is used to complete the last share; yellow solid block protects the online secret-dependent computation that produces the broadcast quantities; and white blocks with dashed borders are unprotected. The dashed TreePRG and $\mathrm{Com}_{i\neq N}$ blocks are unprotected only under the Seed boundary of Section 4.1.

to a masked encoding, but subsequent computations should still exploit its public status whenever possible.

- **Seed values.** A value has type Seed if it is a fresh, per-signature, domain-separated seed or seed-tree node that contains no long-term witness bits. The Seed type is not Pub. In the selective profiles, Seed values are not encoded at the masking layer while they are used only for TreePRG, seed-only commitments, or as inputs to protected share-generation XOF calls. This is a bounded-leakage trade-off following the seed-only boundary used in protected MPCitH implementations such as Picnic [3]: bounded leakage from a single-use seed is treated as a bounded loss in seed hiding rather than as long-term witness leakage. Complete single-trace recovery of a hidden seed or full seed-tree state is not covered by this selective claim. Once seed-derived output is interpreted as a protocol share, e.g., $[wit]_{e,i}$, $[a]_{e,i}$, $[b]_{e,i}$, or $[c]_{e,i}$, the value is promoted to Sec.

  Likewise, Com_N is Sec because it absorbs aux_e, whereas $\mathrm{Com}_{i\neq N}$ remains seed-only under the bounded Seed model.

Two boundary invariants make the protected surface analyzable. First, a Sec value is never released in the clear unless it has become protocol-public or verifier-reconstructible. Second, once a value is known to be Pub, later computations use the cheapest sound public-operand interface. In particular, combining a Pub scalar with a Sec value is evaluated as a share-wise Sec×Pub operation, not as a fresh Sec×Sec gadget. If the implementation lifts the public value to a masked encoding for interface uniformity, the value remains semantically Pub.

Boundary conversions are explicit:

- **Plain → masked encoding.** When a value must enter a protected computation in masked form, we encode it as $\langle x\rangle = (x^{(0)},\ldots,x^{(d)})$. For a Sec value, we sample d fresh masks $r_1,\ldots,r_d \xleftarrow{\$} \mathbb{F}$ and set $x^{(j)} := r_j \quad (1 \le j \le$

$d), x^{(0)} := x - \sum_{j=1}^{d} r_j$. Thus $\sum_{j=0}^{d} x^{(j)} = x$, while individual shares are randomized. For a Pub value, this lifting is used only to satisfy a masked interface; the value remains public and should still be handled through Sec×Pub or affine share-wise operations whenever possible.

- **Masked encoding → plain Pub.** When a masked value becomes public by protocol definition, or is verifier-reconstructible from the transcript and openings, we reconstruct $x := \sum_{j=0}^{d} x^{(j)}$ and handle it as Pub thereafter. This reconstruction is allowed only at such protocol-public boundaries.
- **Sec×Pub operations.** A public affine or scalar map applied to a Sec value is evaluated share-wise and consumes no fresh masking randomness. The result remains Sec whenever the input is secret-dependent. We denote this randomness-free share-wise affine operation by TrivAff.

4.2 Countermeasure for LS1 (Fill_N)

Field and the last-share identity. For LS1, all protocol-layer values live in $\mathbb{F} = \mathbb{F}_{2^{32}}$, hence $x - y = x + y$. For $c = a \cdot b$ and additive MPC sharing $[c] = ([c]_1, \ldots, [c]_N)$,

$$[c]_N \;=\; a \cdot b \;-\; \sum_{i=1}^{N-1} [c]_i \;=\; a \cdot b \;+\; \sum_{i=1}^{N-1} [c]_i \qquad \text{in } \mathbb{F}_{2^{32}}.$$

In the protected implementation we compute $\langle [c]_N \rangle$ without reconstructing any secret.

ISW/SNI profile (fresh masking randomness per multiplication). Using an ISW-style masked multiplication [18], we obtain $\langle a \cdot b \rangle$ from masked inputs and compute

$$\langle [c]_N \rangle := \mathsf{Mul}_{\mathsf{ISW}}(\langle a \rangle, \langle b \rangle) \;+\; \sum_{i=1}^{N-1} \langle [c]_i \rangle.$$

This approach is composable but consumes fresh masking randomness for each multiplication.

Common Shares profile (RMul/RAdd with reusable structured randomness). We follow Wang et al. [27] and use the output-refreshed gadgets RMul/RAdd. With parameter vectors $\gamma_{\mathrm{mul}}, \gamma_{\mathrm{add}} \in \mathbb{F}^d$ and reusable random matrices $\{R_x, R_y, R_z, R\}$, we compute

$$\langle [c]_N \rangle := \mathsf{RAdd}^{R_x,R_y,R_z}_{\gamma_{\mathrm{add}}}\Big(\mathsf{RMul}^{R_x,R_y,R_z,R}_{\gamma_{\mathrm{mul}}}(\langle a \rangle, \langle b \rangle),\; \sum_{i=1}^{N-1} \langle [c]_i \rangle\Big).$$

The soundness of this reuse is exactly the Common Shares MDS condition. Many RMul/RAdd gadgets may reuse the same structured randomness $\{R_x, R_y, R_z, R\}$ only when their gadget-label matrix satisfies the MDS condition required by the Common Shares composition theorem (Theorem 2 in [27]). We therefore choose γ_{mul} and γ_{add} as rows of an MDS-derived schedule over $\mathbb{F}_{2^{32}}$.

4.3 Countermeasure for LS2 ($\mathsf{MPC}_{\mathrm{sec}}$)

LS2 arises in the BN online kernels after Fiat–Shamir fixes the public challenges (r, ε). A naive protected implementation would keep challenge-derived quantities in masked form, inflating the number of Sec×Sec multiplications. We instead adopt a type-driven strategy inspired by protected MPCitH engineering [3] and adapt it to BN arithmetic by reconstructing Pub values aggressively.

We use the Sec/Pub discipline defined above. The key rule for LS2 is public-by-reconstruction: as soon as α, β, or any other BN operand becomes transcript-public or verifier-reconstructible, we reconstruct it and use it as a Pub operand in later computations.

Example: BN online checks. After Hash_1, (r, ε) are Pub. We first compute

$$\langle [\alpha]_i \rangle \leftarrow \varepsilon \cdot \langle [Q(r)]_i \rangle \;+\; \langle [a]_i \rangle,$$
$$\langle [\beta]_i \rangle \leftarrow \langle [S(r)]_i \rangle \;+\; \langle [b]_i \rangle,$$

where multiplication by ε is share-wise. Since the transcript reveals $\alpha = \sum_i [\alpha]_i$ and $\beta = \sum_i [\beta]_i$, we reconstruct α, β immediately and use them as Pub. The remaining BN check is evaluated as

$$\langle [\nu]_i \rangle = \varepsilon \cdot \langle [F \cdot P(r)]_i \rangle - \langle [c]_i \rangle \;+\; \alpha \cdot \langle [b]_i \rangle \;+\; \beta \cdot \langle [a]_i \rangle,$$

where $\alpha \cdot \langle [b]_i \rangle$ and $\beta \cdot \langle [a]_i \rangle$ are Sec×Pub operations and they require no nonlinear masked multiplication and do not consume fresh masking randomness.

4.4 Countermeasure for LS3 (XOF and Com_N)

LS3 concerns Keccak sponge calls whose I/O is secret-bearing: (i) XOF expansions whose outputs are secret MPC shares (XOF) and (ii) last-party commitments that absorb aux_e (Com_N). We therefore apply selective masked Keccak only at these secret-bearing call sites. Transcript-only hashing remains unmasked because its inputs are public, while seed-only TreePRG and $\mathrm{Com}_{i \neq N}$ remain unmasked only under the bounded Seed boundary of Section 4.1. This mirrors the selective-hashing rationale in protected Picnic [3]: random-tape/share-generation outputs and commitments absorbing auxiliary witness-derived data are protected, whereas seed-only hashing can remain outside the masked surface under the bounded seed-leakage assumption.

Tunable masked Keccak backends. We support three backends with increasing robustness and cost: independent-share (IND) [8], Domain-Oriented Masking (DOM) [17], and SNI [5]. We mask exactly those sponge invocations that process secret-bearing bytes (XOF/Com_N), so that the high cost of strong masked hashing does not spill over to purely public transcript hashing.

Table 1: Online fresh masking randomness per signature at L1.

Arithmetic profile	Keccak backend	XOF+Com$_N$ (KiB)[a]	Fill$_N$ (B)	MPC$_{sec}$ (B)	Refresh (B)	Online Total (KiB)
SNI	IND	0	204	3,264	5,100	8.37
	DOM	41,118.75	204	3,264	5,100	41,127.12
	SNI	82,237.50	204	3,264	5,100	82,245.87
SNI+opt	IND	0	204	0	5,100	5.18
	DOM	41,118.75	204	0	5,100	41,123.93
	SNI	82,237.50	204	0	5,100	82,242.68
RNI/RNI+opt	IND	0	16	0	0	0.016
	DOM	41,118.75	16	0	0	41,118.77
	SNI	82,237.50	16	0	0	82,237.52

[a] This column counts only additional online fresh masking randomness used by nonlinear subcomponents inside protected secret-bearing XOF and Com$_N$ sponge calls; it excludes initial Keccak state re-sharing. The initial randomness is 69,904 B per signature for each IND, DOM, and SNI backend; related work may call this offline or initial randomness [19, 24], but we use initial randomness to avoid confusion with SDitH's offline arithmetic phase.

5 Implementation and Evaluation

We evaluate the protected SDitH-Hypercube signer on the L1 parameter set. The evaluation is signer-centric: repeated signing is the side-channel target in our threat model, whereas key generation is one-time and verification does not use the long-term witness. We therefore report signing cost, signing footprint, fresh masking randomness, and first-order leakage detection for the protected signing implementation.

All implementation results in this section are measured on STM32F407DISCOVERY. Unless stated otherwise, memory sizes use binary units, with 1 KiB = 1024 B.

5.1 Profiles and Measurement Scope

We evaluate two largely orthogonal protection choices. The arithmetic profile is applied to the LS1 and LS2 hotspots. SNI denotes the baseline ISW/SNI-style arithmetic masking. RNI denotes our Common-Shares-based arithmetic profile, which amortizes masked multiplications and refreshes across the RMul/RAdd batch. The suffix `+opt` denotes the Sec/Pub optimization: once a value is transcript-public or verifier-reconstructible, we reconstruct it and use it as Pub. Public scalar multiplication is then evaluated share-wise and randomness-free, but the result remains Sec whenever the input is secret-dependent.

The Keccak backend is applied to the LS3 hotspot. We selectively mask only secret-bearing XOF and Com$_N$ calls; sponge calls that process only public transcript data remain unmasked. We instantiate the protected call sites with three backends: IND, DOM, and SNI. The IND backend is included as a low-cost first-order diagnostic configuration. It provides first-order independence at the protected call sites, but it does not provide the same composability guarantees as DOM or SNI. We therefore treat DOM and SNI as the higher-assurance deployment profiles.

Cycle counts are measured on the target board for the complete signing routine. Flash is counted as `.text + .rodata + .ARM.exidx + .data`. Static RAM is counted as `.data + .bss + .ram2 + .ccm`. The unprotected baseline also follows an embedded low-stack implementation style. Therefore, it uses little stack, but some temporary workspace has already been moved to static RAM. This baseline is therefore a fair low-stack signing baseline, not a minimum-RAM reference build.

5.2 Fresh Masking Randomness

Table 1 reports online fresh masking randomness per signature. SDitH seeds, salt, Fiat–Shamir challenges, and protocol tapes are unchanged and are not included. Here, protocol tapes denote the deterministic per-signature randomness expanded from SDitH seeds and consumed by the MPC-in-the-Head simulation. Our countermeasures do not reduce, bias, or reuse these protocol-level random values.

The main arithmetic takeaway is visible from the IND rows, where the Keccak backend contributes no additional online sponge randomness under the table's accounting convention. The unoptimized SNI arithmetic profile requires 204 B for Fill$_N$, 3,264 B for MPC$_{sec}$, and 5,100 B for explicit refreshes, for a total of 8.37 KiB per signature. The `+opt` profile removes the MPC$_{sec}$ term entirely, reducing the arithmetic total to 5.18 KiB.

Common Shares changes where arithmetic masking randomness is charged. In the RNI profile, the online MPC$_{sec}$ protected computation is still present, but its RMul/RAdd calls do not sample new masking randomness: they reuse the Common Shares randomness already charged to Fill$_N$ under the same MDS schedule. In RNI+opt, the MPC$_{sec}$ entry is zero for a different reason: the Sec/Pub typing removes the online Sec$\times$Sec gadgets, as in SNI+opt. Similarly, the Refresh entry is zero for both RNI and RNI+opt: the RNI Refresh calls used by RMul/RAdd reuse the Fill$_N$ Common Shares randomness

Table 2: End-to-end signing cost and footprint on STM32F407DISCOVERY.

Keccak setting	Metric	Arithmetic profile			
		SNI	SNI+opt	RNI	RNI+opt
IND	Cycles (Mcyc)	915.8 [1.84×]	913.7 [1.84×]	933.5 [1.88×]	917.7 [1.85×]
	Footprint (KiB)	138.9 / 135.0 / 7.96	138.9 / 135.0 / 7.96	140.4 / 135.1 / 7.96	138.6 / 136.7 / 7.96
DOM	Cycles (Mcyc)	1,235.4 [2.49×]	1,233.3 [2.48×]	1,253.1 [2.52×]	1,237.3 [2.49×]
	Footprint (KiB)	146.9 / 135.0 / 7.96	146.9 / 135.0 / 7.96	148.4 / 135.1 / 7.96	146.6 / 136.7 / 7.96
SNI	Cycles (Mcyc)	1,524.3 [3.07×]	1,522.1 [3.06×]	1,542.0 [3.10×]	1,526.2 [3.07×]
	Footprint (KiB)	154.1 / 135.0 / 7.96	154.1 / 135.0 / 7.96	155.6 / 135.1 / 7.96	153.9 / 136.7 / 7.96
Unprotected baseline	Cycles (Mcyc)	496.6 [1.00×]			
	Footprint (KiB)	69.0 / 110.4 / 1.04			

Mcyc denotes 10^6 cycles. Slowdowns are normalized to the unprotected signer. In the footprint row, values are reported as flash/static-RAM/signing-stack footprint, all in KiB. The unprotected baseline is the build with no arithmetic masking and no masked-Keccak backend.

and are therefore not charged as a separate refresh stream. Thus, the only fresh arithmetic masking randomness charged to the RNI rows is the 16 B in Fill_N.

The table groups RNI and RNI+opt because their fresh-randomness totals are identical, although the reason for the zero $\mathsf{MPC}_{\mathsf{sec}}$ entry differs and their costs and footprints differ in Table 2. The reuse is confined to the arithmetic batch of a single signature: the Common Shares randomness is sampled freshly per signature and is never reused across signatures.

The security condition for this reuse is the MDS condition of Common Shares: the RMul/RAdd gadget labels must form an MDS-derived parameter matrix. In our implementation, the Common Shares gadget labels are fixed by the MDS schedule, while the reusable random matrices are freshly sampled per signature.

Once arithmetic masking randomness is amortized, masked Keccak dominates the randomness budget. With DOM-Keccak, the protected $\mathsf{XOF+Com}_N$ call sites add 41,118.75 KiB per signature. With SNI-Keccak, they add 82,237.50 KiB. Thus, Common Shares and Sec/Pub typing remove the arithmetic bottleneck, but strong masked sponge backends remain the dominant source of fresh masking randomness. This is why selective masking of secret-bearing sponge calls is necessary rather than merely an implementation optimization.

5.3 End-to-End Signing Cost and Footprint

Table 2 reports end-to-end signing cycles and footprint on STM32F407DISCOVERY.

The unprotected low-stack signer takes 496.6 Mcycles and has a footprint of 69.0/110.4/1.04 KiB for flash/static-RAM/signing-stack. The fastest protected setting is SNI+opt with the IND Keccak backend: it signs in 913.7 Mcycles, a 1.84× slowdown, with a footprint of 138.9/135.0/7.96 KiB. The fastest Common Shares setting is RNI+opt with IND-Keccak; it signs in 917.7 Mcycles (1.85×) and uses 138.6/136.7/7.96 KiB.

The stronger Keccak backends increase both runtime and code size. With DOM-Keccak, the fastest profile is SNI+opt, which signs in 1,233.3 Mcycles (2.48×), while the Common Shares profile RNI+opt signs in 1,237.3 Mcycles (2.49×). With SNI-Keccak, the protected profiles fall in a narrow range from 1,522.1 to 1,542.0 Mcycles. The RNI+opt+SNI-Keccak point signs in 1,526.2 Mcycles (3.07×) and has a footprint of 153.9/136.7/7.96 KiB.

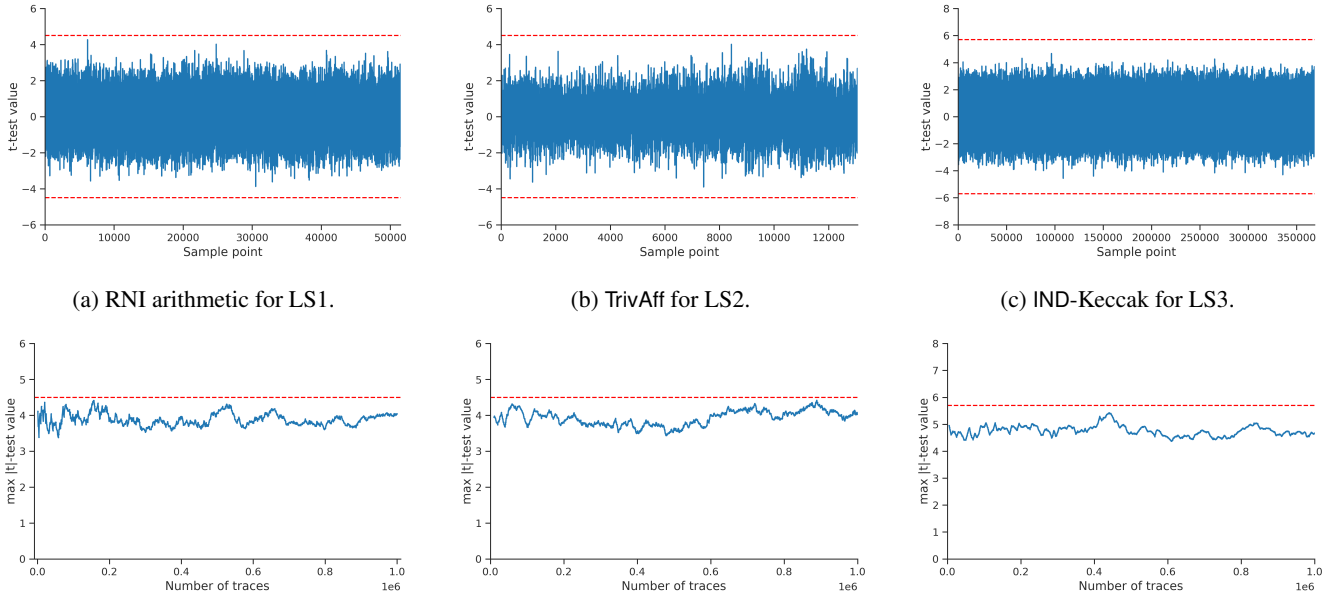
The memory overhead is moderate in static RAM and larger in flash. For RNI+opt with SNI-Keccak, flash grows from 69.0 to 153.9 KiB (2.23×). Static RAM grows from 110.4 to 136.7 KiB (1.24×). The signing stack grows from 1.04 to 7.96 KiB. The large stack ratio is a consequence of the low-stack baseline: large reference temporaries are moved into static workspace, whereas the protected signer still carries a deeper masked call chain.

Two trends are stable across Table 2. First, the Keccak backend is the main performance knob: for RNI+opt, moving from IND to DOM and then to SNI increases signing time from 917.7 to 1,237.3 and 1,526.2 Mcycles. Second, for a fixed Keccak backend, the arithmetic profile has a secondary effect on total signing time. This matches Table 1: once arithmetic masking randomness is amortized, the protected sponge implementation dominates the remaining cost.

5.4 Leakage Validation

We validate the first-order protected versions on an STM32F407DISCOVERY board. The target is clocked at 24 MHz during trace acquisition, and EM traces are collected at a sampling rate of 250 MS/s using a near-field probe and a PicoScope 5244D oscilloscope. This gives approximately 10.4× oversampling. We run two first-order TVLA experiments using the Welch t-test defined in Section 2.5. The first experiment is a fixed-vs-random input test for the three protected hotspots. The second experiment is a fixed-vs-random key test with randomized messages for the RNI+opt+IND-Keccak protected signer.

The labels LS1–LS3 refer to the attack-critical hotspots identified in Section 3. Here we evaluate the protected ver-



(a) RNI arithmetic for LS1.

(b) TrivAff for LS2.

(c) IND-Keccak for LS3.

(d) Max-$|t|$ curve for protected LS1.

(e) Max-$|t|$ curve for protected LS2.

(f) Max-$|t|$ curve for protected LS3.

Figure 4: First-order fixed-vs-random TVLA for the protected versions of the three attack-critical hotspots. The top row shows point-wise Welch t-statistics over aligned trace segments. The bottom row shows the maximum absolute t-value over the same segment as the trace count increases.

sions corresponding to those hotspots. LS1 is the Fill_N region, where the unprotected signer manipulates nonlinear $\mathsf{Sec}\times\mathsf{Sec}$ arithmetic; the protected implementation uses the RNI arithmetic implementation. LS2 is the $\mathsf{MPC}_{\mathsf{sec}}$ region, where the unprotected signer evaluates secret-dependent BN arithmetic; after Sec/Pub typing, the tested version uses randomness-free share-wise $\mathsf{Sec}\times\mathsf{Pub}$ affine arithmetic. LS3 is the secret-bearing $\mathsf{XOF+Com}_N$ sponge region; in the TVLA experiment below, this region is protected with the IND-Keccak backend.

The dashed red lines in Figure 4 show the detection thresholds used for each aligned trace segment. Following Ding et al. [11], these thresholds are selected according to the number of tested sample points and the trace-count sweep. We use $\theta_{\mathrm{LS1}} = 4.5$ for LS1 and $\theta_{\mathrm{LS2}} = 4.5$ for LS2. The LS3 Keccak segment contains many more sample points, so we use the adjusted threshold $\theta_{\mathrm{LS3}} = 5.7$ to avoid inflating the false-positive rate when taking a maximum over the longer segment.

Figure 4 shows that the protected regions exhibit no persistent first-order leakage under our measurement setup. In the final 1M-trace point-wise tests, LS1 and LS2 remain within the $\theta = 4.5$ threshold, and LS3 remains within the adjusted $\theta = 5.7$ threshold. The max-$|t|$ curves are convergence diagnostics: they recompute the maximum statistic over the aligned trace segment on increasing trace prefixes.

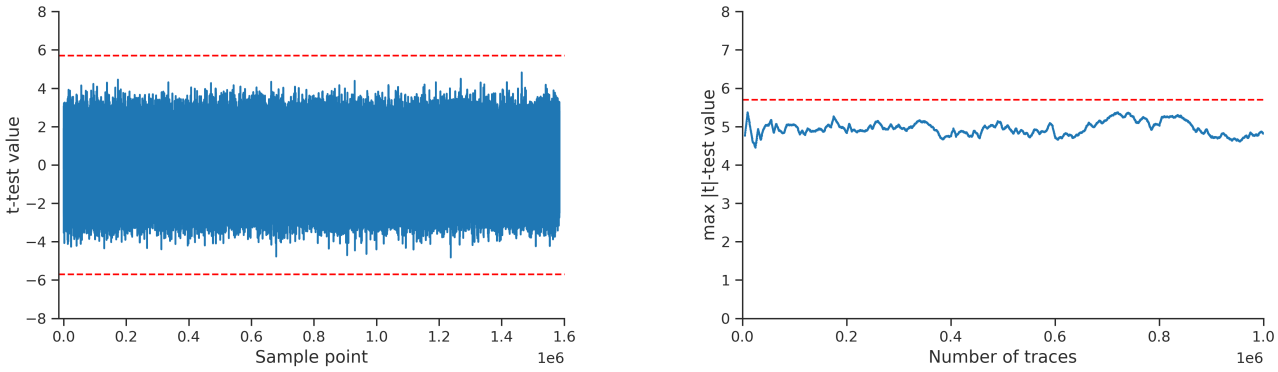
We also run a fixed-vs-random key experiment for the RNI+opt+IND-Keccak protected signer. In this experiment, the fixed class uses a fixed signing key and randomized messages, while the random class samples a randomized signing key and also uses randomized messages. This test targets key-dependent leakage without turning message-dependent variation into a fixed distinguisher. The tested trace segment spans the protected Fill_N region (last-party share completion) and the protected Com_N sponge calls.

Figure 5 shows the corresponding fixed-vs-random-key result. Both the final point-wise t-statistics and the max-$|t|$ curve remain below $\theta_{\mathrm{key}} = 5.7$. Under this experiment, we therefore do not observe a persistent first-order key-dependent leakage peak in the protected RNI+opt+IND-Keccak signer.

TVLA is a leakage-detection test, not a proof of side-channel security. Passing first-order fixed-vs-random TVLA under this setup does not rule out higher-order leakage, horizontal attacks, different preprocessing, different probes, different microarchitectures, or active faults. The experiment is a necessary sanity check for the selected protection boundaries, but not a comprehensive leakage-absence guarantee.

5.5 Scope and Practical Limits

Our countermeasures are implementation-level. They compute the same SDitH transcript, openings, and verifier inputs as the unprotected signer. The SDitH parameters, Commit-and-Prove structure, Fiat–Shamir challenges, seeds, salt, and protocol tapes are unchanged. Randomness reduction concerns only masking randomness: Common Shares reuses masking randomness inside RMul/RAdd gadgets under the



(a) Masked SDitH (part).

(b) Max-$|t|$ curve for Masked SDitH (part).

Figure 5: First-order fixed-vs-random key TVLA for the RNI+opt+IND-Keccak protected signer with randomized messages. The measured trace segment covers last-party share completion and the protected commit implementation.

MDS conditions, and Sec/Pub typing removes unnecessary online masked arithmetic once values are public or verifier-reconstructible.

The masking claims require high-entropy and independent masks. If the TRNG output used for masks is biased, repeated, or correlated across signatures, the masking shares may lose statistical independence, and correlations can accumulate across signatures. Such failures invalidate the SNI/RNI assumptions. Fault injection, clock or voltage glitching, and invasive probing are outside our passive leakage model. Standard mitigations such as redundancy, control-flow checks, infective checks, or recomputation can be combined with our masking strategy, but evaluating active-fault resistance is separate from the present work.

The measurements identify three practical operating points. The fastest protected signer is SNI+opt with IND-Keccak, which signs in 913.7 Mcycles; this is the performance-oriented configuration, but IND provides only first-order independence and lacks the composability guarantees of DOM and SNI. The Common Shares profiles provide the low-randomness arithmetic option: RNI+opt reduces online arithmetic masking randomness to 16 B/signature and signs in 917.7 Mcycles with IND-Keccak. For high-assurance deployments, the relevant backend is SNI-Keccak. In that setting, SNI+opt is marginally faster (1,522.1 Mcycles), while RNI+opt keeps the Common Shares randomness reduction and signs in 1,526.2 Mcycles with a footprint of 153.9 KiB flash, 136.7 KiB static RAM, and 7.96 KiB signing stack. Thus, protected signing fits on the target Cortex-M4 platform, and the remaining embedded bottleneck is masked Keccak rather than Common Shares arithmetic.

6 Conclusion

In this paper, we present a protocol-compatible protection architecture for SDitH-Hypercube against side-channel leakage in MPCitH-based signing. Our design targets the three attack-critical signer hotspots—last-party completion, BN arithmetic, and secret-bearing Keccak calls—without changing SDitH parameters, Fiat–Shamir flow, transcript format, openings, or the verifier interface. By combining Common Shares, Sec/Pub typing, and selective masked Keccak, the protected signer reduces arithmetic masking randomness from 8.37 KiB/signature to 16 B/signature and removes 3,264 B/signature from the online BN kernels. End-to-end measurements on Cortex-M4 show signing costs of 913.7–1,542.0 Mcycles within 155.6 KiB flash, 136.7 KiB static RAM, and 7.96 KiB stack, while the 1M-trace first-order TVLA experiments no longer detect the leakage peaks observed in the unprotected implementation. The results show that arithmetic masking is practical for SDitH-Hypercube and that masked Keccak becomes the dominant remaining deployment cost under our passive first-order leakage model.

Ethical Considerations

Stakeholders. The stakeholders of this work include implementers of post-quantum signature schemes, embedded-device vendors, standards bodies, evaluators of side-channel resistance, and users who may rely on post-quantum signatures in deployed systems. The work does not involve human subjects, personal data, or third-party systems.

Potential negative outcomes. The main risk is dual use: leakage localization may help an attacker prioritize measurements on unprotected SDitH implementations. We mitigate this risk by framing the analysis around defensive boundaries, by evaluating countermeasures, and by not releasing attack automation for key recovery against third-party devices. All experiments were conducted on our own hardware in a controlled laboratory setting.

Responsible disclosure. Our experiments target our own implementation and public reference-style code, not deployed

third-party devices. No vulnerability in a live third-party system was exploited. Artifacts are released to support reproducibility of the defensive claims.

Decision to publish. We publish because the defensive benefit—showing how to harden an emerging post-quantum signature on constrained microcontrollers—outweighs the residual risk, and because the attack surface is already understood in the literature.

Open Science

To support reproducibility and facilitate follow-up research, we release our code, scripts, and experimental artifacts at: https://doi.org/10.5281/zenodo.20432040.

Acknowledgments

We thank all the anonymous reviewers and the shepherd for their valuable comments. This work is supported by the National Key R&D Program of China (Grant No. 2024YFA1013000), the National Natural Science Foundation of China (Grant No. U2336207), the National Cryptologic Science Fund of China (Grant No. 2025NCSF01013), and the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM114).

References

[1] Carlos Aguilar Melchor, Slim Bettaieb, Loïc Bidoux, Thibauld Feneuil, Philippe Gaborit, Nicolas Gama, Shay Gueron, James Howe, Andreas Hülsing, David Joseph, Antoine Joux, Mukul Kulkarni, Edoardo Persichetti, Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue. The syndrome decoding in the head (sd-in-the-head) signature scheme: Algorithm specifications and supporting documentation, version 2.0. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/sdith-spec-round2-web.pdf, February 2025. NIST Post-Quantum Cryptography Project, Round 2.

[2] Carlos Aguilar Melchor, Thibauld Feneuil, Nicolas Gama, Shay Gueron, James Howe, David Joseph, Antoine Joux, Edoardo Persichetti, Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue. The Syndrome Decoding in the Head (SD-in-the-Head) Signature Scheme: Algorithm Specifications and Supporting Documentation – Version 1.1. https://www.sdith.org/docs/sdith-v1.1.pdf, November 2023.

[3] Diego F. Aranha, Sebastian Berndt, Thomas Eisenbarth, Okan Seker, Akira Takahashi, Luca Wilke, and Greg Zaverucha. Side-channel protections for Picnic signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 239–282, 2021.

[4] Diego F. Aranha, Johan Degn, Jonathan Eilath, Kent Nielsen, and Peter Scholl. FAEST for memory-constrained devices with side-channel protections. *Cryptology ePrint Archive*, 2025. URL: https://eprint.iacr.org/2025/1261.

[5] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 116–129, 2016.

[6] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *IACR International Conference on Public-Key Cryptography*, pages 495–526. Springer, 2020.

[7] Ryad Benadjila and Thibauld Feneuil. Breaking the myth of MPCitH inefficiency: Optimizing MQOM for embedded platforms. *Cryptology ePrint Archive*, 2026. URL: https://eprint.iacr.org/2026/078.

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Building power analysis resistant implementations of Keccak. In *Second SHA-3 candidate conference*, 2010.

[9] Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Marco Palumbi, and Lucas Pandolfo Perin. Enabling PERK and other MPC-in-the-Head signatures on resource-constrained devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(4):84–109, 2024.

[10] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 342–375. Springer, 2020.

[11] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In *International Conference on Smart Card Research and Advanced Applications*, pages 105–122. Springer, 2017.

[12] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In *Annual International Cryptology Conference*, pages 541–572. Springer, 2022.

[13] Thibauld Feneuil, Matthieu Rivain, and Auguste Warmé-Janville. Masking-friendly post-quantum signatures in the Threshold-Computation-in-the-head framework. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):667–710, 2025.

[14] Tim Gellersen, Okan Seker, and Thomas Eisenbarth. Differential power analysis of the picnic signature scheme. In *International Conference on Post-Quantum Cryptography*, pages 177–194. Springer, 2021.

[15] Julie Godard, Nicolas Aragon, Philippe Gaborit, Antoine Loiseau, and Julien Maillard. Single trace side-channel attack on the MPC-in-the-Head framework. In *International Conference on Post-Quantum Cryptography*, pages 267–293. Springer, 2025.

[16] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[17] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of KECCAK. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 205–212. IEEE, 2017.

[18] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[19] Vahid Jahandideh, Bart Mennink, and Lejla Batina. Probing secure composability without fresh randomness: Theory and application to Ascon. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):972–1009, 2025.

[20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on KECCAK. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):243–268, 2020.

[21] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4, 2019.

[22] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[23] National Institute of Standards and Technology (NIST). Post-quantum cryptography: Additional digital signature schemes (round 2 additional signatures). https://csrc.nist.gov/Projects/pqc-dig-sig, 2024. Accessed: 2026-02-01.

[24] Artemii Ovchinnikov, Aein Rezaei Shahmirzadi, and Siemen Dhooghe. Uniform sharing in multiple stages: NullFresh for arbitrary functions. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(2):893–927, 2026.

[25] Linda Pirker, Quinten Norga, Suparna Kundu, Anindya Ganguly, Barry van Leeuwen, Angshuman Karmakar, and Ingrid Verbauwhede. HYPERSHIELD: Protecting the hypercube MPC-in-the-head framework against differential probing adversaries without masking. *Cryptology ePrint Archive*, 2026. URL: https://eprint.iacr.org/2026/081.

[26] Okan Seker, Sebastian Berndt, Luca Wilke, and Thomas Eisenbarth. SNI-in-the-head: Protecting MPC-in-the-Head protocols against side-channel analysis. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1033–1049, 2020.

[27] Weijia Wang, Chun Guo, Yu Yu, Fanjie Ji, and Yang Su. Side-channel masking with common shares. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):290–329, 2022.