A Systematic Evaluation of Fault Attacks and Countermeasures on ML-DSA Rejection Sampling

Authors: Yulong Meng **Affiliation**: Henan Normal University

Abstract: This paper presents a systematic evaluation of fault attacks on the rejection sampling of ML-DSA (FIPS 204) and of the countermeasures proposed against them. Using the identity $\mathbf{Az}-c\mathbf{t} = \mathbf{Ay}-c\cdot\mathbf{s}_2 = \mathbf{r}$, we prove that the quantity compared by the "third rejection test" of patent WO2025261784A1 is exactly $\mathbf{r}_1$, which is already produced during signing, while check 2 passing is what implies $\mathbf{r}_1 = \mathbf{w}_1$. The third check therefore re-verifies a condition that check 2 guarantees, rather than being an independent test. We further show that the six embodiments claimed by the patent reduce to only two independent conditions, both covering only skips of the second check. An experiment with 30,000 fault injections confirms that all six embodiments detect the skip of the $\mathbf{z}$-norm check — the route that directly recovers the signing key $\mathbf{s}_1$ — at a rate of zero. Building on this blind spot, we fully recover $\mathbf{s}_1$ against a third-check-protected implementation (all 1024 coefficients of the four polynomials) under randomized signing, the default form in FIPS 204. We then report a complete evaluation matrix over three fault models and two implementation variants, and show that the third check and sign-then-verify are complementary with neither being sufficient on its own: the latter fails entirely against the second-check skip on the reference implementation (all faulty signatures pass verification), while the cheapest embodiment of the former detects every exploitable signature, whereas against the $\mathbf{z}$-check skip only sign-then-verify is effective. Their union covers every attack path we evaluate, and deploying either one alone leaves a complete attack path open. Overhead measurements show that this embodiment costs 0.04–0.05% of a signing on x86-64 and 0.83% on Cortex-M4, about 460–590 and 48 times less than sign-then-verify on the same platforms (22–24% and 40.0%, respectively). We therefore recommend deploying both, at a combined cost that is almost entirely sign-then-verify's.

Keywords: post-quantum cryptography; ML-DSA; fault attack; rejection sampling; fault countermeasure; lattice-based implementation security

---

1 Introduction

1.1 Research Background

ML-DSA (formerly CRYSTALS-Dilithium [BDK+21]) was published by NIST as the FIPS 204 standard in 2024 [FIPS204] and has been incorporated into the NSA's CNSA 2.0 algorithm suite. Compared with other post-quantum signature schemes, ML-DSA offers a relatively balanced trade-off between computational efficiency and signature size, and is therefore widely expected to be used in most scenarios requiring quantum-safe signatures, including embedded devices, smart cards, and IoT terminals.

In contrast to its relatively mature theoretical security, the security of ML-DSA under **physical attacks** is still being rapidly understood. In recent years, a series of works has shown that lattice signature implementations are susceptible to fault injection attacks, with the attack surface gradually expanding from early differential fault analysis of deterministic signatures to NTT operations, coefficient sampling loops, and finally the rejection sampling stage. Among these, [Polytope] was the first to systematically exploit **rejected signatures**: the adversary uses faults to skip the norm checks in rejection sampling, so that candidate signatures that should have been discarded are output, and each such signature leaks a linear inequality on the private key; after collecting enough inequalities, linear programming recovers the key on an ordinary desktop machine within tens of minutes. That work explicitly points out that what must be protected is the **execution of the checks** itself, not merely the values manipulated in the signature computation.

1.2 Problem

Rejection sampling is a critical component of both the security proof and the correctness of ML-DSA. The signing algorithm contains three groups of checks (see Section 2.1); if any of them is not satisfied, the current candidate signature is discarded and a new one is sampled. Discarded candidate signatures carry information about the private key, and rejection sampling ensures that this information does not appear in the output — this is precisely the security role of the mechanism, and it explains why skipping its checks causes key leakage.

In response to this threat, countermeasure proposals centered on redundant checks have recently emerged. Patent WO2025261784A1 [Patent] (applicant Thales Dis France SAS; inventors Azevedo Oliveira, Calle Viera, Cogliati and Vigilant) proposes adding a "third rejection test" on top of the original checks: before outputting a signature, check whether $\mathbf{w}_1$ equals HighBits($\mathbf{Az} - c\mathbf{t}$); if not, restart signature generation. The patent gives 6 embodiments.

Notably, **the patent's own abstract states the four fault classes it claims to defend against**: the first rejection tests on the second test vector $\mathbf{r}_0$ and **on the first test vector z** (our checks 2 and 1), and the second rejection tests on the fourth vector $\mathbf{h}$ and on the value $c \cdot \mathbf{t}_0$ (the two parts of check 3). In other words, **the countermeasure claims coverage of all three original checks.**

Two further points of context are worth recording. First, three of the inventors — Azevedo Oliveira, Calle Viera and Cogliati — **are also authors of [Polytope]**; the very people who proposed the attack. Second, Section 6.4 of [Polytope] attacks precisely the skip of check 1.

However, **the actual coverage of this countermeasure has never been independently evaluated**. Which faults does it really defend against? What is the relation among the 6 embodiments? What does it cost? These questions have no answer in the public literature.

1.3 Our Work and Contributions

This paper provides a systematic, independent evaluation of this countermeasure and makes the following four contributions:

(1) We prove that the third check merely re-verifies the condition guaranteed by check 2, and that it has a structural blind spot. Using the identity $\mathbf{Az} - c\mathbf{t} = \mathbf{Ay} - c \cdot \mathbf{s}_2 = \mathbf{r}$, we prove that the HighBits($\mathbf{Az} - c\mathbf{t}$) compared by this check is exactly $\mathbf{r}_1$ from the signing process, while check 2 passing implies $\mathbf{r}_1 = \mathbf{w}_1$. The countermeasure therefore decides only a necessary condition guaranteed by check 2, and in essence covers only skips of check 2. Further analysis shows that the 6 embodiments actually provide only 2 independent conditions: {E1, E2, E3, E4} reduce to the same decision, E5 has exactly the same detection count as the original check 2, and E6 is a tautology.

(2) We experimentally verify the blind spot and give an end-to-end break. In an experiment with about 30,000 fault injections, all 6 embodiments detect the skip of check 1 at a rate of zero. Building on this, we fully recover the signing key $\mathbf{s}_1$ against a **third-check-protected implementation** — all 1024 coefficients of the 4 polynomials are recovered correctly. We also characterise the sample complexity of this attack: with an optimising solver objective, recovery in practice requires only about **7,000 exploitable faulty signatures** (our experiment collects 88,261, a comfortable margin). The attack is carried out under randomized signing, the default form in FIPS 204.

(3) We give a complete evaluation matrix covering three attack paths and two implementation variants. The evaluation covers 6 embodiments, sign-then-verify, and double computation — three classes of countermeasures — and distinguishes between the specification implementation and the reference implementation. The results show that the coverage of the two main countermeasures is **complementary and neither is dispensable**: on the reference implementation, sign-then-verify fails entirely against the skip of check 2 (100% of faulty signatures pass verification), whereas E1, the best embodiment of the third check, detects exactly all exploitable signatures; conversely, when check 1 is skipped, E1 detects nothing while verification is effective. Their union covers all three attack paths, while deploying either one alone leaves a complete attack path open.

(4) We give measured overheads and a deployment recommendation. Per-check overhead attribution shows that E1 is the cheapest of all variants (0.04% ~ 0.05% of a signing) and that its detection capability is exactly the same as that of E2, E3, and E4 — that is, if the patent deployed only E1, the cost would drop from 3.41% ~ 3.69% to 0.04% ~ 0.05% with no change in protection. E1's overhead is far below that of sign-then-verify — **about 460 ~ 590 times lower on x86-64 (0.04%**

~ 0.05% versus 22% ~ 24%) and about 48 times lower on Cortex-M4 (measured 0.83% versus 40.0%) — while the latter precisely fails to cover the attack path that E1 covers. We therefore recommend deploying both.

1.4 Organization

Section 2 gives the mathematical background on ML-DSA, the hint mechanism, the fault models, and rejection sampling attacks; Section 3 surveys the two threads of attacks and countermeasures and positions the contributions of this paper; Section 4 defines the threat model and the evaluation scope, and argues for the physical feasibility of the attack types considered; Section 5 gives the reduction analysis of the third check and the evaluation matrix; Section 6 gives the end-to-end break; Section 7 gives measured overheads (on the two platforms x86-64 and Cortex-M4); Section 8 gives the coverage complementarity, deployment recommendations, and limitations; Section 9 concludes.

2 Preliminaries

2.1 ML-DSA Signing and Rejection Sampling

ML-DSA [FIPS204] is based on the Fiat-Shamir with aborts paradigm, with security reduced to the MLWE and MSIS problems over module lattices [BDK+21]. Its key generation produces short vectors $\mathbf{s}_1 \in S_\eta^\ell$ and $\mathbf{s}_2 \in S_\eta^k$ and computes $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. To compress the public key, only the high bits $\mathbf{t}_1$ of $\mathbf{t}$ and the matrix seed ρ are published, i.e. $\mathrm{pk} = (\rho, \mathbf{t}_1)$; the low bits $\mathbf{t}_0$ are kept in the private key along with $\mathbf{s}_1$ and $\mathbf{s}_2$.

The signing process iterates with rejection sampling until it produces a candidate signature satisfying all conditions:

1. Sample the blinding vector $\mathbf{y} \in \tilde{S}_{\gamma_1}^\ell$ and compute $\mathbf{w} = \mathbf{A}\mathbf{y}$;
2. Decompose $\mathbf{w}$ to obtain $(\mathbf{w}_1, \mathbf{w}_0) = \mathrm{Decompose}(\mathbf{w})$, and derive the challenge polynomial c from $\tilde{c} = H(\mu \| \mathbf{w}_1)$;
3. Compute $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$ and $\mathbf{r} = \mathbf{w} - c \cdot \mathbf{s}_2$, and let $(\mathbf{r}_1, \mathbf{r}_0) = \mathrm{Decompose}(\mathbf{r})$;
4. Perform the three groups of checks; if any is not satisfied, discard the current candidate and resample $\mathbf{y}$:

Check 1 (z-norm check): $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ **Check 2 ($\mathbf{r}_0$-norm check)**: $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$ **Check 3 (norm check on $c \cdot \mathbf{t}_0$)**: $\|c \cdot \mathbf{t}_0\|_\infty < \gamma_2$, and the Hamming weight of the hint does not exceed ω

Among the three groups of checks, check 1 ensures that the distribution of $\mathbf{z}$ does not leak $\mathbf{s}_1$; check 2 ensures that $\mathrm{HighBits}(\mathbf{A}\mathbf{y} - c \cdot \mathbf{s}_2) = \mathrm{HighBits}(\mathbf{A}\mathbf{y})$, i.e. that $\mathbf{w}_1$ does not leak $\mathbf{s}_2$, and at the same time enables verification to recover $\mathbf{w}_1$ with the help of the hint. Together they constitute the security role of rejection sampling: **discarded candidate signatures carry information about the private key, and rejection sampling guarantees that this information never appears in the output**.

This role can be stated more concretely. In signing, $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$ is "a linear response to the public challenge c using the private key $\mathbf{s}_1$"; if the norm of $\mathbf{z}$ is too large, part of the information about $\mathbf{s}_1$ can be inferred from $\mathbf{z}$ and c alone. Likewise, if the low bits of $\mathbf{r} = \mathbf{w} - c \cdot \mathbf{s}_2$ are too large, then $\mathbf{w}_1$ carries information about $\mathbf{s}_2$. Precisely for this reason, rejection sampling is not an efficiency expedient but the source of the zero-knowledge property: **it discards all candidate signatures that carry leakage and outputs only those samples that carry no exploitable information**. This also explains why skipping its checks causes key leakage — it removes exactly this filter.

All experiments in this section are based on ML-DSA-44 (i.e. Dilithium-2), whose main parameters are given in Table 1.

Table 1 Main parameters of ML-DSA-44

Parameter	Value	Meaning
q	8,380,417 $= 2^{23} - 2^{13} + 1$	Modulus
n	256	Polynomial degree
k, ℓ	4, 4	Module dimensions
η	2	Bound on the private key coefficients

Parameter	Value	Meaning
τ	39	Number of nonzero coefficients of the challenge polynomial c
β	$78 = \tau\eta$	Relevant norm bound
γ_1	2^{17}	Bound on the blinding vector $\mathbf{y}$
γ_2	$(q-1)/88$	Low-bit decomposition base, $\alpha = 2\gamma_2$
ω	80	Upper bound on the Hamming weight of the hint
d	13	Bit width of Power2Round

2.2 The Hint Mechanism

ML-DSA uses two decomposition functions, Power2Round and Decompose. For even $\alpha \mid q-1$, Decompose writes $\mathbf{r} \in \mathbb{Z}_q$ as $\mathbf{r} = \mathbf{r}_1 \cdot \alpha + \mathbf{r}_0$, where $\mathbf{r}_0 = \mathbf{r} \bmod^{\pm} \alpha \in (-\alpha/2, \alpha/2]$; $\mathbf{r}_1$ is called the high bits and $\mathbf{r}_0$ the low bits. In signing, $\alpha = 2\gamma_2$.

MakeHint and UseHint are used to compute $\text{HighBits}(\mathbf{Az} - c \cdot \mathbf{t}_1 \cdot 2^d)$ without knowing $\mathbf{t}_0$, with the core property

$$\text{UseHint}(\text{MakeHint}(\mathbf{z}, \mathbf{r}, \alpha), \mathbf{r}, \alpha) = \text{HighBits}(\mathbf{r} + \mathbf{z}, \alpha)$$

This identity is the basis of verification correctness and the key tool for the analysis in Section 5.

2.3 Specification Implementation and Reference Implementation

ML-DSA has two functionally equivalent ways of writing the hint computation:

- **Specification formulation** (hereafter spec, see FIPS 204 and the algorithm specification document): the hint is computed from the inputs $(\mathbf{w} - c \cdot \mathbf{s}_2)$ and $(\mathbf{w} - c \cdot \mathbf{s}_2 + c \cdot \mathbf{t}_0)$;
- **Reference implementation formulation** (hereafter ref, see the pq-crystals reference implementation): the hint is computed from the inputs $(\mathbf{w}_1)$ and $(\mathbf{w}_0 - c \cdot \mathbf{s}_2 + c \cdot \mathbf{t}_0)$, saving one Decompose call.

When all checks execute normally, the two are functionally equivalent. However, **when check 2 is skipped they are no longer equivalent**: under the spec formulation a faulty signature fails verification, whereas under the ref formulation a faulty signature still passes verification. In the coverage matrix of Section 8 this difference separates two attack paths, so we evaluate the two implementation variants separately.

2.4 Fault Models

We consider **instruction-skip faults**: the adversary uses clock glitches, voltage glitches, laser or electromagnetic pulses, and similar means to make the processor skip one instruction or one function call. The distinguishing feature of this fault class is that what is skipped is **control flow**, not data or arithmetic operations — a skipped check changes no computed value, so countermeasures against data tampering or arithmetic errors are ineffective against it.

Accordingly, we evaluate the following three fault models:

Model	Meaning
No fault	All checks execute normally (used to test for false positives)
Skip check 1	The branch checking the norm of $\mathbf{z}$ is skipped; checks 2 and 3 execute normally
Skip check 2	The branch checking the norm of $\mathbf{r}_0$ is skipped; checks 1 and 3 execute normally

2.5 Mathematical Basis of Rejection Sampling Attacks

After a check is skipped, the output condition on candidate signatures is relaxed, thereby leaking linear inequalities on the private key.

Proposition 4 (spec variant, attacking $\mathbf{s}_2$) Suppose check 2 is skipped and the signature is output. Since the $\mathbf{w}_1'$ computed during verification may then differ from the $\mathbf{w}_1$ computed during signing, [Polytope] proves that the signature at positions where $\mathbf{w}_1 - \mathbf{w}_1'$ is nonzero yields an inequality on $\mathbf{s}_2$:

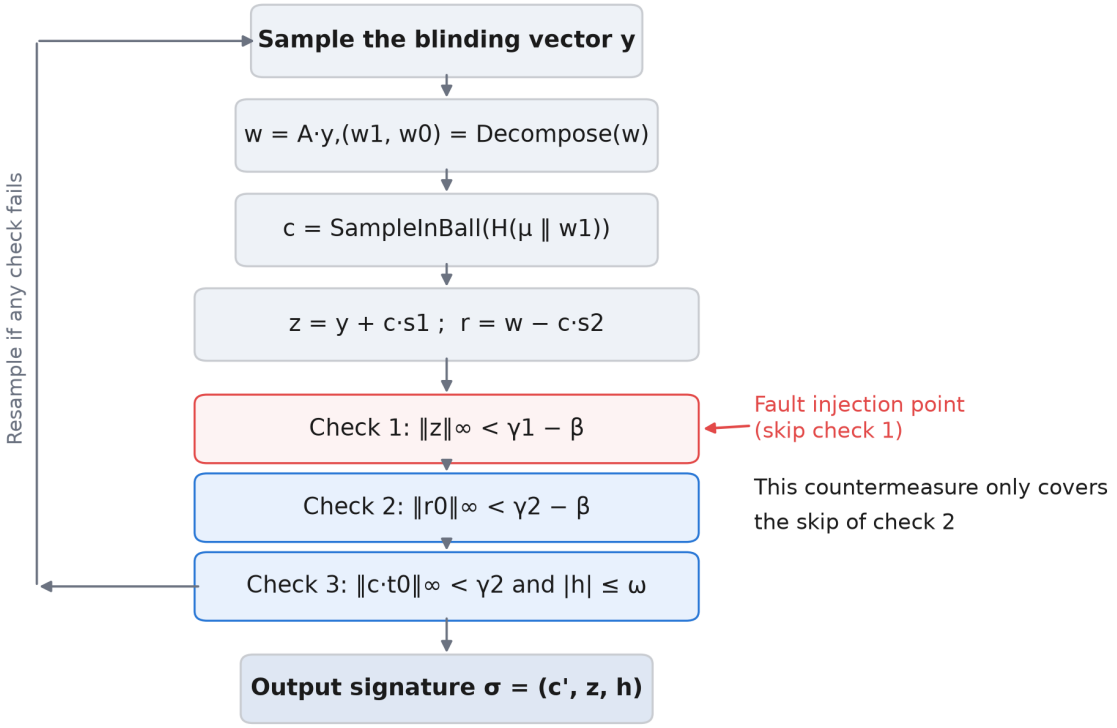


Figure 1 ML-DSA signing flow, the three rejection sampling check points, and fault injection locations

When the difference $\mathbf{w}_1 - \mathbf{w}'_1$ is +1: $(c \cdot \mathbf{s}_2)_i \geq \gamma_2 - \mathbf{r}_{0i} \geq 0$ When the difference $\mathbf{w}_1 - \mathbf{w}'_1$ is -1: $(c \cdot \mathbf{s}_2)_i \leq -\gamma_2 - \mathbf{r}_{0i} \leq 0$

Proposition 6 (attacking $\mathbf{s}_1$) If check 1 is skipped and the output signature satisfies $\|\mathbf{z}\|_\infty \geq \gamma_1$, then the out-of-range coefficients of $\mathbf{z}$ yield inequalities on $\mathbf{s}_1$ (see Section 6.1). This route **does not require knowledge of $\mathbf{t}_0$**, and the recovered $\mathbf{s}_1$ suffices to forge signatures.

2.6 Key Recovery from Inequalities

Each of the above inequalities is a linear constraint on the 256 coefficients of one private key polynomial. Since the private key coefficients are bounded ($\|s\|_\infty \leq \eta$), these constraints together define a bounded convex polytope. Once enough inequalities have accumulated that this polytope contracts to a unique integer point, the private key can be recovered by linear programming. [Polytope] reports a requirement of about ten thousand inequalities per polynomial; our own measurements indicate that the threshold is lower — at about 2×10^4 signatures (about 5.1×10^3 inequalities per polynomial), solving with several different random objectives already yields the **same vertex**, coinciding with the true solution (Section 6.3). This is the solving basis of the attack used in Section 6 of this paper.

3 Related Work

3.1 Fault Attacks

Fault attacks on lattice signature implementations have a fairly long history. Early work [BP18] gave differential fault analysis of deterministic lattice signature schemes; [RJH19] gave instruction-skip attacks on Dilithium on the pqm4 platform.

Around 2023, the attack surface was significantly broadened: [Ravi2023] was the first to analyze the fault sensitivity of the NTT itself, showing that a single fault can zero out all twiddle factors of the NTT; [Ulitzsch2023] proposed loop-abort attacks that zero the coefficients of the blinding vector by aborting the coefficient sampling loop; [ElGhamrawy2023] used a small number of instruction skips to reduce the MLWE problem to a smaller RLWE problem and solved it practically.

From 2024 to 2025, attacks further concentrated on the rejection sampling stage: [Krahmer2024] proposed correction attacks that use verification as a correction oracle and pointed out that sign-then-verify can be bypassed by "ineffective faults" in certain variants; [Polytope] was the first to systematically exploit **rejected signatures**, recovering the key on an ordinary desktop machine by linear programming, and explicitly pointed out that what must be protected is the execution of the checks itself and not merely the values manipulated; [Diverse2025] gave multiple techniques covering skipping, aborting, and zeroing, able to break a class of protected implementations on Cortex-M4 with a high success rate.

On the side-channel side, [Zhou2025] addresses **passive leakage** from the rejection sampling stage: the authors use power leakage of NTT operations to extract rejected challenges and responses and cast key recovery as an integer linear programming problem; that work gives a worst-case complexity analysis and validates it on an ARM Cortex-M4 implementation of Dilithium-2 using **real power traces**, making it the first practical side-channel key recovery attack targeting the rejection sampling process.

The most recent work pushes fault attacks into the **challenge sampling** stage: [Shin2026] points out that the challenge sampling process is itself a practical attack surface, gives a practical differential fault attack on deterministic ML-DSA in which **a single faulty signature suffices to recover a private key usable for forging signatures**, and presents both simulated and real fault injection (clock glitch) experiments along with corresponding countermeasures. That work operates on a **different attack surface** from this paper (challenge sampling vs. rejection sampling checks), so the number of faulty signatures required differs structurally; see the discussion in Section 6.4.

3.2 Fault Countermeasures

Research on fault countermeasures for lattice signatures lags somewhat behind attacks. Generic means include sign-then-verify, double computation, and shuffling, whose effectiveness and limitations are systematically discussed in [Krahmer2024]; dedicated countermeasures for NTT implementations have also been proposed, for example [TECS2024] uses redundant bitslices to improve the resistance of the NTT to electromagnetic faults.

There are also several countermeasures targeting specific attacks: [RJH19] proposes moving the addition of $\mathbf{z}$ into the NTT domain so that fault-affected signatures are discarded by subsequent checks with high probability, but according to later work this modification can introduce a new attack surface; [HP23] proposes lifting operations into an extension ring and verifying consistency, introducing redundancy into the NTT coefficients; its cost depends strongly on architectural parameters such as register width, and its error detection is probabilistic; [Bauer2024] instead protects the NTT by polynomial evaluation and interpolation, which provably detects a single fault in a coefficient, and gives its concrete application to Kyber and Dilithium — on an ARM Cortex-M4, applied to Dilithium, the measured relative overhead is about **72%**; [Mishra2024] proposes building a probabilistic countermeasure framework from rejection sampling, binding the sampling of the blinding vector to the presence of faults so that faulty signatures are not output at the cost of a very large number of retries.

On the engineering side, patent WO2025261784A1 [Patent] proposes the aforementioned "third rejection test", a relatively recent countermeasure proposal centered on redundant checks in this field.

On the masking side, [SUCRE] (TCHES 2026) is a recent work that directly targets the rejection sampling stage. It gives a masked gadget that **applies a random permutation to the coefficients while unmasking**; because the permutation preserves the infinity norm, rejection sampling proceeds as usual without a mask-domain conversion. That work gives a security proof in the probing model and points out that the permutation destroys the correspondence between rejected $\mathbf{z}$ values and the challenge c, thereby defeating existing integer-programming-based side-channel attacks. Note that what SUCRE addresses is the **passive leakage** threat — rejected values observed via probing or side channels — whereas what this paper considers is **active faults** — the execution of checks being skipped. The two belong to different threat categories, so SUCRE is outside the evaluation scope of Section 4.3 of this paper.

Furthermore, the two also belong to different categories in their countermeasure approach: **SUCRE destroys the usability of the leakage through a random permutation and is a randomization-based countermeasure**, whereas **the third check evaluated in this paper attempts to detect faults through redundant comparison and is a redundancy-based countermeasure**. The two are not mutually exclusive and can be combined in deployment.

Table 2 summarizes related work and the attack of this paper along four dimensions — attack surface, fault model, sample size, and hardware validation — in order to position the contributions and the gaps

of this paper.

Table 2 Multi-dimensional comparison of related work and the attack of this paper

Work	Attack surface	Fault model	Samples required	Recovery compute time	Real hardware validation
[Polytope] §6.1	Rejection sampling check 2	Check skip	1.25×10^6 signatures	Median 180 s	× Simulation only
[Polytope] §6.4	Rejection sampling check 1	Check skip	2×10^6 signatures	51.9 s	× Simulation only
[Krahmer2024]	Addition of $\mathbf{z}$	Skip addition	About 10^3 signatures	Not reported	✓ Clock glitch
[Zhou2025]	Rejection sampling (side channel)	No fault (passive observation)	Not reported	Seconds to minutes	✓ Real power traces
[Shin2026]	Challenge sampling	Fault injection	**1 signature** (deterministic signing configuration)	Not reported	✓ Clock glitch
[Ulitzsch2023]	Sampling/unpacking loop of $\mathbf{y}$	Loop abort (coefficients zeroed)	**5 (best) ~ 3,466 (weakest)**	Minutes	✓ Clock glitch
This paper §6	**Rejection sampling check 1 (protected by the third check; randomized signing configuration)**	Check skip	2.4×10^5 signatures (about 7×10^3 exploitable; Section 6.3)	Solve in seconds to minutes	× Simulation; countermeasure overhead measured on Cortex-M4

The definitions of "samples required" and "recovery compute time" in the table follow the respective original papers: for example, the time reported by [Polytope] is the solving time after inequality extraction is complete; under the same definition, the solving time in this paper is a few seconds (the corresponding measurement for the $\mathbf{s}_2$ attack in Section 6.3 is 7 s), while 30 min is the full pipeline (including signature generation). "Not reported" means that the paper gives no directly comparable figure, not that no comparison is possible.

Note: the sample-size range of [Ulitzsch2023] is taken from its Table 1 under different fault-strength configurations, 5 being the best case and 3,466 the weakest. The 2.4×10^5 in the row for this paper follows from the measured yield: recovering the key needs about 7×10^3 exploitable signatures, and exploitable signatures make up about 2.94% of all signatures, so $7 \times 10^3 / 0.0294 \approx 2.4 \times 10^5$ (a conservative estimate at the **current** yield; a better harvesting strategy would lower it further).

As Table 2 shows, the number of samples required spans five orders of magnitude — from the single signature of [Shin2026] to the millions of [Polytope]. **This paper falls in roughly the same range as [Ulitzsch2023]** (about 7×10^3 versus 5 ~ 3,466), even though the two attack surfaces are entirely different: each faulty signature here leaks only **a single inequality**, whereas [Ulitzsch2023] leaks **an entire linear system**. **The difference stems from the information structure of the attack surface, not from the merits of the methods**, and Sections 6.3 and 6.4 discuss this specifically.

The "attack surface" column of the table also determines the boundary of a countermeasure's effectiveness: countermeasures that **protect arithmetic** (such as NTT-domain addition or CRT checks) can only address attacks on operations, whereas countermeasures that **protect control flow** (such as sign-then-verify or redundant checks) are the ones that can address attacks on the execution of checks, and vice versa (see Section 4.3). This paper and [Polytope] belong to the latter.

3.3 Locating the Gap

In summary, existing work has two shortcomings. **First, the granularity of evaluation is single**: most existing evaluations examine the protection effect of "a single countermeasure against a single attack", and no work cross-evaluates multiple rejection-sampling-related countermeasures against multiple attack paths within the same experimental framework, so there is a lack of comparable data on "what each countermeasure actually covers". **Second, recent proposals lack independent evaluation**: the third check proposed in patent [Patent] has no public independent security or overhead analysis, and the relations among its 6 embodiments have not been discussed.

This paper addresses both points: Section 5 gives the reduction analysis and cross-evaluation matrix of this countermeasure, Section 6 gives the end-to-end break, Section 7 gives measured overheads, and Section 8 gives the coverage complementarity and deployment recommendations.

4 Threat Model and Scope

4.1 Adversary Capabilities

We assume the adversary has the following capabilities:

- the ability to inject a **single instruction-skip fault** into the signing device, so that a specified instruction or function call is not executed;
- the ability to obtain all signatures output by the device, including signatures that should have been discarded under the influence of the fault;
- knowledge of the public parameters, the public key, and the messages used.

Regarding $\mathbf{t}_0$: following the convention of the literature, we treat it as a quantity that can be publicly recovered. There are two reasons: first, the security proof of ML-DSA does not rely on the secrecy of $\mathbf{t}_0$; second, prior work [Uncompress] shows that $\mathbf{t}_0$ can be recovered from a reasonable number of signatures. We therefore do not include the secrecy of $\mathbf{t}_0$ among the protection goals.

4.2 Deployment Assumption

All experiments in this paper are carried out under **randomized (hedged) signing**, the default deployment form of FIPS 204. This deserves emphasis: randomized signing randomizes the blinding-vector nonce together with the message on every signing, making the attack harder than against deterministic signing. The attack conclusions of this paper therefore do not depend on the convenient assumption of deterministic signing and hold directly for real deployments.

4.3 Scope: Countermeasures Not Evaluated

The object of evaluation in this paper is countermeasures that **protect the execution of the rejection sampling checks**. One class of countermeasures, although also often listed as "lattice signature fault protection", targets a different fault category and is outside the scope of this paper:

- **[RJH19]'s NTT-domain addition**: its goal is to have "faults on the addition operation itself" discarded by subsequent checks with high probability. The fault in this paper skips a check rather than the addition, and the addition itself still executes correctly, so this countermeasure has no effect on the attacks in this paper.
- **[HP23]'s CRT lifting check**: its goal is to detect **errors in computation results**. Skipping a check changes no computation result, so this check cannot be triggered.
- **[Bauer2024]'s NTT evaluation/interpolation check**: likewise, it protects the NTT's computation results rather than the execution of checks, so a skipped check cannot trigger it either.

In other words, these three countermeasures protect arithmetic, whereas the target of the attacks in this paper is control flow. Our explicit exclusion of them in Section 4.3 is on the grounds of fault-category definition, not out of deliberate omission.

4.4 On Simulated Faults

We simulate instruction skips by modifying code paths and do not use real fault injection equipment. This follows the same paradigm as the simulation parts of [Polytope] and [Krahmer2024]. There are two reasons

for this choice: first, the purpose of this paper is to evaluate the **coverage of the countermeasure** — a question at the level of decision logic, for which simulated faults suffice to give a definite answer; second, real injection experiments require clock glitching or electromagnetic equipment and constitute an independent experimental dimension whose results are affected by device and process variation, making them unsuitable as the sole basis for a coverage conclusion.

4.5 Argument for Attack Feasibility

Since the attack evaluation in this paper is based on simulated faults, a natural question is: are such attacks realizable on real hardware? This section is devoted to answering that question, beginning by **defining the goal of the argument**.

The goal of the argument is that "the attack type is physically possible", not that "the simulation results of this paper can be reproduced directly on any platform". The two propositions belong to different levels. The former is an existence question about the attack category; the latter is a question of specific parameters on a specific platform — the latter depends on numerous factors such as chip packaging, the power delivery network, clock tree design, the temporal resolution and injection position of the glitching equipment, and varies with device batch and environmental conditions. Coverage evaluation of a countermeasure need only rest on the former, because "whether a countermeasure covers a certain fault class" and "with what probability that fault can be successfully injected" are two independent questions.

Evidence from the literature. Existing real fault injection work has already shown that the fault type required here is achievable on commercial microcontrollers:

- [Krahmer2024] achieved an addition skip in Dilithium on a real ARM microcontroller using **clock glitches** and completed key recovery with it; that work reports both simulated and real fault results, which corroborate each other.
- [Diverse2025] realized three classes of faults on the ARM Cortex-M4 platform — instruction skip, loop abort, and zeroing — reporting success rates between 29%~63%, and broke both the reference implementation and an implementation claiming protection against skip-addition attacks.
- [Shin2026] performed **real clock glitch experiments** on ML-DSA and reports being able to identify "whether the fault was injected as intended" with **100% accuracy**. This result shows that **controllable, repeatable** fault injection into the ML-DSA signing flow on commercial microcontrollers is a capability that already exists today.
- [Polytope] uses the same instruction-skip attack model, and its feasibility argument cites existing experimental work with clock glitches, voltage glitches, lasers, and electromagnetic pulses.

The fault required here is within their capabilities. The attack in this paper only needs to **skip one branch instruction** (i.e. the execution of a rejection sampling check); it does not require precise control over the clock cycle in which the fault occurs, nor does it depend on a specific data pattern or cache state. What existing work has achieved is a finer fault than this (a skip accurate to a specific operation). Therefore, on any injection platform capable of a single instruction skip, the fault model of this paper holds.

On boundaries. We do not claim that this attack can be realized with the same probability on **any** platform: the difficulty of fault injection does indeed differ across chips, and certain packaging or power delivery designs significantly raise the injection threshold. But the conclusion of this paper — "the third check has zero coverage of the skip of check 1" — depends only on the fact of whether the check is executed, and is independent of the way the fault is injected, the injection difficulty, and the success probability. In other words, **the coverage conclusion is platform-independent**; only the injection cost is platform-dependent, and injection cost is part of the attack cost and does not affect the judgment of whether the countermeasure is effective.

A note on attack cost. The attack of Section 6 of this paper requires collecting about 8.8×10^4 exploitable signatures (2.94% of all signatures), which demands that the adversary have **reliable and repeatable** injection capability rather than sporadic faults. This is an additional requirement on the adversary's capability, but it does not constitute a limitation on the conclusions of this paper: the 100% fault-identification rate reported by [Shin2026] shows that "reliable and repeatable" is exactly the level that current injection techniques have already reached. The difference in the number of samples required stems from the difference in attack surface and has been discussed in Section 6.4. Note the conventions differ: our 7×10^3 counts **exploitable signatures** (the corresponding number of injections is about

2.4×10^5), whereas [Shin2026] and [Krahmer2024] report 1 to 10^3 signatures.

In summary, evaluating the coverage of a countermeasure by simulated faults is appropriate; injection validation on real platforms is an independent experimental dimension and is left as future work.

5 The Structural Blind Spot of the Third Check

5.1 The Third Check and Its Key Identity

Patent WO2025261784A1 of Thales proposes adding a "third rejection test" on top of the two original rejection sampling checks of ML-DSA. Its Embodiment 1 states: before outputting a signature, check whether the vector $\mathbf{w}_1$ computed during signature generation equals HighBits($\mathbf{Az} - c\mathbf{t}$); if not, discard that candidate signature and restart signature generation. The intent of the patent is that even if the first or second original check is skipped by a fault, an invalid candidate signature will still be rejected at the third check.

The key to understanding what this check actually does lies in the following identity. From the ML-DSA key relation $\mathbf{t} = \mathbf{As}_1 + \mathbf{s}_2$ and $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$ in signing, we have

$$\mathbf{Az} - c\mathbf{t} = \mathbf{A}(\mathbf{y} + c \cdot \mathbf{s}_1) - c(\mathbf{As}_1 + \mathbf{s}_2) = \mathbf{Ay} + c \cdot \mathbf{As}_1 - c \cdot \mathbf{As}_1 - c \cdot \mathbf{s}_2 = \mathbf{Ay} - c \cdot \mathbf{s}_2$$

This identity uses only the two defining relations ($\mathbf{t} = \mathbf{As}_1 + \mathbf{s}_2$ and $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$) and ring algebra; it **holds unconditionally, independently of whether the rejection sampling checks are executed** — skipping a check only affects whether a candidate signature is output and does not change the above algebraic relation. It is the corollary about check 2 in the next section, not this identity, that requires an additional condition.

It should be noted that **this identity is the basis of the ML-DSA verification procedure** (the verifier computes exactly $\mathbf{Az} - c\mathbf{t}$ and recovers $\mathbf{w}_1$ with the help of the hint; see [BDK+21] Section 1 and [FIPS204] Section 3.6); **it is not the contribution of this paper**, which lies in using it to determine the actual semantic role of the patent's third check.

In the signing process, let $(\mathbf{w}_1, \mathbf{w}_0) = \text{Decompose}(\mathbf{w})$ with $\mathbf{w} = \mathbf{Ay}$, and let $(\mathbf{r}_1, \mathbf{r}_0) = \text{Decompose}(\mathbf{r})$ with $\mathbf{r} = \mathbf{w} - c \cdot \mathbf{s}_2$. Then

$$\text{HighBits}(\mathbf{Az} - c\mathbf{t}) = \text{HighBits}(\mathbf{Ay} - c \cdot \mathbf{s}_2) = \mathbf{r}_1$$

that is, the quantity compared by the third check is exactly $\mathbf{r}_1$, which has already been computed during signing; the check is equivalent to deciding whether $\mathbf{r}_1$ equals $\mathbf{w}_1$.

5.2 Reduction: The Third Check Re-verifies Check 2

Consider the semantics of the original check 2. Check 2 decides $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$, and its role is stated explicitly in a comment in the reference implementation as "check that subtracting cs2 does not change high bits of $\mathbf{w}$". This statement can be made precise as the following lemma.

Lemma 1 Let $\mathbf{w}_0 = \text{LowBits}(\mathbf{w}) \in (-\gamma_2, \gamma_2]$, $\alpha = 2\gamma_2$, and assume $\|c \cdot \mathbf{s}_2\|_\infty \le \beta$. Let $\mathbf{r} = \mathbf{w} - c \cdot \mathbf{s}_2$ and $(\mathbf{r}_1, \mathbf{r}_0) = \text{Decompose}(\mathbf{r})$. If $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$, then $\mathbf{r}_1 = \mathbf{w}_1$.

Proof Two cases: $\mathbf{w}_0 - c \cdot \mathbf{s}_2$ either does not leave the interval $(-\gamma_2, \gamma_2]$, or it does (overflow and underflow are symmetric; we treat overflow below). In the first case the low-bit decompositions of $\mathbf{r}$ and $\mathbf{w}$ agree, so their high bits are equal, i.e. $\mathbf{r}_1 = \mathbf{w}_1$, and the claim holds.

If an overflow occurs, i.e. $\mathbf{w}_0 - c \cdot \mathbf{s}_2 > \gamma_2$, then $\mathbf{r}_0 = \mathbf{w}_0 - c \cdot \mathbf{s}_2 - \alpha$. From $\mathbf{w}_0 \le \gamma_2$ and $c \cdot \mathbf{s}_2 \ge -\beta$ we get

$$\mathbf{r}_0 \le (\gamma_2 + \beta) - 2\gamma_2 = -\gamma_2 + \beta,$$

hence $|\mathbf{r}_0| \ge \gamma_2 - \beta$, contradicting the assumption $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$. The underflow case ($\mathbf{w}_0 - c \cdot \mathbf{s}_2 \le -\gamma_2$) is symmetric and likewise yields $|\mathbf{r}_0| \ge \gamma_2 - \beta$. Both out-of-range cases contradict the assumption, so only the first case can occur and the lemma holds. ■

Remark 1 The converse of Lemma 1 **does not hold**. When $\mathbf{w}_0 - c \cdot \mathbf{s}_2$ has not left the interval but has fallen into the β-neighborhood of the boundary $\pm\gamma_2$, we still have $\|\mathbf{r}_0\|_\infty \ge \gamma_2 - \beta$; check 2 then rejects, while $\mathbf{r}_1 = \mathbf{w}_1$ still holds. In other words, $\mathbf{r}_1 = \mathbf{w}_1$ is a **necessary but not sufficient condition** for check 2 to pass; check 2 is strictly stronger than this equality.

This one-directionality is crucial to the conclusions of this section and can be directly confirmed by

experiment: when check 2 is skipped, about **56.98%** of signatures are rejected by check 2, while only **3.59%** of them satisfy $\mathbf{r}_1 \neq \mathbf{w}_1$ — the gap comes precisely from the situation described in Remark 1 (see Section 5.5 for details).

What the third check decides is exactly this necessary condition $\mathbf{r}_1 = \mathbf{w}_1$. This is both why it is effective when check 2 is skipped and the source of its blind spot.

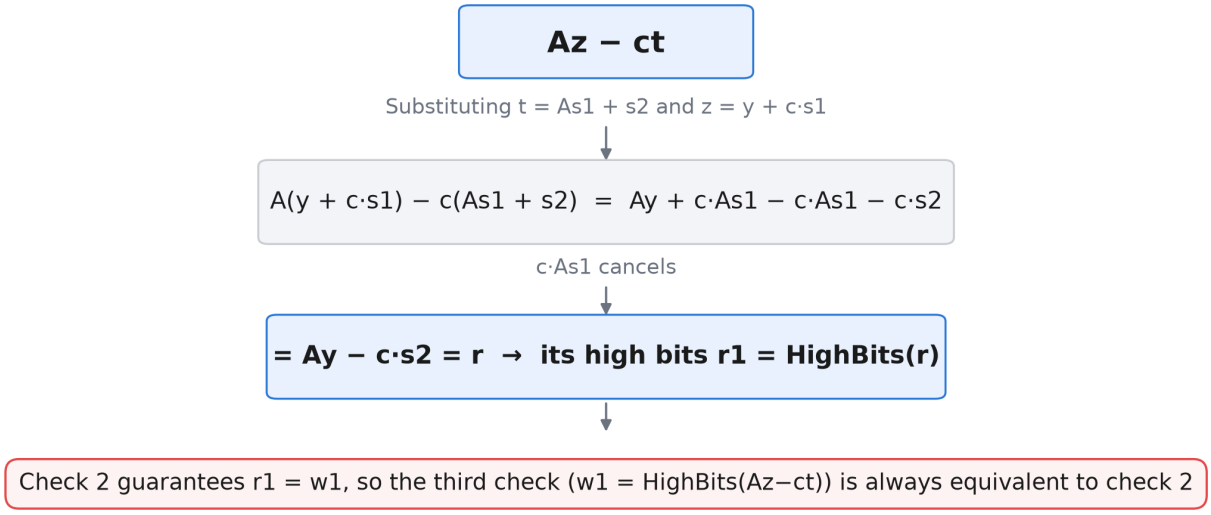


Figure 2 Derivation of the relation between the third check and check 2

Two corollaries follow directly:

- **Corollary 1**: When the adversary skips check 2, $\mathbf{r}_1 = \mathbf{w}_1$ is no longer guaranteed. For those candidate signatures with $\mathbf{r}_1 \neq \mathbf{w}_1$, the third check rejects them. This makes the patent's countermeasure partially effective on the "skip check 2" attack path.
- **Corollary 2**: When the adversary skips check 1 (the $\mathbf{z}$-norm check), check 2 still executes as usual, so $\mathbf{r}_1 = \mathbf{w}_1$ necessarily holds and the third check always passes — **zero detection** of that fault.

Corollary 2 holds because skipping check 1 does not change any quantity involved in the third check's decision. Check 1 only decides the norm of $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$ itself; $\mathbf{w}$, c, $\mathbf{s}_2$ and the $\mathbf{r}$ and $\mathbf{w}_1$ derived from them are all unaffected by that fault. The two quantities compared by the third check therefore remain equal.

This has a direct security consequence: skipping check 1 is precisely the attack path that can **directly recover the signing key** $\mathbf{s}_1$. By Proposition 6 of [Polytope], when a signature is output with $\|\mathbf{z}\|_\infty \geq \gamma_1$, each out-of-range coefficient of $\mathbf{z}$ gives a linear inequality on $\mathbf{s}_1$:

$$(c \cdot \mathbf{s}_1)_i \geq \mathbf{z}_i - \gamma_1 \geq 0 \quad \text{when } \mathbf{z}_i \geq \gamma_1, \qquad (c \cdot \mathbf{s}_1)_i \leq \mathbf{z}_i + \gamma_1 \leq 0 \quad \text{when } \mathbf{z}_i \leq -\gamma_1$$

This attack does not need to know $\mathbf{t}_0$, and the recovered $\mathbf{s}_1$ suffices to forge signatures. The patent's third check provides no protection whatsoever against this route.

5.3 Relations Among the Six Embodiments

The patent gives 6 embodiments. We analyze what each variant actually computes (writing $\mathbf{r} = \mathbf{w} - c \cdot \mathbf{s}_2$ and $(\mathbf{r}_1, \mathbf{r}_0) = \text{Decompose}(\mathbf{r})$):

Variant	Claim	Patent formulation	Condition actually decided
E1	2	Compare $\mathbf{w}_1$ with HighBits($\mathbf{Az} - c\mathbf{t}$)	$\mathbf{w}_1 = \mathbf{r}_1$
E2	3	Compare $\mathbf{w}_1$ using UseHint($\mathbf{h}, \mathbf{Az} - c \cdot \mathbf{t}_1 \cdot 2^d$)	$\mathbf{w}_1 = \mathbf{r}_1$ (when check 2 passes)

Variant	Claim	Patent formulation	Condition actually decided
E3	4	Compare $\mathbf{w}_1$ with HighBits($\mathbf{Ay} - c \cdot \mathbf{s}_2$)	$\mathbf{w}_1 = \mathbf{r}_1$ (identical to the definition of $\mathbf{r}_1$)
E4	5	Decide MakeHint($c \cdot \mathbf{s}_2, \mathbf{w} - c \cdot \mathbf{s}_2$) $= 0$	$\mathbf{w}_1 = \mathbf{r}_1$ (the hint is zero if and only if the two coincide)
E5	6	Decide $\|\mathbf{w}_0 - c \cdot \mathbf{s}_2\|_\infty < \gamma_2 - \beta$	The same condition as check 2 (measured detection counts agree)
E6	7	Recompute an additional hint vector $\mathbf{h}'$ as MakeHint($-c \cdot \mathbf{t}_0, \mathbf{w}_0 - c \cdot \mathbf{s}_2 + c \cdot \mathbf{t}_0$) and check $\mathbf{h}' = \mathbf{h}$	Tautology

For E2: from $\mathbf{t} = \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$ we have $\mathbf{Az} - c \cdot \mathbf{t}_1 \cdot 2^d = (\mathbf{Ay} - c \cdot \mathbf{s}_2) + c \cdot \mathbf{t}_0$, and the verification algorithm itself uses this expression together with the hint to recover $\mathbf{w}_1$, so E2 is equivalent to E1 when check 2 passes.

For E4: this hint records the difference between HighBits($\mathbf{w} - c \cdot \mathbf{s}_2$) and HighBits($\mathbf{w}$), i.e. the difference between $\mathbf{r}_1$ and $\mathbf{w}_1$; it is zero if and only if $\mathbf{r}_1 = \mathbf{w}_1$ (i.e. the two are equal).

For E5: $\mathbf{r}_0 = \text{LowBits}(\mathbf{w} - c \cdot \mathbf{s}_2)$, while $\mathbf{w}_0 = \text{LowBits}(\mathbf{w})$. Within the range of values decided by check 2, $\|\mathbf{w}_0 - c \cdot \mathbf{s}_2\|_\infty$ and $\|\mathbf{r}_0\|_\infty$ take the same value, so E5 is a reimplementation of check 2 rather than a new check.

For E6: claim 7 recomputes an additional hint vector $\mathbf{h}'$ as MakeHint($-c \cdot \mathbf{t}_0, \mathbf{w}_0 - c \cdot \mathbf{s}_2 + c \cdot \mathbf{t}_0$) and checks $\mathbf{h}' = \mathbf{h}$ (the patent writes $\mathbf{w}_0$, the low part of $\mathbf{w}$, as the second argument). These are the same quantities from which $\mathbf{h}$ is computed during signature generation, so the decision is always true and never triggers.

Conclusion: The 6 embodiments claimed by the patent actually provide only 2 mutually independent decision conditions — {E1, E2, E3, E4} reduce to the same condition $\mathbf{r}_1 = \mathbf{w}_1$, and {E5} is identical to the original check 2; E6 does not constitute a valid check. Moreover, both of these conditions **target only the skip of check 2**, and neither covers the skip of check 1.

The two conditions differ in strength, and this difference is of practical significance for assessing their protective capability. By Lemma 1 and Remark 1, the condition decided by E5 (i.e. check 2) is strictly stronger than E1: the former rejects all candidates whose low bits leave the interval, **as well as** those whose low bits, while not leaving the interval, fall into the boundary neighborhood; the latter rejects only the subset in which wraparound actually occurs. E1 is therefore a more permissive check than check 2. This yields two corollaries: first, as protection against "skipping check 1" the two are equally ineffective; second, as protection against "skipping check 2", the strength of E1 alone suffices to stop all exploitable signatures (see Section 5.5), while the part that E5 rejects in addition does not, by Proposition 4 and Proposition 5, carry exploitable information. This difference in strength does not change the count of "two conditions", but it shows that the patent's 6 variants are highly redundant not only in number but also in protection target.

5.4 Experimental Setup

To verify the above analysis, we built a probing signing function on the Dilithium-2 (ML-DSA-44) reference implementation. This function:

- supports the three fault models: no fault, skip check 1 ($\mathbf{z}$-norm check), and skip check 2 ($\mathbf{r}_0$-norm check);
- after each signing computation completes, evaluates the decision conditions of the 6 embodiments one by one and records the results (**recording only, without rejecting on that basis**), in order to measure whether each variant "would reject this signature";
- also records the decision result of the original check 2, as a reference for E5;
- records the coefficients of $\mathbf{z}$ satisfying $|\mathbf{z}_i| \geq \gamma_1$ (the raw material for the Proposition 6 attack).

The experiments use the first key in the KAT key file, generating 30,000 signatures under each fault model. All experiments are carried out under **randomized (hedged) signing**, the default deployment form of FIPS 204 — which means the attack conclusions do not depend on the convenient assumption of deterministic signing.

5.5 Experimental Results

Table 3 gives the detection rates of the 6 embodiments under the three fault models (detection = the variant's decision condition is not satisfied and it would reject that signature).

Table 3 Detection rate of each embodiment of the third check (30,000 signatures per cell)

Embodiment	No fault	Skip check 1	Skip check 2
E1 $\mathbf{w}_1 = \text{HighBits}(\mathbf{Az} - c\mathbf{t})$	0	**0**	3.59%
E2 UseHint comparison	0	**0**	3.59%
E3 $\mathbf{w}_1 = \text{HighBits}(\mathbf{Ay} - c \cdot \mathbf{s}_2)$	0	**0**	3.59%
E4 MakeHint = 0	0	**0**	3.59%
E5 $\|\mathbf{w}_0 - c \cdot \mathbf{s}_2\|_\infty < \gamma_2 - \beta$	0	**0**	56.98%
E6 recomputed-hint comparison	0	0	0
[Reference] Original check 2	0	0	56.98%

Three observations follow from Table 3.

First, there are no false positives. With no fault, the detection count of every variant is 0, showing that these checks do not interfere with signature generation on the normal execution path.

Second, the blind spot is comprehensive. Under the 30,000 injections that skip check 1, the detection count of all 6 embodiments is **0**. Of these, 848 signatures (2.83% of the 30,000 samples in this experiment) contain coefficients satisfying $\|\mathbf{z}\|_\infty \geq \gamma_1$, i.e. are directly exploitable by the Proposition 6 attack; these exploitable signatures **all passed the third check**. This proportion corroborates the 2.94% measured in Section 6.3 over 3×10^6 samples (the two differ by 0.11 percentage points, within sampling error). This shows that the blind spot is not an implementation defect of some variant but a **structural** property of this countermeasure.

Third, the reduction relations are confirmed experimentally. When check 2 is skipped, the detection counts of E1, E2, E3, and E4 are **exactly identical** (all 1076/30000), while the counts of E5 and the original check 2 are **exactly identical** (both 17093/30000). This is consistent with the analysis of Section 5.3: E1–E4 are different algorithmic implementations of the same condition, and E5 is a reimplementation of check 2.

It should be noted that the detection rate of E5 (56.98%) being much higher than that of E1 (3.59%) is not a contradiction. By Lemma 1 and Remark 1, the signatures rejected by check 2 fall into two classes: those whose low bits have wrapped around (where $\mathbf{r}_1 \neq \mathbf{w}_1$), and those whose low bits have not wrapped around but fall into the boundary neighborhood (where $\mathbf{r}_1 = \mathbf{w}_1$ still holds). E1 detects only the former class. The difference between the two (53.39 percentage points) is the proportion accounted for by the second class.

For the adversary, only the first class is useful — by Proposition 4 and Proposition 5 of [Polytope], only signatures with $\mathbf{r}_1 \neq \mathbf{w}_1$ give inequalities on $\mathbf{s}_2$. Hence **E1's 3.59% corresponds exactly to the set of exploitable signatures**, while the part that E5 detects in addition contributes nothing additional to protection (see the overhead comparison in Table 5).

As an independent cross-check, we repeated the skip-check-2 experiment on the reference implementation variant: E1 detected 1,078 out of 30,000 injections, almost the same as the 1,076 on the specification implementation variant. This is consistent with the analysis that "the condition $\mathbf{r}_1 = \mathbf{w}_1$ decided by E1 is independent of the specific implementation"; the difference between the two variants in verification behavior (Section 8) does not affect the decision of this condition. When check 1 is skipped, E1's detection count on the reference implementation variant is likewise zero.

5.6 Discussion

The above results describe an easily overlooked fact: **the effectiveness of "recompute and compare" countermeasures depends entirely on whether the quantity they compare is affected by the target fault**. The third check compares $\mathbf{r}_1$ with $\mathbf{w}_1$; both are determined by $\mathbf{w}$, c, and $\mathbf{s}_2$ and are independent of the norm of $\mathbf{z}$. Therefore any fault that affects only $\mathbf{z}$ and not $\mathbf{w}$, c, $\mathbf{s}_2$ will not be noticed by this check. Skipping check 1 is exactly such a fault.

This also explains why the countermeasure is effective on check 2 but ineffective on check 1: **the condition it decides is exactly the necessary condition guaranteed by check 2**, so it is merely a re-verification of check 2 rather than protection of the rejection sampling mechanism as a whole. Under our experimental conditions, the statement in the patent text about "protecting signature generation against fault attacks" should be understood as "protecting against the skip of check 2".

A generalizable design principle. The above situation is not unique to this patent but can serve as a general lesson for redundancy-based protection: when adding redundant protection to the rejection sampling mechanism, the redundant check must be coupled to the **object** constrained by the check being protected, and must not merely repeat a conclusion that can be derived from that check. In ML-DSA, the object constrained by check 1 is $\mathbf{z}$, the object constrained by check 2 is $\mathbf{r}$, and the object constrained by check 3 is $c \cdot \mathbf{t}_0$ together with the Hamming weight of the hint; the three cannot be derived from one another. Hence any redundant check whose object is $\mathbf{r}_1 = \mathbf{w}_1$, whatever its implementation form, covers only the single path of check 2. To protect all three checks at the same time, the redundancy mechanism must provide an independent, observable consequence for **each** skipped check.

On check 3. Our experiments focus on the two paths of check 1 and check 2, and do not evaluate attacks that skip check 3. By the above principle, however, it can be expected that the third check, whose object of decision is $\mathbf{r}_1 = \mathbf{w}_1$, will likewise have no effect on the skip of check 3, since it involves neither the norm of $c \cdot \mathbf{t}_0$ nor the Hamming weight of the hint. Whether that route can be exploited up to key recovery depends on whether the signatures output after skipping check 3 carry exploitable information about the private key; we leave this as future work.

In the next section we show that this blind spot can be exploited in practice: on an implementation protected by the third check, the adversary can still fully recover the signing key $\mathbf{s}_1$.

6 End-to-End Break

Section 5 showed that the third check detects the fault that skips check 1 at a rate of zero. This section further shows that the blind spot can be exploited in practice up to key recovery, and is not merely a gap in decision logic.

6.1 Attack Construction

The attack follows the Proposition 6 route of [Polytope]. Suppose the adversary uses fault injection to make the signing implementation skip check 1; the implementation then outputs candidate signatures $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ that should have been discarded, where $\mathbf{z}$ may contain coefficients satisfying $|\mathbf{z}_i| \geq \gamma_1$. For each such coefficient we have:

$$(c \cdot \mathbf{s}_1)_i \geq \mathbf{z}_i - \gamma_1 \geq 0 \quad \text{when } \mathbf{z}_i \geq \gamma_1, \qquad (c \cdot \mathbf{s}_1)_i \leq \mathbf{z}_i + \gamma_1 \leq 0 \quad \text{when } \mathbf{z}_i \leq -\gamma_1$$

The derivation is direct: $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$ and $|\mathbf{y}_i| \leq \gamma_1$, so $(c \cdot \mathbf{s}_1)_i = \mathbf{z}_i - \mathbf{y}_i \geq \mathbf{z}_i - \gamma_1$.

Expanding $(c \cdot \mathbf{s}_1)_i$ as a linear combination of the coefficients of $\mathbf{s}_1$ via ring multiplication, each such inequality becomes a linear constraint on the 256 coefficients of one polynomial of $\mathbf{s}_1$. The challenge polynomial c can be recomputed from the public signature (in ML-DSA, $\tilde{c}$ is derived from a hash of the message and $\mathbf{w}_1$, and c is generated from $\tilde{c}$ via SampleInBall), so the adversary can construct these constraints without any secret. When enough inequalities have been collected that the convex polytope containing $\mathbf{s}_1$ has dimension 0, linear programming yields $\mathbf{s}_1$ — the coordinate range of the polytope is $[-\eta, \eta]^{256}$, and $\eta = 2$ in Dilithium-2.

6.2 On the Compression of z in Signatures

We must explain how the adversary obtains $\mathbf{z}$. The reference implementation packs $\mathbf{z}$ in a signature using 18-bit compression (because $\gamma_1 = 2^{17}$), storing each coefficient in 18 bits in the form $\gamma_1 - \mathbf{z}_i$. For normal coefficients satisfying $|\mathbf{z}_i| < \gamma_1$ this packing is lossless; but when check 1 is skipped and $\mathbf{z}_i$ leaves $(-\gamma_1, \gamma_1]$, $\gamma_1 - \mathbf{z}_i$ becomes negative and wraps around modulo $2^{18} = 2\gamma_1$.

Our experiments (consistent with Section 6.4 of [Polytope]) are carried out at the **specification level**, i.e. assuming the adversary can obtain uncompressed $\mathbf{z}$. It should be noted that porting the attack in full to the compressed-packing reference implementation also requires handling the ambiguity below.

Let the observed value obtained by unpacking be $\mathbf{z}'_i = \gamma_1 - (\text{packed value} \bmod 2\gamma_1)$. A direct check shows that this map sends two values differing by $2\gamma_1$ to the same observed value: $\mathbf{z}$ and $\mathbf{z} \pm 2\gamma_1$ are indistinguishable. This raises a practical problem.

In the adversary's data (output signatures with check 1 skipped), the coefficients fall into two classes: those that are **not wrapped**, whose true value $\mathbf{z}_i$ may lie anywhere in $(-\gamma_1, \gamma_1)$, including the two segments $[\gamma_1 - \beta, \gamma_1)$ and $(-\gamma_1, -\gamma_1 + \beta]$ — these should have been rejected by check 1 but **carry no Proposition 6 inequality**; and those that are **wrapped**, whose true value satisfies $|\mathbf{z}_i| \geq \gamma_1$ and which **carry an inequality**. The problem is that after unpacking, the two classes fall into **exactly the same observed interval**:

True value $\mathbf{z}_i$	Class	Observed value $\mathbf{z}'_i$
-131062	Not wrapped (carries no inequality)	-131062
131082	**Wrapped (carries an inequality)**	**-131062**

That is, **from the signature alone it is impossible to determine which class a coefficient with $|\mathbf{z}'_i| \geq \gamma_1 - \beta$ belongs to**; nor does the sign of the observed value indicate the direction of the wrap (both an overflow and an unwrapped negative value appear negative).

Disambiguation is impossible. One natural idea is to use the fact that the absolute value of $(c \cdot \mathbf{s}_1)_i$ is bounded by β (see Section 2.1) to filter out wrong hypotheses: if, under a hypothesis of wrapping in some direction, the resulting bound falls outside $[-\beta, \beta]$, that hypothesis must be false. But this filter can only rule out an **impossible direction**; it cannot separate misclassification from a true wrap — the observed value $\mathbf{z}'_i = -131062$ could be either an unwrapped -131062 or a wrapped 131082, and the correct interpretation of each **lies within the bound**.

Consequence: misclassifications make up the vast majority of the candidate set. The coefficients falling into the β-neighborhood of the boundary (i.e. producing a misclassification) far outnumber those that truly wrap. Modeling $\mathbf{y}$ as uniform on $(-\gamma_1, \gamma_1]$, the wrap probability is about $E[(c\cdot\mathbf{s}_1)^+]/(2\gamma_1)$ while the boundary-neighborhood probability is about $\beta/(2\gamma_1)$; since $\beta = 78$ is far larger than the typical value of $(c \cdot \mathbf{s}_1)^+$, the latter is about twenty times the former. We added coefficient classification statistics over **all signatures (without any filtering)** to the faulty-signature generation tool; over 5×10^5 signatures and 5.12×10^8 coefficients the measured result is: wrapped coefficients (i.e. $|\mathbf{z}| \geq \gamma_1$) account for **0.0029%**, and boundary-neighborhood coefficients (i.e. $|\mathbf{z}| \in [\gamma_1 - \beta, \gamma_1)$) account for **0.0570%**.

The denominator of 95.2% needs to be stated. Only these two classes of coefficients ($|\mathbf{z}| \geq \gamma_1 - \beta$, together 0.0599%) give rise to ambiguity after unpacking; misclassifications account for $0.0570/(0.0029{+}0.0570) = \mathbf{95.2\%}$ of them. If all 5.12×10^8 coefficients are taken as the denominator, misclassifications account for only 0.057%. **It is the former that is the meaningful measure for the adversary** — he can construct constraints only from these two classes of coefficients, and about 95% of them are misclassifications.

What this means. On the compressed-packing reference implementation, naively unpacking and constructing inequalities yields a constraint set of which about **95%** are wrong; and a single wrong inequality can make the linear program infeasible. Completing the attack on that implementation therefore requires **strong outlier handling** (for example a maximum satisfiable subset or robust linear programming, so that a solution can still be found when correct constraints are in the minority), which is substantial algorithmic work rather than an engineering detail. We list it as future work.

Relation to loop-abort attacks. The compressed packing of $\mathbf{z}$ is not the only risky path described in this section. As noted in Section 3.1, [Ulitzsch2023] uses clock glitches to **abort the coefficient sampling loop for $\mathbf{y}$**, so that the coefficients at the skipped positions remain at their initial value of zero, thereby obtaining a system of equations of the form $\hat{\mathbf{z}} = c\cdot\mathbf{s}_1 + \mathbf{y}$ (in which some coefficients of $\mathbf{y}$ are known to be zero), which is solved by integer linear programming; that work completed an end-to-end validation on a real Cortex-M4. Both paths are adjacent to the packing stage of $\mathbf{z}$, but **the fault point and the difficulty differ**:

- [Ulitzsch2023] acts on **the sampling/unpacking loop itself**, and the fault produces a **deterministic, known perturbation** (coefficients set to zero), so the system of equations can be built directly;
- what this section discusses is the wraparound that occurs when $\mathbf{z}$ is packed **after check 1 has been skipped**, whose difficulty lies in the fact that the unpacked observed value **cannot be**

distinguished from the boundary neighborhood — i.e. the 95.2% misclassification quantified in this section.

The relation between the two is complementary rather than duplicative: [Ulitzsch2023] demonstrated that "the packing/unpacking stage can be exploited directly by faults, and that this is already feasible on real hardware", whereas what this section points out is **a solving obstacle caused by packing ambiguity on a different path**. Together they show that the compressed packing of $\mathbf{z}$ is a stage of ML-DSA implementations that concentrates risk and is easily overlooked in protection.

Summary. Under the specification-level premise, the conclusion of this paper about the exploitability of the blind spot is definite; porting the attack in full to the compressed-packing reference implementation requires additionally handling the ambiguity above, which is a separate engineering and algorithmic effort.

6.3 Experimental Procedure and Results

The experimental procedure has three steps: generate faulty signatures, filter exploitable signatures, and construct and solve the linear program. All experiments are carried out on the same implementation and under the same randomized signing configuration as described in Section 5; the results are given in Table 4.

Table 4 Experimental results of the end-to-end attack (Dilithium-2)

Stage	Result	Time
Generate faulty signatures	3,000,000	About 12 minutes
Filter exploitable signatures	88,261 (2.94%)	Included in the previous step
Generate linear inequalities	89,575 (per polynomial 22,611 / 22,679 / 22,087 / 22,198)	—
Inequality construction and LP solving	All 1024 coefficients of the 4 polynomials **correct**	About 18 minutes (of which LP solving is seconds)
Total	Full recovery of $\mathbf{s}_1$	About 30 minutes

Here "exploitable signature" means a signature containing at least one coefficient satisfying $|\mathbf{z}_i| \geq \gamma_1$, accounting for 2.94% of all signatures. Each such signature contributes on average about 1.01 inequalities (89,575 / 88,261), so about 2.2×10^4 inequalities are available per polynomial.

Solving uses the HiGHS linear programming solver provided by SciPy, and the four polynomials are solved within seconds; the bulk of the 18 minutes is spent recomputing the challenge polynomials from the 88,261 signatures and constructing the inequality matrix.

The number of samples actually required is far smaller. Repeated measurements over disjoint corpus blocks show that the LP method used here already recovers all coefficients at about 1.4×10^4 exploitable signatures (about 3.6×10^3 inequalities per polynomial), and succeeds reliably at about 2×10^4 signatures; at that size, the LP vertices obtained under several different random objectives already coincide exactly with the true solution. The 8.8×10^4 margin in Table 4 is therefore roughly **four- to sixfold**, not twofold. If the solving objective is changed from "find a feasible point" to "minimise the total inequality violation", i.e.

$$\min \textstyle\sum_k e_k \quad \text{s.t.} \quad C\mathbf{x} - \mathbf{e} \leq \mathbf{b},\ \mathbf{e} \geq 0,\ \mathbf{x} \in [-\eta, \eta]^{256} \text{ integer},$$

and the resulting integer program is solved by HiGHS branch-and-bound (invoked through SciPy's `milp`, with a 600-second limit per polynomial), the requirement drops further to about 7×10^3 signatures. At $N = 7{,}000$ this formulation has about 2.8×10^3 constraints and 256 integer variables per polynomial (plus an equal number of continuous slack variables). The constraint count exceeds the "about 1.01 inequalities per signature" quoted above because that figure counts only coefficients with $|\mathbf{z}_i| \geq \gamma_1$ (the convention of Table 4), whereas the solver also uses the boundary-neighbourhood coefficients $|\mathbf{z}_i| \in [\gamma_1 - \beta, \gamma_1)$ discussed in Section 6.2 — since $\mathbf{y}_i = \mathbf{z}_i - (c \circledast \mathbf{s}_1)_i$ must lie in the sampling range $[-\gamma_1 + 1, \gamma_1]$, these too yield valid inequalities. Concretely, $7{,}000 \times 1.01/4 \approx 1{,}770$ come from the former and the boundary neighbourhood contributes a further $\approx 1{,}010$ (about 36%), giving about 2.8×10^3 in total. Using only the former gives 3 of 4 instances at the same size. **15 of the 16** polynomial instances across 4 keys are solved within a

600-second limit (**median about 26 s, slowest 304 s**), the single failure being key 1, polynomial 1, for which the solver's best objective value within the limit was 20 (i.e. 20 inequalities still violated), so no feasible solution was found — a limitation of the time limit, not a proof of infeasibility; a longer limit suffices. The objective is the **total violation (ℓ_1)** rather than e.g. ℓ_∞ because it yields additive slacks and hence a usable gradient for branch-and-bound; the pure feasibility form (objective identically zero) degenerates to enumeration on the same solver (Section 6.4).

How much information does the inequality channel carry? This question can be decided rigorously. Let $C\mathbf{x} \leq \mathbf{b}$ be the **signature-derived inequalities** available to the adversary. If two distinct integer points $\mathbf{x}, \mathbf{x}' \in [-\eta, \eta]^{256}$ satisfy all of them, then **these inequalities alone** cannot distinguish the two. Hence exhibiting, at some size N, a candidate solution that satisfies every inequality proves that the inequality channel is insufficient at that size. This is an **existence certificate**, independent of any heuristic or solver.

The search space for such candidates can be made very small. The challenge polynomial c has only $\tau = 39$ nonzero coefficients (15.2% of 256), so for a perturbation that changes a single coefficient, $\mathbf{d} = \Delta\mathbf{e}_j$, the convolution $(c \circledast \mathbf{d})_i$ vanishes with probability 84.8% — in which case that inequality is **satisfied regardless of the value of $\mathbf{s}_1$**. Such a perturbation therefore survives any single inequality with probability $1 - \frac{\tau}{n} \cdot \frac{1}{2} \cdot \Pr[\delta = 0]$: the challenge touches that coordinate with probability $\tau/n = 15.2\%$, and even then the perturbation is excluded only if the sign is adverse **and** $\delta = 0$, with $\Pr[\delta = 0] = 0.0815$ measured. The survival probability is thus about 0.994 — far above the 1/2 of a random half-space (estimating "if touched, a coin flip" would give 0.924, which is an incorrect lower bound). This is the consequence of the constraints being tangent to the true solution (Section 2.6): they are almost blind along individual coordinate directions, which also makes single-coefficient perturbations the hardest family to rule out.

Exhaustively testing all 256×2 candidates $\mathbf{s}_1 + \Delta\mathbf{e}_j$ gives the following. The test covers 4 keys (ρ, $\mathbf{s}_1$ and $\mathbf{s}_2$ are generated deterministically with the standard KAT tool `PQCgenKAT_sign_Modified2` from a fixed entropy source; records 0–3 are used, with 4×10^5 to 1.2×10^6 faulty signatures generated per key) over several disjoint corpus blocks. **Across the four keys (0/1/2/3), the last surviving candidate appears at N = 2,800/3,000/3,500/3,800 respectively**, after which each dies out; the very last one occurs for key 3 at $N = 3{,}800$ (about 1.5×10^3 inequalities per polynomial), polynomial 0 — changing coefficient 96 from -1 to 0 satisfies every constraint, with maximum slack exactly 0. At $N = 4{,}000$, none of the **48 tested instances** (4 keys × 3 disjoint corpus blocks × 4 polynomials) admits such a candidate. **That is: in the worst case (key 3), the signature-inequality channel alone determines $\mathbf{s}_1$ uniquely only from about 3,800 signatures onward; the other keys have lower thresholds (2,800–3,500).**

That threshold, however, describes the information structure of the inequality channel, not a lower bound on the cost of the attack. The adversary also knows $\mathbf{t}_1 = \text{Power2Round}(\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2)_1$, a very strong condition: each coefficient of $\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ must fall in a window of width 2^{13} whereas $q \approx 2^{23}$, so a single coefficient is hit with probability about 2^{-10} and all $kn = 1024$ of them with probability about 2^{-10240}. The space of $\mathbf{s}_2'$ has size $5^{kn} = 5^{1024}$ (each coefficient of each polynomial independently takes a value in $[-2, 2]$), so taking the union over $\mathbf{s}_2'$ still leaves an upper bound of $5^{1024} \cdot 2^{-10240} \approx 2^{-7864}$.

Three remarks. First, this is a statement **averaged over the random matrix $\mathbf{A}$**: since $\mathbf{A}$ is fixed by the public seed ρ, whether a candidate matches is a deterministic event; what the estimate says is that *for all but a negligible fraction of matrices* $\mathbf{A}$ *no such matching exists*, while the deterministic fact for our own instance is covered by direct measurement (below). Second, the argument holds for **any nonzero perturbation**: $\mathbf{Ad}$ is uniform for random $\mathbf{A}$ and the argument does not depend on the support size of $\mathbf{d}$, so it is not limited to the single-coefficient family we exhaustively tested. Third, direct measurement confirms it. Deciding whether a candidate is consistent does **not** require enumerating all 5^{1024} values of $\mathbf{s}_2'$: the condition **separates over coefficients** — it requires each of the 1024 coefficients to fall in its window, and the 1024 coefficients of $\mathbf{s}_2'$ are independent, so a candidate is consistent **iff for every coefficient there exists** $v \in [-\eta, \eta]$ **matching the high bits**, at a cost of only $O(kn(2\eta + 1))$. By this exact criterion, across the 16 combinations (4 keys × 4 polynomials): at $N = 1{,}000$ the number of candidates admitted by the inequalities ranges from 11 to 55 per combination (mean about 23; the spread is group-by-group fluctuation with **no systematic link** to the key or the polynomial — the four polynomials of a single key already cover the whole range, e.g. 25/19/55/11 for key 1), and **none** is consistent with the public key. Since the constraint set grows monotonically with N, the candidate set at $N = 1{,}000$ is the largest over all sizes; it contains **375 candidates**, of which **0 pass** the public-key

test — **including the very last candidate, at** $N = 3{,}800$.

The sample complexity of this attack is therefore a computational question, not an information-theoretic one. Two distinct quantities must be kept apart:

- **(i) The information-theoretic threshold of the inequality channel (about 3,800):** below this size the inequalities alone cannot exclude candidate solutions. It describes the information structure of that one leakage channel and is **unrelated** to how many samples the adversary actually needs.
- **(ii) The sample complexity of the attack (lower bound unknown):** the size at which solving becomes computationally feasible. Since the adversary holds both the public key and the inequalities, a joint solve could in principle push this below 3,800; the lower bound for (ii) is therefore not only unknown but **cannot** be obtained by information-theoretic arguments, because the public key already determines $\mathbf{s}_1$.

The public key does not simply hand over the answer because recovering $(\mathbf{s}_1, \mathbf{s}_2)$ from $(\rho, \mathbf{t}_1)$ is an MLWE-type hard problem — hard in **computational complexity**, not in information (otherwise the public key would leak the private key). What the fault-derived inequalities supply is not "more information" but a **computationally feasible recovery channel**; the threshold reported here (about 7×10^3) measures the size at which that channel becomes solvable.

In short: **3,800 is the information-theoretic threshold of the inequality channel, whereas 7,000 is the smallest size at which the current solver exploits that channel.** A better solver may push the latter lower; whether it can go below 3,800 depends on whether the public-key test can be combined with the solve (item (ii) above) and is **not** limited by the channel threshold.

6.4 Comparison with Prior Work

Section 6.4 of [Polytope] likewise gives an attack on $\mathbf{s}_1$, targeting a specification implementation **without any protection**: the authors collected 2×10^6 faulty signatures on that implementation, obtained 13,584 inequalities, and recovered $\mathbf{s}_1$.

Our attack is identical to theirs in algebraic construction; the difference lies in the attack target: **the target of our attack is an implementation with the patent's third check added**. Because of the blind spot described in Section 5, that countermeasure poses no obstacle whatsoever to this attack, and the detection count in our experiments is 0. In other words, what we demonstrate is not "what happens when there is no protection" but "what still happens after this protection is added".

A note on the number of signatures. Recent work [Shin2026] reports that a single faulty signature suffices to recover the private key, in sharp contrast to the 8.8×10^4 in this section. We must explain here why the two are not directly comparable.

Note that the single-signature conclusion of [Shin2026] is obtained under a **deterministic signing** configuration; whether its attack can still achieve key recovery with a single signature under randomized (hedged) signing is not discussed in that paper. All experiments in this paper are carried out under the randomized configuration (see Section 4.2).

The difference is structural, not a matter of attack efficiency. It comes from the attack surface itself: once challenge sampling is affected by a fault, a single signature can leak a complete linear relation sufficient to determine the key material; whereas skipping a rejection sampling check leaks at most **one linear inequality on the 256 coefficients of a single private key polynomial** per output signature, and about ten thousand such inequalities must accumulate before the polytope containing the private key is reduced to dimension 0 (see Section 2.6). This is an inherent property of inequality-based attacks and is independent of the implementation or of countermeasures.

A cost spectrum under three leakage models. Putting [Zhou2025], [Shin2026], and this paper together reveals a clear pattern: the number of samples an attack requires is determined by **how much information each leakage provides**, which in turn is determined by the leakage model.

Work	Leakage model	Information about the private key provided per sample	Samples required
[Shin2026]	Fault on challenge sampling	One complete linear relation sufficient to determine the key material	**1**
[Zhou2025]	Side-channel observation of rejected values	High-Hamming-weight constraints on rejected challenges and responses	A few traces
This paper / [Polytope]	Fault skipping a rejection sampling check	At most one inequality (256 coefficients of a single polynomial)	$10^5 \sim 10^6$ ([Polytope]) / about 10^4 (this paper, Section 6.3)

The difference among the three stems from **what the adversary can see**: [Shin2026] and [Zhou2025] observe rejected values or intermediate quantities **inside** the signing flow, whereas the adversary of this paper and of [Polytope] can only see **published** signatures. Without access to internal values, each signature leaks at most one inequality. Note, however, that this bounds the *form* of the leakage, not the *number* of samples required: how many inequalities recovery needs depends on the solving method, and Section 6.3 shows that optimisation-based solvers reduce the required signature count to about 7×10^3.

Our yield is comparable to that of similar work. We obtain 89,575 inequalities from 3×10^6 signatures (about 2.24×10^4 per polynomial), a yield of about 2.99%; the comparable attack of [Polytope] obtained 13,584 per polynomial from 2×10^6 signatures, a yield of about 2.7%. The two are comparable. The reason we collected 3×10^6 is to leave margin for solving.

It is important here to distinguish the **yield** from the **requirement**. The proportion of exploitable signatures does have an upper bound — it is determined by the natural probability of $\|\mathbf{z}\|_\infty \geq \gamma_1$ (measured at about 2.8%) — but this constrains only how many inequalities each signature supplies on average, not how many inequalities recovery needs. The latter depends on the solving objective and on the geometry of the constraint set, and Section 6.3 shows that it can be improved. **The bound on the yield only lets one write a lower bound of the form "inequalities required ÷ yield", and the number of inequalities required is not a constant** — it depends on the solving method and, as Section 6.3 shows, can be improved substantially. The bound on the yield therefore does not constitute a lower bound on the sample complexity.

6.5 Impact Chain

After recovering $\mathbf{s}_1$, the adversary can combine it with public information to complete a full key recovery. From $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ together with the public $\mathbf{t}_1$ (and $\mathbf{t}_0$, which can be recovered from signatures, see [Uncompress]) we obtain $\mathbf{s}_2 = \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0 - \mathbf{A}\mathbf{s}_1$, i.e. the adversary can further recover $\mathbf{s}_2$ and thereby obtain the complete signing key, and can thus forge signatures on arbitrary messages.

Therefore, although the attack surface targeted by the patent's third check is partially closed, the attack route corresponding to its blind spot can still break through this countermeasure completely.

7 Overhead Evaluation

Whether a countermeasure can be deployed in practice depends on its cost. This section gives measured overheads for the countermeasure variants, complementing the cost dimension of the "systematic evaluation".

7.1 Measurement Method

Measurements are performed on the Dilithium-2 reference implementation on an x86-64 platform, reading the timestamp counter with rdtsc. Because rdtsc counts at a constant reference frequency, the reported values are **reference cycles** rather than core cycles, and the absolute values vary with the platform and the processor's frequency state.

The signing algorithm contains a rejection sampling loop, so the cost of a single signing is itself variable. To suppress scheduling interference and frequency drift, each variant is measured over 5 repeated rounds of

5,000 signatures each, taking the minimum. In addition, to separate the overhead of each countermeasure from the measurement code, we time and accumulate the evaluation of each check individually inside the signing function.

7.2 Per-Check Overhead Attribution

Table 5 gives the additional overhead of each embodiment within the signing flow. Because rdtsc counts at a fixed reference frequency while the processor core frequency varies with load and thermal state, the absolute cycle counts are **not reproducible across runs** — our measured baseline signing drifts between $1.05\times10^6 \sim 1.11\times10^6$ reference cycles. This table therefore reports the ranges of **4 independent runs**, and the percentages in it are always computed with the **baseline within the same run** as the denominator, so as to eliminate cross-run drift.

All per-check figures have been **corrected for instrumentation overhead**: in the same run we measured, with exactly the same probe structure (two timer reads, one accumulation, one store) but evaluating a no-op, a calibration value of **30 ~ 33 cycles/signature**, which is subtracted from each check's reading.

Table 5 Overhead attribution per check (per signature; 4 independent runs, minimum of 5 rounds × 5,000 signatures each; instrumentation overhead deducted)

Variant	Additional cycles	Share of signing overhead
E1 $\mathbf{w}_1 = \mathbf{r}_1$ comparison	425 – 547	**0.04% – 0.05%**
E5 $\|\mathbf{w}_0 - c\cdot\mathbf{s}_2\|_\infty < \gamma_2 - \beta$	4,610 – 5,105	0.43% – 0.46%
E2 UseHint comparison	8,468 – 9,257	0.78% – 0.84%
E4 MakeHint comparison	10,892 – 12,240	1.01% – 1.11%
E6 recomputed-hint comparison	12,493 – 13,594	1.15% – 1.23%
Total, six embodiments	36,888 – 40,743	3.41% – 3.69%
[For comparison] sign-then-verify (one full verification)	245,884 – 259,334	22.2% – 23.6%

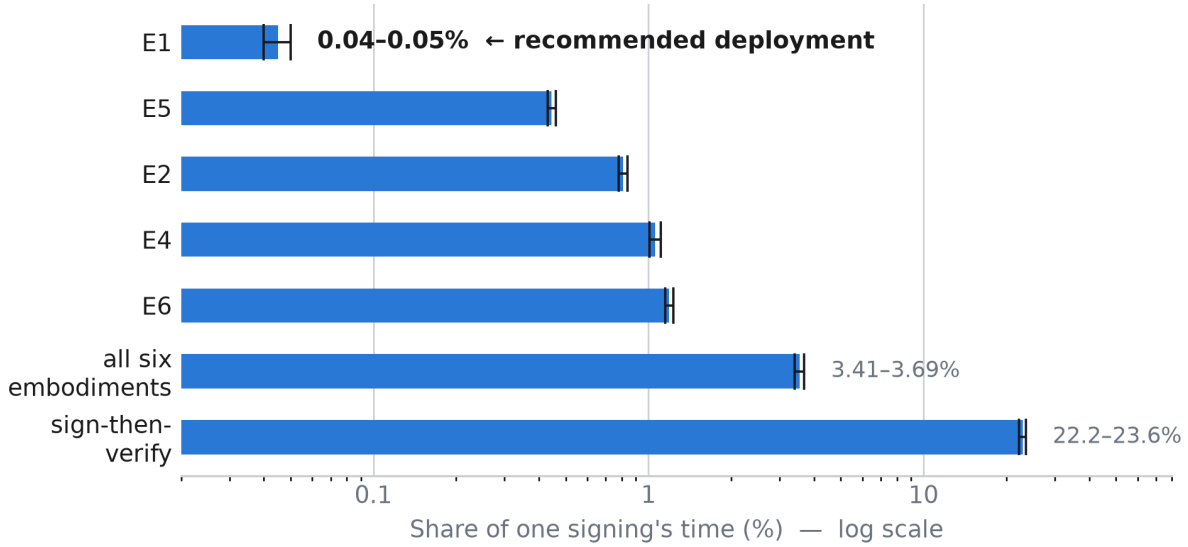


Figure 3 Overhead comparison of the countermeasure variants (log scale)

7.3 Key Observation: The Cheapest Variant Suffices

Table 5 yields a conclusion of direct relevance to engineering practice: **E1 is the cheapest of all 6 embodiments, and its detection capability is exactly the same as that of E2, E3, and E4**.

Section 5 already proved mathematically that E1–E4 reduce to the same decision condition, and confirmed experimentally that the four have bit-for-bit identical detection counts (all 1076/30000). Combined with

the overhead data of this section, the extra overhead of E2, E4, and E6 relative to E1 (about 0.7 ~ 1.2 percentage points) buys **the same detection capability**; and although E5 detects more signatures (56.98% versus 3.59%), the part it detects in addition does not, by Proposition 5, carry exploitable information, so its 0.43% ~ 0.46% overhead buys no additional practical protection.

Therefore, if one decides to deploy the countermeasure proposed by the patent, one should **deploy only E1**: the cost drops from 3.41% ~ 3.69% to 0.04% ~ 0.05% with no change in protective capability.

7.4 Cost Comparison with Sign-then-Verify

As a control, we measured a representative of the other class of countermeasures — sign-then-verify. This scheme performs a full public verification immediately after each signing and does not output the signature if verification fails. On this platform, **an independently measured full verification costs 245,884 ~ 259,334 cycles, equivalent to 22.2% ~ 23.6% of one signing**; the scheme therefore adds about 22% ~ 24% overhead to each signing. (We also measured this increment by the differential method of "subtracting two independent builds", and the result fluctuated widely between 5%~28% — Section 7.6 explains why that method is unreliable, and we consistently rely on direct measurement.)

This yields a noteworthy comparison: **E1's overhead is far lower than that of sign-then-verify, while the two happen to cover different attack paths** (see Section 8). On this platform the two differ by about 460 ~ 590 times (i.e. nearly two orders of magnitude); Section 7.5 gives the corresponding data on Cortex-M4 (about 48 times). This difference is the direct basis for the deployment recommendation in Section 8.

7.5 Measurements on Cortex-M4

To give direct data on an embedded platform, we repeated the above measurements on real hardware.

Platform and configuration. The target board is an STM32F407G-DISC1 (Cortex-M4F, 1 MB Flash / 192 KB SRAM), and the implementation measured is pqm4's `ml-dsa-44/m4f`, i.e. ML-DSA-44 optimized with hand-written assembly for Cortex-M4. Following the pqm4 benchmarking convention, the processor runs at **24 MHz** — the clock is deliberately lowered to obtain zero wait states, which makes the cycle counts deterministic. The timing source is the DWT cycle counter; each configuration collects 200 signatures, about 850 rejection sampling iterations.

Table 6 Measurements on the Cortex-M4 platform (STM32F407G-DISC1 @ 24 MHz)

Operation / countermeasure	Cycles	Relative to bare signing
keypair	1,355,710	—
sign (bare)	**3,657,338**	—
verify (independently measured)	1,351,518	+37.0%
with E1 countermeasure	**+30,472**	**+0.83%**
sign-then-verify (combined measurement)	5,118,559	+40.0%

On the variation of these numbers. The `sign (bare)` and `sign-then-verify` rows **fluctuate from run to run**: ML-DSA uses randomised signing, so the number of rejection-sampling iterations differs for each signature and the cost of a single signing is not a constant. In the official pqm4 benchmarks the same implementation shows `sign` of **1,812,557 / 3,943,121 / 17,009,165 cycles** (MIN / AVG / MAX over 1000 executions), a spread of about **9×**; Table 6 reports one sample of 200 signings from this work. **The per-item countermeasure overhead is not affected** — the 2,578 cycles for E1 is a per-call net value obtained by averaging over calls and subtracting the empty-instrumentation calibration, and is independent of the iteration count.

Three points follow from Table 6.

First, **the measured overhead of E1 on the embedded platform is +0.83%**. It consists of three parts: a single `chknorm` call costs 2,578 cycles (256 coefficients, or 10.07 cycles/coefficient, instrumentation overhead deducted, see Section 7.6); it is called on average 11.8 times per signature; the total is 30,472 cycles. This figure is significantly higher than the 0.04% ~ 0.05% on x86-64, because the pqm4 `m4f` implementation has hot spots such as the NTT written in assembly while `chknorm` is a pure C loop —

the numerator is unoptimized and the denominator is optimized, so the ratio naturally rises. But 0.83% is still of "negligible" magnitude.

Second, **the overhead of sign-then-verify is +40.0%**. This result has an independent cross-check: the official benchmark data shipped with the pqm4 repository show that on the same platform verify = **36.1%** of sign (official figures 1,421,623 / 3,943,121 cycles), while our measurement is 37.0%; the two agree. Thus "one full verification added to each signing" necessarily entails about +36% overhead arithmetically.

Third, **the overhead of a single `chknorm` call has an independent order-of-magnitude cross-check**. The official pqm4 benchmarks publish only **scheme-level** (keygen/sign/verify) data, not per-function data, so a per-item comparison against the official figures is not possible (the cross-check in this section can only use the official verify/sign ratio). As a substitute, we performed an **instruction-level accounting** of the code under test: in pqm4's `ml-dsa-44/m4f`, `poly_chknorm` is a **pure C loop** (unlike hot spots such as the NTT, which are written in assembly), and each coefficient takes roughly 1 memory access, 1 arithmetic shift right, 1 bitwise AND, 1 subtraction, 1 comparison, and 1 branch, which together with loop overhead amounts to about **9 ~ 10 instructions**; on the single-issue, in-order Cortex-M4 this corresponds exactly to an order of magnitude of about 9 ~ 10 cycles/coefficient. The measured net value of **10.07 cycles/coefficient** is consistent with that order of magnitude, which shows that this reading is not a measurement artifact.

Hence **two platforms independently yield the same conclusion: E1 is about 460 ~ 590 times cheaper than sign-then-verify (x86-64) and about 48 times cheaper (Cortex-M4)**. The absolute values vary with the platform and the run state, but the order-of-magnitude difference is robust.

Cost comparison against an arithmetic countermeasure. These figures can be set against published data for a **different class of fault countermeasure** on the same platform: [Bauer2024] reports about **72%** overhead for its NTT evaluation/interpolation countermeasure on the Cortex-M4 applied to Dilithium, whereas the joint deployment recommended here (E1 + sign-then-verify) costs about **41%** on the same platform (40.0% + 0.83%). We stress that **the two protect against different fault classes** — [Bauer2024] targets arithmetic faults in the NTT computation, while this paper targets the skipping of rejection-sampling checks. They are therefore **not substitutes but composable**. A deployer needing to cover both classes would pay the sum, about 113%. This further underscores why keeping E1 at 0.83% matters: it is a negligible share of that total, yet it closes exactly the attack path on which sign-then-verify fails.

7.6 Notes on Measurement Methodology

Why direct measurement is authoritative. All per-item overhead data in this paper come from **instrumenting and timing the target code inside the signing function**, not from "comparing the total time of two builds". The latter looks direct but is severely distorted in practice: on the M4, comparing the total time of the baseline build with the E1-augmented build gives an E1 overhead of only +0.21%, whereas timing directly inside the function gives +0.83% for the same code — a 4-fold difference. The distortion arises because adding one line of checking changes the **code layout** — function alignment, inlining decisions, and instruction cache placement all change accordingly, and these side effects are mixed with the overhead of the target check and cannot be separated. The same phenomenon is observed on x86-64 (the differential method gives a sign-then-verify increment fluctuating between 5%~28%, while independent measurement of verification gives 22% ~ 24%, see item 5 of Section 8.4). Therefore **for any measurement involving a small overhead, in-function instrumentation should be used rather than cross-build differencing**.

Calibration of instrumentation overhead. The per-item timings of Section 7.5 are taken inside the signing function, so the raw readings include the overhead of the instrumentation itself. To separate that overhead, in the same run we also executed **empty instrumentation** — identical in structure to the real measurement (two timer reads, one accumulation, one count) but performing no check. The calibration result is **43.0 cycles** per call, **1.6%** of E1's raw reading (2,621 cycles); the 2,578 cycles and +0.83% reported in Table 6 are both **net values after deducting instrumentation**.

Platform note and limitations. The absolute values in Table 5 and Table 6 belong to two different platforms and should not be compared directly across platforms; what is comparable is the relative relation within each platform. In addition, we did not perform fault injection on real hardware — the scope qualifications stated in Section 8.4 apply equally to this section.

8 Deployment Recommendations and Discussion

The previous two sections gave the effectiveness and the cost of the countermeasures. This section synthesizes them into conclusions usable in engineering practice, and discusses limitations.

8.1 Coverage of the Two Countermeasures: Complementary and Mutually Indispensable

Table 7 summarizes the behavior of the three countermeasures evaluated in this paper on the three attack paths. The criterion is: whether the countermeasure can prevent the adversary from obtaining exploitable signatures.

Table 7 Countermeasure × attack-path coverage matrix

Attack path	sign-then-verify	Third check E1	Double computation
Skip check 1 (attack $\mathbf{s}_1$)	**Effective** (100% of exploitable signatures fail verification)	**Zero detection**	Ineffective
Skip check 2 / specification implementation (attack $\mathbf{s}_2$)	**Effective** (3.59% fail verification)	Effective (detects 3.59%)	Ineffective
Skip check 2 / reference implementation (attack $\mathbf{s}_2$)	**Ineffective** (30000/30000 all pass verification)	**Effective** (detects all 1078 exploitable signatures)	Ineffective

The third row of the table is a situation specific to the reference implementation and is the most noteworthy row. The reference implementation's MakeHint formulation makes the faulty signatures it outputs **still pass public verification**, so the countermeasure that uses verification as its criterion fails completely: of 30,000 injections, all 30,000 faulty signatures pass verification, and 1078 of them (3.59%) are exploitable. The third check E1 detects exactly these 1078 — its decision condition $\mathbf{r}_1 = \mathbf{w}_1$ coincides completely with the exploitability condition stated in Proposition 5.

Conversely, the first row of Table 7 gives the opposite situation: when check 1 is skipped, verification is effective while the third check fails completely.

Taking the three rows together: **the union of the two countermeasures covers all three attack paths, and there is no attack path that both would miss; whereas deploying either one alone leaves a complete, practically exploitable attack path** — skipping check 1 breaks the third check (Section 6 has given the end-to-end verification), and skipping check 2 on the reference implementation breaks sign-then-verify. This is the direct basis for the deployment recommendation of this paper.

8.2 Deployment Recommendations

From the above complementarity, our engineering recommendation is: **deploy E1 and sign-then-verify together**.

- E1 costs 0.04% ~ 0.05% (x86-64) and covers the skip-check-2 attack on the reference implementation (the path on which sign-then-verify fails)
- sign-then-verify costs 22% ~ 24% (x86-64) and covers the skip of check 1 and the skip of check 2 on the specification implementation
- the total cost of joint deployment comes almost entirely from sign-then-verify (about 22% ~ 24%), with E1's 0.04% ~ 0.05% amounting to an almost cost-free supplement

It is worth emphasizing the nature of this combination: **the more expensive party (sign-then-verify) has a complete bypass path, while the party that closes that path (E1) has negligible cost**. This suggests that when evaluating rejection-sampling-related countermeasures, overhead should not be used as the criterion for selection — a far cheaper scheme may cover attacks that the more expensive one cannot.

8.3 Why Double Computation Does Not Apply

The "double computation" column of Table 7 is entirely ineffective, and the reason deserves separate explanation because it concerns the deployment form of ML-DSA.

The idea of double computation is to execute signing twice in a row and compare the results. It does not hold for ML-DSA, in two cases:

- **Randomized (hedged) signing**: this is the default form in FIPS 204 and the configuration used in all experiments in this paper. Each signing draws rnd from the randomness source to participate in deriving the blinding vector, and the nonce is randomized together with the message, so **the outputs of two honest executions are different to begin with** and there is no baseline for comparison.
- **Deterministic signing**: the inputs of the two executions are identical, so the same persistent fault (for example a device continuously affected by glitching, or the check skip simulated in this paper) yields **exactly the same erroneous output**, the comparison agrees, and the fault goes unnoticed. Double computation can only detect transient faults.

Our experiments confirm the first case: under the randomized configuration, the output difference rate between two consecutive signings is **100%** under all three fault models, i.e. the comparison always "fails" and the countermeasure cannot be implemented. The second case is a direct corollary: identical inputs and an identical fault necessarily produce identical outputs.

8.4 Limitations

The conclusions of this paper are subject to the following limitations, which future work will address one by one:

1. **Fault injection is simulated, not performed on real hardware.** Following the same paradigm as the simulation parts of [Polytope] and [Krahmer2024], we simulate instruction-skip faults by modifying code paths (the overhead data, by contrast, have already been measured on the two platforms x86-64 and Cortex-M4, see Sections 7.1~7.5). Section 4.5 has given an argument for the physical feasibility of this attack type — the goal of the argument is that "the attack type is physically possible", not that "the simulation results of this paper can be reproduced directly on any platform"; real fault injection requires clock/voltage glitching or electromagnetic equipment and is an independent experimental direction.
2. **Multi-fault models are not evaluated.** Our evaluation is based on the single-fault assumption. If the adversary can inject multiple faults (for example skipping a check and a countermeasure check at the same time), all redundancy-based countermeasures degrade; quantifying this degradation is future work.
3. **The attack has not yet been completed on the compressed-packing reference implementation.** The experiments of Section 6.3 are carried out at the specification level (consistent with Section 6.4 of [Polytope]), i.e. assuming the adversary can obtain uncompressed $\mathbf{z}$. The analysis and measurements of Section 6.2 show that porting this attack to the compressed-packing reference implementation requires handling about 95% misclassified constraints, which is a separate algorithmic effort.
4. **Masked implementations are not covered.** On a masked implementation, fault injection related to rejection sampling must affect all shares simultaneously, which is significantly harder; we do not pursue this here.
5. **The x86-64 overhead figures drift across runs.** rdtsc counts at a fixed reference frequency while the processor core frequency varies with load and thermal state; the baseline signing time of the same code swings between $1.05 \times 10^6 \sim 1.11 \times 10^6$ reference cycles, and the cross-run range of the verification share is 22.2% ~ 23.6%. We therefore report the x86-64 figures as **ranges** rather than single points (Table 5), and all percentages use the baseline within the same run as the denominator. On the Cortex-M4 side the processor runs at a fixed 24 MHz and the DWT cycle counter is read directly, so the figures are deterministic. The qualitative conclusion of Table 5 — that E1 is about two orders of magnitude cheaper than sign-then-verify — is robust across multiple runs on both platforms, but **the specific percentages should not be cited as precisely reproducible constants**.

9 Conclusion and Outlook

This paper has presented a systematic evaluation of fault attacks and countermeasures on the rejection sampling of ML-DSA. There are three main conclusions.

First, the "third rejection test" countermeasure proposed in patent WO2025261784A1 has a structural blind spot. Using the identity $\mathbf{Az} - c\mathbf{t} = \mathbf{Ay} - c \cdot \mathbf{s}_2 = \mathbf{r}$, we prove that the HighBits($\mathbf{Az} - c\mathbf{t}$) compared by this check is exactly $\mathbf{r}_1$ from the signing process, and that the original check 2 passing implies $\mathbf{r}_1 = \mathbf{w}_1$. The third check therefore decides only a necessary condition guaranteed by check 2. Further analysis shows that the 6 embodiments claimed by the patent actually provide only 2 independent conditions: {E1, E2, E3, E4} reduce to the same decision, E5 has exactly the same detection count as check 2, and E6 is a tautology; moreover, both conditions cover only the skip of check 2. In an experiment with about 30,000 fault injections, all 6 embodiments detect "skip check 1" at a rate of zero.

Second, this blind spot can be exploited in practice up to full key recovery. On an implementation protected by the third check, with an optimising solver objective the adversary needs only about 7×10^3 exploitable faulty signatures (our experiment collects 88,261 to leave a comfortable margin) to fully recover all 1024 coefficients of the signing key $\mathbf{s}_1$ — our analysis of the signature-inequality channel shows that the channel alone determines $\mathbf{s}_1$ uniquely only from about 3.8×10^3 signatures onward, whereas the public key already determines $\mathbf{s}_1$ information-theoretically (though the adversary cannot use it directly — that is an MLWE-type hard problem); **the sample complexity is thus a computational question rather than an information-theoretic one** (Section 6.3), in about 30 minutes end to end. Combined with existing $\mathbf{t}_0$ recovery methods, the adversary can further recover $\mathbf{s}_2$ and forge signatures on arbitrary messages. The attack is carried out under randomized signing and holds directly for the default deployment form of FIPS 204.

Third, the coverage of existing countermeasures is complementary and neither is dispensable, and joint deployment is a feasible solution. The evaluation matrix shows: on the reference implementation, sign-then-verify fails entirely against the skip of check 2 (100% of faulty signatures pass verification), while E1, the best embodiment of the third check, detects exactly all exploitable signatures; conversely, when check 1 is skipped, E1 detects nothing while verification is effective. E1 costs only 0.04% ~ 0.05% of a signing (x86-64) and 0.83% (Cortex-M4), about 460 ~ 590 times and 48 times less than sign-then-verify on the same platforms (22% ~ 24% and 40.0%, respectively). We therefore recommend deploying both; the total cost comes almost entirely from sign-then-verify, with E1 closing the gap where verification fails at essentially no cost.

Future work includes five aspects: fault injection validation on real platforms; porting the attack to the compressed-packing reference implementation (requiring handling of about 95% misclassified constraints); quantifying the degradation of each countermeasure under multi-fault models; analysis of the analogous problem on masked implementations; and an independent evaluation of the probabilistic countermeasure framework [Mishra2024].

References

[Bauer2024] Bauer S, De Santis F, Koleci K, Aghaie A. A fault-resistant NTT by polynomial evaluation and interpolation. Cryptology ePrint Archive, Report 2024/788, 2024.

[BDK+21] Bai S, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D. CRYSTALS-Dilithium: algorithm specifications and supporting documentation (version 3.1). 2021.

[BP18] Groot Bruinderink L, Pessl P. Differential fault attacks on deterministic lattice signatures. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(3): 21–43.

[Diverse2025] Yuan H, Liu Y, Ming J, Zhou Y. Diverse fault attacks on Dilithium and variant implementation: skipping, aborting and zeroing. In: 2025 IEEE 24th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2025: 2986–2993.

[ElGhamrawy2023] ElGhamrawy M, Azouaoui M, Bronchain O, Renes J, Schneider T, Schönauer M, Seker O, van Vredendaal C. From MLWE to RLWE: a differential fault attack on randomized & deterministic Dilithium. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(4): 262–286.

[FIPS204] National Institute of Standards and Technology. FIPS 204: Module-lattice-based digital signature standard. Federal Information Processing Standards Publication, 2024.

[HP23] Heinz D, Pöppelmann T. Combined fault and DPA protection for lattice-based cryptography. IEEE Transactions on Computers, 2023, 72(4): 1055–1066.

[Krahmer2024] Krahmer E, Pessl P, Land G, Güneysu T. Correction fault attacks on randomized CRYSTALS-Dilithium. Cryptology ePrint Archive, Report 2024/138, 2024.

[Mishra2024] Mishra N, Mukhopadhyay D. Probabilistic algorithms with applications to countering fault attacks on lattice based post-quantum cryptography. Cryptology ePrint Archive, Report 2024/551, 2024.

[Patent] Method for protecting against fault attacks an execution of a generation of a CRYSTALS-Dilithium signature. International patent application WO2025261784A1. Applicant: Thales Dis France SAS. Inventors: Azevedo Oliveira P, Calle Viera A, Cogliati B M, Vigilant D. Application No. PCT/EP2025/065511; priority EP 24305977.1 (2024-06-20). Published 2025-12-26.

[Polytope] Azevedo-Oliveira P, Calle Viera A, Cogliati B, Goubin L. Finding a polytope: a practical fault attack against Dilithium. In: Public-Key Cryptography – PKC 2025. Cham: Springer, 2025. (also Cryptology ePrint Archive, Report 2025/195)

[Ravi2023] Ravi P, Yang B, Bhasin S, Zhang F, Chattopadhyay A. Fiddling the twiddle constants: fault injection analysis of the number theoretic transform. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(2): 447–481.

[RJH19] Ravi P, Jhanwar M P, Howe J, Chattopadhyay A, Bhasin S. Exploiting determinism in lattice-based signatures: practical fault attacks on pqm4 implementations of NIST candidates. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (AsiaCCS). New York: ACM, 2019: 427–440.

[Shin2026] Shin W, Jeon S, Bae D, Park S, Kim H. Public coefficient matters: a practical differential fault attack on ML-DSA and HAETAE. Cryptology ePrint Archive, Report 2026/1344, 2026.

[SUCRE] Belaïd S, Benadjila R, Devevey J, Guerreau M, Legavre T, Martinelli A, Ricosset T, Rivain M, Rossi M. ML-DSA masking sweetened with SUCRE: shuffle-and-unmask countermeasure for rejection sampling. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2026(1): 618–659.

[Zhou2025] Zhou Y, Wang W, Sun Y, Yu Y. Rejected challenges pose new challenges: key recovery of CRYSTALS-Dilithium via side-channel attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2025(4): 817–847.

[TECS2024] Singh R, Islam S, Sunar B, Schaumont P. Analysis of EM fault injection on bit-sliced number theoretic transform software in Dilithium. ACM Transactions on Embedded Computing Systems, 2024, 23(2). DOI: 10.1145/3583757.

[Ulitzsch2023] Ulitzsch V Q, Marzougui S, Bagia A, Tibouchi M, Seifert J P. Loop aborts strike back: defeating fault countermeasures in lattice signatures with ILP. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(4): 367–392.

[Uncompress] Azevedo Oliveira P, Calle Viera A, Cogliati B, Goubin L. Uncompressing Dilithium's public key. Cryptology ePrint Archive, Report 2024/1373, 2024.