Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# Resolving an Apparent Key-Dependent Timing Side-Channel in Bouncy Castle ML-DSA-65 Signing: A Pre-Registered Discrimination

**Arpan Sharma**

arpansharma073@gmail.com

Independent Researcher

**Research Article**

**Keywords:** side-channel analysis, timing attacks, ML-DSA, rejection sampling, constant-time implementation, post-quantum cryptography

**Posted Date:** September 9th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-10910294/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** No competing interests reported.

# Resolving an Apparent Key-Dependent Timing Side-Channel in Bouncy Castle ML-DSA-65 Signing: A Pre-Registered Discrimination

Arpan Sharma
Independent Researcher, Dallas, TX, USA
arpansharma073@gmail.com
ORCID: 0009-0006-9303-7102

**Abstract**

ML-DSA (FIPS 204) signs by Fiat–Shamir with aborts, so the number of rejection-sampling iterations, and with it the signing time, varies from signature to signature. The scheme is designed so that the rejection probability does not depend on the secret key, but a per-key timing comparison over a finite message corpus will show an apparent key-dependence regardless: the keys happen to draw different iteration counts. We observed such a signal in Bouncy Castle's ML-DSA-65, a 1.35% spread in mean signing time across twenty keys over 1,200,000 signatures. We resolve it under a pre-registered protocol that records each signature's exact work profile, so timing is compared within identical work under thresholds frozen in advance. Signing time is explained by iteration count ($r = 0.958$ across per-key means). The per-key iteration spread is indistinguishable from sampling noise (one-way ANOVA $F = 1.20$, $p \approx 0.24$, replicated on fresh keys), confirming at scale that ML-DSA's rejection rate does not depend on the key. After conditioning on the work profile, the key term explains 0.0002% of residual variance, and no cross-key comparison passes the pre-registered effect-size bar. Bouncy Castle's ML-DSA-65 signing is clean at this resolution; its one value-dependent branch, an early-exiting norm check, contributes on the order of a nanosecond. Applied unchanged to ML-DSA-44 and -87, the protocol finds no implementation channel at any of the three standardized parameter sets. A fixed-versus-random-key leakage test on such a scheme flags sampling variation as leakage unless it conditions on the work performed.

**Keywords:** side-channel analysis; timing attacks; ML-DSA; rejection sampling; constant-time implementation; post-quantum cryptography

## 1 Introduction

Post-quantum signatures are entering the software that signs firmware, certificates, and code, and with them comes the question that has shadowed every deployed signature scheme: does the time a signer takes leak its private key? For lattice signatures the history is instructive. BLISS, the most efficient pre-standardization lattice scheme, used Gaussian rejection sampling whose data-dependent control flow leaked the secret through cache timing and branch tracing, and was broken in practice that way [1, 2]. ML-DSA (FIPS 204 [3]), standardized from CRYSTALS-Dilithium [4], was designed in that light. It keeps the Fiat–Shamir-with-aborts structure introduced by Lyubashevsky [5, 6]: sample a masking vector $y$, compute a candidate response $z = y + cs_1$, and reject and retry whenever a norm check would let $z$ leak information about $s_1$. But it samples $y$ uniformly rather than from a Gaussian, and with a margin $\beta$ chosen so that the acceptance probability of the check on $z$ is the

same for every key. For each coefficient, $y$ is uniform on a range at least $\beta$ wider than the acceptable window on each side, and $\|cs_1\|_\infty \leq \beta$ always, so the number of acceptable $y$ values, and hence the probability of passing, does not depend on the value of $cs_1$ at all. The iteration count is therefore meant to be a random variable whose distribution is the same for all keys. Its expected value for a given parameter set is a constant of the design, about 5.1 for ML-DSA-65 [7], not a property of the secret.

That design property is what makes ML-DSA's timing assessment subtle. The iteration count is not a secret by design: its distribution is meant to be the same for every key, so revealing it discloses nothing about the key. What is not allowed is for that distribution, or for the time per iteration, to depend on the key. A timing study therefore cannot simply measure that signing time varies and call it leakage, because it varies for every key. Nor can it compare mean signing times across keys and call a difference key-dependence, because over any finite message corpus twenty keys will draw twenty different samples of the iteration count, and their mean times will differ by sampling variation alone. A first-order leakage test in the style of TVLA [8] or `dudect` [9], run fixed-key versus random-key, will flag that difference. It is correct that the distributions differ for the sample in hand; it is uninformative about whether anything depends on the key.

We met this situation concretely. Bouncy Castle [10], the dominant third-party Java cryptography provider, exhibits a measurable spread in mean ML-DSA-65 signing time across keys, about 1.35% across twenty keys over 1,200,000 signatures in the campaign reported here, and a first-order test flags it. Three explanations are possible. The rejection rate could genuinely depend on the key, which would mean the design property above fails in practice, a result of independent interest. The implementation could add a key-dependent timing channel on top of the algorithm, through a value-dependent branch, a data-dependent memory access, or a non-constant-time inner routine. Or the spread could be sampling variation of a key-independent process, in which case there is no channel of any kind. Distinguishing the three is the actual measurement problem, and it is not solved by a $t$-test on raw signing times.

To answer it without the freedom to choose a favorable analysis after seeing the data, we (i) instrument the signing loop to record every signature's work profile, its iteration count and the multiset of stages at which iterations rejected, so that "same algorithmic work" is an observed fact rather than an assumption; (ii) freeze the strata, statistics, thresholds, replication requirement, and the mapping from outcome to conclusion in version control before examining any cross-key timing; and (iii) run, besides the discovery campaign, a placement control that re-allocates the same keys in a fresh process and a replication on entirely fresh keys, so that a per-key timing rank that is really a memory-placement artifact can be told apart from a value-dependent channel that would follow the key.

**Contributions.**

- A pre-registered discrimination protocol for the timing of a rejection-sampling signature scheme (Section 3): record each signature's work profile, test for a residual cross-key timing effect *within* identical work profiles under thresholds and a decision tree fixed in advance, and separate placement artifacts from value-dependent channels with a placement control and a fresh-key replication.
- An empirical confirmation, at the scale of 1,200,000 signatures per campaign, that ML-DSA-65's rejection rate does not depend on the key (Section 4): a one-way analysis of variance of iteration count by key finds no key effect ($F = 1.20$, $p \approx 0.24$; replication $F = 1.14$, $p \approx 0.30$), and the measured mean iteration count matches the design's expected repetition count. The apparent per-key timing spread is the sampling variation this design property predicts.

- The pre-registered result that Bouncy Castle's ML-DSA-65 signing carries no implementation timing channel: after conditioning on the work profile, the key term explains 0.0002% of residual variance, no within-stratum comparison passes the effect-size bar, and the residual sits at the noise floor of twenty per-key means with a ranking that reproduces only marginally across re-allocation. We localize the one value-dependent branch, an early-exiting norm check, at about a nanosecond. We disclosed the study to the maintainer, who confirmed the result.
- A replication of the whole protocol, unchanged, across the ML-DSA family (Section 4.1): the same frozen decision rule applied to ML-DSA-44 and ML-DSA-87 finds no implementation timing channel at any standardized parameter set, and the placement control and replication requirement are shown catching a borderline per-key spread at ML-DSA-44 and correctly classifying it as an artifact.
- A caution for side-channel practice (Section 5): a two-key first-order leakage test on a managed runtime reports a 1%-scale difference as leakage even when both classes hold the same key in two instances, so a same-key negative control is mandatory, and a test over a finite corpus reports iteration-sampling variation as key-dependence. Conditioning on the work performed, and testing the iteration count itself for a key effect, is what turns the observation into a decidable question.

## 2 Background

**Rejection sampling and its key-independence.** In Fiat–Shamir with aborts [5, 6], the signer's response $z = y + cs_1$ would leak $s_1$ if $y$ were reused or if $z$ were released whenever it fell outside the range that makes its distribution independent of $s_1$; rejection sampling discards such candidates. In ML-DSA [3, 4], $y$ is uniform in $[-\gamma_1 + 1, \gamma_1]$ per coefficient and $z$ is accepted only if $\|z\|_\infty < \gamma_1 - \beta$, with $\beta$ an upper bound on $\|cs_1\|_\infty$. Because the sampling range exceeds the acceptance window by at least $\beta$ on each side, every shift $cs_1$ of magnitude at most $\beta$ leaves the same number of acceptable $y$ values, and the acceptance probability of this check is the same for every key. A second check, $\|r_0\|_\infty < \gamma_2 - \beta$ on the low bits of $w - cs_2$, is argued independent of $s_2$ on the heuristic that $w = Ay$ is close to uniform. Two further checks, on $\|ct_0\|_\infty$ and on the hint weight, involve $t_0$ and fail rarely. The consequence is that the iteration count $N$ is geometric with a parameter that is a constant of the parameter set, and its expected value, about 5.1 for ML-DSA-65, appears in the design document as a fixed cost rather than a per-key quantity; the authors state the point directly, that "the expected number of iterations is independent of the secret key" [7]. Signing time is then, to first order, a per-iteration cost times a key-independent random count.

**Side channels on lattice signatures.** The uniform rejection sampling above exists because the Gaussian alternative leaked. BLISS's Gaussian sampler and its rejection step were attacked through cache timing [1] and through branch tracing and electromagnetic emanation [2], recovering keys from the data-dependent control flow of rejection itself. Dilithium's design removes that dependence from the rejection decision; the remaining side-channel work on Dilithium has accordingly targeted other parts of the computation: deterministic-signing fault attacks [11], profiled power attacks on intermediate values [12, 13], and masked implementations to resist them [14]. Most recently the rejection loop itself has become a target, through power rather than time: single-trace leakage from the polynomial arithmetic of rejected iterations reveals the rejected challenges and responses, from which the key is recovered [15, 16]. Those attacks read intermediate values; the timing of the rejection loop, the channel BLISS fell to, is expected to be clean in ML-DSA by construction.

That expectation is what this paper tests end to end, on a software implementation running on a managed runtime, where a timing channel could re-enter through the implementation even though the algorithm excludes it.

**Detecting timing leakage.** The standard tools test whether a timing distribution depends on a secret input. `dudect` [9] compares two input classes with a Welch $t$-test on percentile-cropped samples, avoiding a device model; test-vector leakage assessment (TVLA) [8] uses the same first-order statistic with a fixed-versus-random class split. Systematic tooling built on these detectors has recently been run across the NIST additional-signature candidates, finding timing issues that developers then patched [17]. Applied to ML-DSA signing with the key as the secret, both will report a difference between a fixed key and random keys over a finite corpus, because the per-key mean iteration counts of the sample differ. The discrimination this paper needs is conditional, on the work actually performed, and is built on top of these detectors rather than replacing them.

**The measurement environment.** The study runs on the JVM, whose managed runtime (just-in-time compilation, garbage collection, and thread migration between performance and efficiency cores) adds run-to-run and within-run drift that a native constant-time harness does not face. This drift is itself a confound: a per-key timing *rank* can arise from where a key's objects land in the heap in a given process rather than from the key's values, and can then look like a channel. Controlling for it is a first-class part of the design.

# 3 Method

**The discrimination question, made precise.** Each iteration of the signing loop runs a fixed structural sequence and exits at a definite stage: a reject at one of the three norm checks or the hint check, or the terminal accept. Define a signature's *work profile*

$$\texttt{profile_key} \;=\; N : (r_z, r_{r_0}, r_{ct_0}, r_{\text{hint}}, 1), \tag{1}$$

its total iteration count together with the multiset of reject-stages (the final 1 is the accept). Two signatures with the same `profile_key` executed the same branch structure, and therefore the same count of every expensive operation in the loop: the SHAKE blocks of mask expansion, the number-theoretic transforms, the pointwise products, and the decompose and hint passes. This is verified by reading the source, not assumed. The pre-registered question is then sharp: *after conditioning on `profile_key`, does mean signing time still depend on the key?* If no, any key-to-time relationship is carried entirely by the iteration count (**Outcome A**); if yes, and it replicates, there is an implementation residual the algorithm does not require (**Outcome B**). Whether the iteration count itself depends on the key is a separate, prior question, which we address directly in Section 4 with an analysis added after the pre-registered decision tree had resolved.

**What can vary within a fixed work profile.** Reading the (byte-identical across 1.84 and 1.85) source, exactly two value-dependent quantities can vary within one `profile_key`, and both are instrumented. The first is the early exit of the norm check `Poly.checkNorm`, which scans coefficients and returns on the first out-of-bound one, so the number of coefficients scanned depends on the *position* of the first violating coefficient, a secret-dependent value. This is the leading candidate for an implementation residual, and it is localized in advance. The second is the SHAKE-byte consumption of `SampleInBall`, whose rejection loop reads a variable number of bytes driven by the commitment hash, so a per-*key* dependence is not expected but is recorded and controlled. Everything else in the

loop body is full-scan with a fixed per-coefficient cost. Stratifying by `profile_key` therefore holds the algorithm fixed, and any surviving cross-key difference is by construction an implementation effect.

**Instrumentation.** A write-only tracer records, per signature, the iteration count, the reject-stage of each iteration, the coefficients scanned across all `checkNorm` calls, and the `SampleInBall` byte count. No recorded value is ever read back by the algorithm, so the instrumented and stock builds produce byte-identical signatures; this is checked as a gate. Because these counts are deterministic functions of the (key, message) pair, the work profile carries no timing component and can be collected on any host, which is what lets us condition timing on algorithmic work exactly rather than approximately. Timing itself is measured separately on the stock, signed jar, with the tracer off the classpath.

**Pre-registration.** The strata, statistics, thresholds, corrections, replication requirement, and the outcome-to-conclusion mapping were frozen in version control before any cross-key timing distribution was examined, and the file could afterwards only be appended to. A within-stratum cross-key difference counts as *present* only if it passes all three of a Holm-adjusted $p < 0.05$, a $|t| > 4.5$, and a |Cliff's $\delta$| $\geq 0.147$ (a small-effect floor), over the family of (key-pair $\times$ stratum) tests. In the analysis-of-covariance model time $\sim$ iterations + C(key), the key term counts only if it is significant *and* explains at least 1% of residual variance after controlling for iterations. The decision tree is: if the ANCOVA key term is absent, Outcome A is terminal; otherwise, test within identical work profiles, and Outcome B requires that a surviving residual also replicate on fresh seeds with the same sign and comparable magnitude.

**Campaigns and controls.** Each campaign signs 20 keys $\times$ 60,000 messages (1,200,000 signatures). The *discovery* campaign uses keys drawn from seeds 100–119 and a fixed message corpus. The *placement control* re-runs the same keys and corpus in a fresh JVM with a shuffled object-allocation order: a per-key timing rank driven by heap placement should move, while one driven by key values should follow the key. The *replication* campaign uses entirely fresh keys (seeds 200–219) and a fresh corpus. Four rig controls validate the timing harness end to end: a deliberately non-constant-time comparison (positive control), a constant-time comparison (negative control), and ML-DSA signing of a fixed versus a random message. A further set of two-key tests uses the `dudect`-style two-class harness with 250,000 signatures per run: key A versus key B (the test that first produced the apparent signal), key A versus a second instance of key A (a same-key negative control), and key B versus key A with the allocation order swapped. Timing runs use a fixed heap and execute one at a time under `caffeinate` on an actively cooled host.

**Environment.** Apple M4 (Mac16,10), macOS 27.0, OpenJDK 21.0.9. Bouncy Castle 1.85 (`bcprov` SHA-256 `20af26bf...`); version 1.84 (`64d6c5a6...`) is used only for the cross-version gates. The ML-DSA signing path is byte-identical between the two versions, and deterministic (hedged-off) signing is engaged so that a fixed (key, message) pair yields a byte-identical signature. The instrumentation, harness, analysis, gzipped raw per-signature data for all three campaigns, the rig controls, and the frozen pre-registration are public in the artifact repository [18].

# 4 Results

**The work profile reproduces exactly.** Because the work profile is deterministic in (key, message), it must reproduce on any correct setup, and it does: over the 1,200,000-signature discovery campaign the overall mean iteration count is 5.134, and the reject-stage shares are 47.3% at $\|z\|_\infty$, 52.6% at $\|r_0\|_\infty$, 0.00% at $\|ct_0\|_\infty$, and 0.09% at the hint check. The mean of 5.134 agrees with the expected repetition count of about 5.1 that the Dilithium specification gives for this parameter set [7], and the reject-stage shares confirm that the two key-independent-by-design checks account for essentially all rejections, with the $t_0$-dependent checks negligible.

**The per-key iteration spread is sampling noise, as the design predicts.** Across the twenty keys, the per-key mean iteration count spans 5.095 to 5.184, a spread of 1.74%, and it is this spread that a naive reading would call key-dependence. Tested directly, it is not. A one-way analysis of variance of iteration count by key, over the 1,200,000 signatures, gives $F(19, 1{,}199{,}980) = 1.20$, $p \approx 0.24$, against a 5% critical value of 1.59; the key accounts for 0.0019% of the variance in iteration count. The within-key standard deviation of the geometric count is 4.61 iterations, so the standard error of a per-key mean over 60,000 messages is 0.019, and the observed spread of 0.089 is 4.7 standard errors, above the expected range of about 3.7 for twenty noise means but within its ordinary variation, as the $F$-test's $p \approx 0.24$ quantifies. The replication campaign, on twenty fresh keys and a fresh corpus, gives $F = 1.14$, $p \approx 0.30$, with a spread of 1.29% (3.5 standard errors, below the expected range of 3.7 for twenty noise means). The per-key iteration spread is therefore consistent with every key sharing one iteration distribution, which is the design property of Section 2 observed empirically at scale. This analysis was not part of the pre-registered decision tree, which took the observed iteration counts as given and asked only about timing conditional on them; it was added afterwards to test the design property directly, and we label it exploratory for that reason.

**Time is explained by iterations.** Signing time tracks the iteration count closely: across the twenty per-key means, the correlation between mean iteration count and mean signing time is $r = 0.958$, the per-iteration-count medians are monotone, and a linear fit gives about 52,235 ns per iteration. The per-key mean signing time spans 345,211 to 349,898 ns, a spread of 1.35%: this is the signal a first-order test flags, and it is the per-key iteration spread of the previous paragraph multiplied through by the per-iteration cost. The pre-registered question is whether anything survives once iterations, and the full work profile, are conditioned out.

**The key term is absent, in every campaign.** Table 1 gives the pre-registered statistics for all three campaigns. In the ANCOVA, the key block explains 0.0002% of residual variance in discovery, and 0.0001% and 0.0002% in the placement control and replication, four to five orders of magnitude below the 1% bar. This alone makes Outcome A terminal at the first step of the decision tree. The within-stratum test agrees: across the roughly three thousand (key-pair × stratum) comparisons in each campaign, *zero* pass the pre-registered "present" thresholds, and the largest effect size seen anywhere is a Cliff's $\delta$ of 0.099, below the 0.147 floor.

**The residual is at the noise floor, and its ranking barely reproduces.** A small per-key spread remains in the purest stratum, where the algorithm is held completely fixed and even the early-exit scan count is constant: 448 ns (0.351%) in discovery, 283 ns in the placement control, and 171 ns in the fresh-key replication (Table 1). Two further measurements, added after the pre-registered tree resolved, say what that spread is. First, its size. The standard error of a per-key

Table 1: Pre-registered discrimination statistics, three campaigns of 1,200,000 signatures. The ANCOVA key term is the residual variance explained by key after conditioning on iterations; "present" counts cross-key tests passing Holm $p < 0.05$ *and* $|t| > 4.5$ *and* $|\delta| \geq 0.147$. The pure stratum is `1:0-0-0-0-1` (accept on the first iteration, with the norm-scan count constant, so the early-exit branch contributes nothing). The standard error of a per-key mean in the pure stratum is about 77 ns, so twenty pure-noise means would span about 290 ns.

| | **Discovery** | **Placement control** | **Replication** |
|---|---|---|---|
| Keys / corpus | 100–119 / A | 100–119 / A (shuffled) | 200–219 / B |
| ANCOVA key variance | 0.0002% | 0.0001% | 0.0002% |
| Cross-key tests "present" | 0/2986 | 0/2986 | 0/2916 |
| Max \|Cliff $\delta$\| | 0.099 | 0.093 | 0.084 |
| Pure-stratum per-key spread | 448 ns (0.351%) | 283 ns (0.247%) | 171 ns (0.150%) |

cropped mean in this stratum is about 77 ns, so twenty means of pure noise would span about 290 ns; the placement control sits exactly there, and discovery sits above it by a factor of one and a half. A one-way analysis of variance of the cropped time by key shows that each run carries small key-locked offsets beyond sampling noise ($F = 2.7$ and 4.3, explaining 0.02–0.04% of variance), which is what a signer instance parked in one place for a whole run would produce. Second, its reproducibility. A value-dependent channel would rank the keys the same way in every run; a placement artifact would not. Across the re-allocation the per-key ranking reproduces only marginally (Spearman $\rho = 0.49$, Kendall $\tau = 0.34$ over twenty keys, at the edge of the five-percent level), and within the discovery run itself it barely reproduces at all (split-half $\rho = 0.16$ to 0.22). The one quantity that varies within this stratum, the SampleInBall byte consumption, differs by 0.09 bytes across keys and accounts for under 2 ns. The residual therefore behaves as run-specific placement and drift at the level of a few standard errors. The data bound any key-linked component to a small fraction of 0.35% of signing time, far below every pre-registered threshold; they do not exclude one at that scale, and Section 6 says so. No cross-key comparison in the pure stratum survives the thresholds in any campaign.

**A two-key first-order test flags the harness, not the key.** The signal that first prompted this study was a two-key test: key A versus key B over random messages in a two-class `dudect`-style harness. At 250,000 signatures it reports leakage, with a Welch $t$ of $-4.9$ on a 1.4% difference in mean time (338,183 versus 342,888 ns) and Cohen's $d = 0.02$. The two keys' iteration distributions are identical (means 5.146 and 5.146 over 60,000 messages each), so this difference is not iteration sampling. The same-key control resolves it: with key A in *both* classes, as two signer instances, the harness reports leakage just as confidently, a Welch $t$ of 4.3, a Mann–Whitney $z$ of 14.1, and a cropped-$t$ maximum of 22.6 on a 1.1% class difference with $d = 0.017$. Swapping the allocation order (key B in class 0) gives $d = 0.006$, and a repeat of key A versus key B gives $d = 0.007$, with the sign of the class difference changing from run to run. The two-instance harness therefore carries a run-specific class asymmetry of roughly 0.4–1.1% that is unrelated to the key, presumably the placement of the two signer instances, and at a quarter of a million samples per run a first-order test turns that asymmetry into a confident verdict. Every one of these differences sits far below the pre-registered effect-size floor of 0.147, which is why the protocol treats them as negligible by construction; the control shows that they are also not about the key.

**The rig is validated.** The four controls confirm the harness measures what it claims. The non-constant-time positive control leaks loudly ($|t| = 7745$); the constant-time negative control is

Table 2: The frozen discrimination statistics replicated across the ML-DSA family. The protocol, thresholds, and decision tree are identical to Table 1; only the parameter set differs. Each campaign is 1,200,000 signatures. "Iteration ANOVA" is the one-way $F$ of iteration count by key (the design-property check; 5% critical ≈1.59); "ANCOVA key" is the residual time variance explained by key after conditioning on iterations; "present" counts cross-key tests passing all three pre-registered thresholds, out of roughly 2,600 to 3,000 (key-pair×stratum) tests per campaign; $\rho$ is the Spearman rank persistence of the pure-stratum per-key timing between discovery and the placement control.

| | **ML-DSA-44** | **ML-DSA-65** | **ML-DSA-87** |
|---|---|---|---|
| Mean iterations (design) | 4.37 (4.25) | 5.13 (5.1) | 3.91 (3.85) |
| Iteration ANOVA $F$ (disc./rep.) | 4.49/1.92 | 1.20/1.14 | 1.34/1.74 |
| ANCOVA key, discovery | 0.0001% | 0.0002% | 0.0002% |
| ANCOVA key, placement | 0.0002% | 0.0001% | 0.0000% |
| ANCOVA key, replication | 0.0002% | 0.0002% | 0.0001% |
| Present (disc./place./rep.) | 0/2/0 | 0/0/0 | 0/0/0 |
| Max $\|\delta\|$ (d./p./r.) | .097/.152/.091 | .099/.093/.084 | .084/.090/.084 |
| Pure spread, disc. | 108 ns | 448 ns | 331 ns |
| Rank persistence $\rho$ | 0.48 | 0.49 | 0.48 |

clean ($|t| = 0.78$); ML-DSA signing a *fixed* message repeatedly is clean ($|t| = 1.40$), because the same message drives the same iterations and hence the same time; and ML-DSA signing a fixed versus a random message leaks ($|t| = 201$), which is precisely the message-driven iteration dependence, detected where it should be. A rig that catches a real leak, stays silent on constant-time code, and localizes ML-DSA's timing dependence to the message rather than to hidden state is one whose null on the key question can be trusted.

### 4.1 Replication across the ML-DSA family

To test whether the result is specific to ML-DSA-65, we applied the identical protocol, unchanged, to the other two standardized parameter sets, ML-DSA-44 and ML-DSA-87. Only the parameter argument to the key generator differs; the drivers, the instrumentation, the frozen thresholds, and the decision tree are the same code. The thresholds were pre-registered for ML-DSA-65 and reused without modification, so this is a replication of the decision rule on new parameter sets, not an independent pre-registration. Each set received the full battery: a discovery campaign, a placement control, and a fresh-key replication of 1,200,000 signatures each. Table 2 collects the frozen statistics alongside the ML-DSA-65 column from Table 1.

Three things hold across the family. First, the design property reproduces: the measured mean iteration count matches the Dilithium specification's expected repetitions at every level. Second, and decisively, the ANCOVA key term is negligible in every campaign at every parameter set, explaining between 0.0000% and 0.0002% of residual variance after conditioning on iterations, four to five orders of magnitude below the 1% bar, so Outcome A is terminal at the first step of the decision tree throughout. The implementation carries no key-dependent timing channel at any of the three standardized parameter sets.

Two observations refine that conclusion, and both are what the protocol was built to surface. The within-stratum test returns *zero* cross-key comparisons above the pre-registered thresholds in every discovery and every replication campaign; the one exception is the ML-DSA-44 placement control, where two of 2,850 comparisons cross the effect-size floor (maximum $|\delta| = 0.152$). Every one of them involves a single key that is slow only in that run, and none reproduces in the -44

discovery or the fresh-key replication (maximum $|\delta| = 0.097$ and 0.091). The decision rule requires a surviving residual to replicate with the same sign and comparable magnitude before it counts; these do not, so the rule classifies them as placement artifacts. This is exactly the role the placement control and the replication requirement were pre-registered to play: they catch a borderline per-key spread produced by one run's memory placement and stop it from being read as a channel.

The second observation is about the algorithm, not the implementation. At ML-DSA-65 and -87 the per-key iteration spread is at or near the sampling-noise floor (one-way ANOVA $F = 1.20$ and 1.34 in discovery, 1.14 and 1.74 on fresh keys), reproducing the key-independence of the rejection rate that the design intends. At ML-DSA-44 the same test is significant ($F = 4.49$ in discovery, $F = 1.92$ on fresh keys), so the per-key spread there exceeds pure sampling noise; the effect is real but negligible in magnitude, the key explaining 0.007% of the variance in iteration count. It is a property of the scheme at its smallest parameter set, present in the deterministic work profile that any conformant implementation reproduces, and it induces no implementation timing channel: the key-to-time relation it creates runs entirely through the iteration count, which is Outcome A, and the within-stratum test that conditions on the exact work profile stays clean. Whether the residual reflects the heuristic, rather than exact, key-independence of the low-order norm check at the smallest dimensions is a question for the scheme's analysis, not for this implementation.

### 4.2 Hedged signing

The study so far uses deterministic signing, where a fixed (key, message) pair yields a fixed work profile and a fixed signature. ML-DSA also defines a hedged mode that draws fresh randomness per signature; the mask seed then depends on that randomness, so the rejection-loop iteration count becomes an i.i.d. draw independent of both key and message, and no deterministic work profile exists to join timing against. The key question is then tested directly on timing, by a one-way ANOVA of per-signature signing time by key. We ran the discovery and placement campaigns for ML-DSA-65 in hedged mode (1,200,000 signatures each), on the stock jar with the tracer off the classpath.

The mean signing time does not depend on the key. The ANOVA gives $F(19, 1{,}175{,}980) = 0.52$ in discovery and $F = 0.89$ in the placement control, both below the 1.59 critical value, with the key explaining under 0.002% of variance. The per-key mean-time spread is larger than under deterministic signing (0.83% against the 0.35% of the deterministic pure stratum), because hedging adds per-signature iteration variance to every key; but it is sampling noise, sitting below the $3.2\,\mu$s span expected of twenty noise means, and its per-key ranking does not persist across the placement control (Spearman $\rho = 0.22$, below the significance threshold). Hedged signing carries no key-dependent timing channel either. This test has less resolution than the deterministic one, because randomizing the iteration count per signature inflates the within-key variance, so a per-key effect below about 0.8% of the mean would not be resolved here; but the deterministic analysis already bounds the implementation channel far more tightly by conditioning on the exact work, and hedging only adds key-independent randomness on top of it.

## 5 Discussion

**What the result means, and does not.** Taken together, the three findings leave no detectable key-dependent timing channel in Bouncy Castle ML-DSA-65 signing. The iteration count does not depend on the key beyond sampling variation, which is the design property holding in practice; the time per unit of work does not depend on the key beyond the platform's noise floor, which is the implementation being clean; and the apparent 1.35% per-key spread that prompted the study is

the product of the two, twenty keys drawing twenty samples of one distribution. No change to the implementation is warranted on the strength of this signal. The statement is about a key-dependent *signing-time* channel under a black-box timing model on this platform; it is not a claim about cache, power, or electromagnetic channels, which remain the live attack surface for Dilithium [12, 13], nor about other providers, and it does not certify the absence of every conceivable micro-architectural effect. It does establish, to the resolution of a million-plus-signature campaign, that the visible signal is neither the algorithm leaking nor the code leaking.

**The one value-dependent branch.** The norm check `Poly.checkNorm` genuinely early-exits on the first out-of-bound coefficient, so it is not constant-time, and it was the leading a-priori candidate for a residual. Its measured contribution, isolated within strata where the scan count varies, is on the order of a nanosecond of per-key timing spread, several orders of magnitude below the tens of microseconds a single loop iteration costs and far beneath the platform's own run-to-run drift. It is not a usable channel. A branchless norm check, one that scans all coefficients or accumulates the out-of-bound condition without an early return, would drive even this to zero, and is worth having as defense in depth for a library that may later run on quieter or non-JVM platforms; but the data do not make it a correctness or security fix.

**Responsible disclosure.** We reported the measured signal and the analysis to the Bouncy Castle maintainers. The maintainer's assessment matched the finding: alert but not alarmed, with an interest in driving even the nanosecond-scale norm-check contribution to zero as a hardening measure. Because the study concludes there is no channel and identifies nothing exploitable, there is no vulnerability to embargo; the value of the disclosure is the confirmation that a careful, pre-registered look at a plausible-looking timing signal found the implementation clean.

**A caution for side-channel practice.** The reusable part of this work is the method, and its most transferable lesson is a negative one. Rejection-sampling schemes have a legitimately variable running time, so a fixed-versus-random-key test over a finite corpus compares two samples of a key-independent count and reports their difference as leakage; that report is true of the sample and says nothing about the key. Worse, on a managed runtime the two-class harness itself contributes a class asymmetry of order 1% from instance placement, which the same test also reports as leakage, so a same-key negative control, the secret in both classes, is a precondition for reading any two-key verdict at all. The observation becomes decidable only by conditioning: record the work each signature performed, test timing within identical work, and test the work itself for a key effect with an analysis that accounts for sampling variation. Pre-registering the thresholds removes the freedom to read a preferred conclusion out of a noisy residual, and the placement control distinguishes a memory-layout artifact from a value-dependent channel, a confound specific to managed runtimes that a native constant-time study never faces. The same ingredients apply to any Fiat–Shamir-with-aborts scheme, and to the variable-time paths of hash-based signing, on any managed runtime.

## 6 Threats to Validity

*Timing model and platform.* The study measures wall-clock signing time on one aarch64 host under a black-box model; it does not address cache, power, or electromagnetic channels, and absolute magnitudes are host-specific. The defended claim is the *relative* one, that no cross-key effect survives conditioning on the work profile. *Managed-runtime drift.* JVM thermal and scheduler drift set

the noise floor; we mitigate it with a fixed heap, single-process runs on an actively cooled host, and, decisively, the placement control and replication, which convert drift from a confound into an observable whose behavior distinguishes it from a channel. *Power of the key-effect test.* The analysis of variance on iteration count was added after the pre-registered tree resolved and is reported as exploratory. With twenty keys and a per-key standard error of 0.019 iterations, a true key effect would need to shift per-key means by roughly the observed noise range, about 1% of the mean, to be detected; a far smaller real dependence, for instance through the heuristically argued $r_0$ check or the rare $t_0$-dependent checks, is bounded by these data but not excluded. The twenty keys share one message corpus; because the masking seed in deterministic signing is derived from the key as well as the message, a message has no common "difficulty" across keys, so the shared corpus induces no correlation between per-key means, and any residual message effect would only inflate the within-key variance and make the test conservative. *Run-to-run level shifts.* The absolute mean signing time differs substantially between separate two-key runs (280 to 377 $\mu$s), reflecting the host's thermal and scheduler state; each two-key verdict compares its two interleaved classes within one run, so these shifts do not create a class difference, but they are a reminder that absolute timings from this platform are not comparable across runs. *Scope.* Results cover Bouncy Castle's ML-DSA-44, -65, and -87 under deterministic signing (Section 4.1) and ML-DSA-65 under hedged signing (Section 4.2); other providers and other schemes are out of scope, though the protocol transfers. *Resolution.* A null bounds, rather than eliminates, the possibility of a channel below the campaign's resolution; the pre-registered effect-size floor states exactly how small an effect the study treats as negligible. In the purest stratum the data bound any key-linked residual to a small fraction of 0.35% of signing time; a ranking that reproduces only marginally across re-allocation is consistent with placement and drift, but does not exclude an effect at the 0.1% scale.

# 7 Conclusion

ML-DSA's rejection loop makes its signing time legitimately variable, and that turns any timing measurement into a discrimination problem. We met an apparent key-dependent signal in Bouncy Castle ML-DSA-65, a 1.35% spread in mean signing time across twenty keys, and resolved it with a pre-registered protocol: instrument the loop to record each signature's exact work, test for a cross-key timing effect within identical work under thresholds fixed in advance, and add a placement control and a fresh-key replication. Across three campaigns of 1,200,000 signatures, signing time is explained by iteration count; the per-key iteration spread is indistinguishable from sampling noise, confirming at scale the design property that ML-DSA's rejection rate does not depend on the key; and after conditioning on work, the key term explains 0.0002% of variance, nothing passes the pre-registered effect-size bar, and the residual sits at the noise floor of the per-key means with a ranking that barely reproduces. There is no detectable channel; the implementation is clean at the campaign's resolution; the one non-constant-time branch contributes about a nanosecond and is a defense-in-depth item, not a fix. Applied unchanged to ML-DSA-44 and ML-DSA-87, the protocol returns the same verdict at every standardized parameter set, and its placement control and replication requirement are seen classifying a borderline per-key spread at ML-DSA-44 as an artifact rather than a channel. The protocol, and the caution that fixed-versus-random-key tests will flag sampling variation on any rejection-sampling scheme, are what let this study earn a negative result instead of asserting one.

## Declarations

**Funding.** No funding was received for this work.

**Competing interests.** The author has no competing interests. The author reported the measured signal and the analysis of Section 4 to the Bouncy Castle maintainers, as described in Section 5, and has no affiliation with the project.

**Data and code availability.** The instrumentation, the timing harness, the frozen pre-registration, the analysis scripts, the gzipped raw per-signature data for all campaigns (including the ML-DSA-44, -87, and hedged-mode batteries), the rig controls, the two-key control runs, and the rank-persistence and family analyses are publicly available at https://github.com/Arpan0995/pqc-jvm-sidechannel (under `results/bc185/`; commit `e0f8702`).

**Ethics approval.** Not applicable; no human or animal subjects.

## References

[1] L. Groot Bruinderink, A. Hülsing, T. Lange, and Y. Yarom, "Flush, Gauss, and Reload: A cache attack on the BLISS lattice-based signature scheme," in *Cryptographic Hardware and Embedded Systems, CHES 2016*, ser. LNCS, vol. 9813, Springer, 2016, pp. 323–345.

[2] T. Espitau, P.-A. Fouque, B. Gérard, and M. Tibouchi, "Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1857–1874.

[3] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard," FIPS 204, August 2024. https://doi.org/10.6028/NIST.FIPS.204

[4] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS–Dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 1, pp. 238–268, 2018.

[5] V. Lyubashevsky, "Fiat–Shamir with aborts: Applications to lattice and factoring-based signatures," in *Advances in Cryptology, ASIACRYPT 2009*, ser. LNCS, vol. 5912, Springer, 2009, pp. 598–616.

[6] V. Lyubashevsky, "Lattice signatures without trapdoors," in *Advances in Cryptology, EUROCRYPT 2012*, ser. LNCS, vol. 7237, Springer, 2012, pp. 738–755.

[7] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS–Dilithium: Algorithm specifications and supporting documentation (version 3.1)," NIST Post-Quantum Cryptography Standardization, Round 3 submission, February 2021. https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf

[8] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST Non-Invasive Attack Testing Workshop (NIAT)*, 2011.

[9] O. Reparaz, J. Balasch, and I. Verbauwhede, "Dude, is my code constant time?" in *Proc. Design, Automation & Test in Europe (DATE)*, 2017, pp. 1697–1702.

[10] Legion of the Bouncy Castle, "Bouncy Castle cryptographic APIs for Java." https://www.bouncycastle.org/

[11] P. Ravi, M. P. Jhanwar, J. Howe, A. Chattopadhyay, and S. Bhasin, "Exploiting determinism in lattice-based signatures: Practical fault attacks on pqm4 implementations of NIST candidates," in *Proc. ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2019, pp. 427–440.

[12] S. Marzougui, V. Ulitzsch, M. Tibouchi, and J.-P. Seifert, "Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all," IACR ePrint 2022/106, 2022. https://eprint.iacr.org/2022/106

[13] A. Berzati, A. Calle Viera, M. Chartouny, S. Madec, D. Vergnaud, and D. Vigilant, "Exploiting intermediate value leakage in Dilithium: A template-based approach," IACR ePrint 2023/050, 2023. https://eprint.iacr.org/2023/050

[14] V. Migliore, B. Gérard, M. Tibouchi, and P.-A. Fouque, "Masking Dilithium: Efficient implementation and side-channel evaluation," in *Applied Cryptography and Network Security, ACNS 2019*, ser. LNCS, vol. 11464, Springer, 2019, pp. 344–362.

[15] Y. Zhou, W. Wang, Y. Sun, and Y. Yu, "Rejected signatures' challenges pose new challenges: Key recovery of CRYSTALS-Dilithium via side-channel attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2025, no. 4, pp. 817–847, 2025. https://doi.org/10.46586/tches.v2025.i4.817-847

[16] Z. Liu, A. Wang, C. Wei, Y. Ding, J. Zhang, A. Liu, and L. Zhu, "Release the power of rejected signatures: An efficient side-channel attack on the ML-DSA cryptosystem," IACR ePrint 2025/582, 2025. https://eprint.iacr.org/2025/582

[17] O. Adjonyo, S. Bardin, E. Bellini, G. N. Dione, M. F. Al Ameen, R. Merget, F. Recoules, and Y. Sellami, "Systematic timing leakage analysis of NIST PQDSS candidates: Tooling and lessons learned," in *Sixth NIST PQC Standardization Conference*, September 2025. arXiv:2509.04010.

[18] A. Sharma, "pqc-jvm-sidechannel: a pre-registered timing discrimination for Java ML-DSA." GitHub repository, 2026. [Online]. Available: https://github.com/Arpan0995/pqc-jvm-sidechannel