Incomplete Ciphertext Comparison in ML-KEM: From an IND-CCA2 Break to Key Recovery

Bhabani Sankar Das

007bsd@proton.me

Abstract. ML-KEM is IND-CCA2 secure only because of one check inside decapsulation: the receiver re-encrypts the message it recovered and returns the true shared secret only if the result matches the received ciphertext exactly. This is the Fujisaki–Okamoto (FO) check. wolfSSL implemented it in hand-written SIMD assembly, and on two backends it compared *fewer than all* of the ciphertext bytes. The x86-64 AVX2 path compared 1536 of 1568 bytes; the ARM64 NEON path compared roughly half.

These bugs were documented as a weakening of IND-CCA2 security, in that a tampered ciphertext can slip past the check. We show they are worse than that. The bytes the check skips carry the tail of the decryption noise, and that noise is an exact linear function of the secret key. An attacker who varies those unchecked bytes and watches the decapsulation output reads the noise off one coordinate at a time. Stacking the measurements gives an overdetermined linear system in the secret, which we solve by ordinary least squares with no lattice reduction.

The measurement is a plaintext-checking oracle, the same primitive that key-mismatch attacks use. What is new is where it comes from. Reading it off the unchecked v-tail, rather than from chosen sparse-u ciphertexts, means it survives even when u is fully validated, as on AVX2, so the standard "validate all of u" hardening does not close it. The price is queries, 10^5 to 10^6 against a few thousand for key-mismatch, so the contribution is reach rather than efficiency.

We recover most of the ML-KEM-1024 private key end-to-end against the shipped binaries on both backends: 98.0 % of the 2048 secret coefficients at 400 ciphertexts on AVX2, and 98.5 % at 600 on NEON, reaching the full key with more ciphertexts (the verified reference model recovers all 2048 at about 1300 ciphertexts). The cost appears to track the *geometry* of which bytes go unchecked more than their number: NEON leaves about 2.5× more coordinates unchecked than AVX2 yet needs more ciphertexts. We conclude that an incomplete FO comparison is a key-recovery vulnerability, and should be triaged as one.

Keywords: ML-KEM · Kyber · Fujisaki–Okamoto transform · implicit rejection · chosen-ciphertext attack · plaintext-checking oracle · key recovery · implementation attack

1 Introduction

An incomplete ciphertext comparison in ML-KEM's decapsulation looks like a minor robustness flaw. It is a private-key recovery. This paper establishes that the incomplete-comparison bug class is key recovery rather than the distinguishing break it has been treated as, and demonstrates the recovery against shipped code.

ML-KEM [Nat24] is IND-CCA2 secure only because of one check inside decapsulation. The receiver decrypts the incoming ciphertext with the IND-CPA scheme, re-encrypts the recovered message with the same randomness, and returns the true shared secret only if the re-encryption reproduces the ciphertext byte for byte. Any mismatch triggers implicit

rejection and a pseudorandom secret is returned instead. This is the Fujisaki–Okamoto (FO) transform [FO99, HHK17], and the comparison is the whole of it in practice.

wolfSSL [wol26] implemented that comparison in hand-written SIMD assembly, and on two backends the comparison covered only part of the ciphertext. On x86-64 AVX2, `mlkem_cmp_avx2` compared 1536 of the 1568 bytes of an ML-KEM-1024 ciphertext (CVE-2026-10097 [MIT26a]). On ARM64 NEON, the comparison covered roughly half (CVE-2026-6330 [MIT26b], reported independently by Carlini [Car26]). Both were documented as weakening IND-CCA2 security: a tampered ciphertext can pass implicit rejection.

We show the consequence is stronger. The bytes the check skips carry the compressed tail of the decryption noise, and that noise is an exact linear function of the secret key. An attacker who varies those unchecked bytes and observes the decapsulation output measures the noise directly, one coordinate at a time, then recovers the key by ordinary least squares. We demonstrate ML-KEM-1024 key recovery end-to-end against the shipped vulnerable binaries on both backends, and full recovery in the verified reference model.

Contributions.

- An incomplete FO comparison is a *direct, per-coordinate oracle* for the decryption noise. This is neither the rare-failure regime of decryption-failure attacks [DGJ$^+$19] nor the chosen-sparse-u regime of plaintext-checking and key-mismatch attacks [RRCB20, QCZ$^+$21, BDHD$^+$19], which the AVX2 bug rejects.
- A severity reclassification: this bug class is *key recovery*, not merely a weakening of IND-CCA2 security, and should be triaged accordingly.
- A leakage-geometry result: on the two geometries we examine, recovery cost tracks *which* bytes are unchecked more than how many. NEON leaves about 2.5× more coordinates unchecked than AVX2 yet needs more ciphertexts.
- End-to-end key recovery against two shipped SIMD backends (AVX2 and NEON) of one library.

2 Background

We assume familiarity with ML-KEM [Nat24], the standardized form of Kyber [BDK$^+$18], and describe only what the attack needs.

ML-KEM-1024 and the FO check. The IND-CPA scheme works over $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$ with $q = 3329$. The secret key is a short vector $\mathbf{s} \in R_q^4$; the public key is $(\mathbf{A}, \mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e})$ with $\mathbf{e}$ a short error, both sampled from a centered binomial distribution CBD. Encryption of a message m samples randomness $\mathbf{r}$ and produces a ciphertext $(\mathbf{u}, v)$; decryption recovers m' by rounding $v - \mathbf{s}^\top \mathbf{u}$. Decapsulation wraps this in the FO transform: it decrypts m', re-encrypts m' with derived randomness to obtain $(\mathbf{u}', v')$, and returns the real shared secret only if $(\mathbf{u}', v')$ equals the received ciphertext; otherwise it returns a pseudorandom value (implicit rejection).

Ciphertext layout. An ML-KEM-1024 ciphertext is 1568 bytes: $c_1 = \mathsf{Compress}_{11}(\mathbf{u})$ occupies 1408 bytes and $c_2 = \mathsf{Compress}_5(v)$ occupies 160 bytes. The compressed v-tail is what our attack exploits on both backends (it is entirely unchecked by the AVX2 bug and partly unchecked by the NEON bug). Compression is lossy; we write c_u and c_v for the compression errors, which are public once the (attacker-chosen) ciphertext is fixed.

Threat model. We assume a *static* (reused) ML-KEM key and a plaintext-checking oracle: for each submitted ciphertext the attacker learns one bit, the decoded message bit m'_j that the returned shared secret is derived from. Under implicit rejection the decapsulator always returns *some* secret, so this bit is not the real-versus-reject distinction directly; it is recovered from any downstream use of the derived key (a MAC or AEAD verification, a decrypt-and-respond step). Our experiments read the bit from a *constructed* oracle, the pre-fix build queried against an exported keypair; that a fielded HPKE [BBLW22] or KEMTLS [SSW20] deployment exposes the same bit through its AEAD path we argue in Section 4.2, we do not demonstrate it. The setting is the standard one for key reuse: HPKE static recipients, KEMTLS long-term keys, and pinned or embedded keys. Ephemeral hybrid TLS 1.3 shares use a fresh key per handshake and are not key-recoverable; there the bug is only a distinguishing break.

3 The Vulnerability

Both defects are in wolfSSL's constant-time ciphertext comparison, reached on every decapsulation. We confirmed the exact unchecked byte ranges by sweeping single-byte mutations across all 1568 positions of a re-encrypted ciphertext and recording which positions failed to trigger implicit rejection.

3.1 AVX2 (x86-64): 1536 of 1568 bytes

`mlkem_cmp_avx2` compares only the first 1536 bytes, so the final 32 are never checked. Those bytes are the tail of c_2: about 51 compressed v-coefficients, at indices ≈ 205–255. A ciphertext that differs from the expected re-encryption only in bytes 1536 to 1567 is therefore accepted and the real shared secret returned, while all of c_1 and the first 128 bytes of c_2 are still compared.

A single-byte position sweep against the pre-fix build confirms this exactly: a mutation is accepted only when it lands in one of the 32 tail positions (bytes 1536 to 1567), and rejected everywhere else. The same fix (PR #10430) also corrects an unrelated defect, an incorrect 5-bit decompression of v. We treat the two independently and use only the comparison defect; because we measure against the live binary, that decompression behaviour is part of the oracle we model, not a discrepancy from it.

3.2 NEON (ARM64): roughly half

In the NEON comparison, `ins v9.b[0], v8.b[1]` was used where `ext v9.16b, v8.16b, v8.16b, #8` was intended, discarding the high 64 bits of the 128-bit comparison accumulator. After the `ld4` deinterleaving this leaves bytes 32 to 63 of each 64-byte chunk unchecked: 784 of 1568 bytes, about half. The position sweep is the ground truth: it flags exactly 125 unchecked v-coefficients, in three bands $[52, 101] \cup [154, 203] \cup [231, 255]$. The byte-chunk account above predicts this same pattern from the instruction semantics, and the sweep confirms it.

4 From Incomplete Comparison to Key Recovery

The classical key-recovery attacks on ML-KEM submit chosen sparse-u ciphertexts, isolating one secret coefficient per query (Section 8). On AVX2 that route is closed: the bug validates all of $\mathbf{u}$, and flipping a single bit of $\mathbf{u}$ on the live binary is rejected every time. The NEON bug is broader and leaves about half of $\mathbf{u}$ unchecked, so that closure is AVX2-specific, but our attack does not depend on it either way. On both backends we read the key off the unchecked v-tail instead. The attacker honestly encrypts a message of its choice,

which fixes $\mathbf{r}, \mathbf{e}_1, e_2$ and hence the compression errors c_u, c_v, and then modifies only the unchecked tail bytes.

4.1 The decryption-noise identity

At decapsulation the received ciphertext is decompressed to $(\mathbf{u}', v')$ and the message is recovered coordinatewise as $m'_j = \mathsf{Compress}_1\big(v'_j - (\mathbf{s}^\top \mathbf{u}')_j\big)$. The signed distance of the argument from the $\mathsf{Compress}_1$ boundary is the decryption noise, and it is an exact linear function of the secret.

Lemma 1. *Let $(\mathbf{u}, v)$ encrypt a message m with randomness $(\mathbf{r}, \mathbf{e}_1, e_2)$, and let $(\mathbf{u}', v') = (\mathbf{u} + c_u,\, v + c_v)$ be its decompression at decapsulation, where c_u, c_v are the compression errors. Then*

$$v'_j - (\mathbf{s}^\top \mathbf{u}')_j \;=\; \mathsf{Decompress}_1(m)_j + \delta_j, \qquad \delta_j = \big(\mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top(\mathbf{e}_1 + c_u) + e_2 + c_v\big)_j,$$

and δ_j is an exact integer linear form in $(\mathbf{s}, \mathbf{e})$.

Once the attacker-chosen ciphertext is fixed, the randomness $\mathbf{r}, \mathbf{e}_1, e_2$ and the compression errors c_u, c_v are all fixed and known, so the 2048 unknowns enter linearly: $e_2 + c_v$ is the constant term, $\mathbf{r}$ multiplies $\mathbf{e}$, and $-(\mathbf{e}_1 + c_u)$ multiplies $\mathbf{s}$. The secret, the randomness, and the compression errors are all small, so δ_j stays below $q/4$ except with the negligible ML-KEM decryption-failure probability. Its balanced representative therefore never wraps modulo q, and the equality holds over $\mathbb{Z}$. We verified the identity to maximum error 0 over random keys and ciphertexts against the kyber-py reference implementation [Pop24]; the derivation is in Section A.

4.2 Measuring the noise

By Lemma 1, $m'_j = \mathsf{Compress}_1(\mathsf{Decompress}_1(m)_j + \delta_j)$, so the decoded bit is fixed by which side of the $\mathsf{Compress}_1$ boundary δ_j falls on. Because the v-tail bytes are not compared, the attacker can set the compressed value of v_j to any of its $2^{d_v} = 32$ levels without triggering implicit rejection. This adds a known offset Δ_j to v'_j, making the decoded bit $\mathsf{Compress}_1(\mathsf{Decompress}_1(m)_j + \delta_j + \Delta_j)$. Sweeping Δ_j carries the argument once around $\mathbb{Z}_q$, so m'_j flips twice, at the two $\mathsf{Compress}_1$ boundaries ($q/4$ and $3q/4$); the transition nearest the known nominal $\mathsf{Decompress}_1(m)_j$ locates the boundary and measures δ_j to the compression granularity, about $\pm q/64$.

The observable. The per-query signal is one bit: which side of the $\mathsf{Compress}_1$ boundary δ_j falls on, i.e. the decoded value m'_j. This is a *plaintext-checking* oracle, the same primitive Ravi et al. [RRCB20] obtain by side channel; what is new here is how it is exposed (through the unchecked v-tail rather than chosen sparse-u ciphertexts), not the oracle itself. Reading it needs no decapsulation-internal information: under implicit rejection the decapsulator always returns *some* secret, and an attacker recovers the bit from any downstream use of the derived key (a MAC or AEAD verification, a decrypt-and-respond step).

Query cost. Locating one boundary is a search over the 32 tail levels: about 5 decapsulation queries by binary search, or 32 by a linear sweep. Each ciphertext supplies one equation per unchecked coordinate, so recovery costs (ciphertexts) × (unchecked coordinates) × (boundary queries), on the order of 10^5 to 10^6 decapsulations for the counts we run (Section 5).

Table 1: Fraction of the 2048-coefficient secret $(\mathbf{s}, \mathbf{e})$ recovered against the live binaries (wolfSSL commit `d278da09d`), by ciphertext count. AVX2 is native x86-64; NEON is ARM64 under QEMU. Dashes mark checkpoints outside that backend's committed grid: the two runs used different checkpoint grids, and NEON under QEMU was too slow to re-run at the AVX2 grid.

Ciphertexts	AVX2 (native) %	NEON (QEMU) %
100	65.5	68.9
200	87.8	85.2
300	–	94.1
350	96.9	–
400	98.0	96.0
500	–	97.2
600	–	98.5

4.3 Recovering the key by regression

Each unchecked coordinate contributes one linearly independent equation per ciphertext (51 on AVX2), so stacking the measurements gives a system

$$\mathbf{M}\mathbf{x} = \boldsymbol{\delta} + \boldsymbol{\varepsilon}$$

in the 2048 unknowns $(\mathbf{s}, \mathbf{e})$. The design matrix $\mathbf{M}$ has rows built from the known $\mathbf{r}$ and $-(\mathbf{e}_1 + c_u)$, $\boldsymbol{\delta}$ is the exact decryption noise of Lemma 1, and $\boldsymbol{\varepsilon}$ is the $\pm q/64$ measurement error. The system reaches full rank at about 45 ciphertexts, matching the equation-limited floor $\lceil 2048/51 \rceil = 41$; a control fed the exact inner product recovers all of $\mathbf{s}$ by 50 (Section 5). We solve by ordinary least squares and round to the CBD support $\{-2, \ldots, 2\}$. Clamping to that support buys little over plain rounding, because the residual error sits in noise-dominated coordinates.

The recovery is lattice-free by design. An LWE instance is a linear system that only the error term makes hard [Sch23]; because the oracle measures that error to the compression granularity, subtracting the measured value leaves a low-noise system that ordinary least squares solves directly, with no shortest-vector step. The cost of that choice is a short tail of noise-dominated coordinates that plain rounding leaves wrong. Two paths close it. The first is simply more ciphertexts; in the verified reference model this reaches all 2048 coefficients. The second is faster and stays lattice-free: public-key error correction, a greedy search over the uncertain $\mathbf{s}$-coordinates against the constraint $\mathbf{e} = \mathbf{t} - \mathbf{A}\mathbf{s}$ in the CBD support, which has a known zero-loss optimum. Neither path uses lattice reduction.

Two regimes separate as a result. At the equation-limited floor the noise-free system is already determined; the number of ciphertexts the *live* binary needs is set instead by measurement noise, and that figure is the one that reflects attack difficulty.

5 Results

We built a decapsulation oracle on the pre-fix wolfSSL build (commit `d278da09d`) that exports its keypair so recovery can be scored against ground truth, and ran the full attack against the shipped SIMD assembly: AVX2 natively on x86-64, NEON on ARM64 under QEMU [Bel05] emulation. Each run performs a self-check, confirming the reconstructed noise matches the algebraic value to the measurement precision, before reporting any recovery figure.

The reported figure is the full $(\mathbf{s}, \mathbf{e})$ recovery; the decryption key $\mathbf{s}$ alone tracks it closely ($1002/1024 = 97.9\,\%$ on AVX2 at 350 ciphertexts; $1007/1024$ on NEON at 600). An

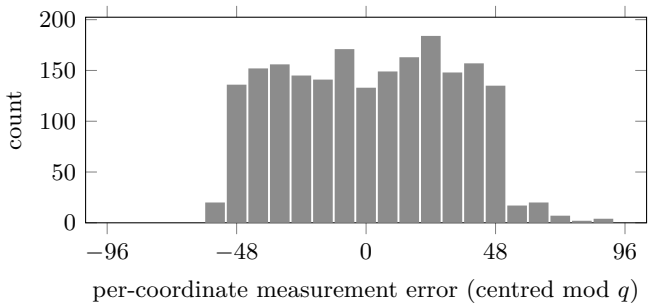

Figure 1: Signed per-coordinate measurement error against the live AVX2 binary (2040 samples): mean +1.05, std 31.2 ($q/64 = 52$). Centred on zero, so the recovery is variance-limited and reaches the full key with more ciphertexts.

exact-measurement control that feeds the true $(\mathbf{s}^\top \mathbf{u})_j$ in place of the swept measurement recovers 1024/1024 of $\mathbf{s}$ at 50 ciphertexts, isolating measurement noise as the only obstacle.

The oracle's exported key is used only to score recovery, never in the attack itself; every run passes a ground-truth self-check before reporting a figure. End to end each attack issues on the order of 10^5 to 10^6 decapsulation queries.

The numbers above are the demonstrated difficulty on the shipped binaries. A compressed reference model (kyber-py) with idealized $\pm q/64$ measurement noise reproduces the same identity and rank floor, and recovers the full secret $(\mathbf{s}, \mathbf{e})$ across independently sampled keys: 2047/2048 coefficients by 700 ciphertexts and all 2048 by about 1300. Its ciphertext counts run optimistic, because the live measurement noise is not the clean i.i.d. $\pm q/64$ of the model.

The live noise is, however, *zero-mean*. Over 2040 per-coordinate measurements against the live AVX2 binary the signed error has mean +1.05 and standard deviation 31.2 (Fig. 1; $q/64 = 52$). So the recovery is variance-limited, not bias-limited: the stranded few percent is averaging noise, not a floor, and it clears with more ciphertexts, as the reference model confirms (all 2048 coefficients). This is what backs the full-key claim on the shipped binary. The live counts in Table 1 are a query-budget snapshot, not a ceiling.

6 Discussion

6.1 Why NEON needs more ciphertexts

NEON leaves about 125 coefficients unchecked, in three non-contiguous bands, against roughly 51 in one contiguous band for AVX2 (about 2.5× more), yet it needs more ciphertexts. Two effects pull against each other. First, NEON's per-coordinate measurement is noisier: each harness self-checks its reconstructed noise against the algebraic truth, and the mean absolute per-coordinate error is about 25 on AVX2 and about 50 on NEON, i.e. close to $\pm q/64$ and $\pm q/32$. Second, the larger NEON leak *lowers* its rank floor ($\lceil 2048/125 \rceil = 17$ against 41), which should help.

A controlled ablation on the reference model separates the two. We run each coordinate set at each precision and read the recovery after 200 ciphertexts (Table 2); the absolute figures are model figures and run optimistic (Section 5), but the comparison across conditions is what we rely on.

Holding precision fixed, the larger NEON leak helps, as its lower floor predicts: at $\pm q/64$, recovery rises from 94.3 to 99.3 %. Holding geometry fixed, halving the precision costs 9 to 17 points. The precision penalty wins. NEON's realistic setting (125 coeffs at $\pm q/32$, 90.3 %) trails AVX2's (51 coeffs at $\pm q/64$, 94.3 %), so NEON needs more

Table 2: Ablation on the reference model: fraction of $(\mathbf{s}, \mathbf{e})$ recovered after 200 ciphertexts, for each leakage geometry at each measurement precision (one shared key; the cross-condition ordering shown is stable across the keys we spot-checked). More coordinates help at fixed precision; halving precision hurts more, so the realistic NEON setting ($\pm q/32$) trails the realistic AVX2 one ($\pm q/64$).

Leakage geometry	$\pm q/64$ %	$\pm q/32$ %
AVX2 (51 coeffs, 1 band)	94.3	77.3
NEON (125 coeffs, 3 bands)	99.3	90.3

ciphertexts, reproducing the live ordering (Table 1). On the two geometries and the keys we examined, then, recovery cost tracks measurement precision rather than the raw number of leaked coordinates; we report this as the pattern the data show, not as a law established over a multi-key sample (Section 6.2).

6.2 Applicability and severity

The attack needs a reused key and a decapsulation oracle, so it does not touch ephemeral hybrid TLS 1.3, where each handshake draws a fresh key and the bug is only a distinguishing break. Static-key deployments are the ones at risk. All 10^5 to 10^6 decapsulations must land on one key before it rotates, so the strongest case is a pinned or embedded key that never rotates, common in wolfSSL's IoT and embedded base; HPKE static recipients and KEMTLS long-term keys are reachable when the key is long-lived, less so under frequent rotation. This recovery capability is why CVE-2026-10097 was rated High (CVSS 4.0 score 8.3, CWE-697), above the distinguishing-only framing.

The result says nothing about correct ML-KEM implementations or the standard itself. It is a logic bug, not a side channel, and it is bounded to implementations that compare fewer than all ciphertext bytes.

Scope of the demonstration. Three limits bound what we show, as opposed to the mechanism, which is exact. First, recovery runs end-to-end on the live binaries to 98 % of the secret; the full 2048/2048 is reached in the verified reference model, and on the binary the closing few percent follow from the zero-mean measurement noise of Fig. 1 rather than a run we carried to completion. Second, the oracle is constructed: it reads the plaintext-checking bit from the pre-fix build against an exported keypair, so leakage of the same bit through a fielded HPKE or KEMTLS AEAD path is argued, not demonstrated. Third, the leakage-geometry comparison of Section 6.1 rests on one key, with the cross-condition ordering spot-checked over a handful of others rather than averaged over a multi-key sample. None of the three touches the noise identity or the rank floor, which are exact; they bound the empirical reach of the demonstration, not the attack it demonstrates.

7 Mitigation and Disclosure

The fix is to compare the entire ciphertext in constant time, including the tail; a comparison routine should be tested on differences confined to its last bytes. Both defects were reported privately, assigned CVEs, and fixed in wolfSSL 5.9.2.[1]

[1] AVX2 comparison in PR #10430 (affecting 5.7.0–5.9.1); NEON comparison in PR #10192 (affecting 5.7.4–5.9.1).

8 Related Work

The closest positive precedent is decryption-failure attacks [DGJ+19]: they exploit the same decryption-noise term, a linear function of the secret, but only through the rare event of that term crossing the decoding boundary and causing a failure, so recovery there is statistical and needs far more ciphertexts. An incomplete comparison instead makes that same term directly measurable, so no failures are needed.

Plaintext-checking and key-mismatch attacks [RRCB20, QCZ+21, BDHD+19] recover the key in a few thousand queries by submitting chosen sparse-u ciphertexts to isolate one coefficient per query. Our attack uses the same plaintext-checking oracle (one bit per query, which side of the rounding boundary the noise falls on) but obtains it from the unchecked v-tail rather than sparse-u. That difference is the point: the AVX2 bug validates all of $\mathbf{u}$ and rejects the sparse-u ciphertexts this family needs, yet the v-tail oracle survives, so the standard "validate all of u" hardening does not close it. We pay for that reach in queries (10^5 to 10^6 against their few thousand), so the contribution is reach, not efficiency.

Key-recovery oracles built from the FO transform's timing have also been demonstrated on FrodoKEM [GJN20]. Implementation-level key recovery in the same spirit as ours includes KyberSlash [B+24] and the liboqs ML-KEM timing leak (CVE-2024-36405) [MIT24], both timing side channels rather than logic bugs. Recent work on ML-KEM implementation security (chosen-ciphertext side-channel attacks [QGZ+26], hardware FO-verification protections [RMDF26], the secrecy of the encapsulation coin [TBL26], and fault attacks on the seed [KVRBW25]) targets different defects; none exploits an incomplete comparison.

9 Summary and Conclusion

An incomplete ciphertext comparison in the FO check is not merely a weakening of IND-CCA2 security: the unchecked coordinates leak the decryption noise, and that noise recovers the secret key by ordinary least squares. We demonstrated ML-KEM-1024 key recovery against two shipped SIMD backends (AVX2 and NEON) of one library (98 % of the secret end-to-end and the full key in the verified reference model), and showed the cost tracks the geometry of the unchecked bytes more than their count.

Implementations and reviewers should treat any FO comparison that covers fewer than all ciphertext bytes as a key-recovery vulnerability, and comparison routines should be tested specifically on differences confined to the ciphertext tail. Two questions remain open: whether other hand-written KEM backends and libraries carry similar incomplete comparisons, which a differential tail-sweep audit would answer, and how far the leakage-geometry effect generalises, which the ablation of Section 6.1 would settle.

10 Misc

The attack code, the decapsulation oracle, and full reproduction instructions for both backends are available at `https://github.com/007bsd/ml-kem-key-recovery`.

Acknowledgements

We thank Nicholas Carlini (Anthropic) [Car26] for the independent discovery and report of the NEON comparison flaw (CVE-2026-6330), the sibling on which the two-backend analysis here builds.

The write-up was prepared with AI writing assistance (Anthropic's Claude). The vulnerability analysis, the experiments, and all results and claims are the author's own and were verified by the author.

References

[B+24] Daniel J. Bernstein et al. KyberSlash: Exploiting secret-dependent division timings in Kyber implementations. Cryptology ePrint Archive, Paper 2024/1049, 2024.

[BBLW22] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. Hybrid public key encryption. RFC 9180, 2022.

[BDHD+19] Ciprian Băetu, F. Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In *Advances in Cryptology – EUROCRYPT 2019*, 2019. Cryptology ePrint Archive, Paper 2019/525.

[BDK+18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018. Cryptology ePrint Archive, Paper 2017/634.

[Bel05] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[Car26] Nicholas Carlini. ML-KEM ARM64 NEON incomplete ciphertext comparison (CVE-2026-6330). `https://nicholas.carlini.com/`, 2026. Anthropic.

[DGJ+19] Jan-Pieter D'Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In *Public-Key Cryptography – PKC 2019*, volume 11443 of *LNCS*, pages 565–598, 2019.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology – CRYPTO 1999*, volume 1666 of *LNCS*, 1999.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki–Okamoto transformation and its application on FrodoKEM. In *Advances in Cryptology – CRYPTO 2020*, LNCS, 2020.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki–Okamoto transformation. In *Theory of Cryptography – TCC 2017*, 2017. Cryptology ePrint Archive, Paper 2017/604.

[KVRBW25] Hariprasad Kelassery Valsaraj, Prasanna Ravi, Shivam Bhasin, and Hongjun Wu. When randomness isn't random: Practical fault attack on post-quantum lattice standards. Cryptology ePrint Archive, Paper 2025/2009, 2025.

[MIT24] MITRE. CVE-2024-36405: liboqs ML-KEM compiler-induced timing leak. `https://www.cve.org/CVERecord?id=CVE-2024-36405`, 2024. NIST CVSS 7.5.

[MIT26a] MITRE. CVE-2026-10097: ML-KEM-1024 x64 AVX2 incomplete ciphertext comparison enables IND-CCA2 break and static private-key recovery. `https://www.cve.org/CVERecord?id=CVE-2026-10097`, 2026. Fixed in wolfSSL 5.9.2, PR #10430.

[MIT26b] MITRE. CVE-2026-6330: ML-KEM ARM64 NEON ciphertext comparison compares only part of the input. `https://www.cve.org/CVERecord?id=CVE-2026-6330`, 2026. Reported by N. Carlini; fixed in wolfSSL 5.9.2, PR #10192.

[Nat24] National Institute of Standards and Technology. FIPS 203: Module-lattice-based key-encapsulation mechanism standard. Technical report, NIST, 2024.

[Pop24] Giacomo Pope. kyber-py: a pure-Python implementation of ML-KEM (FIPS 203). `https://github.com/GiacomoPope/kyber-py`, 2024. MIT OR Apache-2.0; used as the reference model.

[QCZ+21] Yue Qin, Chi Cheng, Xiaohan Zhang, Yanbin Pan, Lei Hu, and Jintai Ding. A systematic approach and analysis of key mismatch attacks on lattice-based NIST candidate KEMs. In *Advances in Cryptology – ASIACRYPT 2021*, 2021. Cryptology ePrint Archive, Paper 2021/123.

[QGZ+26] Yuhan Qian, Jing Gao, Yuchen Zhong, Yaoling Ding, Jingjie Wu, Weiping Gong, Zihe Lin, and An Wang. One time is enough: Chosen-ciphertext side-channel attack on ML-KEM cryptosystems. In *Applied Cryptography and Network Security Workshops (ACNS Workshops)*, 2026.

[RMDF26] Davis Ranney, Yashaswini I. Makaram, A. Adam Ding, and Yunsi Fei. Exploring side-channel protections in hardware implementations of PQC ML-KEM verification, 2026. arXiv:2606.31681.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3), 2020. Cryptology ePrint Archive, Paper 2019/948.

[Sch23] Sophie Schmieg. Learning, but with errors. Key Material (blog), 2023.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020. Cryptology ePrint Archive, Paper 2020/534.

[TBL26] Madjid G. Tehrani, William J. Buchanan, and Mouad Lemoudden. On the secrecy of the encapsulation coin in ML-KEM. Cryptology ePrint Archive, Paper 2026/1117, 2026.

[wol26] wolfSSL Inc. wolfSSL embedded TLS/SSL library. `https://github.com/wolfSSL/wolfssl`, 2026.

A Full Derivation of the Noise Identity

We use FIPS 203 notation and prove Lemma 1.

Encryption and decompression. Encryption of m with randomness $(\mathbf{r}, \mathbf{e}_1, e_2)$ produces

$$\mathbf{u} = \mathbf{A}^\top \mathbf{r} + \mathbf{e}_1, \qquad v = \mathbf{t}^\top \mathbf{r} + e_2 + \mathsf{Decompress}_1(m),$$

and transmits $(\mathsf{Compress}_{d_u}(\mathbf{u}), \mathsf{Compress}_{d_v}(v))$. At decapsulation these decompress to $\mathbf{u}' = \mathbf{u} + c_u$ and $v' = v + c_v$, where $c_u = \mathsf{Decompress}_{d_u}(\mathsf{Compress}_{d_u}(\mathbf{u})) - \mathbf{u}$ and c_v likewise are the compression errors, bounded coordinatewise by $\lceil q/2^{d_u+1} \rceil$ and $\lceil q/2^{d_v+1} \rceil$ ($d_u = 11,\ d_v = 5$ for ML-KEM-1024).

Cancelling the mask. Using $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ and $(\mathbf{A}\mathbf{s})^\top \mathbf{r} = \mathbf{s}^\top \mathbf{A}^\top \mathbf{r}$,

$$\begin{aligned} v - \mathbf{s}^\top \mathbf{u} &= \mathbf{t}^\top \mathbf{r} + e_2 + \mathsf{Decompress}_1(m) - \mathbf{s}^\top (\mathbf{A}^\top \mathbf{r} + \mathbf{e}_1) \\ &= (\mathbf{A}\mathbf{s} + \mathbf{e})^\top \mathbf{r} + e_2 + \mathsf{Decompress}_1(m) - \mathbf{s}^\top \mathbf{A}^\top \mathbf{r} - \mathbf{s}^\top \mathbf{e}_1 \\ &= \mathsf{Decompress}_1(m) + \mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top \mathbf{e}_1 + e_2. \end{aligned}$$

The mask $\mathbf{s}^\top \mathbf{A}^\top \mathbf{r}$ cancels exactly; this is why decryption recovers m.

Adding the compression errors.

$$\begin{aligned} v' - \mathbf{s}^\top \mathbf{u}' &= (v + c_v) - \mathbf{s}^\top (\mathbf{u} + c_u) = (v - \mathbf{s}^\top \mathbf{u}) + c_v - \mathbf{s}^\top c_u \\ &= \mathsf{Decompress}_1(m) + \underbrace{\mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top (\mathbf{e}_1 + c_u) + e_2 + c_v}_{\delta}, \end{aligned}$$

which is Lemma 1.

Linearity. In the coefficient basis of $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$, a product ab has $(ab)_j = \sum_i \pm a_i\, b_{(j-i) \bmod 256}$ with a sign flip on the negacyclic wrap. Thus $\mathbf{e}^\top \mathbf{r}$ contributes rows in $\mathbf{e}$ with coefficients drawn from $\mathbf{r}$, $-\mathbf{s}^\top (\mathbf{e}_1 + c_u)$ contributes rows in $\mathbf{s}$ with coefficients from $-(\mathbf{e}_1 + c_u)$, and $e_2 + c_v$ is the known constant. Every one of $\mathbf{r}, \mathbf{e}_1, e_2, c_u, c_v$ is fixed by the chosen ciphertext, so $(\mathbf{s}, \mathbf{e})$ enters linearly with known coefficients.

No wraparound. Each summand of δ is a product or sum of small quantities: $\mathbf{s}, \mathbf{e}, \mathbf{r}, \mathbf{e}_1, e_2$ have coefficients in $\{-\eta, \ldots, \eta\}$ with $\eta = 2$, and c_u, c_v are bounded as above. The magnitude of δ_j stays below $q/4$ except with the ML-KEM-1024 decryption-failure probability, on the order of 2^{-174}. Its balanced representative in $(-q/2,\ q/2]$ therefore equals the integer value and does not reduce modulo q, so Lemma 1 holds over $\mathbb{Z}$.

From identity to measurement. Since $m'_j = \mathsf{Compress}_1(\mathsf{Decompress}_1(m)_j + \delta_j)$, the recovered bit flips exactly as δ_j crosses the $\mathsf{Compress}_1$ boundary. The unchecked v-tail lets the attacker add a known offset to v'_j and sweep it; the offset of the flip nearest the nominal $\mathsf{Decompress}_1(m)_j$ locates the boundary and yields δ_j to the compression granularity, about $\pm q/64$ (Section 4.2).