Phantom Timing Leaks on Cortex-M4: A Pre-registered Evaluation of ML-KEM Decapsulation

Amadou Tidiane Anne[0009-0003-0178-8864]

Univ Brest (UBO), Brest, France
amadou.tidiane.anne@proton.me

Abstract. Fixed-versus-random timing tests in the style of dudect are a cheap first check of constant-time behaviour, and on microcontrollers the cycle counter makes them exact to the cycle. We report a pre-registered evaluation of ML-KEM decapsulation on an STM32F411 (Cortex-M4) covering nine implementations of PQM4 (`clean`, `m4fspeed`, `m4fstack` for ML-KEM-512/768/1024) and a positive control in which we restored the pre-KyberSlash division formulas. Each of the 44 campaigns used in the analysis has 100,000 trials and checks every result.
With invalid ciphertexts (random bytes, or one modified byte), no difference is detected in 18 of 18 campaigns, and both classes always have exactly the same cycle count, while the control is detected every time ($|t|$ up to 1,877).
With fresh *valid* ciphertexts, however, all nine implementations are flagged, with differences of 4 to 15 cycles and $|t| \geq 19{,}000$ (or infinite, when both classes have zero variance). A matched control, in which both classes run the same code before the measured call, removes the difference for seven of the nine. An A/A test, in which both classes decapsulate the same bytes, still yields $|t| = 12{,}864$ for one implementation. Deterministic few-cycle differences can therefore come from the measurement harness alone: on a part with flash wait states, an instruction cache and prefetching, they can appear when the code around the measured call changes. We also show that per-trial correctness checks matter: they exposed a functional bug in PQM4's ML-KEM-1024 `m4fstack` (220 wrong shared secrets out of 49,830), which upstream had fixed shortly before. We report every deviation from the pre-registration, including a build error of our own, and publish data, binaries and code.

Keywords: ML-KEM · Kyber · timing side channel · dudect · constant-time · Cortex-M4 · pre-registration

1 Introduction

ML-KEM [1] is being deployed on embedded devices, where an attacker may have physical access and can time operations precisely. Timing leakage in this algorithm family is real: KyberSlash [2] showed that divisions by the modulus in

several Kyber implementations leak secret-dependent timing, and that the leak can be exploited from decapsulation timings.

A popular first check is the dudect methodology [3]: measure the execution time of the operation for two classes of inputs, typically a fixed input and varying inputs, and compare them with Welch's t-test, flagging $|t| > 4.5$ as in TVLA [4]. On a Cortex-M4, the `DWT_CYCCNT` cycle counter makes such measurements exact to the cycle. With interrupts disabled, many runs of a constant-time routine give *exactly* the same count, so even a single-cycle difference between classes produces a huge t value.

This paper started as a plain evaluation and became a study of that sensitivity. We evaluated `crypto_kem_dec` of nine PQM4 [5] implementations of ML-KEM on real hardware, with three class-1 inputs, under a protocol written down and time-stamped before measuring. One of the three inputs, fresh valid ciphertexts, produced "leaks" in every implementation. Two further pre-registered experiments show that these are, at least in large part, artefacts of the harness.

Contributions.

1. A pre-registered timing evaluation of `crypto_kem_dec` for nine ML-KEM implementations (3 parameter sets × `clean`/`m4fspeed`/`m4fstack`) on an STM32F411. It uses $N = 100{,}000$ trials per campaign, three classes of input and a positive control that re-introduces the KyberSlash divisions. Invalid ciphertexts (implicit-rejection path) produce no detectable difference in any implementation; the control is detected in every class (Sect. 5.2).
2. Evidence that few-cycle, fully deterministic class differences on this platform can come from the harness alone. They appear with fresh valid ciphertexts in all nine implementations (C3). They largely vanish when both classes run the same code before the measured call (C4). And they can appear in an A/A test in which both classes process identical bytes (C5) (Sect. 5.3). We draw practical recommendations for evaluators (Sect. 6).
3. A demonstration that per-trial correctness checks belong in timing harnesses: ours exposed a functional bug in PQM4's ML-KEM-1024 `m4fstack` (Sect. 5.1).
4. A transparent record: pre-registration and two amendments time-stamped with OpenTimestamps, every deviation reported (including one caused by our own build error), and all raw data, binaries and scripts published (Sect. 9).

2 Background

ML-KEM decapsulation. ML-KEM is obtained from a CPA-secure encryption scheme through a Fujisaki–Okamoto transform. Decapsulation decrypts the ciphertext, re-encrypts the result and compares the two ciphertexts. On mismatch it returns a pseudo-random key derived from a secret value (implicit rejection). The comparison and the conditional selection that follows are classic targets for side-channel attacks [6,7], since they reveal whether a ciphertext was accepted.

PQM4. PQM4 [5] collects implementations of post-quantum schemes for the Cortex-M4: `clean` (portable C from PQClean) and hand-optimised implementations with assembly, `m4fspeed` (speed) and `m4fstack` (memory). Several files are shared between implementations through symbolic links.

KyberSlash. KyberSlash [2] reported that `poly_compress`, `poly_tomsg` and related functions computed a division by $q = 3329$ of a secret-dependent value. On processors whose division instruction has a variable latency, such as the Cortex-M4 `UDIV`, this leaks timing information. The fixed code uses multiplications and shifts.

dudect and TVLA. dudect [3] measures the execution time of two classes of inputs, interleaved at random, and applies Welch's t-test. TVLA [4] uses the same statistic and the threshold $|t| > 4.5$. The test detects a difference between the two timing distributions. It cannot tell where the difference comes from, and in particular whether it is caused by the secret data.

Flash memory on the STM32F4. In our configuration (84 MHz), the STM32F411 reads its flash memory with 2 wait states. An accelerator (instruction cache, data cache and prefetch buffer) hides most of this latency [8]. The time of a code fragment therefore depends on its placement in flash and on the accelerator state left by the code executed before it.

3 Threat Model

The attacker can submit chosen ciphertexts to `crypto_kem_dec` and measure its execution time; the secret key is fixed. We use the core cycle counter, which is more precise than what a remote attacker would observe, so our results bound what such an attacker could see. Power and electromagnetic leakage, fault attacks and key generation are out of scope.

4 Experimental Setup

Hardware and software.

- *Board:* NUCLEO-F411RE (STM32F411RE, Cortex-M4F), clocked at 84 MHz from the internal 16 MHz oscillator through the PLL. Flash: 2 wait states, instruction and data caches and prefetch enabled, which is PQM4's configuration for this family.
- *Software:* a fork of PQM4 with support for this board, which upstream does not support; the port is described in the repository. Compiler: GCC 16.2.0 for `arm-none-eabi`, with the PQM4 default flags (`-O3`).
- *Functional check:* before any timing measurement, the nine implementations passed PQM4's test vectors on the board.
- *Host:* a Linux server that only flashes the board and records its UART output; it plays no part in the measurement.

Harness. The harness (`common/dudect_v2.c`) generates a fixed key pair and a fixed valid ciphertext. Then, for each trial, it:

1. draws the class with a seeded xorshift32 generator (seed `0x2026C0DE`);
2. prepares the input;
3. disables interrupts, reads `DWT_CYCCNT`, calls `crypto_kem_dec` once, reads `DWT_CYCCNT` again and re-enables interrupts;
4. checks the result;
5. sends one line `index,class,cycles,ok` over the UART.

The host rejects a campaign if a line is missing, if any check failed, or if the final line does not match.

Input classes. Class 0 is always the fixed valid ciphertext; its check is that the expected shared secret is returned. Class 1 depends on the campaign:

C1 a fresh random byte string (implicit rejection). The check is that the fixed key is *not* returned.
C2 the fixed ciphertext with one random byte replaced by a different value (implicit rejection). Same check as C1.
C3 a fresh valid ciphertext produced by `crypto_kem_enc` just before the call (acceptance). The check is that the encapsulated key is returned.
C4 as C3, but *both* classes call `crypto_kem_enc` before the measured call. Class 0 then copies the fixed ciphertext, class 1 the fresh one, so the code executed before the call is the same.
C5 as C4, but class 1 copies a byte-identical copy of the fixed ciphertext, stored in another buffer. Both classes decapsulate the same bytes (A/A test).

Each campaign has $N = 100{,}000$ trials, about 50,000 per class, interleaved at random.

Positive control. The control `m4fspeedks` is ML-KEM-512 `m4fspeed` with the pre-KyberSlash division formulas restored in `poly_compress`, `cmp_poly_compress`, `poly_tomsg` and `polyvec_compress`. It is compiled with `-Os`: at `-O1` to `-O3`, GCC 16 replaces these divisions by multiplications. We checked by disassembly that the 9 implementations under test contain `udiv` instructions only in `main` (output formatting, outside the measured window) and in the UART set-up, whereas the control also contains them in the four functions above.

Statistics and pre-registration. For each campaign we compute Welch's t between the two classes and flag $|t| > 4.5$. When both classes have zero variance, we report $t = 0$ if their counts are equal and $t = \infty$ otherwise; we never hide degenerate cases. We report minimum, maximum and number of distinct values for each class.

The hypotheses H1–H4 and the decision rules were written down before the campaign (28 October 2026). Two amendments followed: H5 and the C4 class (2 November), then H6 and the C5 class (3 November). Each was written after seeing the previous results and before measuring the new ones. Each document, with the firmware hashes, was time-stamped with OpenTimestamps [9]; they are anchored in Bitcoin blocks 973,549, 974,272 and 974,381 respectively.

Table 1. Welch t between class 0 and class 1 for every campaign ($N = 100{,}000$ each, about 50,000 per class). =: both classes have the same single cycle count (zero variance); ∞: both classes have zero variance and different counts. Bold: $|t| > 4.5$. –: not measured. †KyberSlash positive control.

Set	Impl.	C1	C2	C3	C4	C5
ML-KEM-512	`clean`	=	=	**23 285.5**	−3.5	–
ML-KEM-512	`m4fspeed`	=	=	∞	0.8	0.8
ML-KEM-512	`m4fstack`	=	=	**348 811.0**	1.4	–
ML-KEM-768	`clean`	=	=	**19 492.6**	1.9	–
ML-KEM-768	`m4fspeed`	=	=	∞	**−11 216.0**	=
ML-KEM-768	`m4fstack`	=	=	**43 155.3**	=	=
ML-KEM-1024	`clean`	=	=	**22 789.3**	−0.4	–
ML-KEM-1024	`m4fspeed`	=	=	∞	**−15 863.4**	**12 864.0**
ML-KEM-1024	`m4fstack`	=	=	∞	=	–
ML-KEM-512	`m4fspeedks`†	**−1 877.5**	**−306.2**	**104.4**	**36.2**	–

5 Results

5.1 Integrity and Deviations from the Protocol

A functional bug in PQM4. In the first wave, the C3 campaign of ML-KEM-1024 `m4fstack` failed validation twice, with 220 failed checks each time. All failures are in class 1: 220 of 49,830 fresh valid ciphertexts were decapsulated to a wrong key, and the two attempts are identical line by line, so the failure is deterministic.

The cause is a symbolic link in the PQM4 version we started from: this implementation used the inverse NTT of `m4fspeed`, which assumes tighter input bounds. Upstream had fixed it shortly before our campaign (mupq/pqm4 pull request #407, merged on 17 October 2026)[1], after observing the same symptom. Our pre-registered H4 ("all campaigns are valid") is therefore refuted for the first wave. We applied the upstream fix and measured this implementation again in all classes. All 29 campaigns of the second and third waves are valid at the first attempt.

Our build error. When creating the control, we edited `polyvec.c` through a symbolic link shared with the ML-KEM-512 and -768 `m4fspeed` and `m4fstack` implementations. The first-wave binaries of these four implementations therefore carried the division formula. At `-O3` the compiler had turned it into a multiplication, but the machine code was not the upstream one.

We found the error while checking the first wave. We restored the file and rebuilt all 30 binaries:

– 18 are bit-for-bit identical to those measured;

[1] `https://github.com/mupq/pqm4/pull/407`

Table 2. Cycle counts (minimum–maximum) of class 0 (fixed valid ciphertext) and class 1 in C3, C4 and C5, with the number of distinct values. Each row is a separate binary; C3, C4 and C5 binaries differ in code layout.

Set	Impl.	Class	Class 0	dist.	Class 1	dist.
ML-KEM-512	`clean`	C3	847 712	1	847 699–847 701	2
ML-KEM-512	`clean`	C4	847 700–847 704	3	847 700–847 704	3
ML-KEM-512	`m4fspeed`	C3	408 579	1	408 572	1
ML-KEM-512	`m4fspeed`	C4	408 609–408 614	3	408 609–408 614	3
ML-KEM-512	`m4fspeed`	C5	408 609–408 614	3	408 609–408 614	3
ML-KEM-512	`m4fstack`	C3	410 821	1	410 813–410 814	2
ML-KEM-512	`m4fstack`	C4	410 853–410 855	2	410 853	1
ML-KEM-768	`clean`	C3	1 321 902	1	1 321 890–1 321 892	2
ML-KEM-768	`clean`	C4	1 321 908–1 321 912	3	1 321 908–1 321 912	3
ML-KEM-768	`m4fspeed`	C3	672 934	1	672 927	1
ML-KEM-768	`m4fspeed`	C4	672 929–672 932	2	672 932	1
ML-KEM-768	`m4fspeed`	C5	672 989	1	672 989	1
ML-KEM-768	`m4fstack`	C3	678 845	1	678 839–678 841	2
ML-KEM-768	`m4fstack`	C4	678 843	1	678 843	1
ML-KEM-768	`m4fstack`	C5	678 896	1	678 896	1
ML-KEM-1024	`clean`	C3	1 923 445	1	1 923 430–1 923 432	2
ML-KEM-1024	`clean`	C4	1 923 459–1 923 461	2	1 923 459–1 923 461	2
ML-KEM-1024	`m4fspeed`	C3	1 046 538	1	1 046 533	1
ML-KEM-1024	`m4fspeed`	C4	1 046 489–1 046 490	2	1 046 490	1
ML-KEM-1024	`m4fspeed`	C5	1 046 486	1	1 046 485–1 046 486	2
ML-KEM-1024	`m4fstack`	C3	1 056 059	1	1 056 050	1
ML-KEM-1024	`m4fstack`	C4	1 055 992	1	1 055 992	1

– the 12 others are exactly the affected ones.

The 12 affected campaigns were excluded and measured again. The selection rule is mechanical: a campaign is used only if its firmware hash is among the archived binaries of the amendments. Table 1 lists the 44 campaigns used; the 16 excluded ones are listed with their reason in the data release.

5.2 Invalid Ciphertexts and the Positive Control

In C1 and C2 (Table 1), the two classes of every implementation have *the same single cycle count* over about 100,000 trials. Examples: 408,621 cycles for ML-KEM-512 `m4fspeed` in C1; 1,923,445 for ML-KEM-1024 `clean` in C2. This holds for the 18 campaigns, including the portable `clean` code.

Valid and invalid ciphertexts therefore take exactly the same time: implicit rejection does not show at the cycle level. In particular, the comparison and conditional selection do not depend on whether the ciphertext is accepted, and the content of an invalid ciphertext has no visible effect.

The positive control is detected in every class:

– $t = -1{,}877.5$ in C1 and -306.2 in C2;
– 104.4 in C3 and 36.2 in C4;
– 92 to 176 distinct values in class 1, against one or two in class 0.

Figure 1a shows the spread for C1. The harness and the statistic therefore detect a real, data-dependent timing leak of the KyberSlash type; the null results are not an artefact of a blind measurement.

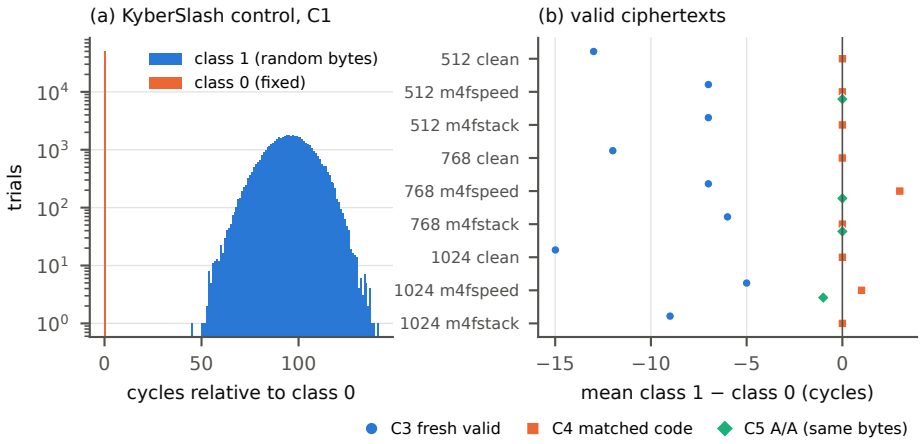

Fig. 1. (a) KyberSlash positive control in C1 (log scale): class 0 has a single cycle count, class 1 spreads over 92 values, 45 to 141 cycles slower. (b) Mean difference between the classes for each implementation with valid ciphertexts. In C3, class 1 runs `crypto_kem_enc` just before the measured call; in C4, both classes do; in C5, both classes also decapsulate the same bytes. A point at 0 means no difference.

Pre-registered verdicts:

- H1 (control detected in C3) is confirmed.
- H2 (no detection in the 27 campaigns C1–C3) is refuted, by the nine C3 campaigns only (Sect. 5.3).
- H3 (zero variance for all `m4fspeed`/`m4fstack` campaigns) is refuted narrowly. In C3, class 1 of ML-KEM-512 `m4fstack` has one trial at 410,813 cycles and 49,829 at 410,814. Class 1 of ML-KEM-768 `m4fstack` has 12 trials two cycles above the 49,818 others.

5.3 Valid Ciphertexts: Phantom Leaks

C3: every implementation is flagged. With fresh valid ciphertexts, all nine implementations show a difference between the classes (Tables 1 and 2). Class 1 is faster by 4 to 15 cycles, out of 0.4 to 1.9 million. For `m4fspeed` the difference is fully deterministic: for ML-KEM-512, class 0 always takes 408,579 cycles and class 1 always 408,572, over 49,830 *different* fresh ciphertexts. Because the class-1 count does not vary with the ciphertext, the difference is unlikely to depend on the ciphertext value.

In C3, however, the code executed just before the measured call differs between the classes (Fig. 1b): class 1 runs `crypto_kem_enc`, class 0 only copies a buffer. In C1 and C2, where both classes run short copy or generation loops, the counts are equal.

C4: matched code before the call. Amendment 1 introduced C4, in which both classes run `crypto_kem_enc` before the measured call. With the same rule ($|t| > 4.5$), the difference disappears for seven of the nine implementations; the three `clean` implementations and the three `m4fstack` ones show no detection. It remains for ML-KEM-768 and ML-KEM-1024 `m4fspeed`, where class 0 is faster by 3 cycles and 1 cycle respectively ($t = -11{,}216$ and $-15{,}863$).

H5(a) is therefore confirmed for 7 of 9 implementations. H5(b), a single common count in both classes, holds only for ML-KEM-768 and -1024 `m4fstack`; elsewhere, rare trials deviate by 1 to 5 cycles in one or both classes. The control is still detected (H1′).

C5: an A/A test. Amendment 2 introduced C5 to separate the two possible explanations of the remaining difference: a dependence on the ciphertext, or the harness. C5 is identical to C4, except that both classes decapsulate the same bytes. The results:

- For ML-KEM-1024 `m4fspeed`, the classes still differ. Class 0 always takes 1,046,486 cycles; class 1 takes 1,046,485 in 49,815 trials and 1,046,486 in 15. This gives $t = 12{,}864$ *for identical data*, with the opposite sign to C4.
- For ML-KEM-768 `m4fspeed`, both classes take exactly 672,989 cycles.
- The two witnesses, ML-KEM-512 `m4fspeed` and ML-KEM-768 `m4fstack`, show no detection, as predicted (H6′).

By the rule fixed in advance, H6 has a *mixed outcome*: the difference persists in only one of the two implementations, with the opposite sign. We therefore attribute neither C4 difference formally.

The C5 result for ML-KEM-1024 nevertheless establishes, without ambiguity, that this harness can produce a perfectly deterministic difference with $|t|$ above 10^4 between two classes that process the same bytes.

Absolute counts depend on the binary. The same operation on the same data takes a different number of cycles in different binaries. ML-KEM-768 `m4fspeed` decapsulating the fixed ciphertext takes:

- 672,934 cycles in the C3 binary;
- 672,929 in the C4 binary;
- 672,989 in the C5 binary.

Before this study, an exploratory version of the harness measured 408,562 cycles for ML-KEM-512 `m4fspeed` (C1-type input), against 408,621 with the current one. The cryptographic code is identical in these binaries; only the surrounding code and its placement change.

6 Discussion

What the results support. On this platform, the nine implementations take exactly the same time for a valid ciphertext and for invalid ones (C1, C2).

Implicit rejection therefore does not leak at the cycle level, and the control shows that a KyberSlash-type leak would have been seen. With valid ciphertexts, the differences we measured are of 1 to 15 cycles and do not vary across tens of thousands of different ciphertexts. The matched control and the A/A test show that differences of this size and form are produced by the harness itself.

What remains open. For ML-KEM-768 `m4fspeed`, the 3-cycle difference of C4 vanished in C5, and our pre-registered rule does not allow us to attribute it. A dependence on the ciphertext would require the fixed ciphertext to behave differently from 49,830 fresh ones that all share one count. We consider this implausible, but this argument was made after seeing the data. A direct test would compare several fixed ciphertexts against each other with one and the same binary.

A plausible mechanism. At 84 MHz the flash has 2 wait states, hidden by the instruction cache and the prefetch buffer. Which flash lines are present, and how the prefetcher is positioned when the measured call starts, depend on the code executed just before it (`crypto_kem_enc`, a copy, a branch) and on where the linker placed everything. A difference of a few cycles over a million-cycle operation is consistent with a handful of cache misses or prefetch stalls. We did not isolate this mechanism. Disabling the caches and prefetch, or running from RAM, would test it.

Recommendations for evaluators.

1. *Match the code executed before the measured call* in both classes. Input generation should be done for both classes, and only the final selection should differ.
2. *Run an A/A test* with the same binary layout. A difference between identical inputs measures the harness, not the target.
3. *Report counts, not only t.* With a cycle counter and zero variance, t is infinite or huge for a one-cycle difference. The minimum, maximum and number of distinct values tell a deterministic one-cycle shift apart from a data-dependent spread such as the 92 to 176 values of the control.
4. *Use a positive control* built from a known leak, so that a null result is informative.
5. *Check every result.* Our checks caught a wrong-output bug that a pure timing harness would have silently measured.
6. *Fix the protocol in advance and record deviations*, including one's own errors.

7 Limitations

- *One platform:* one board, one compiler version, `-O3`, one clock configuration.
- *Fixed key and input:* one key pair and one fixed ciphertext per binary.
- *Sample size:* $N = 100{,}000$ per campaign, enough to detect deterministic differences; rare data-dependent paths below roughly 10^{-5} could be missed.

- *Mechanism not isolated:* no experiment with the flash accelerator disabled.
- *Measurement point:* the cycle counter is internal, so an external attacker would see less.
- *Timing only:* no power, electromagnetic or fault analysis.

8 Related Work

KyberSlash [2] found and exploited division-based timing leaks in Kyber implementations, including on Cortex-M4. Power and electromagnetic attacks on the decapsulation of lattice-based KEMs, in particular on the re-encryption and the comparison, are surveyed and demonstrated in [6,7]. dudect [3] and TVLA [4] define the statistical test used here. PQM4 [5] provides the implementations and the benchmarking framework. To our knowledge, the effect of the code surrounding the measured call on fixed-versus-random timing tests on microcontrollers has not been documented with A/A tests before; we would welcome pointers to such work.

9 Data and Code Availability

The repository contains:

- the PQM4 fork, the harness and the build rules;
- every measured binary with its SHA-256 hash;
- the raw CSV file and metadata of every campaign, including excluded and invalid ones;
- the analysis scripts that produce Tables 1 and 2;
- the pre-registration, the two amendments and their OpenTimestamps proofs.

Repository: `https://github.com/AmadouAnne/embedded-security` (directory `kyber-sca`). Archived version (data, binaries, scripts and pre-registration): `https://doi.org/10.5281/zenodo.23921418`.

References

1. National Institute of Standards and Technology: Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS 203 (2024). `https://doi.org/10.6028/NIST.FIPS.203`
2. Bernstein, D.J., Bhargavan, K., Bhasin, S., Chattopadhyay, A., Chia, T.K., Kannwischer, M.J., Kiefer, F., Paiva, T.B., Ravi, P., Tamvada, G.: KyberSlash: Exploiting secret-dependent division timings in Kyber implementations. IACR TCHES **2025**(2), 209–234 (2025). `https://doi.org/10.46586/tches.v2025.i2.209-234`
3. Reparaz, O., Balasch, J., Verbauwhede, I.: Dude, is my code constant time? In: DATE 2017, pp. 1697–1702 (2017). `https://doi.org/10.23919/DATE.2017.7927267`
4. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation. In: NIST Non-Invasive Attack Testing Workshop, Nara, Japan (2011)

5. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. Second NIST PQC Standardization Conference (2019). IACR ePrint 2019/844
6. Ravi, P., Sinha Roy, S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. IACR TCHES **2020**(3), 307–335 (2020). https://doi.org/10.13154/tches.v2020.i3.307-335
7. Bhasin, S., D'Anvers, J.P., Heinz, D., Pöppelmann, T., Van Beirendonck, M.: Attacking and defending masked polynomial comparison for lattice-based cryptography. IACR TCHES **2021**(3), 334–359 (2021). https://doi.org/10.46586/tches.v2021.i3.334-359
8. STMicroelectronics: RM0383 Reference manual: STM32F411xC/E advanced Arm-based 32-bit MCUs, Rev. 4 (2025)
9. Todd, P.: OpenTimestamps. https://opentimestamps.org