

What Makes Lattice Key Generation Expensive? Controlled Cost Attribution with Structured LWR, ML-KEM, and HAETAE on Cortex-M4

Yan Zhang¹, Meizi Li², and Liang Tan²

¹ Southwest Jiaotong University, Chengdu, China
yan.zhang.ece@outlook.com

² Sichuan Normal University, Chengdu, China
20251393014@stu.sicnu.edu.cn, jkxy_tl@sicnu.edu.cn

Abstract. End-to-end cycle counts quantify lattice key-generation time on a microcontroller, but not which implementation decisions create that cost. We develop a controlled attribution method for public structure, candidate admission, and transform lifetime: how public data are organised, when candidate acceptance is checked, and whether transformed secret state is retained or reconstructed. On a fixed STM32L476RG Cortex-M4 target, paired runs within one executable keep the relevant secret, accepted candidate, or output key unchanged; separate measurements record resource effects. In our Bos-based structured-LWR implementation, reorganising public data exposes expansion and multiplication cost, and same-key, first-attempt comparisons show nearly additive admission and lifetime effects. We then test whether these explanations transfer across algorithms and rejection structures. In ML-KEM-512, whose KeyGen has no candidate rejection, retaining the secret’s NTT-domain representation reduces cycles; a pre-specified ML-KEM-768 test preserves this direction but shows that the saving does not scale with module rank alone. HAETAE-5 tests both decisions inside a retrying KeyGen. Early checking stops rejected candidates before matrix computation, whereas deferred checking repeats matrix work across attempts and, under recomputation, rebuilds transformed secret state. The interaction grows with the retry count, while retention saves cycles at the cost of a larger KeyGen frame. Together, these experiments establish a compositional account of KeyGen cost: public structure defines the downstream computation for each candidate, while candidate admission and transform lifetime determine how many candidates execute it and how often reusable transformed representations are rebuilt. These relations yield directional predictions across the tested KeyGen structures, with paired measurements quantifying their cycle and storage consequences.

Keywords: lattice cryptography · Cortex-M4 · key generation · cost attribution

1 Introduction

KeyGen optimisation on a microcontroller depends on knowing why a cost occurs, not only how large it is. Embedded lattice implementations are usually

evaluated through whole-operation cycle counts, peak stack use, and code size. Benchmarking frameworks such as the eXtended eXternal Benchmarking eXtension (XXBX) and pqm4 make these measurements reproducible on Cortex-M4 devices [17,19,20]. Such totals identify viable operating points, but combine costs arising from the organisation of public data, the placement of rejection, and the reuse of transformed state. A bare-metal STM32L476RG Cortex-M4 provides a stable baseline on which these cycle–stack trade-offs can be observed directly. We use this setting to separate the costs within one controlled implementation line and then test whether selected explanations continue to predict behaviour as parameter sets and rejection structure change across implementations.

Embedded lattice implementations repeatedly exchange storage for recomputation. Work on FrodoKEM, Saber, and Kyber shows how on-demand generation and buffer reuse reduce live storage [9,21,12]. NewHope and Toom–Cook implementations make the complementary choice: retained precomputation or transformed state avoids repeated work [3,8]. Rejection makes these costs depend on how many attempts are required. Compact Dilithium exposes the recomputation accumulated across rejected attempts, while HAETAE work uses rejection-aware passes and early checks to avoid further work on failing candidates [10,16,22]. These strategies produce different speed–memory operating points [15], but whole-operation totals aggregate their effects. A total cycle count alone cannot attribute cost to a particular implementation choice, distinguish interactions between choices, or show whether the same explanation transfers to another KeyGen implementation.

We examine three implementation choices. Public structure changes how public data are expanded and multiplied. Candidate admission selects which sampled candidates reach that work, while transform lifetime controls whether a reusable transformed representation is retained or rebuilt. The admission mechanism differs across the experiments: our Bos-based radical-ring (RR) path adds an optional unit check, whereas the HAETAE-5 paths place its native norm predicate before or after matrix work. In both cases, the cost follows which candidates execute the downstream computation and how often reusable state is reconstructed.

We measure these relationships through interventions that change one computation or live range while preserving the relevant input, candidate, or key. Timing alternatives share one executable, resource costs come from fixed-policy builds or instrumented paired runs, and external predictions are fixed before measurement. We implement the RR and block-ring-module (BRM) structures of Bos et al. alongside a rank-matched Module-LWR baseline, using a common arithmetic and sampling framework to isolate the cost of public-data organisation [11]. We instantiate this comparison at modulus $q = 3329$ and rank $k = 5$. The BRM–Module contrast isolates public expansion and multiplication, while the RR factorial shows nearly additive admission and lifetime effects when the first candidate is accepted.

We select the external cases to test transform lifetime under contrasting KeyGen control flows. ML-KEM-512 provides a standardised setting without candi-

date rejection. A prediction fixed after that measurement tests whether the cycle saving scales proportionally with module rank in ML-KEM-768. HAETAE-5 provides native candidate rejection during KeyGen, allowing admission placement and transform lifetime to interact within the same retry loop; ML-DSA’s candidate-rejection loop instead occurs during signing [24]. Moving HAETAE-5’s norm check after matrix work makes rejected candidates complete that work and, under recomputation, rebuild transformed secret state across the matrix rows. This progression establishes an execution-count account that links each design choice to work introduced, avoided, or repeated. The resulting account predicts which KeyGen path should be faster and when retry behaviour changes the preferred schedule, while paired measurements quantify the corresponding cycle and storage costs on Cortex-M4.

2 Related Work and Attribution Gap

The Learning with Rounding problem and its subsequent security refinements underpin rounding-based lattice constructions [5,4]. Saber established Module-LWR as a practical KEM design, with its Arm implementation generating public-matrix entries just in time [14,21]. Scabbard extends this design space across ring and module LWR, while a recent systematisation places these constructions and their implementation techniques in a common framework [7,6]. Bos et al. add radical-ring and block-ring-module structures [11]. Implementing these structures alongside a rank-matched Module-LWR baseline allows us to vary public-data organisation within a common code base.

Embedded lattice implementations also trade stored state for repeated computation. NTTRU uses the NTT to test invertibility and keeps key material in transformed form [23]. Cortex-M4 arithmetic frameworks and Saber NTT implementations further show how transform-domain accumulation and memory layout shape embedded cost [2,1]. The same general trade-off appears in FrodoKEM, where matrix regeneration and buffer reuse define different operating points, and in Toom–Cook multiplication, where retaining precomputed operands and intermediate state avoids repeated evaluation and interpolation [9,8]. FIPS 203 standardises the Kyber-derived ML-KEM [25]. The earlier Kyber Cortex-M4 implementation generates one-use matrix and noise data on demand while retaining transformed vectors that feed more than one matrix product [12]. Our transform lifetime experiment turns this established storage–recomputation choice into a paired KeyGen measurement.

Rejection adds another dimension because discarded candidates may already have incurred expensive computation. HAETAE rejects KeyGen candidates to bound the secret under challenge multiplication [13]. In signing, compact Dilithium implementations show how repeated attempts amplify regeneration and recomputation costs [16,10]. Hyoungh et al. provide a lower-memory Cortex-M4 implementation of SMAUG-T and HAETAE [18], while Kim et al. combine rejection-aware pass decomposition and component-level early rejection in HAETAE signing with streamed matrix generation across the implementation

[22]. These results establish rejection-aware scheduling as an implementation technique. Our HAETAE-5 experiment instead measures how the placement of its native KeyGen check interacts with retaining or rebuilding transformed state as the number of attempts changes.

Prior work therefore establishes the relevant algebraic structures, storage–recomputation trade-offs, and rejection-aware schedules. We bring these elements into one controlled KeyGen study. Each timing comparison holds fixed the sampled secret, accepted candidate, or output key required by that comparison, and the alternative paths are compiled into one executable. Resource effects are measured with isolated builds or instrumented paired runs, while external predictions are fixed before measurement. This design attributes measured KeyGen cost to the work changed by an implementation decision and tests whether the resulting directional and quantitative predictions survive a new parameter set or execution structure.

3 M4 Implementation

3.1 Structured-LWR Paths

The experimental system combines three structured-LWR paths in one bare-metal Cortex-M4 code base. Two implement the RR and BRM structures of Bos et al.; the third is a rank-matched Module-LWR baseline [11]. Each path operates on degree- $< n$ polynomials modulo q , with multiplication reduced by the relation $y^n = -1$. We write this base ring as

$$R_{q,n} = \mathbb{Z}_q[y]/(y^n + 1).$$

All three paths sample a secret s , expand a public map A from a seed, and form $t = \text{Round}_p(As)$. Their main difference is how much independent public data is needed to represent A and how the resulting multiplication is scheduled.

The Module baseline represents A as a dense matrix $A_{\text{Mod}} \in R_{q,n}^{k \times k}$, so all k^2 entries are expanded independently. BRM reduces this public data by giving the leading block a ring-multiplication structure. At $k = 5$, it uses a $4 + 1$ partition. The leading 4×4 block contains sixteen matrix positions, but all of them are generated from only four public polynomials; the final row and column remain independent:

$$A_{\text{BRM}} = \begin{bmatrix} C(a_0, a_1, a_2, a_3) & u \\ v^\top & \alpha \end{bmatrix},$$

$$C = \begin{bmatrix} a_0 & ya_3 & ya_2 & ya_1 \\ a_1 & a_0 & ya_3 & ya_2 \\ a_2 & a_1 & a_0 & ya_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix}.$$

The four polynomials $a_0, \dots, a_3$ generate the entire 4×4 block; the factors of y appear when multiplication wraps around under the block relation. Together

with the four entries of u , the four entries of v , and α , BRM expands 13 independent public polynomials rather than the 25 required by the dense Module matrix. After this definition, *BRM* denotes the rank-5 instance unless another partition is stated.

Radical Ring-LWR (RR-LWR), shortened to *RR*, goes further by encoding all five components of the public map and secret as elements of one enlarged ring, replacing the matrix-vector product with a single ring product. The enlarged ring is

$$S_{q,n,k} = \mathbb{Z}_q[x]/((x^k - 2)^n + 1) \cong R_{q,n}[x]/(x^k - y - 2).$$

Writing $A = \sum_{i=0}^{k-1} a_i x^i$ and $s = \sum_{i=0}^{k-1} s_i x^i$ packages their base-ring components into single elements of $S_{q,n,k}$. At $k = 5$, the RR, BRM, and Module paths therefore expand 5, 13, and 25 public polynomials, respectively. This progression changes both public expansion and multiplication while preserving the common KeyGen relation.

For the controlled comparison, all three paths use $q = 3329$, rounding modulus $p = 256$, $k = 5$, and a degree-128 base ring. They also share SHAKE256 expansion, the sparse balanced secret sampler, NTT arithmetic, rounding, and key representation. From the same form of 64-byte input, KeyGen derives A and s , computes As , and rounds its coefficients from modulus q to modulus p to obtain t . In our benchmark representation, the secret key also stores the rounding residual

$$e_t = \text{Lift}_p(t) - As \pmod{q},$$

where Lift_p maps the stored coefficients of t back to the q -domain.

Bos et al. provide the RR structure. We add two experimental controls to its implementation: candidate admission and transform lifetime. Candidate admission either accepts a sampled secret immediately or checks whether its radical-ring representation is invertible, resampling after failure. After acceptance, transform lifetime either keeps the NTT-domain representation for public multiplication or discards and rebuilds it from the same coefficient-domain secret. Crossing the two choices gives the four RR paths used below. Paired retain and recompute executions produce identical keys.

3.2 KeyGen Contexts

ML-KEM and HAETAE-5 place these implementation choices in two different KeyGen control flows. ML-KEM has no candidate rejection, so we can compare retaining and rebuilding its transformed secret without retries. HAETAE-5 repeatedly samples candidates until its secret-key bound is satisfied, allowing the position of the acceptance check and the treatment of transformed state to affect the same computation.

ML-KEM samples secret and error vectors, transforms them, and performs its public-key calculation in the NTT domain:

$$\hat{t} = \hat{A} \circ \hat{s} + \hat{e}$$

where $\circ$ denotes the module matrix–vector product using ML-KEM’s NTT-domain polynomial multiplication [25]. The same transformed secret $\hat{s}$ is first consumed by the public-key computation and is later serialised into the decryption key. The gap between these two uses provides a direct test of transform lifetime.

We focus on HAETAE-5’s $d = 0$ KeyGen branch because its acceptance test can be evaluated before matrix computation. Its public matrix has four rows, and the secret vector used in the multiplication contains six polynomials. It samples this vector s_{gen} and an error vector e_{gen} , then evaluates the scheme-specific spectral size measure that bounds how strongly a challenge can amplify that candidate. The candidate is accepted when

$$\mathcal{N}(s_{\text{gen}}, e_{\text{gen}}) \leq \gamma^2 n;$$

here n and γ are the HAETAE-5 dimension and acceptance-bound parameters. Otherwise both vectors are resampled [13]. In the native schedule, this decision is made before the public matrix is expanded. Once a candidate is accepted, the implementation multiplies it by the expanded public matrix, adds the error vector, and applies HAETAE’s public-key scaling:

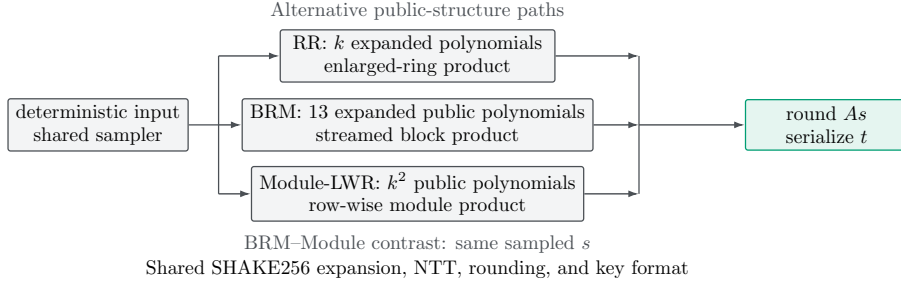
$$\hat{b} = -2(\hat{A}_{\text{gen}} \circ \text{NTT}(s_{\text{gen}}) + \text{NTT}(e_{\text{gen}})) \bmod q.$$

The selected low-stack Cortex-M4 implementation computes the four output rows in sequence [22]. Its native schedule regenerates and transforms the same six accepted secret polynomials separately for each row. Our retention path instead keeps those six NTT-domain polynomials and reuses them across all four rows, while the error polynomial remains local to each output row. The controlled paths independently place the same acceptance test before or after this matrix work. HAETAE-5 therefore reveals what changes when rejected candidates enter an expensive computation and when reusable transformed state must be rebuilt within that computation.

3.3 Embedded Design

The structured-LWR public keys store a seed instead of the expanded public map, so each measured KeyGen reconstructs A from this seed as multiplication proceeds. The structure of A determines both how much public data must be generated and how much temporary state remains live. Module processes one matrix row at a time. It keeps the transformed secret and output vectors in memory, expands and transforms the k public polynomials in the current row, computes one output polynomial, and then reuses the row buffer. BRM needs only one public-polynomial scratch buffer. Each of its four structured generators is expanded once and applied to the first four output accumulators; four independently generated polynomials complete the final column, and five more form the bottom row. RR has no matrix row to stream. It keeps five transformed components of the public ring element, five transformed secret components, and

(a) Public structure implementations



(b) Controlled RR KeyGen paths

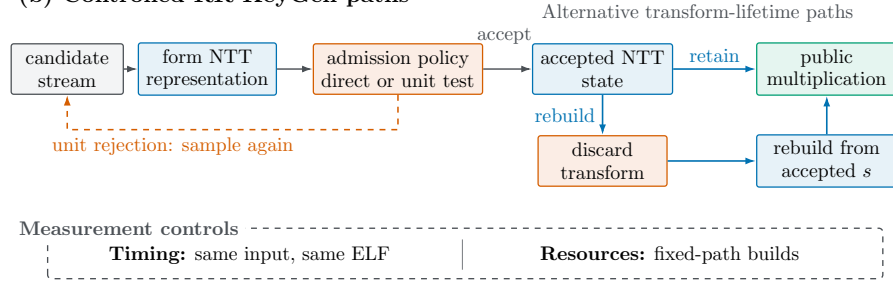


Fig. 1. Structured-LWR implementation paths and controlled RR comparisons. (a) RR, BRM, and Module realise the same KeyGen relation with different public-data expansion and multiplication schedules; the BRM-Module comparison reuses the same sampled secret. (b) RR crosses unit-conditioned admission with retaining or rebuilding the accepted secret’s NTT-domain representation. Timing paths share one executable; fixed-path builds measure resource use.

five output accumulators. At each NTT position, the public and secret components are combined according to the enlarged-ring multiplication to update those accumulators. The completed outputs are then returned to the coefficient domain and rounded.

All working storage uses fixed-size arrays; no heap is used. SHAKE256 expands the public data and samples sparse secrets through an Arm-assembly Keccak permutation. The serialised public key contains only the public seed $seed_A$ used to expand A and the rounded output t ; the expanded public data and its transformed working buffers exist only while KeyGen is running.

In the unit-conditioned RR paths, the accepted secret’s NTT-domain representation is used by both the invertibility test and public multiplication. Retention keeps it live between these uses; recomputation clears it after the test and rebuilds it before multiplication. Both paths release the representation afterwards. Figure 1 connects this lifetime choice to the three structured-LWR implementations and shows the conditions held fixed by the timing and resource comparisons.

4 Attribution Method

4.1 Cost Model

To attribute a cycle difference, we identify both the work changed by an intervention and the number of times that work runs. In the public structure comparison, both paths use the same sampled secret, while the chosen structure changes the public polynomials generated and the multiplication schedule. Candidate admission changes how many sampled secrets reach later computation. Transform lifetime changes how often an existing transformed representation must be rebuilt before another use. When KeyGen can reject and resample candidates, the number of executions must therefore be counted explicitly.

Let m denote the total number of candidate attempts, including the accepted candidate. When the acceptance check precedes matrix work, rejected candidates stop at the check and the downstream matrix computation runs once. Deferring the check sends all m candidates through that computation, adding $m - 1$ executions. The early HAETAE-5 schedule also performs a separate norm-only pass and then replays the accepted candidate. With immediate acceptance, this fixed overhead can make the early schedule slower. Each rejection, however, adds matrix work only to the deferred schedule. Early checking becomes faster once the matrix work avoided after rejection outweighs its fixed extra pass.

For transform lifetime, the relevant event is whether a transformed secret is still available when another operation needs it. In the controlled RR paths, recomputation rebuilds the accepted secret’s NTT-domain representation between the invertibility test and public multiplication, while retention keeps the existing NTT-domain state. ML-KEM uses the same representation for public-key computation and later for secret-key serialisation. Rebuilding it at the second use regenerates and transforms the K secret polynomials, where K is the module rank. HAETAE-5 places this choice inside its four-row matrix calculation. For every candidate that reaches the matrix, retention transforms the six secret polynomials once and reuses them across the rows. Recomputation regenerates and transforms all six separately for each row, producing 24 secret NTTs rather than six and adding 18 NTTs per matrix candidate. Early admission exposes only the accepted candidate to this difference. Deferred admission exposes all m candidates, so every rejection adds another 18 transforms to the recomputation path.

These execution counts give the tests used below. Among RR inputs whose initial candidate is accepted, the invertibility check and the transform rebuild each occur once, so the factorial asks whether their cycle effects combine additively. ML-KEM-512 tests whether rebuilding the transformed secret increases cycles without candidate rejection. Before measuring ML-KEM-768, we fixed a stronger magnitude prediction: if the reconstruction cost per secret polynomial remained unchanged, increasing the module rank from two to three would make the median cycle difference $3/2$ of the ML-KEM-512 value. For HAETAE-5, the model predicts that deferring admission becomes more expensive with each rejection and that the benefit of retention grows on the deferred path while remaining

nearly constant on the early path. The following controls turn these predictions into matched cycle and resource comparisons on the Cortex-M4.

4.2 Platform and Controls

All cycle contrasts are measured on the same bare-metal NUCLEO-L476RG, whose Cortex-M4F runs at 80 MHz. Every build uses Arm GNU Toolchain 14.3.Rel1 and the hard-float ABI. The Bos C sources use `-Os -g0`, and their assembly uses `-O3 -DNDEBUG`; the ML-KEM and HAETAE C sources use `-O3 -g0`. Each primary experiment replays a fixed set of 1,000 inputs or seeds in two separately flashed sessions. The second session repeats the same inputs to check capture reproducibility rather than enlarging the input set. Only KeyGen lies inside the interval timed by the Data Watchpoint and Trace (DWT) cycle counter; input preparation, output comparison, and UART transmission remain outside it. The contrasted paths reside in the same linked executable (ELF) and receive matching inputs. The Bos public structure comparison pairs separate deterministic sweeps by input index; the RR factorial, ML-KEM, and HAETAE-5 experiments alternate or balance path order. The signed within-input cycle difference is the primary paired observation.

Equivalence checks follow the comparison: BRM and Module must use the same sampled secret, while controlled admission and transform lifetime comparisons must preserve the accepted candidate, generated keys, and attempt count where applicable. HAETAE-5 uses one isolated build per factorial cell for path-specific Flash, static RAM, and compiler-frame measurements. Bos resource records instead come from an eight-build matrix covering structure and admission variants, not the four RR lifetime cells. In that matrix, static RAM includes the benchmark harness and compiler frames describe individual functions. ML-KEM uses a custom allocator inside the paired executable to record peak live allocation during KeyGen.

4.3 Bos Structured-LWR

For each input, the BRM and Module paths receive the same sampled secret but organise the public map differently. We use this pair to ask how many additional cycles arise from the Module public expansion and multiplication schedule relative to BRM. Let $C_{\text{BRM}}(i)$ and $C_{\text{Module}}(i)$ denote their cycle counts, and define

$$D_{\text{pub}}(i) = C_{\text{Module}}(i) - C_{\text{BRM}}(i).$$

This paired contrast isolates the difference between their implemented public expansion and multiplication schedules from variation in the secret. The RR paths instead support a factorial comparison of candidate admission and transform lifetime: the transformed candidate lies between an optional invertibility check and public multiplication, allowing the two decisions to be varied together.

All four RR paths reside in one executable and initially sample and transform the same candidate. The unconditioned admission policy accepts that candidate

immediately. The unit-conditioned policy checks whether the transformed candidate is invertible in the RR algebra and resamples after a failure. After acceptance, retain passes the existing NTT-domain representation directly to public multiplication, whereas recompute clears it and transforms the same accepted secret again. Each admission policy is measured once with retention and once with recomputation, giving the four cells of the factorial design.

Within a fixed admission policy, the next comparison asks how many cycles are added when the accepted candidate’s transformed representation is rebuilt rather than retained. For admission policy $a \in \{\text{unconditioned}, \text{unit}\}$ and transform lifetime policy $\ell \in \{\text{retain}, \text{recompute}\}$, let $C_{a,\ell}(i)$ denote cycles on input i . We define the within-policy transform lifetime contrast as

$$D_{\text{ret}}^a(i) = C_{a,\text{recompute}}(i) - C_{a,\text{retain}}(i).$$

Within either admission policy, retain and recompute produce the same accepted key with the same attempt count for all 1,000 inputs, so every input contributes to D_{ret}^a . Across admission policies, the accepted candidate and key remain identical for the 970 inputs that pass the invertibility check on the first attempt. This subset lets us ask how much unit-conditioned admission adds under each lifetime policy and whether that added cost changes between retention and recomputation. We define

$$\begin{aligned} D_{\text{adm}}^\ell(i) &= C_{\text{unit},\ell}(i) - C_{\text{unconditioned},\ell}(i), \\ I_{\text{Bos}}(i) &= D_{\text{ret}}^{\text{unit}}(i) - D_{\text{ret}}^{\text{unconditioned}}(i). \end{aligned}$$

Within this first-attempt subset, D_{adm}^ℓ measures the net cycle effect of enabling unit-conditioned admission under lifetime policy ℓ , and I_{Bos} measures departure from additivity between candidate admission and transform lifetime on the cycle-count scale. The 30 retry inputs remain valid for the within-policy lifetime contrasts but are excluded from D_{adm}^ℓ and I_{Bos} because the two admission policies accept different candidates.

4.4 ML-KEM Tests

We use the portable C implementation in `mlkem-native v1.2.0` and its deterministic ML-KEM KeyGen core [25,26]. Each retain–recompute pair receives the same deterministic 64-byte KeyGen input, formed by concatenating the two 32-byte values d and z , and must produce the same keypair. The retain path keeps the NTT-domain secret vector until secret-key serialisation. The recompute path deliberately clears this transformed secret after public-key computation, then regenerates only the secret polynomials in coefficient form from the still-live noise seed. A study-specific scalar helper applies ML-KEM’s PRF and centred-binomial sampler to one polynomial at a time before the forward NTT. The path then rejoins the retained implementation for coefficient reduction and serialisation; the error vector is not regenerated. Thus, the paired cycle difference measures the complete implementation cost of clearing and rebuilding the

transformed secret, including regeneration, transformation, and the helper’s associated data movement and control flow. Both paths run in one executable with alternating order, and execution stops on any key mismatch. A custom allocator separately records the peak number of simultaneously live allocated bytes during KeyGen.

Before measuring ML-KEM-512, we fixed three requirements for all 1,000 inputs: rebuilding the secret transform must add cycles relative to retaining it, both paths must produce byte-identical keys, and the KeyGen allocation peak must remain unchanged. A separate instrumented executable divides the retained path into measured stages and checks their timing-window coverage of the complete KeyGen execution. For each input, the uninstrumented retained path provides the baseline. We estimate aggregate instrumentation perturbation as the instrumented whole-KeyGen total minus this baseline, and subtract it from both the stage sum and the instrumented total. Coverage is the corrected stage sum divided by the corrected instrumented total, while relative perturbation is the total-cycle difference divided by the baseline. For every input, the calibrated coverage must reach at least 95%, and the absolute relative perturbation may not exceed 2%. This accounting checks timing-window coverage of retained KeyGen; the transform lifetime effect is measured separately by the paired retain–recompute difference.

After obtaining the ML-KEM-512 result, but before collecting ML-KEM-768 board timing for this intervention, we fixed a simple rank-only prediction. The intervention reconstructs one secret polynomial per module component, so the rule treats the cycle difference as proportional to module rank. Increasing the rank from two in ML-KEM-512 to three in ML-KEM-768 therefore predicts a 768 median recompute-minus-retain difference equal to $3/2$ of the 512 median. Agreement requires the measured 768 median to lie within $\pm 2\%$ of that prediction. The ML-KEM-768 experiment therefore tests two claims separately: whether rebuilding the secret transform still adds cycles, and whether the size of that effect scales with module rank alone.

4.5 HAETAE-5 Factorial

ML-KEM isolates transform lifetime without candidate rejection. HAETAE-5 brings candidate admission back into a retry loop, where rejected candidates can repeat both matrix work and transform reconstruction. In the upstream $d = 0$ low-stack path, the norm test runs before matrix work. After acceptance, each of the six secret polynomials is regenerated and transformed separately for each of four matrix rows; the six polynomials are packed into the secret key during the first row. This is the early–recompute path. We derive the other paths by changing when the same norm test runs and how long the candidate’s transformed secret remains live. Deferred checking performs a candidate’s matrix work before testing its norm, so a rejected candidate has already paid that cost. Retention instead forms the six transformed secret polynomials once per candidate reaching matrix work and reuses them across the four rows. Table 1

combines these choices. In the path labels, E and D denote early and deferred admission, while R and T denote recomputation and retention.

Table 1. HAETAE-5 paths formed by candidate admission placement and transform lifetime.

Candidate admission	Recompute per row	Retain across rows
Before matrix work (early)	ER	ET
After matrix work (deferred)	DR	DT

All four paths run in one executable on the same 1,000 deterministic 32-byte seeds. Four Williams-balanced orders, selected by the input index modulo four, rotate every path through every execution position. For each seed, host checks require all paths to accept the same candidate after the same number of attempts and to produce byte-identical public and secret keys. The board runs compare the first four seed bytes, accepted candidate, attempt count, and a 64-bit FNV-1a comparison tag over $pk\|sk$ against that frozen host record.

Within a lifetime policy, deferred minus early measures the net cycle effect of admission placement: $DR - ER$ under recomputation and $DT - ET$ under retention. These contrasts include both the work performed for rejected candidates and the extra norm-only sampling pass used by early checking; deferred checking accumulates the norm during candidate generation for matrix and output work. Within an admission placement, recompute minus retain measures the net cycle effect of rebuilding rather than retaining the secret transform across rows: $ER - ET$ with early checking and $DR - DT$ with deferred checking. These complete-path contrasts include the sampling, packing, clearing, and control flow associated with each schedule as well as its transform count.

The central interaction asks whether retention saves more cycles when the norm check is deferred. We measure that change as

$$I_H = (DR - DT) - (ER - ET).$$

The four cell means support three algebraically independent contrasts. We report the two admission effects, the two lifetime effects, and their interaction to show how each decision changes under the other. Before board timing, we fixed the expected direction of each reported quantity. Deferred checking had to add cycles under both lifetime policies, recomputation had to add cycles under both admission placements, and the retention saving had to be larger under deferred checking. We required both the paired mean and median to follow these directions in both captures of the same ELF and frozen input set.

Retry count provides a diagnostic of the proposed mechanism. Each additional rejected candidate makes the deferred paths repeat matrix and output work. Under recomputation, secret-polynomial regeneration and transformation are additionally repeated for every matrix row; under retention, the transformed secret is formed once per candidate and reused across rows. We therefore fit the four simple effects and their interaction against the recorded attempt count using ordinary least squares with an intercept. The slopes assess the predicted

pattern: effects tied to work repeated by rejected candidates should grow with the attempt count, whereas the early-path retention saving should remain nearly constant because only the accepted candidate reaches matrix work. The median contrasts for first-attempt and retried inputs show what changes once rejection occurs.

A fitted slope alone could be dominated by a small number of inputs with many attempts or conceal departures elsewhere in the retry range. To examine the fit across that range, we group measured-minus-fitted residuals into eight attempt-count bins: 1, 2, 3–4, 5–7, 8–12, 13–20, 21–35, and 36–64. Their medians and 5th–95th percentiles show how closely the linear relation describes each part of the observed distribution. Because the four paths execute sequentially in one ELF, we also check for a systematic association between execution position and cycle count. Within every attempt-count stratum containing at least four seeds, we subtract the corresponding path median and group the residuals by execution position. The range of the four position medians must remain within 2% of that path’s overall median. This threshold requires position-associated variation to remain small relative to total KeyGen cycles.

The same-ELF design gives matched timing, but the executable contains all four paths and cannot assign Flash, static RAM, or stack costs to one of them. We therefore build each path separately with both selectors fixed at compile time. These builds provide per-path Flash, static RAM, and compiler-reported static KeyGen frames. A one-seed smoke run also fills unused stack memory with a known pattern and measures how much is overwritten. This touched-stack watermark checks the compiler-frame difference; it excludes the caller’s existing frame and is not a worst-case stack bound.

5 Results

Table 2 summarizes the primary controlled effects. Contrasts are reported as the first path minus the second; a positive cycle value therefore means that the first path takes longer. Timing uses 1,000 paired inputs, except for the 970 first-attempt Bos admission and interaction comparisons. The HAETAE-5 frame result comes from fixed-path builds.

5.1 Bos Separates Three Sources of KeyGen Cost

The Bos comparisons use two distinct controls. Module and BRM receive the same sampled secret, isolating the implemented public structure paths; the RR factorial crosses the unit-conditioned policy with transform lifetime. Module requires about 0.88 million more cycles than BRM, quantifying the difference between the implemented public-expansion and multiplication paths. In the RR factorial, reconstructing the accepted candidate’s transform adds about 159.4 thousand cycles under either checking policy, while unit-conditioned checking adds about 295.7 thousand cycles under either lifetime policy. The median interaction is only 18 cycles (range 18–24), approximately four orders of magnitude

Table 2. Primary controlled KeyGen effects.

Test	Contrast (first – second)	Matched condition	Result
<i>Bos structured-LWR</i>			
Public structure	Module – BRM	Same sampled secret	Median: +0.879 M cycles
Candidate admission	Unit check – no check	Same first candidate and key	Median: +0.296 M cycles
Transform lifetime	Recompute – retain	Same candidate and key	Median: +0.159 M cycles
Interaction	Lifetime gap: unit – no check	Same first candidate and key	Median: +18 cycles
<i>ML-KEM</i>			
512 lifetime	Recompute – retain	Same input and keypair	Median: +0.171 M cycles; KeyGen allocator peak $\Delta 0$ B
768 rank-only scaling	Recompute – retain	Same input and keypair	Median: +0.167 M Prediction: +0.256 M
<i>HAETAE-5</i>			
Interaction	Lifetime gap: deferred – early	Same candidate, attempts, and keys	Mean: +11.74 M cycles
Retry slope	Recompute – retain, deferred	Same candidate, attempts, and keys	OLS slope: +1.31 M cycles/additional attempt
Retention memory	Retain – recompute	Fixed-path builds	KeyGen frame: +6,160 B

below either main effect. Candidate checking and transform lifetime therefore behave as separable costs when the first candidate is accepted. Retention removes about 10.0% of the recompute-path median at this point.

A separate same-input Bos run measures the unit-conditioned policy effect when that policy retries. Because its two paths then accept different candidates, this comparison is distinct from the same-key admission effect above. Unit conditioning selects a later candidate for 30 of the 1,000 inputs, raising the median policy overhead from about 295 thousand cycles on immediate acceptance to 740 thousand cycles on retries. Those 30 inputs occupy a distinct cost regime because the checking policy changes which candidate reaches public multiplication.

5.2 ML-KEM Confirms the Transform Lifetime Direction

For 1,000 matched ML-KEM-512 inputs, rebuilding the discarded secret transform requires about 171 thousand additional cycles at the median compared with retaining it. The cycle difference is positive for every input, and both paths produce byte-identical keypairs with the same 5,729-byte KeyGen allocator peak.

Retention therefore saves cycles without increasing peak live allocation during KeyGen. In the separate instrumented retained runs, the calibrated timing window covers a median 99.9630% of KeyGen cycles, with a maximum total-cycle perturbation of 0.1452%.

The benefit also holds for ML-KEM-768: recomputation adds a median of about 167 thousand cycles and requires more cycles on all 1,000 inputs, while preserving identical keypairs and the 10,081-byte KeyGen allocator peak. Its magnitude, however, does not follow the increase in module rank. The pre-specified rank-only rule predicts about 256 thousand cycles; the observed median is 35.0% lower, outside the frozen $\pm 2\%$ interval. The two parameter sets also use different secret-sampling parameters, so each additional polynomial does not carry an identical reconstruction workload. Retaining the secret transform saves cycles in both parameter sets, but module rank alone does not determine how much work reconstruction adds.

5.3 HAETAE-5 Exposes a Retry-Dependent Scheduling Crossover

ML-KEM isolates transform lifetime in a KeyGen without candidate rejection. HAETAE-5 introduces a retry loop and tests whether the position of the acceptance check changes the benefit of retaining transformed state. Across all 1,000 seeds, the four experimental paths preserve the accepted candidate, attempt count, public key, and secret key. KeyGen requires 9.973 attempts on average, with a median of seven; 103 seeds accept on the first attempt, consistent with its reported acceptance rate of roughly 10% [22]. The two admission effects, two lifetime effects, and their interaction all follow the pre-specified directions in both board captures.

The placement of the acceptance check produces a scheduling crossover. When the first candidate succeeds, deferred checking requires about 368 thousand fewer cycles because the early path includes a separate norm-only sampling pass. Once candidates are rejected, however, early checking prevents them from entering matrix computation. The deferred-checking penalty consequently grows by about 4.88 million cycles per attempt under recomputation and 3.57 million under retention. Across the measured retry distribution, early checking reduces mean KeyGen cycles by 70.35% under recomputation and 65.09% under retention (Figure 2(a,b)).

The same retry mechanism changes the value of transform retention. With early checking, only the accepted candidate reaches matrix computation, so retention avoids repeated secret-polynomial regeneration and transformation across rows, saving about 1.308 million cycles regardless of the number of rejected candidates. With deferred checking, every candidate reaches matrix computation; each additional attempt therefore repeats this across-row work and increases the retention saving by another 1.308 million cycles. Because the interaction subtracts the early lifetime gap from the deferred gap, its fitted retry slope is the difference between the two lifetime slopes. The resulting increment of about 1.308 million cycles per attempt matches the measured early-path lifetime

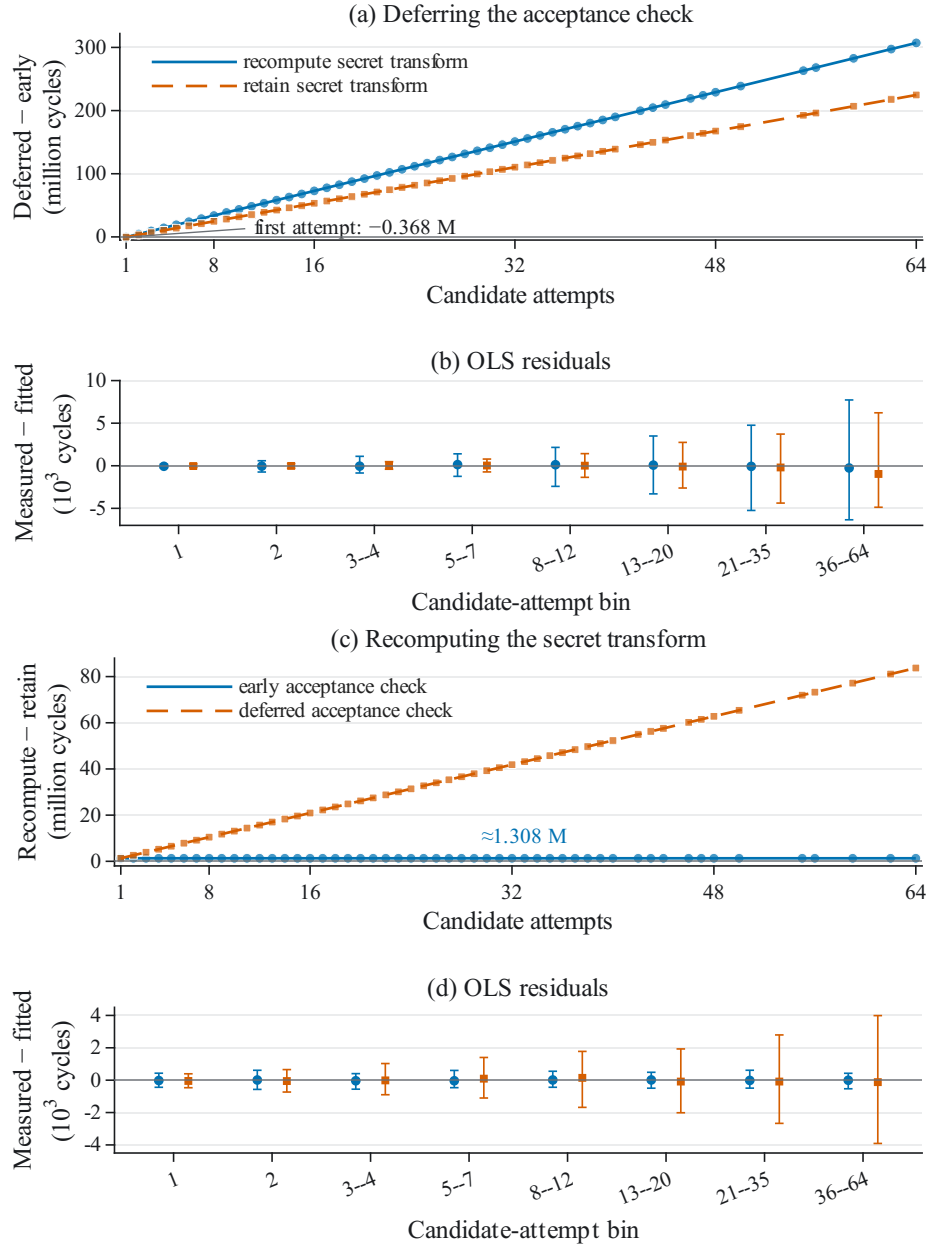


Fig. 2. HAETA5 retry-conditioned effects across 1,000 frozen seeds. (a,c) Markers show attempt-count medians and lines show ordinary least-squares fits to all paired observations. The annotation in (a) gives the immediate-acceptance contrast compressed near zero by the full retry-range scale. (b,d) Points show binned median measured-minus-fitted residuals, with whiskers spanning the 5th–95th percentiles.

gap, as expected when each additional candidate repeats the same across-row re-computation path. The interaction reaches a mean of about 11.74 million cycles (Figure 2(c,d)).

The linear fits track the measurements across all eight retry bins: the absolute value of each bin’s median residual remains below 1,000 cycles, and the range of execution-position residual medians is at most 0.00042% of the corresponding cell median. These diagnostics support the repeated-work interpretation across the observed retry range and show little systematic position-associated variation.

Transform retention also increases the measured stack requirement. In isolated builds, the compiler-reported KeyGen frame rises from 4,272 to 10,432 bytes under either check schedule, a difference of 6,160 bytes. The seed-zero touched-stack measurement reproduces the same differential, while moving the acceptance check leaves both stack measures and static RAM unchanged. The two choices therefore address different costs. Early checking removes work performed for rejected candidates without changing the measured memory requirement, whereas retention removes transform reconstruction but increases the KeyGen frame by 6,160 bytes. The preferred check schedule follows the retry behaviour; the decision to retain transformed state additionally depends on the available stack.

6 Discussion

6.1 Design Rules for KeyGen

KeyGen cost depends on the work attached to each candidate and how often that work is executed or reconstructed. In the first-attempt Bos baseline, candidate checking and transform lifetime contribute nearly additive costs. HAETAE-5 turns them into an interaction: deferred checking lets rejected candidates enter matrix computation, and recomputation repeats secret transforms within it. The ML-KEM tests separate direction from magnitude. Avoiding secret-transform reconstruction saves cycles in both parameter sets, but the saving does not scale with rank alone; the parameter sets also differ in secret sampling.

For rejection-bearing KeyGen, the scheduler therefore balances the fixed cost of early checking against downstream work avoided after rejection. Retention adds the stack trade-off: it avoids reconstruction but keeps transformed state live. The preferred combination follows the retry distribution, downstream work, and available stack.

6.2 Implications for Embedded Evaluation

Benchmark suites such as pqm4 provide comparable cycle, stack, and code-size totals under a common harness [19,20]. Controlled attribution complements them by linking each difference to work moved, retained, or recomputed. This predicts the faster ML-KEM path even when rank-only scaling misses the magnitude; under rejection, interpretation additionally requires the retry distribution and stack cost of retained state.

The coefficients reported here characterize the NUCLEO-L476RG, the pinned implementations, and the fixed toolchain. Repeating the same intervention on another microarchitecture preserves the attribution question while establishing its local cycle-memory balance.

7 Reproducibility

For reproducibility, we archived the exact source revisions, study patches, compiler settings, executable hashes, deterministic inputs, raw board output, and analysis programs used in this study. The reported contrasts can be recomputed from the archived board outputs without access to the development board. The structured-LWR code independently implements the published RR and BRM structures and contains no source code from Bos et al. The package includes this implementation and the recorded upstream revisions and patches for ML-KEM and HAETAE. New captures require the NUCLEO-L476RG, recorded toolchain, ST-LINK programmer, and a 115200-baud serial connection. Each timing experiment was repeated after a fresh flash on the same frozen inputs, producing byte-identical serial output.

Replay checks the matching conditions for each intervention. Bos uses the same sampled secret for the public structure comparison and restricts cross-admission RR contrasts to the 970 first-attempt inputs producing the same key. ML-KEM retain and recompute paths produce identical keypairs for all 1,000 inputs. For HAETAE-5, the host record fixes the accepted candidate, attempt count, and keys; board output supplies the corresponding seed identifier, candidate, attempt count, and 64-bit key tag.

Four isolated HAETAE-5 builds record per-cell Flash, static RAM, and compiler-reported KeyGen frames. Bos supplies a separate eight-build resource matrix covering structure and admission variants rather than the RR lifetime factorial; its static RAM includes the harness and its compiler frames are per function. ML-KEM records peak live allocation during KeyGen. The HAETAE-5 one-seed stack watermark matches the compiler-frame difference but is not a worst-case stack bound.

8 Conclusion

Controlled interventions expose how KeyGen cost is composed. Public structure sets the expansion and multiplication work attached to a candidate. Admission and transform lifetime are nearly additive when the first Bos candidate is accepted, but interact in HAETAE-5 because retries repeat matrix work and transform reconstruction. Matched paired runs carry the lifetime prediction to both ML-KEM parameter sets, while the held-out result shows that module rank alone does not determine its magnitude. Together with separate resource measurements, this attribution provides an execution-level basis for choosing KeyGen schedules and representations on constrained devices.

References

1. Abdulrahman, A., Chen, J.P., Chen, Y.J., Hwang, V., Kannwischer, M.J., Yang, B.Y.: Multi-moduli NTTs for Saber on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2022**(1), 127–151 (2022). <https://doi.org/10.46586/tches.v2022.i1.127-151>
2. Alkim, E., Bilgin, Y.A., Cenk, M., Gérard, F.: Cortex-M4 optimizations for $\{R,M\}$ LWE schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2020**(3), 336–357 (2020). <https://doi.org/10.46586/tches.v2020.i3.336-357>
3. Alkim, E., Jakubeit, P., Schwabe, P.: NewHope on ARM Cortex-M. In: *Security, Privacy, and Applied Cryptography Engineering – SPACE 2016*. pp. 332–349 (2016). https://doi.org/10.1007/978-3-319-49445-6_19
4. Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited: New reduction, properties and applications. In: *Advances in Cryptology – CRYPTO 2013*. pp. 57–74 (2013). https://doi.org/10.1007/978-3-642-40041-4_4
5. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: *Advances in Cryptology – EUROCRYPT 2012*. pp. 719–737 (2012). https://doi.org/10.1007/978-3-642-29011-4_42
6. Basso, A., Bos, J.W., D’Anvers, J.P., Karmakar, A., Bermudo Mera, J.M., Renes, J., Sinha Roy, S., Vercauteren, F., Wang, P., Wang, Y., Zhang, S., Zhong, C.: Using learning with rounding to instantiate post-quantum cryptographic algorithms. *ACM Transactions on Embedded Computing Systems* **25**(4), 1–33 (2026). <https://doi.org/10.1145/3807514>
7. Bermudo Mera, J.M., Karmakar, A., Kundu, S., Verbauwhede, I.: Scabbard: A suite of efficient learning with rounding key-encapsulation mechanisms. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2021**(4), 474–509 (2021). <https://doi.org/10.46586/tches.v2021.i4.474-509>
8. Bermudo Mera, J.M., Karmakar, A., Verbauwhede, I.: Time-memory trade-off in Toom–Cook multiplication: An application to module-lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2020**(2), 222–244 (2020). <https://doi.org/10.46586/tches.v2020.i2.222-244>
9. Bos, J.W., Bronchain, O., Custers, F., Renes, J., Verbakel, D., van Vredendaal, C.: Enabling FrodoKEM on embedded devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2023**(3), 74–96 (2023). <https://doi.org/10.46586/tches.v2023.i3.74-96>
10. Bos, J.W., Renes, J., Sprenkels, A.: Dilithium for memory constrained devices. In: *Progress in Cryptology – AFRICACRYPT 2022*. pp. 217–235 (2022). https://doi.org/10.1007/978-3-031-17433-9_10
11. Bos, J.W., Renes, J., Vercauteren, F., Wang, P.: Structured module lattice-based cryptography: Fine-grained parameter selection without compromising performance. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2026**(3), 1032–1055 (2026). <https://doi.org/10.46586/tches.v2026.i3.1032-1055>
12. Botros, L., Kannwischer, M.J., Schwabe, P.: Memory-efficient high-speed implementation of Kyber on Cortex-M4. In: *Progress in Cryptology – AFRICACRYPT 2019*. pp. 209–228 (2019). https://doi.org/10.1007/978-3-030-23696-0_11
13. Cheon, J.H., Choe, H., Devevey, J., Güneysu, T., Hong, D., Krausz, M., Land, G., Möller, M., Stehlé, D., Yi, M.: HAETAE: Shorter lattice-based fiat–shamir sig-

- natures. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2024**(3), 25–75 (2024). <https://doi.org/10.46586/tches.v2024.i3.25-75>
14. D’Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F.: Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In: *Progress in Cryptology – AFRICACRYPT 2018*. pp. 282–305 (2018). https://doi.org/10.1007/978-3-319-89339-6_16
 15. Fournaris, A.P., Tasopoulos, G., Brohet, M., Regazzoni, F.: Running longer to slim down: Post-quantum cryptography on memory-constrained devices. In: *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. pp. 1–6 (2023). <https://doi.org/10.1109/COINS57856.2023.10189268>
 16. Greconici, D.O.C., Kannwischer, M.J., Sprenkels, D.: Compact Dilithium implementations on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2021**(1), 1–24 (2021). <https://doi.org/10.46586/tches.v2021.i1.1-24>
 17. Hession, B., Kaps, J.P.: Feasibility and performance of PQC algorithms on microcontrollers. *Second NIST Post-Quantum Cryptography Standardization Conference* (2019), <https://csrc.nist.gov/CSRC/media/Events/Second-PQC-Standardization-Conference/documents/accepted-papers/kaps-feasibility-performance-pqc.pdf>
 18. Hyoung, Y., Cho, S., Kim, U., Lee, M., Seo, H., Sim, M.: Memory-efficient implementation of SMAUG-T and HAETAE. *Cryptology ePrint Archive*, Paper 2026/442 (2026), <https://eprint.iacr.org/2026/442>
 19. Kannwischer, M.J., Krausz, M., Petri, R., Yang, S.Y.: pqm4: Benchmarking NIST additional post-quantum signature schemes on microcontrollers. *Cryptology ePrint Archive*, Paper 2024/112 (2024), <https://eprint.iacr.org/2024/112>
 20. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. *Cryptology ePrint Archive*, Paper 2019/844 (2019), <https://eprint.iacr.org/2019/844>
 21. Karmakar, A., Bermudo Mera, J.M., Sinha Roy, S., Verbaauwhede, I.: Saber on ARM: CCA-secure module lattice-based key encapsulation on ARM. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2018**(3), 243–266 (2018). <https://doi.org/10.46586/tches.v2018.i3.243-266>
 22. Kim, Y., van Vredendaal, C., Banegas, G., Seo, S.C.: Low-stack HAETAE for memory-constrained microcontrollers. *Cryptology ePrint Archive*, Paper 2026/635 (2026), <https://eprint.iacr.org/2026/635>
 23. Lyubashevsky, V., Seiler, G.: NTTRU: Truly fast NTRU using NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2019**(3), 180–201 (2019). <https://doi.org/10.46586/tches.v2019.i3.180-201>
 24. National Institute of Standards and Technology: Module-lattice-based digital signature standard. Tech. Rep. FIPS 204, NIST (2024). <https://doi.org/10.6028/NIST.FIPS.204>
 25. National Institute of Standards and Technology: Module-lattice-based key-encapsulation mechanism standard. Tech. Rep. FIPS 203, NIST (2024). <https://doi.org/10.6028/NIST.FIPS.203>
 26. PQ Code Package contributors: mlkem-native v1.2.0. Software release, commit 0ba906cb14b1c241476134d7403a811b382ca498 (2026), <https://github.com/pq-code-package/mlkem-native/releases/tag/v1.2.0>