

X-Wing on Cortex-M85: Architecture-Aware Integration and End-to-End Evaluation

Do-Yun Park¹[0009-0002-2201-6161] and
Hwa-Jeong Seo¹[0000-0003-0069-9061]*

Department of Convergence Security,
Hansung University, Seoul (02876), South Korea
rhcp030418@gmail.com

Abstract. X-Wing combines ML-KEM-768 and X25519 for hybrid key establishment. On Cortex-M85, we adapted the coefficient representation and storage order of an MVE NTT to the existing ML-KEM path and optimized data-format transformations and X25519 arithmetic while preserving draft-10. During encapsulation, the two X25519 results share an inversion. We compared the eight-technique integrated implementation B8 with the baseline A0 in the same executable. Across ten paired comparisons from five independent reflashes of one EK-RA8M1 board, the minimum reported cycle reductions were 18.61% for key generation, 18.89% for encapsulation, 12.25% for decapsulation reusing an expanded key, and 15.22% for decapsulation including seed expansion. Nonvolatile storage increased by 35,692 B, including a 33,024 B fixed-base table. We separately report the cost and validation of B8S, which corrects identified secret-sign-dependent operand-address selections in B8. Some key-varying tests retained timing signals after this correction. Their causes remained partly unresolved, so no whole-program constant-time pass is claimed for B8S.

Keywords: X-Wing · Hybrid KEM · ML-KEM · X25519 · Cortex-M85 · M-profile Vector Extension · Embedded cryptographic optimization

1 Introduction

Hybrid key establishment combines classical and post-quantum cryptography to support the transition to post-quantum security. X-Wing is a key-encapsulation mechanism (KEM) that combines the shared secrets of X25519 and ML-KEM-768 using SHA3-256 [6,8]. Under the security assumptions on the combiner hash, the construction is designed to remain secure if either cryptographic component remains secure.

On embedded devices, the execution time of the full API includes function calls, intermediate buffers, and code and table placement, as well as component

* Corresponding author: hwajeong84@gmail.com.

arithmetic. Component-level optimizations therefore require evaluation within complete X-Wing calls. Renesas offers RA8M1 for network-connected embedded devices, making it a target for evaluating these computational and memory costs [17].

The target Cortex-M85 core supports 128-bit vector operations through the Arm M-profile Vector Extension (MVE, Helium). We use MVE for ML-KEM’s data parallelism and modify arithmetic and instruction order for X25519’s long dependency chains. The full API is measured while preserving the sequential calls to ML-KEM and X25519.

1.1 Related Work

The original X-Wing work presented the X25519/ML-KEM-768 construction and its security rationale, and evaluated optimized implementations [6]. Its implementation combined the components without changing their internals and discussed further optimization by combining internal ML-KEM processing with X-Wing key derivation. The evaluated specification here is the subsequently defined draft-10 algorithm and API [8].

For ML-KEM, pqm4 provides optimized Cortex-M4 code and a common evaluation framework [12]. The number-theoretic transform (NTT) and modular arithmetic are major optimization targets in polynomial multiplication. Helium NTT work introduced MVE-based NTT and polynomial multiplication on Cortex-M55 [7]; pqmx provides these MVE implementations for several Arm cores [21]. SLOTHY uses constraint solving to schedule cryptographic assembly for the target microarchitecture [1,2].

For X25519, prior work optimized Montgomery ladders and field multiplication and squaring for small microcontrollers [13,14,10,9]. A Montgomery ladder performs point addition and doubling for each scalar bit. Precomputed tables for multiplication by a fixed basepoint are described in [11, Section 3.3.2]. Algorithm 2.26 of the same reference describes batch inversion, which shares one inversion across several denominators. The provenance table in Appendix A distinguishes reused code from the changes evaluated here.

1.2 Contributions

We address three questions: how much of the individual optimization benefits remains in the full API; what memory costs are required; and whether address-selection or timing issues remain undetected by functional-equivalence checks and limited timing tests.

We denote the baseline implementation by A0, the performance-evaluation implementation integrating eight techniques by B8, and the variant correcting secret-dependent operand-address selection by B8S. B8’s integrated performance relative to A0 and B8S’s additional cost relative to B8 were evaluated in separate experiments. No direct A0/B8S comparison under identical conditions was performed, so B8’s improvement percentages are not assigned to B8S.

The contributions are as follows.

1. We integrated an MVE NTT with ML-KEM’s existing arithmetic representation, input range, and storage order, and implemented 11 sampling, encoding, and comparison functions using MVE. Direct storage replaces the scalar reordering required after the forward NTT in an intermediate MVE configuration.
2. We optimized X25519 fixed-base computation, shared field arithmetic, and conditional swap, and shared an inversion when normalizing the two X25519 encapsulation results. Existing behavior is preserved for each output when a denominator is zero.
3. We evaluated the integrated effect through direct A0/B8 comparisons, conditional removal experiments on B8, and resource measurements. B8S’s address-selection correction cost and unresolved timing observations are reported separately.

2 Background

2.1 X-Wing

X-Wing combines the shared secrets of ML-KEM-768 and X25519 into one 32 B shared secret [8,15,13]. Let ss_M, ss_X denote the component shared secrets, ct_X the sender’s ephemeral X25519 public key, and pk_X the recipient’s X25519 public key. The combination is

$$ss = \text{SHA3-256}(ss_M \parallel ss_X \parallel ct_X \parallel pk_X \parallel \text{label}).$$

Here $\parallel$ denotes byte-string concatenation, and *label* is the fixed 6 B label specified in draft-10. The inputs are thus the two shared secrets, the two X25519 public keys, and the specification’s label.

Key generation. A 32 B secret seed is first expanded to 96 B using SHAKE256. The first 64 B generates the ML-KEM key pair; the remaining 32 B serves as the X25519 secret scalar. The X25519 public key is obtained by multiplying this scalar by the standard Montgomery basepoint with u -coordinate 9. The external public key concatenates the 1,184 B ML-KEM public key and the 32 B X25519 public key. The external secret key is the original 32 B seed. Multiplication by this predetermined basepoint is called *fixed-base scalar multiplication*.

Encapsulation. ML-KEM produces a ciphertext and shared secret ss_M . One ephemeral X25519 secret scalar is multiplied by the standard basepoint to produce ct_X and by the recipient’s public key pk_X to produce ss_X . The latter operation, which receives the point as a call input, is called *variable-base scalar multiplication*. The two shared secrets and the other combiner inputs defined above are hashed. The returned ciphertext concatenates the ML-KEM ciphertext and ct_X .

Decapsulation. ML-KEM decrypts the ciphertext and compares it with the re-encryption result. For an invalid ciphertext, implicit rejection selects a shared secret derived from a secret rejection value. X25519 performs variable-base multiplication using the recipient’s secret scalar and ct_X from the ciphertext. The two shared secrets then pass through the same combiner.

The internal expanded-key structure contains a 2,400 B ML-KEM secret key, the X25519 secret scalar and public key of 32 B each, and a separately stored 1,184 B ML-KEM public key, totaling 3,648 B. We call measurements that reuse this structure *warm*, and those that expand the external secret seed before decapsulation *cold*. These terms indicate inclusion of key preparation and are distinct from cache state. Table 1 specifies the timed work.

Table 1. API boundaries in this implementation. Key generation does not output a shared secret.

Operation	Input/state at measurement entry	Output and included work
Key generation	Random-number generator	Public key: 1,216 B; secret seed: 32 B
Encapsulation	Recipient public key; random-number generator	Ciphertext: 1,120 B; shared secret: 32 B
Warm decapsulation	Ciphertext; expanded key: 3,648 B	Shared secret: 32 B; expanded key reused
Cold decapsulation	Ciphertext; secret seed: 32 B	Key expansion followed by shared secret: 32 B

In this call structure, fixed-base computation is required for key generation, encapsulation, and cold key expansion. Warm decapsulation reuses the existing public key. Encapsulation also produces two X25519 outputs, an ephemeral public key and a shared secret, whose final conversion can potentially be performed together.

2.2 ML-KEM

ML-KEM uses 256-coefficient polynomials modulo $q = 3329$. Polynomial multiplication consists of a forward NTT, multiplication in the NTT domain, and an inverse NTT [15]. Base multiplication in this NTT multiplies degree-one expressions formed by coefficient pairs; polynomial-vector operations accumulate the results by matrix row. The coefficient-pair order expected at each stage must therefore agree. An MVE NTT output layout that differs from the existing consumer layout requires separate reordering.

Centered binomial distribution (CBD) sampling, serialization, compression and decompression, message conversion, and ciphertext comparison are also repeated. They apply the same operation to independent coefficients and are candidates for vectorization alongside the NTT.

2.3 X25519

X25519 repeats addition, subtraction, multiplication, squaring, and conditional swap in a Montgomery ladder [13]. These operations take place in the finite field $\mathbb{F}_p$, where $p = 2^{255} - 19$, and are referred to as field arithmetic. For ladder projective coordinates (X_M, Z_M) with $Z_M \neq 0$, the affine output coordinate is $u = X_M/Z_M = X_M Z_M^{-1}$. This conversion is called normalization; field division is computed by multiplying by the denominator’s inverse. The X25519 output expression $X_M Z_M^{p-2}$ returns zero when $Z_M = 0$. It does not define an inverse of zero.

Inversion requires multiple multiplications and squarings. When at least two outputs are available, their inversions can therefore be shared. Fixed-base multiplication can also use points precomputed from the standard basepoint. This table does not apply to variable-base multiplication whose input point is the peer’s public key.

2.4 Cortex-M85

Cortex-M85 is an Armv8.1-M in-order core that issues instructions in program order and provides 128-bit vector operations through MVE [3]. These operations can process multiple coefficients or 32-bit words together. Integration must also account for the following constraints.

- The limited scalar register count restricts the intermediate values that can be retained simultaneously.
- MVE and scalar memory accesses are subject to concurrent-issue restrictions.
- Dependencies and issue rules can give different cycle counts to sequences containing the same number of instructions.
- Spills, which temporarily save register values to memory, and constant regeneration can reduce the benefit of rescheduling.

3 Proposed Method

3.1 Overview

A0 builds ML-KEM and Keccak from the pqm4 lineage, X25519 from the Lenngren lineage, scalar C data-format transformations, and newlib-nano memory routines for Cortex-M85. B8 adds the eight changes in Table 2. B8S retains them while replacing operand-address selection based on a secret sign with value swaps at fixed addresses. The table shows applicability; its numbering does not indicate execution order. We first describe component-level layout and computation, then the joint processing of the two encapsulation outputs. All configurations preserve the cryptographic specification and external API.

Table 8 in Appendix A distinguishes reused algorithms and code from the modifications. The full-API comparisons in Section 4.2 evaluate the integrated effect; removal experiments evaluate conditional effects within B8 while retaining the other changes.

Table 2. Eight optimizations applied to Cortex-M85 X-Wing and their operation scope.

No.	Component	Optimization	Operations
1	ML-KEM	MVE NTT with storage matched to subsequent operations	All
2	ML-KEM	11 MVE data-format transformations	All
3	X25519	Per-digit table for 3-bit digits and basepoint $u = 9$	Key generation, encapsulation, cold
4	X25519	Fixed frame and rescheduling for field multiplication	All X25519
5	X25519	Rescheduling of field squaring	All X25519
6	Common	Alignment-aware copying and initialization	All
7	X25519	MVE masked conditional swap	Encapsulation, warm, cold
8	X-Wing/X25519	Shared inversion for two outputs, with zero-denominator handling	Encapsulation

3.2 NTT and Encoding

Integrating an NTT requires matching both the storage order and the mathematical representation of the computed coefficients. An order mismatch causes subsequent functions to read the wrong coefficient positions. A mismatch in the fixed representation factor changes the intended arithmetic even when positions agree. We describe storage matched to the consumers, followed by compatibility of the arithmetic representation and input range.

A0 uses the existing `ntt_fast` and `invntt_fast`. An intermediate configuration first integrating the MVE forward NTT required scalar `rev4` reordering because its output layout differed from that of subsequent operations. B8 uses the pqmx MVE structure-store path to write directly in the required layout [21]. The removed operation is therefore the *reordering after the forward NTT in that intermediate configuration*. Reordering before the inverse NTT remains in B8.

Storage order. Reordering changes memory positions while preserving coefficient values. The `rev4` routine treats 32 consecutive coefficients as one block. Let $a_0, \dots, a_{31}$ be its 16-bit coefficients before reordering, grouped into 32-bit pairs $P_j = (a_{2j}, a_{2j+1})$ ($0 \leq j < 16$). A 4×4 transpose acts on these 16 pairs. Denoting pairs before and after reordering by P_j and P'_j , respectively, gives

$$P'_{4r+c} = P_{4c+r}, \quad 0 \leq r, c < 4.$$

One block contains 32 coefficients, or 64 B, and a 256-coefficient polynomial occupies eight blocks. Figure 1 in Appendix A shows the two layouts of one block. B8's forward NTT stores the coefficients in vector lanes, the individual element positions in a vector, directly in this transposed order. This preserves

the layout expected by subsequent base multiplication, matrix accumulation, and serialization. Before the inverse NTT, one transpose converts the existing array to pqmx’s input layout.

Representation. In modular arithmetic, an integer representing a coefficient may include a fixed factor introduced during multiplication. We call this factor the representation scale. Integration with the existing ML-KEM Plantard modular-arithmetic path requires the inverse transform to produce the scale expected by subsequent computations.

For $q = 3329$, let $R = 2^{16} \bmod q$ and $P = -2^{32} \bmod q$. The upstream inverse supplies the factor R together with normalization by 128 for the seven inverse-NTT layers. The existing Plantard base-multiplication outputs carry a factor P^{-1} , so the adapted inverse must supply P to cancel it. The normalization constants are therefore

$$R 128^{-1} \bmod q = 512, \quad P 128^{-1} \bmod q = 1888.$$

For Barrett multiplication by c , the auxiliary constant is $\text{round}(c 2^{15}/q)$, giving 5040 for $c = 512$ and 18584 for $c = 1888$. These relations determine both replacements in the pqmx inverse transform.

Input coefficients are signed 16-bit integers representing values modulo q ; this inverse-transform path uses centered representatives in $[-1664, 1664]$. The inverse-integration experiment expY checked all residues on the host and compared coefficients on the board. Observed inputs to actual calls also lay within this interval. The result applies to the tested representation and input conditions, not arbitrary 16-bit inputs. Appendix A.3 identifies the source snapshots; `artifact_index.tsv` indexes the expY validation record.

Data processing around the NTT was also vectorized. For CBD sampling, serialization, compression, message conversion, and comparison, A0’s scalar C loops were replaced with fixed-iteration implementations. They use compiler intrinsics, which specify vector instructions in C function form. The replacements cover 11 functions. Multiple coefficients are extracted, compressed, or compared together in 128-bit vectors.

Both optimizations apply to key generation, encapsulation, and warm/cold decapsulation, all of which execute ML-KEM. Mask setup specifying active vector lanes and some byte assembly with high rearrangement costs remain scalar.

3.3 Fixed-Base Arithmetic

X25519 public-key generation multiplies a secret scalar by the standard basepoint with u -coordinate 9. A0 uses a general Montgomery ladder for this operation as well. B8 uses the fixed basepoint to precompute the required points, then selects and adds them at runtime. Following the code’s terminology, we call the fixed-base implementation below a signed comb. Width $w = 3$ means that each digit processes 3 bits. The 86th position accommodates the final carry from the preceding digit. All four candidates are read at each digit, and masks select the

required values. This procedure does not use the secret digit magnitude directly as a table address. B8 selects operand pointers for the chosen values according to the sign, so address order depends on the secret sign even without branches. B8S exchanges the selected table values and mixed-addition intermediates using masks at fixed addresses.

The comb accumulator uses extended Edwards projective coordinates (X, Y, Z, T) , where $x = X/Z$ and $y = Y/Z$, with curve constant $d_E = -121665/121666 \in \mathbb{F}_p$. X25519 outputs a Montgomery coordinate, requiring conversion of the accumulator as $u = (Z + Y)/(Z - Y)$. The function producing a projective result returns numerator $Z + Y$ and denominator $Z - Y$ without completing the division immediately. These values form the first output used for shared inversion in Section 3.5. Appendix A.2 gives the digit representation, table coordinates, 33,024 B table-size calculation, and width-selection limits.

Shared field arithmetic. Fixed-base computation and the general ladder both use field multiplication and squaring. Exponentiation for inversion requires consecutive squarings, so each result is passed directly to the next squaring’s input registers, eliminating intervening stores and reloads. For field multiplication, push/pop operations in the routine body were replaced with stores and loads at fixed stack-pointer (SP) offsets. On entry, 8 B of register saves and a further 40 B reservation establish a 48 B frame for temporaries and return information. SP remains fixed in the routine body.

Field squaring retains the 94 instructions selected for rescheduling and their register assignment while moving independent operations between instructions waiting for earlier results. No separately measured cycle benefit is reported for a scheduling fragment.

As specified in Table 2, the fixed-base table is used for public-key generation, while the field-multiplication and squaring changes apply to all operations that execute X25519.

3.4 Conditional Swap

The Montgomery ladder repeatedly swaps two intermediate results according to a bit of the secret scalar. A conditional swap exchanges the values or leaves them unchanged according to a mask. Each of the two buffers in this implementation contains two field elements: 16 words of 32 bits, or 64 B, per buffer.

A0 performs scalar xor-mask operations word by word. For words a, b and mask m , it computes $tmp = (a \text{ xor } b)$ and m and xors tmp into each of a, b . A zero mask preserves the values; an all-ones mask exchanges them.

B8 performs the same operations using 128-bit MVE vectors. Each group loads one vector from each buffer, applies xor-and-xor, and stores both vectors. Four groups require eight vector loads and eight stores in total. The mask value does not change the instruction flow or processed addresses. This change applies to encapsulation and warm/cold decapsulation, which use the variable-base ladder.

3.5 Paired Inversion

Encapsulation computes two X25519 results with one ephemeral secret scalar. Fixed-base multiplication produces the sender's ephemeral public key; variable-base multiplication produces the shared secret with the recipient. This implementation executes the operations sequentially and retains both projective results just before conversion to the external output format. Joint normalization at this point allows them to share an expensive inversion.

Outputs. Normalization derives the output coordinate u_i from a result held in projective coordinates. For a nonzero denominator, the numerator and denominator (N_i, D_i) satisfy $u_i = N_i/D_i = N_i D_i^{-1}$. Index 1 denotes the ephemeral public key and index 2 the X25519 shared secret.

For the B8/B8S comb accumulator, $(N_1, D_1) = (Z + Y, Z - Y)$; for the variable-base ladder, $(N_2, D_2) = (X_M, Z_M)$. Each denominator D_i is a field element obtained from these coordinates and is distinct from the comb curve constant d_E . All equations below are over $\mathbb{F}_p$ with $p = 2^{255} - 19$; $D_i = 0$ means the zero element modulo p .

Shared inversion. If both denominators are nonzero, $(D_1 D_2)^{-1}$ is computed once. Multiplication by D_2 yields D_1^{-1} , and multiplication by D_1 yields D_2^{-1} . This reduces two separate inversion calls to one. Paired inversion replaces one inversion with additional multiplications [11, Algorithm 2.26].

Zero denominators. If one denominator is zero, their product is zero and the construction cannot be applied directly. To preserve normalization of the other output, a zero denominator is replaced by one during computation only. Define this temporary denominator as $\tilde{D}_i$:

$$\tilde{D}_i = \begin{cases} 1, & D_i = 0, \\ D_i, & D_i \neq 0, \end{cases} \quad i \in \{1, 2\}. \quad (1)$$

Both temporary denominators are nonzero, so the inverse I below is defined. Multiplication gives their respective inverses J_1, J_2 :

$$I = (\tilde{D}_1 \tilde{D}_2)^{-1}, \quad J_1 = \tilde{D}_2 I, \quad J_2 = \tilde{D}_1 I. \quad (2)$$

Here $J_i = \tilde{D}_i^{-1}$. If the original denominator is nonzero, the numerator is multiplied by this inverse. Otherwise, only the corresponding output is set to zero, preserving X25519's output behavior:

$$u_i = \begin{cases} N_i J_i, & D_i \neq 0, \\ 0, & D_i = 0. \end{cases} \quad (3)$$

For example, if $D_1 = 0$ and $D_2 \neq 0$, then $I = D_2^{-1}$. The first output is restored to zero, while the second is computed normally as $N_2 D_2^{-1}$. If both denominators are zero, $I = 1$ is computed before both outputs are restored to zero. No case defines or computes 0^{-1} .

Scope. The zero test reduces a field element to its integer representative in $[0, p)$ and compares all words. Masks implement the cases in the equations; the tests and output masking follow the same instruction flow regardless of input values. Each final u_i is serialized to 32 bytes. This applies to each X25519 result; it does not set the final X-Wing shared secret to zero.

Key generation and warm decapsulation each produce one X25519 result. Cold decapsulation produces two, but the current implementation completes public-key normalization and serialization inside key expansion before passing the expanded key to decapsulation. It therefore does not retain both projective results at a common normalization point and does not use paired inversion.

3.6 Memory and Addressing

X-Wing repeatedly copies and initializes fixed-size internal buffers. B8 replaces actual calls to `memcpy` and `memset` with word-based routines selected by address alignment and length. Byte operations handle addresses that do not satisfy 4 B alignment and the remainder after word processing. Branches depend only on alignment and length. Copies already inlined by the compiler are outside this scope.

Address selection. The branch and address properties of each change were reviewed separately. NTT layout changes and instruction rescheduling process predetermined coefficient positions and instructions. MVE conditional swap and zero-denominator selection use masks. The fixed-base table reads all four candidates at each digit, but B8’s sign handling retains operand-pointer selection. B8S replaces this selection with value swaps at fixed addresses in the C paths producing affine and projective results and in internal mixed addition.

Functional equivalence and assembly inspection at the modified sites support output preservation and correction of those address selections. Timing tests use specified inputs, placements, and calling conditions; Section 4.3 reports the results. These checks do not establish secret-independent address traces on every program path or absence of power and electromagnetic leakage.

4 Evaluation

4.1 Setup

Configurations The main performance evaluation compares A0 and B8 from Section 3. They use the same X-Wing draft-10 specification, inputs, API, and SHA3-256 combination order. The measured operations are key generation, encapsulation, and warm/cold decapsulation, with the timed work specified in Table 1. Symbols and actual calls of both replacement memory routines are checked outside the timed region; call counters are disabled during timing.

The main comparison includes both implementations in one Executable and Linkable Format (ELF) executable, selected by a public execution-mode value.

They share the linking and execution environment, clock, measurement caller, and inputs. Experimental selection and instrumentation code is common to both executions. We therefore compare the implementations within this shared executable rather than treating absolute cycle counts as deployment-library performance.

Platform

- Device: EK-RA8M1, Cortex-M85, 480 MHz.
- Compiler: GCC 13.2.1, `-O2`.
- Code: designated assembly in instruction tightly coupled memory (ITCM); ordinary C code and tables in flash.
- Cryptographic buffers and measurement stack: data tightly coupled memory (DTCM); measurement-control state in SRAM.
- Cycle counter: Data Watchpoint and Trace CYCCNT (DWT CYCCNT).
- Each implementation–operation combination: median of 100 measurements.
- Independent repetition: reflash the same board image and restart measurement.

On the evaluation board, writes near ITCM addresses `0x18` and `0xe0` were observed not to persist. The first 256 B was reserved as linker padding, with no executable code placed there. Before each run, flash was compared with the recorded image and known-answer tests (KATs) were performed. Measurements were not repeated on a second board without this anomaly. Appendix B gives the additional checks.

Measurements In the main comparison, the two implementations in one executable run in the following order. Each position is a *batch of 100 calls* to the operation, summarized by one median:

$$A0 \rightarrow B8 \rightarrow B8 \rightarrow A0.$$

The order is intended to reduce execution-order effects. Five independent re-flashes of the same board image were performed. Positions one and two and positions three and four in each repetition form A0/B8 pairs, giving ten paired comparisons per operation. Let $C_{A0,i}(o)$ and $C_{B8,i}(o)$ denote their 100-call medians for operation o in pair i . The cycle reduction and reported value are

$$r_i(o) = 100 \left(1 - \frac{C_{B8,i}(o)}{C_{A0,i}(o)} \right), \quad r_{\text{reported}}(o) = \frac{\lfloor 100 \min_{1 \leq i \leq 10} r_i(o) \rfloor}{100}. \quad (4)$$

The denominator is the A0 median from the same pair, and r_i is in percent. Results report the range of the ten reductions and their minimum truncated to two decimal places.

Correctness Correctness checks comprise the following items, with their applicability distinguished by experiment. The main comparison’s fixed-input checks and the later expDN checks of all three complete official vectors have the respective scopes given in Appendix B.1 and Section 4.3.

- RFC 7748 X25519 KATs and SHA3-256 KATs;
- ML-KEM-768 encapsulation/decapsulation round trips and implicit rejection of invalid ciphertexts;
- X-Wing draft-10 Appendix C test vectors;
- byte-for-byte comparison of both implementations’ public keys, secret keys, ciphertexts, and shared secrets on eight fixed inputs;
- warm/cold agreement and three low-order X25519 point inputs;
- stack-boundary integrity and agreement between the recorded image and actual flash contents.

Candidate Selection Each candidate is evaluated at three levels. First, the target function is measured to assess feasibility and the effect within that function. The candidate is then integrated into its actual caller, and full key generation, encapsulation, and decapsulation are remeasured. Finally, the effect is reassessed in a diagnostic configuration combining accepted optimizations and in the A0/B8 comparison executable.

Conditional contributions are obtained by comparing B8 with configurations that remove individual techniques in the same executable. In the current encapsulation path, the function producing both projective results directly uses the comb for its fixed-base result. The comb comparison therefore removes paired inversion as well. This is a dependency of the function integration, not a mathematical requirement of batch inversion. Removal of paired inversion alone is also measured. Dependent removal effects are not summed or combined with improvements from other images.

Timing Tests Timing diagnostics divide inputs into two classes. The fixed class holds the field under test constant; the random class varies it. Balanced interleaving tests whether their mean execution times differ. Each experiment defines the fixed and varying fields and distinguishes variation of public inputs from variation of secret inputs.

Mean differences are evaluated using Welch’s t , with individual-call cycle counts as samples. We use $|t| \geq 4.5$ as an investigation threshold without correction for multiple comparisons. The full sample and its first and second halves in collection order are tested to identify differences confined to part of a collection. No outliers are removed. A fixed–fixed negative control checks for differences despite identical inputs. An intentional-delay positive control checks detection of an inserted timing difference. Exceeding the threshold does not establish a particular instruction defect or an exploitable attack. Appendix B.2 gives the statistic and detailed conditions.

4.2 Performance

End-to-End Performance Table 3 directly compares A0 and B8 in the same executable. Following Eq. (4), the reported value is the minimum reduction across ten adjacent pairs, truncated to two decimal places. It is an observed minimum, not a confidence bound for a population of boards. Appendix B.1 describes the functional, image, and stack checks across five reflashes.

Table 3. Direct A0/B8 comparison within one executable. Ranges cover ten pairs of 100-call medians from five reflashes. The reported minimum reduction is truncated to two decimal places.

Operation	Cycles		Reduction (%)	
	A0	B8	Range	Min.
Key generation	871,798–	709,464–	18.616–18.631	18.61
	871,928	709,513		
Encapsulation	1,327,994–	1,077,048–	18.896–18.898	18.89
	1,328,019	1,077,069		
Warm decapsulation	936,330–	821,588–	12.251–12.256	12.25
	936,360	821,624		
Cold decapsulation	1,805,038–	1,530,164–	15.227–15.228	15.22
	1,805,078	1,530,202		

Ablation Each removal configuration was compared with B8 in the same executable in the order B8–removal–removal–B8. Table 4 reports the median cycle difference across ten adjacent pairs formed from 100-call batch medians. Raw logs and complete ranges, including negative values, are preserved in expDM (Appendix B.1). Because of the function dependency described above, the comb row also removes paired inversion and measures a group effect. It is not added to the paired-inversion-only removal row.

The comb/paired-inversion group had the largest removal cost in key generation, encapsulation, and cold decapsulation; MVE data-format transformations contributed the most in warm decapsulation. Paired inversion alone reduced encapsulation by about 33,405 cycles. It does not execute in the other operations, so their small differences in that row are not interpreted as its benefit. Contributions from code placement and measurement variability were not separated. Removing alignment-aware memory routines cost 5,870–13,302 cycles across operations and matched pairs. These effects are conditional on the final configuration, not independent additive components of the total gain.

Table 4. Cycle differences between removal configurations and B8 (removal – B8): medians of ten adjacent paired differences. † denotes an observed difference on a path where the technique does not execute.

Removed technique	Key			
	generation	Encapsulation	Warm	Cold
NTT and storage order	22,709.0	18,046.5	28,142.5	50,743.5
MVE data-format transformations	11,258.0	33,002.5	39,218.5	49,974.0
Comb + paired inversion	92,176.0	125,267.0	–6.0 [†]	91,852.0
Field multiplication	4,305.5	12,056.0	7,744.0	12,015.0
Field squaring	3,973.5	19,770.0	19,392.0	23,314.5
Alignment-aware memory routines	9,249.0	5,890.0	5,952.5	13,289.5
MVE conditional swap	–1.5 [†]	14,338.5	14,440.0	14,339.0
Paired inversion	1.5 [†]	33,405.0	3.5 [†]	0.5 [†]

Resources A0 and B8 were built separately. Each configuration retains only its required scalar or MVE paths and affine or projective output paths. Experimental selection branches and internal comb instrumentation were removed. Replacement memory routines were linked explicitly, and two rebuilds of each executable were byte-identical. Size-measurement executables retain the specified APIs and their dependencies while removing unused code. They include common board support package (BSP) code and deterministic test randomness. Table 5 compares the storage required to add the optimizations under the same build conditions.

Stack usage is measured with a separate caller (probe) that executes only the API and observes the maximum depth of changes to a prefilled canary pattern, the high-water mark. The storage table excludes this probe’s code and buffers.

Table 5. Standalone A0/B8 executable sizes and maximum observed API stack usage (B). Storage for the stack-measurement probe is excluded.

Item	A0	B8	B8 – A0
Nonvolatile storage (text+data)	30,760	66,452	+35,692
Static RAM (data+bss)	1,480	3,496	+2,016
Key generation stack	9,176	9,176	0
Encapsulation stack	6,664	6,664	0
Warm decapsulation stack	6,600	6,600	0
Cold decapsulation stack	10,264	10,264	0
Caller-owned expanded key	3,648	3,648	0

The 33,024 B comb table accounts for about 92.5% of the 35,692 B increase in nonvolatile storage; static RAM increases by 2,016 B. Appendix B.1 gives the

standalone resource experiment’s functional and stack checks. Stack values are observations on tested inputs, not proven worst-case bounds. Storage includes test randomness and the BSP, and is distinct from the resource usage of an entire deployed firmware image.

The 3,648 B expanded key defined in Section 2.1 belongs to the warm caller. It is therefore accounted for separately from standalone-ELF static RAM and the warm stack. Appendix B gives the memory-accounting scope.

B8S Costs This comparison evaluates the additional time and storage required by the address-selection correction in Section 3, relative to B8. Section 4.3 reports output-preservation and timing checks.

Standalone B8S removes B8’s address-selecting mixed addition and the experimental selector. Two complete builds were byte-identical. Nonvolatile storage including the common BSP is 66,668 B, an increase of 216 B over B8; static RAM remains 3,496 B. Caller buffers and runtime stack are separate. The expanded key belongs to the warm caller’s buffer or the cold function’s internal stack. All stack probes passed the bottom-guard check in the 24 KiB stack region. Observed API stack maxima were unchanged from B8; Appendix B gives the measurement scope.

B8 and B8S were compared over five independent reflashes of a new shared executable, with the public mode selected before the timed API call. Each operation and reflash used 100 alternating pairs of individual calls, identical deterministic randomness, and a preceding execution of the same operation. Outputs were compared outside the timed region. This pairs individual calls, unlike the main comparison’s ten pairs of 100-call batches. Table 6 gives the across-reflash ranges of medians for paired cycle and percentage increases. The denominator is the matched B8 call in this executable; these costs are not combined with the main A0/B8 improvement percentages. Appendix B gives the definitions and stack-measurement protocol.

Table 6. Additional B8S cost relative to B8 in the same executable. Ranges cover five reflash-specific medians, each from 100 paired calls.

Operation	Additional cycles	Increase
Key generation	12,205–12,207	1.821–1.823%
Encapsulation	9,970–9,978	1.003%
Warm decapsulation	8–24	0.001–0.003%
Cold decapsulation	12,164–12,199	0.849–0.851%

External Comparison To evaluate differences from an external implementation, B8 and RustCrypto X-Wing were linked into a shared executable distinct from the main comparison. The comparison includes both compiler and implementation differences. Rust exposes no corresponding warm-expanded API, so

Rust warm decapsulation is not compared. Code placement and calling context differ from the A0/B8 image, so these cycle counts are not substituted into Table 3. No claim is made to the best performance among public implementations, and the external implementation’s side-channel properties were not evaluated. Appendix D gives the cycle counts, key-preparation costs, and matched-call boundaries.

4.3 Security

There are two principal findings. B8S corrects the identified secret-dependent address selections, but some full-API timing signals remain unresolved. The evidence proceeds from functional and code review to full-API tests and investigation of causes.

Diagnostic executables and input distributions differ from those of the main performance comparison. Each experiment distinguishes changed conditions from observations. Appendix C gives repetition statistics and collection conditions; Appendix B.1 maps experiment identifiers to raw evidence.

Functional Checks Functional checks, code review, and timing tests address different questions. Functional checks compare outputs before and after optimization. Code review examines whether branches or access addresses vary with secret values. Timing tests investigate differences in mean execution time between two classes under specified input distributions. Evidence is reported separately because one check cannot replace another.

The A0/B8 comparison and standalone executables passed the three official vectors, comparison against an independent RustCrypto implementation on 1,024 inputs, and coefficient, point, and ciphertext boundary tests. Synthetic numerators and denominators were passed directly to paired inversion to compare cases with one, both, or neither denominator zero. These are normalization boundary tests, not a claim that all cases occur during normal encapsulation.

The initial B8 cryptographic timing tests stayed below the investigation threshold; Appendix C.1 reports full-sample statistics.

Despite these below-threshold results, code review found remaining operand-address selection based on a secret sign. Absence of branches does not make access addresses fixed. This distinction motivated the B8S correction.

Address correction. When handling the sign of a selected comb point, B8 used pointers to select which of two operands to use first. B8S fixes the addresses and swaps the values at those addresses with masks. It passes the same values to arithmetic while removing the secret sign’s influence on address selection at these sites.

Specifically, the C functions producing affine and projective results swap the selected $y - x$ and $y + x$ values before passing fixed pointers. Mixed addition swaps two 32 B temporary field elements and multiplies from fixed stack offsets. Final assembly confirmed fixed-address accesses and no conditional branches in

the swaps. Appendix C.2 gives instruction-level checks; Section 4.2 reports the additional cost.

B8S validation. B8S also passed official vectors and independent-implementation comparisons. In addition, 1,251 structured X25519 inputs, including sign and carry boundaries and low-order points, were compared against an independent cryptography/OpenSSL implementation. These checks test output preservation after changing address selection.

Timing tests placed the stack, including comb temporaries, in DTCM and cached SRAM to examine whether operand placement changes the mean timing difference between classes. Each test and placement collected 40,960 samples in total across the classes. Scalar-varying fixed-base and paired X25519 tests and negative controls remained below the threshold in the full sample and both halves at both placements. Positive controls were detected at both placements.

These results concern the modified paths and tested inputs. Full decapsulation also includes ML-KEM and the combiner, so the next tests vary the secret key while timing the complete API.

Key Variation

Seed variation. Full B8S decapsulation was tested with a fixed public ciphertext and varying secret seed. Warm calls receive a key expanded before timing; cold calls expand the seed inside the timed region. Both classes use keys unrelated to the ciphertext recipient, exercising ML-KEM implicit rejection. Warm/cold outputs were compared for every call. The initial warm test in Table 7 exceeded the investigation threshold with full-sample $t = 4.587$.

Generating a key pair from a seed also changes the public key and public matrix seed ρ . Matrix generation uses rejection sampling, accepting only candidate coefficients that satisfy the range condition. Candidate bytes come from an extendable-output function (XOF), which produces the required output length. The number of candidates processed and XOF output blocks required can vary with ρ . The initial observation alone therefore cannot separate secret-dependent effects from public-workload differences.

Field variation. To investigate this distinction, the expanded-key fields were separated. One condition fixes the embedded public key and its hash while varying only secret coefficients, the secret rejection value, and the X25519 scalar. The other fixes secret fields and varies only public fields. Combining fields from different keys in this way produces synthetic expanded keys. These exploratory inputs separate effects but do not represent the distribution of valid key pairs.

Each row contains 40,960 samples in total across the classes, and Welch’s t is dimensionless. The first and second halves partition that collection; the last row is a new independent collection of the secret-field-varying condition. A result was investigated if $|t| \geq 4.5$ in the full sample or either half.

Table 7. Welch t with 40,960 samples per row. All rows measure warm decapsulation except the cold row. Bold: $|t| \geq 4.5$. Halves partition one collection; the last row is an independent repetition.

Test	Full	First half	Second half
Varying secret seed, warm	4.587	4.563	1.997
Varying secret seed, cold	-0.695	-0.598	-0.386
Fixed-key negative control	-0.150	0.189	-0.399
Intentional-delay positive control	-12,837.291	-9,071.144	-9,083.314
Varying secret / fixed public fields	4.410	1.734	4.516
Varying public / fixed secret fields	1.767	1.112	1.389
Independent secret-varying repetition	2.687	2.433	1.368

With fixed public fields and varying secret fields, the initial collection’s second half gave $t = 4.516$. The independent repetition gave full-sample $t = 2.687$, with both halves also below the threshold. Thus the same conditions did not consistently produce signals of the same strength. The initial observation is retained, but is not described as independently reproduced. Appendix C.3 records preregistered versus exploratory tests and sample preservation.

A post hoc analysis examined the relation between public workload and timing variation in the initial seed-varying tests. The high fit shows a relationship between public workload and timing variation, but does not establish public workload as the sole cause. It also does not explain the second-half signal in the separate public-fixed test. The original samples and Welch assessment were therefore neither corrected nor reclassified as a pass. Appendix C.3 gives the model and fit.

Signal Localization In these diagnostics, the outer timer includes stack switching and argument preparation; the inner timer excludes them. Full warm and component functions are called separately, and the outer- and inner-timed full-warm entries also use separate calls. These measurements do not partition one execution: preparation history and intermediate-value addresses differ by call. ML-KEM and combiner inner timing exceeded the investigation threshold. Full warm also exceeded it with the inner timer but remained below it with the outer timer.

The full-warm inner-timing and combiner signals recurred with prediction enabled and fell below the threshold in the full sample and both halves when disabled. However, ML-KEM weakened to $t = -3.053$ even in the prediction-enabled repetition, so its entire original signal is not attributed to branch prediction.

Preparation history. The history-only condition varies the preceding key preparation by class, then restores identical inputs before the measured calls. The secret-varying condition varies actual secret inputs while keeping public fields

fixed. The history-only condition remained below the threshold. The combiner with actual secret variation reproduced a signal in both reflashes and each half (pooled $t = 9.924$). The tested preparation history alone therefore did not explain the observation. The new caller and reference buffers do not fully preserve the earlier context, so this does not exclude every preparation-history effect.

The findings identify some operations with signals and effects of execution context. The particular instructions or microarchitectural states causing the original differences remain unresolved; no whole-program constant-time pass is claimed for B8S. Appendix C preserves the component observations, prediction-setting interventions, preparation-history tests, and ML-KEM stage analysis.

4.4 Limitations

Generalizability. The main experiments used one EK-RA8M1 board. Five reflashes of the same image tested software repeatability. Measurements did not cover a second board without the ITCM write anomaly, another Cortex-M85 SoC, or another clock configuration.

The A0/B8 comparison shows the combined effect of eight changes within one implementation lineage. The RustCrypto comparison includes both implementation and compiler differences and does not rank all public implementations. Without a direct A0/B8S comparison, the available data also do not provide B8S’s improvement relative to A0.

Resource Measurement. Standalone executables include common board-support code and deterministic test randomness. Storage, static RAM, caller buffers, and runtime stack must be distinguished as in Appendix B. Stack measurements are observations on tested inputs, not proven worst-case bounds. Pure-library size and whole-product memory budgets require deployment-specific measurements. Power and energy were not measured, so cycle reductions do not imply proportional energy savings.

Security Validation. Functional checks test output agreement on specified inputs; assembly inspection examines address selection at modified sites. Timing tests cover first-order mean differences under each input distribution and calling context. The remaining B8S key-varying and follow-up signals were not attributed conclusively to particular instructions or microarchitectural states. Public-workload regression did not change the original timing assessment. Other placements, the full distribution of valid key pairs, follow-up causal diagnostics of the cold path, higher-order timing differences, and power and electromagnetic leakage require further evaluation.

5 Conclusion

The benefits of integrating established ML-KEM and X25519 optimizations with Cortex-M85 data representations, storage order, and call boundaries remained

observable across the full X-Wing API. Direct A0/B8 comparisons gave minimum reported cycle reductions of 18.61% for key generation, 18.89% for encapsulation, 12.25% for warm decapsulation, and 15.22% for decapsulation including seed expansion. The comb and paired-inversion group had the largest removal costs in key generation, encapsulation, and cold decapsulation; MVE data-format transformations had the largest cost in warm decapsulation. The 33,024 B fixed-base table accounted for most of the 35,692 B increase in nonvolatile storage.

B8S, which replaces secret-sign-dependent operand-address selection with value swaps, required a further 216 B of storage. A separate B8/B8S comparison showed cost increases of approximately 1.82%, 1.00%, and 0.85% for key generation, encapsulation, and cold decapsulation. These correction costs are not combined with A0/B8 improvement percentages.

Some key-varying tests and follow-up diagnostics retained timing signals after the address-selection correction. Branch-prediction and preparation changes altered the observations, but the causes of the original signals were not fully identified. The results therefore report integrated performance and resource costs together with the validation still required for whole-program constant-time assurance.

References

1. Abdulrahman, A., Becker, H., Kannwischer, M.J., Klein, F.: Fast and clean: Auditable high-performance assembly via constraint solving. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2024**(1), 87–132 (2023). <https://doi.org/10.46586/tches.v2024.i1.87-132>
2. Abdulrahman, A., Kannwischer, M.J., Lim, T.H.: Enabling microarchitectural agility: Taking ML-KEM & ML-DSA from Cortex-M4 to M7 with SLOTHY. In: *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*. pp. 1756–1771. ACM (2025). <https://doi.org/10.1145/3708821.3736210>
3. Arm Limited: Arm Cortex-M85 processor software optimization guide. Tech. Rep. 107950_0100_01_en, Arm Limited (2023), [https://documentation-service.arm.com/static/646bee83bc004c4648165983,issue 01](https://documentation-service.arm.com/static/646bee83bc004c4648165983,issue%201)
4. Arm Limited: Arm Cortex-M85 Processor Devices Generic User Guide (2024), [https://documentation-service.arm.com/static/662a73bfc7622e1c31ae0eb9,rlp1,document 101928_0101_07_en](https://documentation-service.arm.com/static/662a73bfc7622e1c31ae0eb9,rlp1,document%20101928_0101_07_en)
5. Arm Limited: Arm Cortex-M85 Processor Technical Reference Manual (2024), <https://documentation-service.arm.com/static/662a65926862ee0ae48937cd,rlp1>
6. Barbosa, M., Connolly, D., Duarte, J.D., Kaiser, A., Schwabe, P., Varner, K., Westerbaan, B.: X-Wing: The hybrid KEM you’ve been looking for. *IACR Communications in Cryptology* **1**(1) (2024). <https://doi.org/10.62056/a3qj89n4e>
7. Becker, H., Bermudo Mera, J.M., Karmakar, A., Yiu, J., Verbauwhede, I.: Polynomial multiplication on embedded vector architectures. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2022**(1), 482–505 (2021). <https://doi.org/10.46586/tches.v2022.i1.482-505>
8. Connolly, D., Schwabe, P., Westerbaan, B.E.: X-Wing: General-purpose hybrid post-quantum KEM. Internet-Draft `draft-connolly-cfrg-xwing-kem-10` (2026), <https://www.ietf.org/archive/id/draft-connolly-cfrg-xwing-kem-10.html>, work in Progress, published 2 March 2026; expires 3 September 2026
9. Düll, M., Haase, B., Hinterwälder, G., Hutter, M., Paar, C., Sánchez, A.H., Schwabe, P.: High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. *Designs, Codes and Cryptography* **77**(2–3), 493–514 (2015). <https://doi.org/10.1007/s10623-015-0087-1>
10. Fujii, H., Aranha, D.F.: Curve25519 for the Cortex-M4 and beyond. In: *Progress in Cryptology – LATINCRYPT 2017*. LNCS, vol. 11368, pp. 109–127. Springer (2019). https://doi.org/10.1007/978-3-030-25283-0_6
11. Hankerson, D., Menezes, A.J., Vanstone, S.: *Guide to Elliptic Curve Cryptography*. Springer (2004). <https://doi.org/10.1007/b97644>
12. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. *IACR ePrint* 2019/844 (2019), <https://eprint.iacr.org/2019/844>
13. Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security. Tech. Rep. RFC 7748, IETF (2016)
14. Lenngren, E.: X25519 for ARM Cortex-M4. GitHub repository `Emill/X25519-Cortex-M4`, <https://github.com/Emill/X25519-Cortex-M4>,

- evaluated source archived in expDM; see the evidence index. Accessed 27 September 2026
15. NIST: Module-lattice-based key-encapsulation mechanism standard. Tech. Rep. FIPS 203, NIST (2024)
 16. pqm4 contributors: pqm4 source snapshot (2026), <https://github.com/mupq/pqm4/commit/cc2c1b992b60>, commit cc2c1b992b60, 19 June 2026
 17. Renesas Electronics: RA8M1: 480 MHz Arm Cortex-M85 based microcontroller with Helium and TrustZone. <https://www.renesas.com/en/products/ra8m1>, product features and application examples; accessed 7 September 2026
 18. Reparaz, O., Balasch, J., Verbauwhede, I.: Dude, is my code constant time? In: 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1697–1702. IEEE (2017), <https://past.date-conference.com/proceedings-archive/2017/html/0786.html>
 19. RustCrypto Developers: X-Wing implementation. <https://github.com/RustCrypto/KEMs/tree/075018d88f57eb96b1a233ba1e17b0447bb9b666/x-wing>, pinned commit 075018d88f57; Cargo.lock and adapter in artifact, accessed 7 September 2026
 20. slothy-optimizer: pqmx source snapshot for the inverse NTT (2025), <https://github.com/slothy-optimizer/pqmx/commit/1eeaf854e60d>, commit 1eeaf854e60d, 26 August 2025; local adaptation archived in expDM
 21. slothy-optimizer: pqmx: Post-quantum cryptography on Arm Cortex-M. GitHub repository [slothy-optimizer/pqmx](https://github.com/slothy-optimizer/pqmx) (2026), commit 89465b9756cd (15 June 2026)

Appendices

A Implementation

A.1 NTT Layout

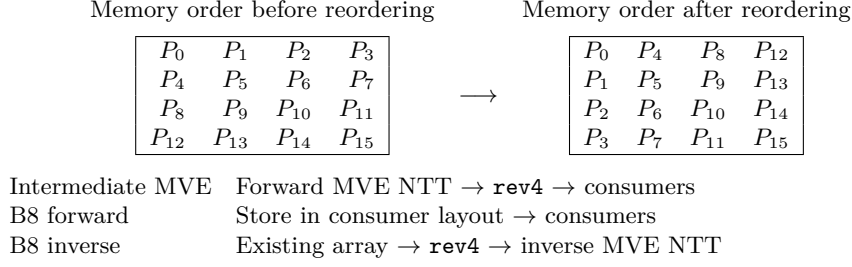


Fig. 1. Coefficient-pair layout within one block. Each grid is read by rows in memory order, and each P_j contains two 16-bit coefficients. B8’s forward NTT stores directly in the right-hand order.

A.2 Comb Representation

Width $w = 3$ means that each digit processes 3 bits. After X25519 scalar clamping, write k with signed radix-8 digits e_i :

$$k = \sum_{i=0}^{85} e_i 8^i, \quad e_i \in \{-3, -2, -1, 0, 1, 2, 3, 4\}.$$

The 86th position accommodates the final carry from the preceding digit. P_E is the implementation’s Edwards basepoint, corresponding to the standard basepoint’s Montgomery u -coordinate 9 after conversion. The table at position i stores the four points $[j8^i]P_E$ ($j = 1, 2, 3, 4$). The point matching the digit’s magnitude is selected, its sign is handled, and it is added. A zero digit selects the identity element and follows the same addition procedure.

The intermediate sum of selected points is the accumulator. The comb accumulator uses extended Edwards projective coordinates (X, Y, Z, T) , where $x = X/Z$ and $y = Y/Z$. T is an auxiliary coordinate used in mixed addition and satisfying $XY = ZT$. Table points store $(y - x, y + x, 2d_E xy)$ derived from affine coordinates. The curve is $-x^2 + y^2 = 1 + d_E x^2 y^2$, with $d_E = -121665/121666 \in \mathbb{F}_p$ [13]. Adding an affine table point to a projective accumulator is called mixed addition.

Each field element occupies eight 32-bit words, so an entry occupies $3 \times 8 \times 4 = 96$ B and the full table $86 \times 4 \times 96 = 33,024$ B.

A wider digit reduces additions but increases the full candidate scan at each position and table storage. Width $w = 3$ was selected as a trade-off using measured table-lookup costs and estimated arithmetic costs for widths 2–5. End-to-end evaluation used the actual 86-position implementation including carry handling. Full-API measurements at every width were not performed to establish a global optimum.

A.3 Provenance

Table 8. Provenance of the evaluated components and scope of our implementation work.

Component	Reused foundation	Work evaluated here
ML-KEM / Keccak	pqm4 implementation lineage	Draft-10 API integration; common baseline configuration
MVE NTT / ordering	pqmx M85 NTT and transposed-layout stores	Plantard representation, range, and storage-order adaptation; full-path integration
Data-format transformations	Standard ML-KEM encoding	MVE implementations of 11 conversion, sampling, and comparison functions
Fixed-base multiplication	Fixed-base precomputation and Lenngren field arithmetic	Width-3 tables, recoding, selection, and mixed-addition integration into X-Wing
Field multiplication / squaring	Lenngren assembly lineage	M85 stack-frame and instruction-schedule modifications
Memory routines	Established aligned copying and initialization	Word-based wrappers and verification of actual linking and calls
MVE conditional swap	Established xor-mask conditional swap	Vector implementation in the variable-base ladder
Paired inversion	Established batch inversion	Projective outputs, zero masks, and encapsulation-path integration
Validation	Known-answer checks and paired measurements	Final removal experiments, resource accounting, and validation of frozen executables

The inverse NTT uses pqmx `1eeaf854e60d`[20] and SLOTHY `ed3b034b3a3f`; the ML-KEM/Keccak source snapshot is pqm4 `cc2c1b992b60`[16]. Frozen source paths and hashes are recorded in `artifact_index.tsv`.

B Measurements

B.1 Evidence

Experiment directories are relative to `measure-harness/artifacts/`. The evidence index `artifact_index.tsv` records file paths and SHA-256 hashes. The main performance and removal results use expDM; standalone resources use expDK; functional checks use expDN; external results use expDO/expDP; and B8S validation and timing diagnostics use expDQ–expDT.

Main checks. All five independent expDM reflashes passed recorded-image/flash agreement, RFC 7748 and SHA3 KATs, full-byte equivalence on eight inputs, rejection and low-order-input checks, agreement of timing-test outputs, and stack-guard checks. This scope is distinct from expDN’s later checks of all three complete official vectors.

Resource checks. Each expDK standalone stack probe passed draft-10 vector excerpts and shared-secret checks, eight-seed round-trip/rejection tests, and zero-point warm/cold checks over two reflashes. The bottom 64 B guard of the 24 KiB canary stack remained intact. These checks are distinct from the expanded expDN validation in Section 4.3.

Board checks. Additional validation checked unchanged flash and executable ITCM bytes before and after execution, the 480 MHz clock, and cache settings. These checks support consistency of the code and placement used, but do not establish that the entire board was free of faults. Measurements were not repeated on a second board without this anomaly.

B.2 Timing Statistic

Mean differences are evaluated using Welch’s t , with individual-call cycle counts as samples. For fixed and random classes F, R , let $\overline{C}_F, s_F^2, n_F$ and $\overline{C}_R, s_R^2, n_R$ denote the sample means, unbiased variances, and sample sizes:

$$t = \frac{\overline{C}_F - \overline{C}_R}{\sqrt{s_F^2/n_F + s_R^2/n_R}}.$$

If the denominator is zero, the code records zero for equal means and signed infinity otherwise.

B.3 Memory Accounting

RAM accounting for C warm decapsulation distinguishes static RAM, the caller-owned 3,648 B expanded key, observed warm stack usage, I/O, and runtime ITCM. The cold function constructs its expanded key on its own stack, so that internal key is not added again to the cold stack value. Any separately retained caller key or application buffers require additional storage. Observed stack usage informs product-specific stack margins but is not a worst-case bound.

B.4 B8S Costs

For pair j in one reflash, let $C_{B8,j}$ and $C_{B8S,j}$ denote the call cycle counts. Their increase ΔC_j and percentage increase g_j are

$$\Delta C_j = C_{B8S,j} - C_{B8,j}, \quad g_j = 100 \frac{\Delta C_j}{C_{B8,j}} \quad (\%).$$

These quantities are computed for each of the 100 pairs before taking their medians separately. The denominator is the matched B8 call in this executable; it is neither a ratio of independently computed implementation medians nor the B8 value in the main performance table.

Standalone B8 and B8S stack usage was measured with identical API boundaries from a separate SRAM caller. Each image used two reflashes, three official seeds, 64 random seeds, and two canary patterns. Only the target API executed on the DTCM stack; an empty call used 0 B. B8S had the same observed API stack maxima as B8 in Table 5; key expansion used 5,512 B in both images. These are observations, not proven worst-case bounds. Caller-owned I/O and the warm 3,648 B expanded key are separate; the expanded key constructed inside cold is already included in its stack.

C Timing Diagnostics

Each diagnostic is a separate experiment. The statistic and investigation threshold follow Section 4.1; Appendix B.1 identifies the frozen evidence.

C.1 B8 Tests

Timing tests collected 40,960 samples in total across the two classes for each image and test, in ten balanced blocks with randomized class order and no outlier removal. The first three tests vary synthetic secret scalars with fixed public points. The ciphertext test varies public input with a fixed key and is not interpreted as a secret-key leakage test. A custom board caller follows dudect’s detection principle [18], calculating Welch’s t for the full sample and both halves with $|t| \geq 4.5$ as the investigation threshold.

C.2 B8S Tests

The scalar assembly is fully unrolled. Its 16 loads and 16 stores use fixed offsets; the mask appears only in eight data ANDs, with no conditional branches. Final assembly of both C functions and mixed addition was also inspected. This review covers modified paths and is not whole-program formal verification.

New tests moved the stack, including comb temporary operands, between DTCM and cached SRAM while preserving other buffer and code placement. Each cell in Table 10 uses 40,960 samples in balanced random order. All 327,680 samples across four tests and two placements were preserved without deletion.

Table 9. Full-sample Welch t ; each cell uses 40,960 samples.

Test	Comparison-image B8	Standalone B8
Fixed-base secret scalar	1.068	0.971
Variable-base secret scalar	-0.284	0.083
Paired X25519 secret scalar	1.369	-0.276
Public ciphertext / warm	0.836	1.423
Fixed-fixed control	-2.045	0.422
Intentional-delay control	-1,487.336	-45,243.561

Table 10. B8S full-sample Welch t by stack placement. Each test and placement uses 40,960 total samples across the two classes.

Test	DTCM t	Cacheable SRAM t
Fixed-base secret scalar	0.671	-0.451
Paired X25519 secret scalar	-0.200	-0.269
Fixed-fixed control	1.587	1.086
Intentional-delay control	-1,489.431	-215.261

ExpDQ DTCM timing collection spanned two sessions after a debugger disconnect. The 17 completed blocks were retained; code, clock, and caches were rechecked before collecting only missing blocks. Completed samples were not deleted or replaced according to their statistics.

C.3 Key Variation

In Table 7, the first four rows were preregistered; the next two were exploratory. The final 40,960 samples form an independent repetition registered after the second-half synthetic-secret signal. None of the 286,720 samples was deleted or replaced, and the full/split-half $|t| \geq 4.5$ criterion was retained. Table 7 gives all full-sample and split-half key-varying statistics.

Public workload. A post hoc analysis examined the relation between public workload and timing variation in the initial seed-varying tests. For each sample i , the counts of 3 B candidate groups T_i and 168 B XOF output blocks B_i required to generate the 3×3 matrix from public ρ were summed. Each 3 B group yields two 12-bit candidates, so T_i differs from the number of accepted coefficients. These counts describe that public matrix generation, not all work in a cold call.

An ordinary least-squares model with an intercept was fitted separately to the 40,960 warm and 40,960 cold samples. Let C_i be measured cycles and $\hat{C}_i$ the model prediction. The fitted coefficients $\beta_0, \beta_1, \beta_2$ have units of cycles, cycles per candidate group, and cycles per XOF block, respectively:

$$\hat{C}_i = \beta_0 + \beta_1 T_i + \beta_2 B_i.$$

With $\overline{C}$ denoting the measured mean, the fit is

$$R^2 = 1 - \frac{\sum_i (C_i - \hat{C}_i)^2}{\sum_i (C_i - \overline{C})^2}. \quad (5)$$

In these samples, R^2 was 0.994342 for warm and 0.996413 for cold, describing the fit of this linear model.

C.4 Components

This experiment investigated the components and calling contexts retaining signals with fixed public fields. B8S cryptographic code was fixed, with 20,480 inputs per independent reflash over two reflashes. Full warm, ML-KEM, X25519, and the combiner were *called separately* for each input. Component outputs were checked, but the measurements do not partition one full execution into consecutive intervals. Each call has its own stack switch and input preparation, changing call history and intermediate-value addresses.

The outer timer includes stack switching and argument preparation; the inner timer excludes them. The full-warm outer/inner entries in Table 11 also use separate calls. Row differences cannot therefore be summed into a total timing difference. Δ is the fixed-class mean minus the random-class mean in cycles; pooled statistics are Welch statistics over both reflashes' samples.

The history control restores identical inputs after different key preparations; the pre-expanded condition prepares a block's keys in advance.

Table 11. Component diagnostics: per-reflash and pooled Welch statistics. Each run has 20,480 samples; the pooled sample has 40,960. Δ is in cycles.

Target	Run 1	Run 2	Pooled	Δ
Full warm / outer	0.812	1.685	1.770	+0.922
Full warm / inner	8.066	9.154	12.178	+5.858
ML-KEM / inner	-9.274	-8.846	-12.813	-6.023
X25519 / inner	0.477	-1.235	-0.519	-0.017
Combiner / inner	-15.348	-14.007	-20.753	-0.896
Empty call / inner	0.000	0.000	0.000	+0.000
Same inputs / different history	1.566	0.213	1.262	+0.624
Pre-expanded key set / outer	-0.033	-0.925	-0.681	-0.333

C.5 Branch Prediction

Across 64 pairs in expDS's original-timer `same_driver` records, CCR was 0x000e0201 with prediction enabled and 0x000a0201 with prediction disabled. Collection code read the register before and after each block. These values differ

only in bit 18, the TRM’s CCR.BP control [5, Section 6.6]. The Generic User Guide instead marks this field RAZ/WI (read as zero, writes ignored) [4, p. 518]; the setting description follows the recorded readings.

The original timer binary was reused byte for byte, changing only CCR.BP with Data Independent Timing (DIT) disabled. The same 8,192 inputs per setting were tested in alternating order, with unchanged code, stack, addresses, caches, and the low-overhead branch (LOB) setting.

Table 12. Branch-prediction intervention with the same timer and 8,192 identical inputs per setting. BP denotes branch prediction; t is dimensionless, and Δ is the fixed-minus-random mean in cycles.

Target	BP on		BP off	
	t	Δ	t	Δ
Full warm / outer	2.501	+2.929	0.125	+0.014
Full warm / inner	5.571	+5.931	-0.120	-0.013
ML-KEM / inner	-3.053	-3.212	0.530	+0.045
X25519 / inner	0.939	+0.067	-0.997	-0.071
Combiner / inner	-9.573	-0.925	-1.106	-0.010
Empty call / inner	-1.000	-0.002	0.000	+0.000
Same inputs / different history	1.677	+1.889	0.956	+0.104
Pre-expanded key set / outer	-1.356	-1.491	0.548	+0.060

Paired BP-off – BP-on differences for identical inputs also showed class effects (full warm $t = -5.556$, combiner $t = 9.417$), confirming a change in observed effects with the prediction setting.

C.6 Preparation History

The next question is whether changing only preceding key preparation produces a signal. Two independent B8S reflashes separated preparation history from actual inputs. Each run collected 4,096 input rows per setting, pairing identical inputs across BP on/off. A new SRAM caller and input-comparison loops change execution history, and this effect is acknowledged.

The history condition assigns fixed/random classes only to preparation while making measured inputs identical. The secret condition varies actual secrets with public fields fixed. In the history condition, class-specific seeds are used only for key expansion and warm calls, after which measured inputs are restored to fixed values. The 1,120 B ciphertext, 3,648 B expanded key, and secret combiner inputs are compared outside timing. Public keys, public matrix seeds, and public ciphertexts are fixed. Even with identical immediate preparation, the main negative control can retain different earlier reference-generation histories. A strict control identical from seed generation onward was therefore collected separately.

Table 13. Conditions varying only preparation history or actual secret inputs, pooled across two reflashes. BP denotes branch prediction; t is dimensionless, and Δ is the fixed-minus-random mean in cycles.

Condition	Target	BP on		BP off	
		t	Δ	t	Δ
History	Full warm	-0.022	-0.025	-0.242	-0.093
History	ML-KEM	4.000	+4.764	-0.597	-0.189
History	Combiner	-2.760	-0.137	1.568	+0.026
Secret	Full warm	-2.190	-2.585	1.579	+0.665
Secret	ML-KEM	3.527	+4.265	0.100	+0.035
Secret	Combiner	9.924	+0.372	0.346	+0.007

The secret-varying combiner signal recurred in the full sample and both halves of both reflashes. However, the three history-only targets and matched history-minus-negative differences on identical inputs did not exceed the preregistered $|t| \geq 4.5$ threshold. This neither excludes every preparation history nor proves equivalence. All nine strict-negative-control channels stayed below the threshold in full and split-half class tests.

C.7 ML-KEM Stages

ML-KEM was decomposed into decryption, the G hash, a function combining re-encryption and comparison, rejection-secret generation, and final selection, passing actual intermediate values in order. Each row checked agreement with original ML-KEM and recombined X-Wing outputs. Variable preparation changes key preparation immediately before G by class; fixed preparation holds that procedure constant. G is SHA3-512.

Table 14. ML-KEM stage diagnostics after variable preparation, using separate calls and pooling two reflashes. BP denotes branch prediction; t is dimensionless, and Δ is the fixed-minus-random mean in cycles.

Stage	BP on		BP off	
	t	Δ	t	Δ
Internal decryption	1.960	+0.599	0.833	+0.033
G hash	12.625	+0.791	-0.713	-0.007
Re-encryption/comparison	-0.854	-1.018	-0.257	-0.082
Rejection-secret generation	0.960	+0.086	0.738	+0.033
Final selection	-0.027	-0.000	-0.243	-0.001
Original ML-KEM	5.728	+7.407	1.038	+0.271

After variable preparation, G reproduced $t = 8.274/9.591$ in the two runs and pooled $\Delta = +0.791$, $t = 12.625$. Both halves of each run also exceeded the threshold.

Each stage uses a separate DTCM call stack and SRAM intermediate buffers. Pooling the undecomposed ML-KEM samples from both runs gave $t = 5.728$ with variable preparation and $t = 5.991$ with fixed preparation. Separately, the runs gave $t = 4.440$ and 3.652 for variable preparation, and $t = 4.000$ and 4.472 for fixed preparation, all below the threshold. These pooled results provide weaker repetition evidence than G , which reproduced a signal in each run. The effect sign also differs from the earlier component diagnostic, so neither G nor preparation history is established as the cause of the entire original ML-KEM signal. Stage times were not added to end-to-end performance.

Changed observations under separate PMU instrumentation neither resolve the main timer’s signal nor establish a particular internal stage as its cause.

D External Comparison

D.1 Performance

To evaluate differences from an external implementation, B8 and RustCrypto X-Wing were linked into a shared executable distinct from the main comparison. RustCrypto was pinned to commit `075018d88f57` [19], using Rust 1.98.1/LLVM 22.1.8, Cortex-M85 opt-level 2, and a fixed Cargo.lock. C used GCC 13.2.1 O2. Both shared the BSP, deterministic randomness adapter, memory wrappers, and DTCM caller buffers and stack. Designated assembly resided in ITCM; ordinary C/Rust code and tables resided in flash. The comparison includes both compiler and implementation differences.

All five independent reflashes passed the three complete official Rust KAT vectors, full-output C/Rust equivalence on eight inputs including tampered and zero-point ciphertexts, public-key boundary checks, output comparisons outside each timed pair, and stack guards.

Table 15 reports across-reflash ranges of 100-call medians with alternating call order and warmed instruction and data caches. Key generation includes generation of 32 random bytes; encapsulation includes public-key validation and 64 random bytes. Rust cold decapsulation constructs its API key object before timing, but each timed decapsulate call includes internal secret-seed expansion. One-time object construction is excluded. Rust exposes no corresponding warm-expanded API, so Rust warm decapsulation is not compared.

Table 15. External API cycle counts in a new common ELF. Ranges cover 100-call medians across five independent reflashes.

Operation	C/assembly B8	RustCrypto + adapter
Key generation	668,938–668,957	3,565,942–3,566,202
Encapsulation	993,006–993,096	4,675,357–4,676,104
Cold decapsulation	1,432,570–1,432,614	7,378,017–7,378,504

The comparison limits in Section 4.2 apply to these results.

D.2 Key Preparation

A separate SRAM caller measured Rust key-object construction, a cold call on an existing object, and their sequential seed-to-secret combination in the external comparison binary. Table 16 gives ranges of operation-specific 100-sample medians across five independent reflashes. The three official vectors and operation order were cycled, and the same operation was executed once immediately before each measurement. The timed region includes stack-switching call overhead and the small frame of the seed-to-secret wrapper. This measures software first-use cost with warmed caches, not power-on or cold-cache latency. It exposes object-preparation cost omitted from Table 15; differing inputs and callers prevent substitution into or simple addition to the earlier cycle counts.

Table 16. Rust key-preparation and seed-to-secret cycle counts in the external-comparison binary.

Operation	Median range
Key-object construction	3,522,857–3,523,102
Cold with existing object	7,301,388–7,302,191
Seed-to-secret (object construction + cold)	10,824,364–10,824,946