

Optimized NEON Implementation of MAYO on Apple M1

Minwoo Lee¹ , Minjoo Sim¹ , Siwoo Eum¹ , and Hwajeong Seo² 

¹ Department of Information Computer Engineering, Hansung University, Seoul, Republic of Korea

{minunejip, minjoos9797, shuraatum}@gmail.com

² Department of Convergence Security, Hansung University, Seoul, Republic of Korea
hwajeong84@gmail.com (corresponding author)

Abstract. MAYO is one of nine third-round candidates in NIST’s process for additional post-quantum signatures. Its third-round specification of August 2026 changed the level-1 parameters and added a hash-derived linear term Λ to the verification equation, so earlier optimised implementations no longer match it. We present an optimised NEON implementation of MAYO Round 3 for the Apple M1 and compare it with the upstream NEON code on the same machine. A generator emits the sixteen GF(16) kernels of each parameter set as assembly whose output is byte-identical to that of the upstream intrinsics. Apart from the Λ kernel, the large kernels reach 86–96 % of the speed an operation-count model predicts. A transposed constant-time echelon form needs 55–70 % fewer cycles than the upstream one. An eight-way interleaved AES expands the public matrices at 0.18 instead of 0.43 cycles per byte. The linear term is expanded with NEON; with our changes off, its portable-C AES derivation takes 10–22 % of signing and 18–37 % of verification. As medians over seven function alignments, compact-key signing is 1.99–2.12 times faster than upstream on all four sets, and verification 2.11–2.62 times. A constant-time campaign with matched null runs finds the linear-algebra tests indistinguishable from their nulls, at 25 % power for a twofold dispersion. The MAYO₂ signature test differs. Verification, which holds no secret, shows a key-fixed timing effect on public data, upstream as well; the signature test does not separate it from a secret one. All measurements are on one M1 Pro.

Keywords: MAYO · Post-quantum signatures · ARMv8 NEON · Apple M1 · Constant-time implementation

1 Introduction

NIST’s process for additional post-quantum signature schemes, opened in 2022 [28], moved fourteen candidates into its second round in October 2024 [2] and nine into its third round in May 2026 [3]. MAYO [8] is among the nine. It is a multivariate scheme in the Oil and Vinegar family [21], and it signs by solving a linear system over $\mathbb{F}_{16}$ whose matrices are expanded from short seeds with AES.

Its oil space is smaller than the number of equations, which keeps its public keys short: 1,456 bytes for MAYO₁. The third-round specification of August 2026 [11] moved the two level-1 parameter sets, MAYO₁ to $(n, m, o, k) = (88, 80, 8, 10)$ and MAYO₂ to $(86, 64, 13, 5)$. In response to improved claw-finding attacks it also added a linear term to the verification equation, with a matrix A derived per signature from the message hash by a second AES-128-CTR keystream. The same specification made signing without the materialised matrix L the default for the compact secret key (Section 2.1). The upstream library [34] implements all of this, including a NEON backend, in the version pinned for this paper.

Optimised implementations of MAYO exist for AVX2 and the Cortex-M4 [9], for other Cortex-M cores [17] and for hardware [18]. The wider Oil and Vinegar family has been implemented with NEON [12] and AVX2 [4], and related multivariate schemes have been ported to ARMv8 as well [22, 23]. All of these, however, predate the third round. For Round 3 the only optimised ARMv8 figures we are aware of are the designers’ own NEON numbers, measured on an Apple M3 [11]. To our knowledge, the only ARMv8 implementation of Round 3 is the upstream backend itself, whose kernels, echelon form and signing path were reworked for the round. The Apple M1 is a natural target for a closer look: its Firestorm core carries the Cryptography Extensions and the SHA-3 extension, issues four SIMD operations per cycle, and has surprised constant-time code before [13, 31]. This paper reports what an implementation written for this core achieves against the upstream NEON code on the same machine.

Against that baseline, with the same compiler and cycle counter, we report medians over seven function alignments. Our implementation signs 1.99–2.12 times faster from the compact key, verifies 2.11–2.62 times faster and generates keys 1.60–1.79 times faster on the four parameter sets (Section 5.2).

Contributions. Our contributions are the following; the first three account for nearly all of that gain.

- **Generated GF(16) kernels** (Section 3). A Python generator emits the sixteen GF(16) bilinear kernels of every parameter set as assembly with fully unrolled bodies. Their output is byte-identical to that of the upstream intrinsics on pseudo-random inputs, and the large kernels take 21–39 % fewer cycles in isolation (6.5–39 % over all sixteen).
- **Transposed echelon form** (Section 4.1). A rewritten constant-time echelon form writes its result transposed for a column-wise back substitution and needs 55–70 % fewer cycles than the upstream one, which was itself vectorised.
- **Key expansion and signing structure** (Sections 4.2 and 4.3). The key-expansion path keeps eight AES blocks in flight and expands A with NEON; in our tree with every change off, the portable C derivation of A takes 10–22 % of signing and 18–37 % of verification. An entry barrier guards it against a layout-dependent slow state of the AES loop, which we characterise. The signing composite is arranged per parameter set.
- **Constant-time evaluation** (Section 5.3). A campaign with matched null runs and paired permutation tests finds the arms that hold only secret data indistinguishable from their nulls. The one contrast that differs, the MAYO₂

signature, coincides with a timing effect on public data, largely fixed by the key, that the test does not separate from a secret component. Upstream verification shows such an effect too.

Our Keccak, unlike the portable upstream one, uses the SHA-3 extension [7]; no step measures it, and single builds put its effect at 1.3% at most.

Everything was measured on one Apple M1 Pro with one compiler version, so the results describe that core only; the designers’ Apple M3 figures are context, not a comparison. Every number comes from a ledger generated from the logs. The headline tables come from one session on a tree that passed every correctness check; the diagnostic builds taken afterwards are marked as such (Section 5.1). The implementation, the scripts and the session logs are publicly available.³ Section 2 recalls the scheme and the upstream backend, Section 3 describes the kernel generator, Section 4 the echelon form, the AES path and the signing composite, and Section 5 the measurements.

2 MAYO Round 3 and Its NEON Implementation

2.1 The scheme and the Round 3 parameters

MAYO [8] is an Oil and Vinegar signature scheme whose oil space is smaller than the number of equations, so that the public map $\mathcal{P}: \mathbb{F}_{16}^n \rightarrow \mathbb{F}_{16}^m$ alone cannot be inverted. Signing instead inverts the *whipped* map $\mathcal{P}^*$, which combines $\mathcal{P}(x_i)$ and the differentials $\mathcal{P}'(x_i, x_j)$ of k inputs with fixed $m \times m$ matrices E_{ij} . The public key consists of a seed and $P_3 = \text{Upper}(O^\top P_1 O + O^\top P_2)$, where the secret oil space is $O \in \mathbb{F}_{16}^{v \times o}$ with $v = n - o$. The operator *Upper* folds each entry below the diagonal onto its mirror entry above it. The matrices P_1 (upper triangular, $v \times v$) and P_2 ($v \times o$) are expanded from the seed with AES-128-CTR [29]. Every entry is an m -vector of $\mathbb{F}_{16}$ elements that occupies $m/2$ bytes. The secret key is a 24- to 40-byte seed from which O and the public seed are derived with SHAKE256 [27].

To sign, the implementation hashes the message together with a salt into a target $t \in \mathbb{F}_{16}^m$ and samples k vinegar vectors $v_i \in \mathbb{F}_{16}^v$. It computes the m -vectors $v_i^\top P_1 v_j$ and $M_i = v_i^\top L$ for $L = (P_1 + P_1^\top)O + P_2$. The whipping schedule assembles a linear system of m equations in ko unknowns, which is solved in constant time by a row echelon form followed by back substitution (Section 4.1). A rank-deficient system causes a restart with fresh vinegar; the restart is public because the number of attempts is independent of the final signature [11]. The signature is the salt and the vectors $s_i = (v_i + O x_i \parallel x_i)$. Verification expands P_1 and P_2 from the public seed, evaluates the whipped map on s and compares the result with t (Algorithms 1 and 2).

Round 3 of the specification [11] changed the scheme in three places. First, the two level-1 parameter sets moved: MAYO₁ from $(n, m, o, k) = (86, 78, 8, 10)$ to $(88, 80, 8, 10)$, and MAYO₂ from $(81, 64, 17, 4)$ to $(86, 64, 13, 5)$ [10]. MAYO₃

³ <https://anonymous.4open.science/r/mayo-m1-r3-FDC2>

Table 1. Round 3 parameter sets, sizes and keystream lengths; KiB rounded up.

set	n	m	o	k	pk (B)	sig (B)	esk (KiB)	epk (KiB)	$P_1 \ P_2$ (B)	Λ (B)
MAYO ₁	88	80	8	10	1,456	464	152	153	155,200	3,520
MAYO ₂	86	64	13	5	2,928	239	115	117	116,800	2,752
MAYO ₃	118	108	10	11	2,986	681	368	371	376,164	6,372
MAYO ₅	154	142	12	12	5,554	964	823	828	841,847	10,934

and MAYO₅ kept theirs; Table 1 lists the resulting sizes, with esk and epk the expanded secret and public keys. Second, the verification equation acquired a linear term. A matrix $\Lambda \in \mathbb{F}_{16}^{m \times n}$ is expanded per signature with AES-128-CTR under a 16-byte key and a 16-byte IV taken from the same SHAKE256 output that yields t . Verification then checks $\mathcal{P}^*(s) + \sum_i E_{i0} \Lambda s_i = t$. The specification motivates the term by improved claw-finding attacks, which lower the attack cost by up to a factor 2^{24} for MAYO₅. The specification expects the term to cost almost nothing. Third, the whipping schedule became an MDS schedule: every square block submatrix of the symmetric block matrix (E_{ij}) is now invertible.

Round 3 also changed the implementation strategy. Signing from the compact secret key no longer materialises L but folds its products into the signature computation. The specification credits this to Hiroshi Amagasa and reports it as 20–30 % more efficient for a single signature from a compact key [11]. Signing from a pre-expanded key still uses a materialised L . Every comparison in this paper is against the upstream implementation that contains this change [34].

2.2 The upstream NEON implementation

The upstream library [34] ships a NEON backend written in C with intrinsics. It keeps every m -vector nibble-packed, two field elements per byte, in $\lceil m/32 \rceil$ 16-byte registers. In memory, upstream stores an m -vector as 64-bit limbs of sixteen nibbles each. Multiplication by a scalar is a table lookup [12] on the odd (low) and the even (high) nibble plane of a source register, in upstream’s one-based naming. Two scalars share one 16-byte table, the products of the first in its low nibbles and those of the second in its high nibbles. After the accumulation, SLI and SRI recombine the two planes into packed columns. Figure 1 shows the arrangement in the form our kernels use. The representation follows the nibble-sliced design of the earlier AVX2 and Cortex-M4 implementations [9]. P_1 is read directly in the packed layout of the keystream, whose stride of $m/2$ bytes is not a multiple of 16 bytes for MAYO₁, MAYO₃ and MAYO₅. P_2 , in contrast, is spread to the limb stride, except in verification, which reads it packed as well.

Algorithms 1 and 2 give signing from the compact secret key and verification as our implementation performs them, at the level of the kernel calls. The upstream backend has the same structure and differs only where the next sections say so and in Keccak, for which our tree uses the SHA-3 extension [7] in place of upstream’s portable code. Section 5.2 bounds its effect. All arithmetic is a small set of kernels that multiply a matrix of m -vectors by a matrix of scalars, named

Algorithm 1 Sign from the compact secret key

Require: seed_{sk}, message μ ; route B (MAYO₁, MAYO₃) or A (MAYO₂, MAYO₅)

- 1: (seed_{pk}, O) $\leftarrow$ SHAKE256(seed_{sk}) $\triangleright O \in \mathbb{F}_{16}^{v \times o}$
- 2: $P_1 \parallel P_2 \leftarrow$ AES-128-CTR(seed_{pk}), both left packed $\triangleright$ Section 4.2
- 3: **route B only:** $L' \leftarrow P_1 O + P_2$, in place of P_2 $\triangleright P_1 O$, Section 3
- 4: $h \leftarrow$ SHAKE256(μ); salt $\leftarrow$ SHAKE256($h \parallel R \parallel$ seed_{sk}), R random; $t \parallel k_A \parallel iv_A \leftarrow$ SHAKE256($h \parallel$ salt) $\triangleright t \in \mathbb{F}_{16}^m$
- 5: $\Lambda = (\Lambda_v \parallel \Lambda_o) \leftarrow$ AES-128-CTR(k_A, iv_A) $\triangleright v$ vinegar, o oil m -vectors; Section 4.2
- 6: **for** ctr = 0, 1, ..., 255 **do**
- 7: $V \parallel r \leftarrow$ SHAKE256($h \parallel$ salt $\parallel$ seed_{sk} $\parallel$ ctr) $\triangleright$ rows $v_0, \dots, v_{k-1}$ of V ; $r \in \mathbb{F}_{16}^{ko}$
- 8: $P_v \leftarrow P_1 V^\top$; column 0 of $P_v \leftarrow \Lambda_v$ $\triangleright P_1 V^\top$, Section 3
- 9: $Q \leftarrow V P_v$ $\triangleright V P_1 V^\top$, Section 3; $Q_{ij} = v_i^\top P_1 v_j$, Q_{i0} also holds $\Lambda_v v_i$
- 10: **route A:** $M \leftarrow V P_2$; $U \leftarrow P_v + P_1^\top V^\top$; $M \leftarrow U^\top O$ $\triangleright V P_2, P_1^\top V^\top, U^\top O$
- 11: **route B:** $M \leftarrow V L'$; $M \leftarrow P_v^\top O$ $\triangleright V L, U^\top O$; routes: Section 4.3
- 12: $M_0 \leftarrow \Lambda_o$ $\triangleright$ block 0; $\Lambda_v O$ arrives via P_v (line 8)
- 13: $y \leftarrow t - \sum_i E_{ii} Q_{ii} - \sum_{i < j} E_{ij} (Q_{ij} + Q_{ji})$ $\triangleright$ whipping schedule
- 14: $A \leftarrow$ the matrix of $x \mapsto \sum_i E_{ii} M_i x_i + \sum_{i < j} E_{ij} (M_i x_j + M_j x_i)$ $\triangleright$ Section 4.3
- 15: $\tilde{A} \leftarrow$ ECHELONFORM [$A \mid y - Ar$], written transposed $\triangleright$ Section 4.1
- 16: **if** rank $A < m$ **then** clear M, A ; **continue** $\triangleright$ public restart; Section 4.1
- 17: $x \leftarrow$ BACKSUBSTITUTION($\tilde{A}, r$); **break** $\triangleright$ Section 4.1
- 18: **end for**
- 19: **if** no attempt succeeded **then return** $\perp$
- 20: **return** $\sigma \leftarrow (s_0, \dots, s_{k-1}, \text{salt})$ with $s_i \leftarrow (v_i + O x_i \parallel x_i)$

Algorithm 2 Verify from the compact public key

Require: cpk = (seed_{pk}, P_3), message μ , signature $\sigma = (s_0, \dots, s_{k-1}, \text{salt})$

- 1: $P_1 \parallel P_2 \leftarrow$ AES-128-CTR(seed_{pk}), both left packed $\triangleright$ Section 4.2
- 2: $P_3 \leftarrow$ the upper-triangular P_3 of cpk, unpacked to the m -vector stride
- 3: $t \parallel k_A \parallel iv_A \leftarrow$ SHAKE256(SHAKE256(μ) $\parallel$ salt) $\triangleright t \in \mathbb{F}_{16}^m$
- 4: $\Lambda = (\Lambda_v \parallel \Lambda_o) \leftarrow$ AES-128-CTR(k_A, iv_A) $\triangleright n$ m -vectors; Section 4.2
- 5: $S \leftarrow (s_0 \parallel \dots \parallel s_{k-1}) \in \mathbb{F}_{16}^{n \times k}$; $S_1, S_2 \leftarrow$ its v vinegar and o oil rows
- 6: $W_1 \leftarrow P_1 S_1 + P_2 S_2$ $\triangleright P_1 S_1 + P_2 S_2$, Section 3; P_2 read packed
- 7: $W_2 \leftarrow P_3 S_2$ $\triangleright P_3 S_2$, Section 3
- 8: column 0 of $W_1 \leftarrow \Lambda_v$; column 0 of $W_2 \leftarrow \Lambda_o$ $\triangleright n$ m -vector additions
- 9: $Q \leftarrow S_1^\top W_1$; $Q \leftarrow S_2^\top W_2$ $\triangleright V P_1 V^\top, S_2^\top (P S)_2$; Q_{i0} also holds $\Lambda_s s_i$
- 10: $y \leftarrow \sum_i E_{ii} Q_{ii} + \sum_{i < j} E_{ij} (Q_{ij} + Q_{ji})$ $\triangleright$ whipping sum = $\mathcal{P}^*(s) + \sum_i E_{i0} \Lambda_s s_i$
- 11: **return** accept **if** $y = t$, reject otherwise

in the comments of the algorithms and listed in Tables 7 and 8 of the appendix. On the compact-key paths the linear term needs no kernel of its own. In verification, A is added into column 0 of W_1 and W_2 , the two halves of PS (line 8 of Algorithm 2). The multiplication by $S^\top$ then produces the terms As_i from products it computes anyway, and the whipping multiplies them by $E_{0i} = E_{i0}$. In signing, the vinegar part A_v rides in column 0 of $P_1V^\top$, and the oil part A_o in the first block of M . On the compact-key path the backend signs without L . It accumulates $P_1^\top V^\top$ on top of $P_1V^\top$ and multiplies the result by O . This is the implicit- L strategy of Round 3 and route A of Algorithm 1.

Two parts of the backend leave throughput unused. The AES-CTR loop encrypts one block per iteration and runs at 0.429 cycles per byte. The A keystream falls back to the bitsliced portable C AES of BearSSL (Section 4.2). Replacing the AES path alone removes 57% of the public-key expansion and 37–56% of verification (Section 5.2). Section 3 turns first to the kernels; Section 4 then treats the echelon form (Table 3), the AES path and the signing composite.

3 Generating the GF(16) Kernels

Every multiplication of a matrix of m -vectors by a matrix of scalars in key generation, signing and verification is taken by one of sixteen kernels (Algorithms 1 and 2). A Python generator emits each kernel of each parameter set as AArch64 assembly; unlike SLOTHY [1], it applies no search-based scheduling. We call these kernels the generated tier. Each kernel body is fully unrolled over chunks, the 16-byte slices of an m -vector, and over column pairs. Loop bounds, strides and offsets over rows and source pairs are constants or closed-form functions of the row counter, which takes the compiler’s register allocator out of the loop. The kernels compute every stored product of the upstream intrinsics in exact GF(16) arithmetic, so their output is byte-identical (Section 3.3). The gain lies in register allocation and addressing, the three-input EOR3 fold, the merged tail and store-first accumulation, whose shares we did not separate. The fourteen large kernels, all but P_3S_2 and $S_2^\top(PS)_2$ of verification, take 21–39% fewer cycles in isolation. With the per-set signing route and the packed- P_2 first touch (Section 4.3), the tier takes 11–28% off signing (Table 2).

3.1 Kernel shapes and the accumulator budget

The sixteen kernels come in three shapes. Eight sweep a row of P_1 or P_3 , both upper triangular, against o or k scalar columns. Seven multiply a rectangular matrix of m -vectors by k or o scalar columns, counting both variants of VL , read-modify-write and store-first. One handles the single product A_vO of the pre-expanded-key path. The fused $(P_1 + P_1^\top)O$ and the read-modify-write VL are tested but not called by the measured build; the signing route and m select among the others. In the inner step all share (Figure 1), a source m -vector meets one column pair, that is, one 16-byte table, and contributes two register-wide products to two accumulators. The nibble planes of a chunk are prepared once

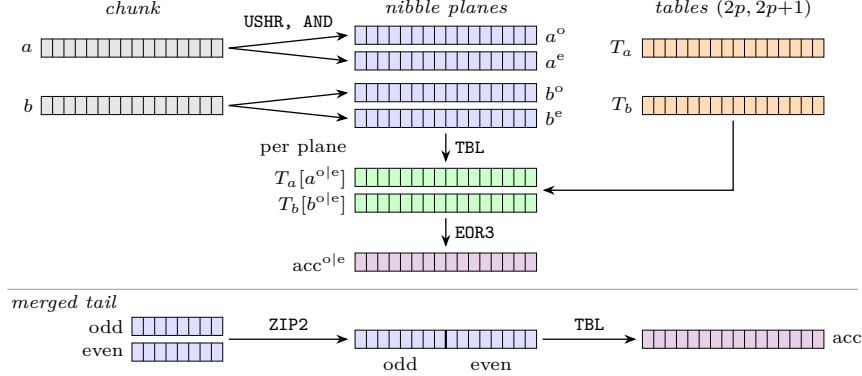


Fig. 1. Inner step of a generated kernel, and the merged tail.

per source and looked up in every table of the row. Two sources are folded with one EOR3 from the SHA-3 extension [6]; without it, the tier uses the intrinsics. The first source of a row writes its accumulators, so there is no zeroing pass, except in the two $(P_1 + P_1^\top)O$ kernels, whose source groups can be empty.

The accumulators of a row, one per column pair, nibble plane and chunk, stay in v0 to v22 (Figure 2), the registers left after the mask, four plane, two table and two scratch registers. On this core, whose SIMD file issues four operations per cycle [20], a microbenchmark sustains 3.97 EOR and 3.98 TBL per cycle but only 2.97 loads and 1.95 stores. With the nibble preparation amortised over a row, the kernels are bound by SIMD issue, not memory. A spilled accumulator adds a load and a store but no SIMD operation, so spills cost little. Accumulators beyond these 23 registers live in stack slots, updated through a register that is provably dead there. This happens only in the flat $(P_1 + P_1^\top)O$ kernels of MAYO_{2,3,5}, which take one pass per row, and in MAYO₁’s k -kernels, where blocking would rerun the source loop for a short tail pass measured to gain nothing.

3.2 Chunk-blocked passes and the merged tail

Each remaining kernel whose flat allocation would spill processes a row in passes over groups of chunks (Algorithm 3). The whole source loop runs for the chunks of one group, whose accumulators then fit in registers, and those chunks are disentangled before the next group starts. Groups are formed greedily in chunk order with at most 21 accumulators, the 23 registers less v18 and v19. These two then hold the tables of a second column pair, so the tables of two pairs arrive with one LDP each. Only table loads and loop control repeat per pass, and blocked kernels use no stack slots.

On MAYO₁, MAYO₃ and MAYO₅, the packed m -vectors of 40, 54 and 71 bytes are not multiples of 16. The final 16-byte load therefore overlaps the previous one, and a straightforward kernel multiplies the overlapping bytes twice. The generator instead gathers the 8, 6 or 7 unique tail bytes into one register, odd nibbles in its low half and even nibbles in its high half (the ZIP2 in Figure 1).

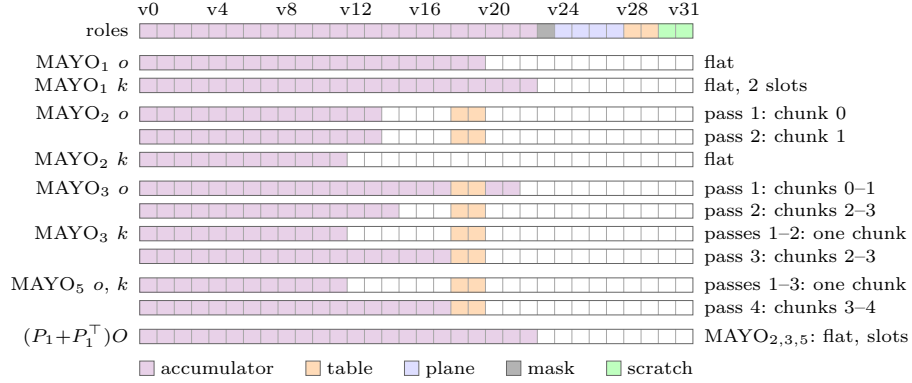


Fig. 2. Register allocation of the generated kernels, per set and pass.

Algorithm 3 Schedule of a generated GF(16) kernel

Input: per output row r : sources $a_1, \dots, a_N$ ($N \geq 1$), m -vectors in 16-byte chunks $a_i[\kappa]$, $\kappa = 0..C-1$; tables $T_{i,p}$ per source i and column pair p , entry x holding $x s_{i,2p}$ in its low and $x s_{i,2p+1}$ in its high nibble; accumulators $\mathcal{A}_{p,\kappa}^o, \mathcal{A}_{p,\kappa}^e$ per column pair and chunk, only $\mathcal{A}^o$ for the merged tail; blocked: over 23 accumulators on MAYO_{2,3,5}, except $(P_1 + P_1^T)O$ (flat, zeroed first); flat kernels spill any excess to stack slots

Output: $R_r[j] \leftarrow \bigoplus_i s_{i,j} a_i$ for each column j ; stored, or folded into existing data when the kernel accumulates

```

1:  $\mathcal{G} \leftarrow$  greedy chunk groups of  $\leq 21$  accumulators if blocked, else one group
2: for each output row  $r$  do
3:   for each group  $g \in \mathcal{G}$  do  $\triangleright$  one pass: the whole source loop over the chunks of  $g$ 
4:     for  $i \leftarrow 1, 3, 5, \dots$  up to  $N$  do  $\triangleright$  two sources per step; no  $a_{N+1}$  for odd  $N$ 
5:       for each chunk  $\kappa \in g$  do  $\triangleright$  nibble planes prepared once per source
6:          $a^o, a^e \leftarrow a_i[\kappa] \wedge 0x0F$ ,  $a_i[\kappa] \gg 4$ ; likewise  $b$  from  $a_{i+1}[\kappa]$   $\triangleright$  AND, USHR
7:         if merged tail:  $a^o \leftarrow$  upper halves of  $a^o, a^e$ ; likewise  $b^o$   $\triangleright$  ZIP2
8:         for each column pair  $p$  do
9:           load  $T_{i,p}, T_{i+1,p}$   $\triangleright$  LDR; LDP of two pairs when v18/v19 are free
10:           $t \leftarrow T_{i,p}[a^o]$ ,  $u \leftarrow T_{i+1,p}[b^o]$   $\triangleright$  TBL; no  $u$  without  $a_{i+1}$ 
11:           $\mathcal{A}_{p,\kappa}^o \leftarrow \mathcal{A}_{p,\kappa}^o \oplus t \oplus u$   $\triangleright$  EOR3, or EOR;  $i = 1$ :  $\leftarrow t \oplus u$  unless zeroed
12:          unless merged tail: likewise  $a^e, b^e \rightarrow \mathcal{A}_{p,\kappa}^e$   $\triangleright$  even plane
13:        end for
14:      end for
15:    end for
16:    for each column pair  $p$ , chunk  $\kappa \in g$  do  $\triangleright$  disentangle: SLI, SRI; STR
17:       $R_r[2p][\kappa] \leftarrow (\mathcal{A}^o \wedge 0x0F) \vee (\mathcal{A}^e \ll 4)$ ,  $R_r[2p+1][\kappa] \leftarrow (\mathcal{A}^e \wedge 0x0F) \vee (\mathcal{A}^o \gg 4)$ 
18:      merged tail:  $\mathcal{A}^e \leftarrow$  halves of  $\mathcal{A}^o$  swapped; as above, 8-byte stores  $\triangleright$  EXT, STR
19:    end for
20:  end for
21: end for

```

Table 2. Generated kernels against upstream; model range excludes the A kernel.

set	large kernels, isolated		tier, end to end	
	change	% of model	sign	verify
MAYO ₁	−36.6 to −28.8 %	86.3–92.3 %	−24.7 %	−24.0 %
MAYO ₂	−26.0 to −20.9 %	92.1–96.3 %	−10.8 %	−12.2 %
MAYO ₃	−39.3 to −30.7 %	89.4–95.3 %	−28.4 %	−27.8 %
MAYO ₅	−37.4 to −32.4 %	91.0–95.7 %	−27.5 %	−25.2 %

One lookup per table then feeds one accumulator, whose disentangled result is stored as an 8-byte tail. The stored products are unchanged, so the output stays byte-identical, while one lookup and one accumulator per column pair are saved on every source of every row.

3.3 Byte identity, wiring and speed

A test program links the upstream C kernels and the generated assembly, fills the operands from a fixed-seed pseudo-random generator and compares the results byte for byte. Each kernel runs once with a random and once with a zero accumulator prefill. Accumulating kernels see the prefill on both sides. Kernels that store first, fused ones included, are compared with a C kernel on zeros or on the unpacked P_2 , showing that their output ignores the prefill. The packed VP_2 kernel’s padding bytes are masked. All sixteen kernels agree on all four sets in both runs. The generator reproduces its own output byte for byte, and the full scheme built on the tier passes the known-answer tests (Section 5.1).

Three edits wire the tier in. Macros re-point the kernels of the upstream signing composite, which remains as the fallback. The P_3 , materialised- L and verification composites of the NEON header are replaced with versions calling the generated kernels. Our own signing, key-generation and compact-expansion composites (Section 4.3), the path the measured build takes, call them too. Because the first kernel into a buffer stores, the clears of $P_1V^\top$, PS and $S^\top PS$ disappear, including that of the whole PS buffer in verification.

The tier columns of Table 2 compare placement ensembles over seven function alignments (Section 5.1). Tables 7 and 8 in the appendix list all sixteen kernels. The model approximates the operations of our own formulation. Per source m -vector and register, it has one shift and one mask for the nibble planes, plus two TBL and one EOR3 per column pair, that is, $3\lceil c/2 \rceil + 2$ operations for c columns. It runs over 2.5, 2, 3.5 and 4.5 effective registers for the four sets, the merged tail counting as half. At four SIMD operations per cycle, it gives the time of a kernel that did nothing else. The thirteen large kernels other than the A kernel reach 86–96 % of it on every set. This is not a hardware bound, but it suggests that further gains would need fewer operations, not a better schedule. The two small kernels of verification reach 57–80 %, which we attribute to rows too short to amortise the nibble preparation. Together they take 1.29–2.45 % of a verification. The A kernel reaches 79–92 %, below the large kernels on all

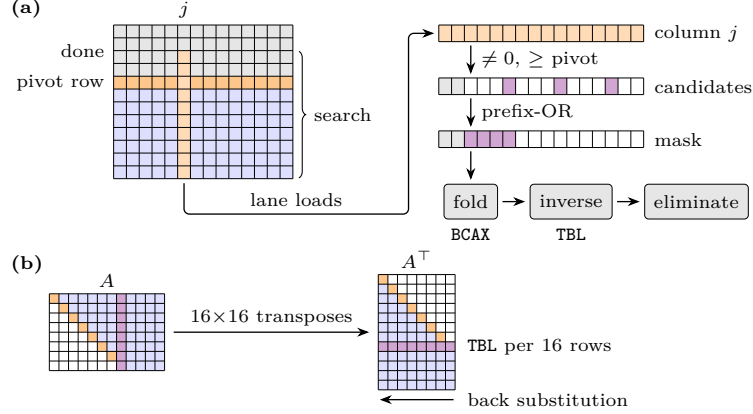


Fig. 3. Transposed echelon form: column step, write-back and back substitution.

sets but MAYO₅. MAYO₂ gains the least. Its vectors of $m = 64$ are exactly two registers with no tail to merge, its o -kernels need two passes, and its k -kernels amortise each prepared chunk over only three column pairs.

4 Linear Algebra and Key Expansion

Three changes outside the kernel tier account for the rest of the gain. The echelon form that solves the signing system is rewritten around a transposed output. The AES-based expansion of the public matrices and of the linear term is rewritten around the throughput of the AES units. The composite that assembles the signing system is rearranged per parameter set. Section 5.2 measures each change.

4.1 Transposed echelon form and vectorised back substitution

Signing solves $Ax = y$ through the augmented $m \times (ko + 1)$ matrix $[A | y]$ over $\mathbb{F}_{16}$, one field element per byte. The pivot positions of this matrix are secret, so the solver must touch the same bytes in the same order whatever the matrix contains. The upstream Round 3 code already does this well. It selects pivots with masks computed directly from the matrix, and it inverts the pivot through a table lookup instead of an exponentiation chain. It also back-substitutes on a copy of the matrix transposed in 16×16 blocks. In the tree with every change off, the solver, back substitution included, takes 14–19% of a signature, comparable to the AES-CTR expansion. Nothing outside the GF(16) products and the AES expansions costs more.

Our echelon form keeps the upstream row-wise elimination and changes how each column step is carried out. Figure 3 and Algorithm 4 give the step. The pivot search gathers the pivot-column bytes of sixteen consecutive rows into one register with lane loads at public addresses. It compares their row indices against the secret pivot-row counter broadcast into a register, and the bytes against zero.

Algorithm 4 Transposed constant-time echelon form, column step j

Input: public column $j \in \{0, \dots, ko\}$ of $[A \mid y]$; each row of A held right-aligned in $R = \lceil (ko+1)/16 \rceil$ registers $A[i][0..R-1]$; the pivot-row registers $Piv[0..R-1]$; the secret pivot-row counter p (pivots found so far); multab(v) is the table of the sixteen $\mathbb{F}_{16}$ multiples of the element in the lanes of v , so that $\text{TBL}(\text{multab}(v), X) = v \cdot X$ lane-wise; inv₁₆ is the 16-entry $\mathbb{F}_{16}$ inverse table; the index ρ ranges over registers $s..R-1$

- 1: $s \leftarrow$ register of column j (one body per s , chosen by a **switch** on j); $Piv[\rho] \leftarrow 0$;
 $\pi \leftarrow 0$; $found \leftarrow 0$ $\triangleright$ public register start
- 2: $w_0 \leftarrow \max(0, j+m-ko-1)$; $w_1 \leftarrow \min(m-1, j)$; $w_2 \leftarrow \min(m-1, w_1+32)$ $\triangleright$ public
- 3: **for** $r_0 = w_0, w_0+16, \dots$ **while** $r_0 \leq w_2$ **do** $\triangleright$ pivot search, sixteen rows per vector
- 4: $b_i \leftarrow A[r_0+i][j]$ for $i = 0..15$ $\triangleright$ LD1 lane loads, public addresses
- 5: $c_i \leftarrow [r_0+i \geq p] \wedge [r_0+i \leq w_2]$; $nz_i \leftarrow c_i \wedge [b_i \neq 0]$ $\triangleright$ vs. DUP(p); secret
- 6: $pre_i \leftarrow \bigvee_{i' \leq i} nz_{i'}$; $sel_i \leftarrow c_i \wedge \neg(pre_{i-1} \vee found)$ $\triangleright$ prefix-OR: four EXT/ORR
- 7: $Msk[r_0+i] \leftarrow sel_i$; $\pi \leftarrow \pi \oplus \sum_i (sel_i \wedge b_i)$; $found \leftarrow found \vee \bigvee_i nz_i$ $\triangleright$ π secret
- 8: **end for**
- 9: **for** each block r_0 ; rows i in fours **while** $r_0+i \leq w_2$ **do** $\triangleright$ fold, public bounds
- 10: $Piv[\rho] \leftarrow Piv[\rho] \oplus (sel_i \wedge A[r_0+i][\rho])$ $\triangleright$ BCAX with $\neg \text{DUP}_i(Msk)$
- 11: **end for**
- 12: $T \leftarrow \text{multab}(\text{TBL}(\text{inv}_{16}, \text{DUP}(\pi)))$; $Piv[\rho] \leftarrow \text{TBL}(T, Piv[\rho])$ $\triangleright$ $Piv \leftarrow \pi^{-1} Piv$
- 13: **for** $r = w_0, \dots, w_1$ **do** $\triangleright$ window rows: take Piv or eliminate
- 14: $g \leftarrow [r > p]$; $q \leftarrow [r = p] \wedge found$ $\triangleright$ secret masks, DUP into lanes
- 15: $T_r \leftarrow \text{multab}(\text{LD1R}(A[r][j]) \wedge g)$; $A[r][\rho] \leftarrow \text{BSL}(q, Piv[\rho], A[r][\rho]) \oplus \text{TBL}(T_r, Piv[\rho])$ $\triangleright$ public address
- 16: **end for**
- 17: **for** $r = w_1+1, \dots, m-1$, in fours **do** $\triangleright$ $r > p$: no mask
- 18: $T_r \leftarrow \text{multab}(\text{LD1R}(A[r][j]))$; $A[r][\rho] \leftarrow A[r][\rho] \oplus \text{TBL}(T_r, Piv[\rho])$
- 19: **end for**
- 20: $p \leftarrow p + (found \wedge 1)$ $\triangleright$ secret counter; arithmetic, no branch
- 21: **after** column ko : transpose every 16×16 block of A in registers (TRN) and store it as sixteen column vectors $\triangleright$ A^T : every column contiguous

From the comparison, a prefix-OR over lanes in four EXT/ORR steps, shifted by one lane and joined with the earlier blocks, yields the mask “rows from the pivot row to the first non-zero row”. Every row still receives exactly the mask upstream computes, and neither a branch nor an address depends on a secret. The fold that assembles the pivot row broadcasts the stored masks per row and accumulates with BCAX. As upstream, one table lookup inverts the pivot, but the result stays in a lane and builds its multiplication table without a round trip through a general-purpose register. The register loop of a step starts at the register holding column j , a public index. Like upstream, the search stops 32 rows past the window of possible pivot rows, which misses a pivot with negligible probability; rows past the window are eliminated without a mask, four at a time. Finally the echelon matrix is written back transposed through 16×16 register transposes, so that every column of A is contiguous in the output.

The transposed output is what the back substitution needs. It walks the rows from the last upward and, for each candidate column, updates the right-hand

Table 3. Echelon form on random full-rank systems, median of 1500 runs.

set	$m \times (ko + 1)$	cycles			instructions	
		upstream	ours	change	upstream	ours
MAYO ₁	80×81	52,598	23,439	−55.4 %	281,030	112,445
MAYO ₂	64×66	35,450	15,843	−55.3 %	177,620	74,239
MAYO ₃	108×111	150,227	45,071	−70.0 %	604,127	225,427
MAYO ₅	142×145	252,767	84,173	−66.7 %	1,240,807	440,675

side y by a multiple of that column. On a column-contiguous matrix this is one table lookup per sixteen rows. The update is a dependency chain through y . Reading $y[\text{row}]$ back as a scalar after the previous vector stores appears to stall on a partial store-to-load forwarding. We therefore reload the whole vector that contains it and broadcast the lane with a lookup at a public lane index. We also form the multiplier and its table in lanes, so that the chain never leaves the SIMD file. The masks and the XORs are upstream’s, and every address is public. A rank-deficient system returns to the caller for a public restart, as upstream.

The new echelon form is checked against the upstream one entry for entry on 400 matrices per parameter set, including matrices with duplicated rows, with a zero column and with sparse entries, without a mismatch. Table 3 shows the effect: 55 % fewer cycles on the level-1 sets and 67–70 % on MAYO₃ and MAYO₅, with 2.4 to 2.8 times fewer instructions. At its step of the decomposition (Table 5) the change removes 14.5–19.3 % of the signing time and leaves the other operations untouched. The solver is then 10–19 % of our faster signature, the back substitution only 0.7–2.6 %. The gain comes at the cost of stack, since the transposed copy grows the peak stack of signing by 2.2–3.5 %.

4.2 Key expansion: AES-CTR, the linear term, and the entry barrier

The upstream NEON AES-CTR encrypts one block per iteration and derives each counter from the previous one. This loop-carried chain presumably bounds it, although all four SIMD pipelines of the core execute AES instructions [20]. We therefore apply the usual block interleaving [16], as the upstream x86 builds do with VAES [11]. We keep $W = 8$ counter blocks in flight and issue each round for all of them before the next. The key schedule and the round function are the upstream ones, so the keystream is byte-identical. On the 155,200-byte expansion of MAYO₁’s $P_1 \parallel P_2$ the interleaved loop runs at 0.183 cycles per byte and 7.67 instructions per cycle, against 0.429 and 4.82 for the upstream loop. Its spread over four function alignments, a smaller ensemble, is 0.1 %, against 2.2 % for a four-way version. Sixteen-way interleaving is slower again at 0.211 cycles per byte, presumably because its sixteen blocks and eleven round keys leave too few of the 32 vector registers. Every set spends 57 % less time on its keystream.

The linear term Λ is an AES-128-CTR keystream, which upstream computes through the ECB entry point over explicitly formed counter blocks, so that every

backend produces the same stream. However, the upstream header that selects the AES implementation maps the ECB entry point to the portable C code unless the x86 AES-NI build is enabled. The NEON build therefore runs the C AES on $nm/2$ bytes per signature and per verification. We add an ECB routine with the same key schedule and round function as the CTR loop and route the expansion to it; the stream is unchanged. This closes a gap in the upstream backend, not an algorithmic one. An open upstream pull request proposes ARMv8 AES for the same entry point [25]. In our build the expansion of A is about 1–2% of a signature and 2–5% of a verification. With the C implementation it is 10–22% of a signature and 18–37% of a verification. The C figures come from a later phase profile of the tree with every change off. The decomposition of Section 5.2 measures the AES changes of this section as one step.

The CTR loop is also where code placement enters the picture. On this core the same instruction stream runs mostly in one of two throughput states, 0.18 or 0.23 cycles per byte. The code path into the loop sets the state. Figure 4 shows the first of two sweeps, with medians of 300 runs per stub at the MAYO₅ length. We enter the routine through sixteen stubs whose branch into it sits at each 4-byte offset of a 64-byte block. Without any barrier, the expansion at this length is slow at fourteen offsets, fast at one and in between (0.20 cycles per byte) at the remaining one. At the MAYO₁ length it is fast at all sixteen. An instruction synchronisation barrier, `isb`, at the entry of the routine makes all sixteen fast at both lengths. A repeat sweep at the MAYO₅ length left two stubs up to 9% slower than the rest, though still under 0.20 cycles per byte, and at the MAYO₁ length the barrier adds bumps of up to 3%. End to end, the barrier is worth more than the roughly 50 cycles per call it cost in a loaded isolated run. Without it, the shipping tree measured over the seven function alignments of Section 5.1 spreads by up to 28% on the public-key expansion and by 8–12% on verification. Signing is 4–6% slower on every set at every alignment. The isolated sweep does not reproduce this uniform penalty, so the barrier’s end-to-end effect is measured, not explained. With the barrier every spread is below 0.6% (Table 4).

The barrier is not a guarantee, however. When the ECB entry point that the linear term needs gave the key schedule a second caller, the compiler stopped inlining it. The loop then ran 27–28% slower at every keystream length with the barrier in place. Its retired-instruction count, 217 thousand to 1.18 million, exceeded that of the inlined layout by at most 44. It was slow at six of the seven alignments of the ensemble, and a second barrier before the loop did not help. The same outlined layout without any barrier was fast at the one alignment measured. Forcing the key schedule inline restored the fast state at all seven alignments, and that is the shipped layout. We do not know the mechanism; the counts rule out extra work and suggest a cause in the front end. The state of this loop depends on the whole code path into it, and an unrelated edit can flip it. Section 5 therefore reports the spread of every operation over function alignments.

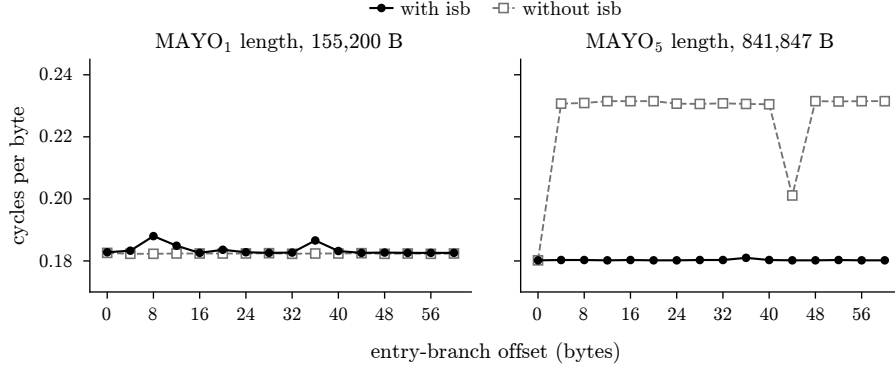


Fig. 4. AES-CTR throughput against the entry-branch offset, with and without barrier.

4.3 Signing structure

The composite that turns the vinegar vectors into the signing system (Algorithm 1) is rearranged in three ways. First, on MAYO₃ and MAYO₅ the count m is not a multiple of 16 elements, one 64-bit limb, so their packed m -vectors end in a partial limb. There, the upstream code spreads P_2 from the packed keystream to a padded stride in a separate pass. We leave P_2 packed: the P_1O kernel spreads it on first touch, from the last row upward, and route A’s VP_2 kernel reads it packed. The spread row r always lies above every packed row $\leq r$ by the size difference of P_1 , so nothing is overwritten before it is read. The separate pass thus disappears from key generation and the compact-key expansion.

Second, the choice between signing with and without L is made per parameter set (the two routes of Algorithm 1). The upstream implicit- L path, route A, does no GF(16) work in the compact-key expansion and computes $P_1^\top V^\top$ and $U^\top O$ in the signature. Route B, in contrast, keeps $L' = P_1O + P_2$ in the expanded key, half the products of the materialised L . It completes $M = VL' + P_v^\top O$ in the signature, where $P_v = P_1V^\top$ is needed anyway. Which route is faster appears to depend on the balance of o and k . Measured as single builds of the shipping tree with the route switched, B wins on MAYO₁ and MAYO₃ by 3.2 and 2.5% of a signature. It loses by 18.2% on MAYO₂, whose $o = 13$ makes L' expensive relative to $k = 5$. On MAYO₅ the two routes are within 0.7%, which single builds cannot resolve, so we keep the upstream route A. The shipping build fixes the route per set at compile time, and the pre-expanded-key API keeps the upstream layout and the materialised L . In the final build the L' computation is 17.6% of a MAYO₁ signature and 22.7% of a MAYO₃ signature. The ladder of Section 5.2 counts the route and the packed- P_2 first touch in its kernel step.

Third, the function that assembles the matrix A from the M_i blocks copies the k blocks once into a limb-major scratch. The shifted accumulation of the whipping schedule then loads pairs of columns as one 128-bit operand and performs the 16×16 nibble transposes in registers. It unpacks to bytes before the reduction by the field polynomial. That reduction becomes table lookups on

whole byte rows. The polynomial’s tail coefficients are public parameters of the set, and the two values 0 and 1 are specialised away. The output is byte-identical by construction, and the change is worth 1.1–3.1 % of a signature. Together with the store-first kernels of Section 3.3, which remove the clearing of the intermediate buffers, this leaves a signature dominated by the products M and $VP_1V^\top$, which take 37–66 % of it.

5 Evaluation

5.1 Setup and methodology

All measurements were taken on one Apple M1 Pro on mains power with low-power mode off. The tree was compiled with Apple clang 17.0.0 (clang-1700.6.4.2) at -O3 with `-march=armv8.2-a+crypto+sha3`. Cycles are read through the kperf interface of the upstream benchmark harness [34], as in our earlier work on this core [24]. It counts only the benchmark thread’s user-mode cycles; the thread requests a performance core through the user-interactive QoS class. The counter configuration is verified against a workload of known instruction count before every session. Medians are over 2000 runs end to end, 41 for kernels, 1500 for the echelon form and 1000 for the AES loop. Signing repeats one fixed message under one random key per process, and verification one signature. Phase profiles average 400 runs, minus the measured cost of the probe pairs. The baseline is the upstream NEON build of the pinned commit [34] with the same compiler, flags and machine. “Keypair” is key generation; “expand sk” and “expand pk” are the expansions of the pre-expanded-key API, and “expand sk” materialises L on every route. “Sign” and “verify” start from the compact keys and include their expansion, as in the specification’s fused columns. Function alignment alone moves an operation by a few percent on this core, and code layout biases measurements elsewhere [14, 26]. Tables 4 and 9 therefore report placement ensembles: seven builds that differ only in `-falign-functions` $\in \{4, 8, 16, 32, 64, 128, 256\}$. Link order, data alignment and environment stay fixed. We report the median of the seven builds’ medians and their spread, $(\max - \min)/\min$. The decomposition and the route comparison are single builds.

The tree passed the Round 3 known-answer tests on all four sets, with Keccak built with and without the SHA-3 extension, on the intrinsics tier, and under UndefinedBehaviorSanitizer with the alignment check disabled. It also passed the echelon-form comparison of Section 4.1, the kernel oracle and generator idempotence check of Section 3.3, and 64 cross-verification checks against the upstream tree in both directions with single-bit tampering. Our files compile without warnings under `-Wall -Wextra`. Nothing was edited during the session, whose ensemble binaries were compiled from the gated tree just before each run. The end-to-end, tier, kernel, echelon-form and constant-time tables come from this one session, run only after every check above passed. The entry-barrier, key-schedule and route measurements, the phase profile of the tree with every change off and the three MAYO₂ controls of Section 5.3 are diagnostic, added later on

Table 4. End-to-end cycles, upstream against ours, medians over seven function alignments.

set	operation	upstream	ours	speedup	spread (ours)	published (M3)
MAYO ₁	keypair	126,971	70,940	1.79×	0.25 %	123,271
	expand sk	159,767	91,909	1.74×	0.58 %	155,312
	expand pk	66,234	28,476	2.33×	0.14 %	63,395
	sign	371,655	177,626	2.09 ×	0.37 %	381,390
	verify	218,842	92,355	2.37 ×	0.18 %	213,224
MAYO ₂	keypair	114,849	71,961	1.60×	0.15 %	112,322
	expand sk	137,273	87,687	1.57×	0.14 %	141,107
	expand pk	50,398	21,647	2.33×	0.21 %	47,918
	sign	216,955	103,701	2.09 ×	0.47 %	214,151
	verify	132,781	50,703	2.62 ×	0.11 %	128,003
MAYO ₃	keypair	342,680	192,319	1.78×	0.48 %	334,458
	expand sk	474,443	266,215	1.78×	0.22 %	457,789
	expand pk	159,859	68,434	2.34×	0.05 %	152,837
	sign	930,327	438,946	2.12 ×	0.27 %	892,572
	verify	527,701	237,236	2.22 ×	0.08 %	512,092
MAYO ₅	keypair	804,262	457,214	1.76×	0.19 %	795,090
	expand sk	1,106,302	655,344	1.69×	0.11 %	1,083,099
	expand pk	356,920	152,366	2.34×	0.04 %	341,523
	sign	1,868,326	937,302	1.99 ×	0.14 %	1,789,597
	verify	1,056,854	501,429	2.11 ×	0.12 %	1,033,879

the same machine. The profile and the controls waited for a quiet machine; the others ran under load.

5.2 Results

Table 4 is the main result. Signing from the compact key is 1.99–2.12 times faster than the upstream build on the four sets, verification 2.11–2.62 times, key generation 1.60–1.79 times and the expansion of the public key 2.33–2.34 times. The largest spread of any of our cells is 0.58 %, and of the upstream cells 0.99 %. The last column gives the designers’ Apple M3 means [11], from a different chip and compiler build. Unlike our medians, they include restarted signatures, and our upstream baseline lies within -2.7 to $+5.2$ % of them. Of the three scheme operations, verification gains most. It consists almost entirely of AES expansion and GF(16) kernels, and the upstream C expansion of Λ weighs most on it (Section 4.2). Key generation and the secret-key expansion gain least (1.57–1.79×). Half of key generation is the computation of P_3 , whose kernels improve by 22–37 %, and the secret-key expansion is dominated by the $(P_1 + P_1^\top)O$ kernel that materialises L . MAYO₂ gains the least on both because its kernels gain the least (Section 3.3); that kernel gains 26 % there, 33–38 % elsewhere.

Table 5. Cycles with the changes switched on one at a time, single builds.

set	operation	all off	+ AES path	+ kernels	+ echelon form	+ assembly of A	total
MAYO ₁	keypair	126,195	88,443 (−29.9 %)	70,934 (−19.8 %)	70,889 (−0.1 %)	70,985 (+0.1 %)	1.78×
	expand sk	158,718	120,476 (−24.1 %)	91,759 (−23.8 %)	91,862 (+0.1 %)	91,797 (−0.1 %)	1.73×
	expand pk	66,198	28,485 (−57.0 %)	28,459 (−0.1 %)	28,467 (+0.0 %)	28,480 (+0.0 %)	2.32×
	sign	369,177	272,413 (−26.2 %)	214,582 (−21.2 %)	183,536 (−14.5 %)	177,761 (−3.1 %)	2.08×
	verify	218,193	121,491 (−44.3 %)	92,243 (−24.1 %)	92,252 (+0.0 %)	92,360 (+0.1 %)	2.36×
MAYO ₂	keypair	113,908	85,433 (−25.0 %)	71,949 (−15.8 %)	71,925 (−0.0 %)	71,993 (+0.1 %)	1.58×
	expand sk	136,468	107,717 (−21.1 %)	87,698 (−18.6 %)	87,735 (+0.0 %)	87,734 (−0.0 %)	1.56×
	expand pk	50,048	21,658 (−56.7 %)	21,655 (−0.0 %)	21,657 (+0.0 %)	21,637 (−0.1 %)	2.31×
	sign	214,915	140,571 (−34.6 %)	127,677 (−9.2 %)	106,289 (−16.8 %)	105,150 (−1.1 %)	2.04×
	verify	132,196	57,802 (−56.3 %)	50,742 (−12.2 %)	50,864 (+0.2 %)	50,657 (−0.4 %)	2.61×
MAYO ₃	keypair	341,472	250,126 (−26.8 %)	192,610 (−23.0 %)	192,451 (−0.1 %)	192,314 (−0.1 %)	1.78×
	expand sk	473,388	381,556 (−19.4 %)	266,024 (−30.3 %)	266,455 (+0.2 %)	266,339 (−0.0 %)	1.78×
	expand pk	159,864	68,428 (−57.2 %)	68,430 (+0.0 %)	68,455 (+0.0 %)	68,441 (−0.0 %)	2.34×
	sign	927,601	729,258 (−21.4 %)	554,499 (−24.0 %)	447,686 (−19.3 %)	438,898 (−2.0 %)	2.11×
	verify	527,296	328,702 (−37.7 %)	237,048 (−27.9 %)	236,947 (−0.0 %)	237,209 (+0.1 %)	2.22×
MAYO ₅	keypair	802,694	598,146 (−25.5 %)	457,426 (−23.5 %)	456,786 (−0.1 %)	457,661 (+0.2 %)	1.75×
	expand sk	1,105,465	899,753 (−18.6 %)	655,334 (−27.2 %)	655,000 (−0.1 %)	655,105 (+0.0 %)	1.69×
	expand pk	356,940	152,422 (−57.3 %)	152,382 (−0.0 %)	152,339 (−0.0 %)	152,383 (+0.0 %)	2.34×
	sign	1,863,183	1,480,918 (−20.5 %)	1,123,109 (−24.2 %)	952,856 (−15.2 %)	938,766 (−1.5 %)	1.98×
	verify	1,056,336	670,136 (−36.6 %)	501,626 (−25.1 %)	501,579 (−0.0 %)	501,880 (+0.1 %)	2.10×

Table 5 attributes the gain. The ladder starts from our tree with every change off, which is the upstream code except for our Keccak with the SHA-3 extension. No step measures that Keccak; against the upstream single build it saves at most 1.3 %. The changes are then enabled one at a time: the AES path, the generated kernels, the echelon form, and the assembly of A . Each parenthesis gives the change from the previous column. The kernel step also contains the per-set signing route and the packed- P_2 first touch; the route alone is measured in Section 4.3. For verification the AES path removes 37–56 % and the kernels a further 12–28 %; the two other changes are not on its path. For signing, the totals agree with Table 4 to within 0.05 in speedup, but each step is a single build and may carry a placement effect. The first step is largely the gap in the

upstream backend (Section 4.2). The rest of the step comes from the interleaved counter mode and its entry barrier.

The generated GF(16) tier was also measured against the intrinsics tier as full ensembles (Table 9 in the appendix); the tier brings its own signing composites, route and packed- P_2 first touch included. On MAYO₂ the intrinsics tier’s secret-key expansion is bimodal, with cells near 108 and 125–127 thousand cycles and a spread of 17 %, so its gain is relative to a bimodal median.

5.3 Constant time

By design, though without a machine check, the implementation has no secret-dependent branch, memory index or address. Every mask is value-equal to the upstream one’s, though several are computed differently (Section 4.1). macOS on Apple silicon has no instrumentation-based checker, so a statistical test asks whether the machine nevertheless takes different times for different secrets. It follows the style of *dudect* [30] and the fixed-versus-random t -test [15]: two input classes, cycle counts of many samples, Welch’s t [35] on the class means. One such run is not a verdict. At 150,000 samples it resolves a few cycles, where buffer placement and the rest of the machine can dominate, and the t of a fixed environmental offset grows with the sample count [33]. The campaign therefore repeats every arm 16 times with a fresh fixed secret and class sequence. Each set has four arms and three matched nulls, the two echelon-form arms sharing one. The arms are the echelon form on a fixed random against fresh random matrices, the same on a fixed rank-deficient matrix, the full solver including the rank check and the back substitution, and the signature. Each real run is paired with a null run in which both classes get the same secret and every instruction, address and refresh stays identical. The verdict is an exact sign-flip permutation test on the sixteen paired differences. The mean test, on t , would reveal a harness asymmetry as a consistent sign; the dispersion test, on $|t|$ and one-sided, would reveal a first-order leak as a larger real dispersion; Δ is real minus null. The p -values are Holm-corrected [19] within each family of sixteen contrasts. Sixteen repetitions give a smallest attainable p of 2^{-16} one-sided and 2^{-15} two-sided.

Before the timed call, only the bytes of the secret depend on the class. The echelon-form arms generate the sample’s matrix in place with one generator pass for either class and differ only in the seed. The degenerate arm uses the same pass with a branch-free mask. All class buffers are aligned to the 16 KiB page, since a 4 KiB alignment produced fixed class offsets in earlier experiments. The signature arm signs under a fixed key against a pool of 64 random keys, rotated per sample and refreshed on a public schedule. It stages both candidates for every sample and selects with a byte mask. Its null uses 64 copies of the fixed key at the same addresses with the same refresh work. The statistic is Welch’s t at a crop of 1.05 times the run’s own median, which excludes the restarted signatures, a public event, and the operating-system tail. The linear-algebra arms have no restarts, so there the crop removes only that tail. Recomputed at crops of 1.2 to 3 times the median, none of their contrasts differs from its null and no real run reaches $|t| = 4.5$. The MAYO₂ signature contrast differs only at the 1.05 crop; wider

Table 6. Paired real-versus-null tests, sixteen repetitions per arm, Holm-corrected.

set	arm	mean test		dispersion test		verdict
		$\overline{\Delta t}$	p	$\overline{\Delta t }$	p (one-sided)	
MAYO ₁	echelon form	−0.014	1.000	+0.318	0.860	not distinguishable
	echelon form, degenerate	−0.122	1.000	+0.263	1.000	not distinguishable
	full solver	−0.787	0.580	−0.020	1.000	not distinguishable
	signature	+0.180	1.000	+0.590	0.745	not distinguishable
MAYO ₂	echelon form	−0.030	1.000	+0.325	0.429	not distinguishable
	echelon form, degenerate	−0.671	0.414	+0.247	1.000	not distinguishable
	full solver	+0.176	1.000	−0.201	1.000	not distinguishable
	signature	−0.402	1.000	+1.769	0.004	differs
MAYO ₃	echelon form	+0.088	1.000	−0.212	1.000	not distinguishable
	echelon form, degenerate	−0.015	1.000	−0.068	1.000	not distinguishable
	full solver	+0.355	1.000	+0.175	1.000	not distinguishable
	signature	−0.322	1.000	+0.460	0.962	not distinguishable
MAYO ₅	echelon form	−0.362	1.000	−0.367	1.000	not distinguishable
	echelon form, degenerate	−0.447	1.000	−0.264	1.000	not distinguishable
	full solver	+0.394	1.000	−0.063	1.000	not distinguishable
	signature	+0.291	1.000	+0.323	0.860	not distinguishable

crops, which add the operating-system tail, lose it. Runs take 150,000 samples, or 300,000 for the signature, with the data-independent-timing bit clear [5].

Table 6 gives the sixteen contrasts, and Figure 6 in the appendix the per-repetition values. Fifteen are not distinguishable from their nulls by either test, including every arm that handles only secret data. The null runs stay well inside the conventional threshold (largest $|t|$ 2.5, standard deviations 0.6–1.2). One contrast differs: the signature of MAYO₂ fails the dispersion test with a Holm-corrected p of 0.004. Twelve of its sixteen real runs have $|t| \geq 2$, with mixed signs, and their standard deviation is 2.8 against 0.96 for the nulls. The mean test, in contrast, is flat. The class means differ by 0.2 to 7.9 cycles in a signature of 103,534, i.e. by at most 0.008 %.

Three further experiments on MAYO₂ probe this effect. First, it is not specific to our kernels. With the upstream kernels our tree shows a larger dispersion (standard deviation 3.73, uncorrected one-sided $p = 0.0002$). On the upstream tree the dispersion lies above its nulls but is not significant (1.97 against 1.17, $p = 0.14$). Second, it is not a per-process artefact. When four separate processes sign under the same seed-derived key, five of six key and tier pairs keep the sign of t in all four. Third, a similar effect appears without any secret. Timing *verification* of a fixed public key and signed message against random ones shows a larger dispersion still. Its standard deviation is 7.0 on our build, 9.7 on the intrinsics tier and 3.7 on the upstream build, each with $p \leq 0.0002$ and every null at about 1. Its mean test on our build is marginal ($p = 0.023$ two-sided, uncorrected). The largest class-mean offsets are 0.014–0.022 % of a verification, and four of six key and tier pairs keep their sign across processes. Verification indexes a 256-byte table by public signature nibbles, so its effect need not share a mechanism with signing.

The reading consistent with all three is that the time of branch-free code on this core appears to depend slightly on the values it handles, a known class of behaviour [32]. In units of t its dispersion is largest on the intrinsics tier and smallest on the slower upstream tree; in cycles the upstream offsets are the

largest. The M1’s data memory-dependent prefetcher [31] is an unlikely cause: it follows only values whose bits 55 to 32 match their load address [13], which random field data almost never do. A fixed-versus-random-key signature test does not separate a secret-dependent component from the public-data component of the same key. Our measured claim is therefore twofold. The arms that hold only secret data are not distinguishable from their matched nulls on any parameter set; their largest observed class-mean offsets are 4.04 cycles and 0.0072 % of the operation. A timing effect on public data of about 0.01–0.02 %, largely fixed by the key, exists on this platform and also appears in the upstream verification. Three limitations remain. The dispersion test is the weaker one at sixteen repetitions: on the measured null spread, a twofold inflation of one arm’s dispersion survives the correction in 25 % of synthetic campaigns, and a threefold one in 68 %. The kernels that multiply by O and V are tested only inside the signature arm; key generation and signing from a pre-expanded key are not covered. The three controls cover MAYO₂ only.

5.4 Code size and stack

The generated assembly holds 27.4–78.6 KiB of text per set, and the page-rounded benchmark binaries grow from 80–96 to 112–160 KiB. Peak stack is measured with a canary watermark on a thread whose stack we own. For signing it grows by 2.2–3.5 % because of the transposed echelon matrix, from 226,544 to 234,512 bytes on MAYO₁. It changes by at most 1.0 % for key generation and verification. Unlike upstream, our one-shot Keccak and AES keep their state on the stack.

6 Conclusion

Against the upstream NEON implementation on the same Apple M1 Pro, our implementation of MAYO Round 3 signs 1.99–2.12 times, verifies 2.11–2.62 times and generates keys 1.60–1.79 times faster on all four parameter sets. A generated assembly tier for the GF(16) kernels reproduces the upstream arithmetic byte for byte. Apart from the A kernel, its large kernels reach 86–96 % of the speed an operation-count model predicts. A transposed echelon form needs 55–70 % fewer cycles than the upstream one. An AES path expands the public matrices at 0.18 instead of 0.43 cycles per byte, and NEON expands the linear term; with our changes off, its portable-C AES derivation takes 10–22 % of signing and 18–37 % of verification. A signing composite is arranged per parameter set. Single builds switch these changes on one at a time, with the per-set route and the packed- P_2 first touch inside the kernel step. The headline tables come from one gated session, the diagnostic runs of Sections 4 and 5.3 from later sessions, and the end-to-end and tier tables report medians over seven function alignments.

The constant-time campaign, with matched null runs and paired permutation tests, distinguishes no arm that holds only secret data from its null, at 25 % power for a twofold dispersion. The MAYO₂ signature arm differs. Controls on MAYO₂

show a timing effect of about 0.01–0.02 % on public data, largely fixed by the key, which also appears in the upstream verification; the signature test does not separate it from a secret component. Such an effect bounds what a fixed-versus-random-key test can establish on MAYO₂ on this platform. The measurements cover one chip and one compiler version.

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 (2024). <https://doi.org/10.46586/tches.v2024.i1.87-132>
2. Alagic, G., Bros, M., Ciadoux, P., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., Waller, N.: Status report on the first round of the additional digital signature schemes for the NIST post-quantum cryptography standardization process. NIST Internal Report 8528, National Institute of Standards and Technology (Oct 2024). <https://doi.org/10.6028/NIST.IR.8528>
3. Alagic, G., Bros, M., Ciadoux, P., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., Waller, N.: Status report on the second round of the additional digital signature schemes for the NIST post-quantum cryptography standardization process. NIST Internal Report 8610, National Institute of Standards and Technology (May 2026). <https://doi.org/10.6028/NIST.IR.8610>
4. Amagasa, H., Ueno, R., Homma, N.: AVX2 implementation of QR-UOV for modern x86 processors. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2026**(2), 325–346 (2026). <https://doi.org/10.46586/tches.v2026.i2.325-346>, also IACR ePrint 2025/1599
5. Apple Inc.: Writing ARM64 code for Apple platforms. Apple Developer Documentation (Xcode), <https://developer.apple.com/documentation/xcode/writing-arm64-code-for-apple-platforms>, section “Enable DIT for constant-time cryptographic operations”. Accessed 26 September 2026
6. Arm Ltd.: Arm Architecture Reference Manual for A-profile architecture. Arm Ltd. (Jun 2026), <https://developer.arm.com/documentation/ddi0487/mc/>, Arm DDI 0487, Issue M.c
7. Becker, H., Kannwischer, M.J.: Hybrid scalar/vector implementations of Keccak and SPHINCS⁺ on AArch64. In: Isobe, T., Sarkar, S. (eds.) *Progress in Cryptology – INDOCRYPT 2022. Lecture Notes in Computer Science*, vol. 13774, pp. 272–293. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22912-1_12
8. Beullens, W.: MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In: AlTawy, R., Hülsing, A. (eds.) *Selected Areas in Cryptography – SAC 2021. Lecture Notes in Computer Science*, vol. 13203, pp. 355–376. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99277-4_17, also IACR ePrint 2021/1144
9. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: Nibbling MAYO: Optimized implementations for AVX2 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2024**(2), 252–275 (2024). <https://doi.org/10.46586/tches.v2024.i2.252-275>, also IACR ePrint 2023/1683

10. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: MAYO. Round 2 Version. Specification submitted to the NIST Post-Quantum Cryptography Standardization Process for Additional Digital Signature Schemes (Feb 2025), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/mayo-spec-round2-web.pdf>
11. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: MAYO. Round 3 Version. Specification submitted to the NIST Post-Quantum Cryptography Standardization Process for Additional Digital Signature Schemes (Aug 2026), <https://pqm Mayo.org/assets/specs/mayo-round3.pdf>
12. Beullens, W., Chen, M.S., Hung, S.H., Kannwischer, M.J., Peng, B.Y., Shih, C.J., Yang, B.Y.: Oil and vinegar: Modern parameters and implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2023**(3), 321–365 (2023). <https://doi.org/10.46586/tches.v2023.i3.321-365>, also IACR ePrint 2023/059
13. Chen, B., Wang, Y., Shome, P., Fletcher, C.W., Kohlbrenner, D., Paccagnella, R., Genkin, D.: GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 1117–1134. USENIX Association, Philadelphia, PA (Aug 2024), <https://www.usenix.org/conference/usenixsecurity24/presentation/chen-boru>
14. Curtsinger, C., Berger, E.D.: STABILIZER: Statistically sound performance evaluation. In: Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2013). pp. 219–228. ACM, Houston, TX, USA (Mar 2013). <https://doi.org/10.1145/2451116.2451141>
15. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation. In: NIST Non-Invasive Attack Testing Workshop (NIAT 2011). Nara, Japan (Sep 2011), https://csrc.nist.gov/CSRC/media/Events/Non-Invasive-Attack-Testing-Workshop/documents/08_Goodwill1.pdf
16. Gouvêa, C.P.L., López, J.: Implementing GCM on ARMv8. In: Nyberg, K. (ed.) *Topics in Cryptology – CT-RSA 2015*. Lecture Notes in Computer Science, vol. 9048, pp. 167–180. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16715-2_9
17. Gringiani, A., Meneghetti, A., Signorini, E., Susella, R.: MAYO: Optimized implementation with revised parameters for ARMv7-M. *Cryptology ePrint Archive*, Paper 2023/540 (2023), <https://eprint.iacr.org/2023/540>
18. Hirner, F., Streibl, M., Krieger, F., Mert, A.C., Roy, S.S.: Whipping the multivariate-based MAYO signature scheme using hardware platforms. In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24). pp. 3421–3435. ACM, Salt Lake City, UT, USA (2024). <https://doi.org/10.1145/3658644.3690258>, also IACR ePrint 2023/1267
19. Holm, S.: A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics* **6**(2), 65–70 (1979), <https://www.jstor.org/stable/4615733>
20. Johnson, D.: Firestorm Overview, Apple M1 microarchitecture and instruction tables; accessed 26 September 2026. <https://dougallj.github.io/applecpu/firestorm.html>
21. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced Oil and Vinegar signature schemes. In: Stern, J. (ed.) *Advances in Cryptology – EUROCRYPT '99*. Lecture Notes in Computer Science, vol. 1592, pp. 206–222. Springer, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_15

22. Kwon, H., Kim, H., Sim, M., Lee, W.K., Seo, H.: Look-up the rainbow: Table-based implementation of Rainbow signature on 64-bit ARMv8 processors. *ACM Transactions on Embedded Computing Systems* **22**(5), 1–19 (2023). <https://doi.org/10.1145/3607140>
23. Lee, M., Sim, M., Eum, S., Seo, H.: High- and low-rank optimization of SNOVA on ARMv8: From high-security applications to IoT efficiency. *Electronics* **14**(13), 2696 (2025). <https://doi.org/10.3390/electronics14132696>
24. Lee, M., Sim, M., Eum, S., Seo, H.: ARMv8/NEON optimization of NCC-Sign for mixed-radix NTT: Cycle-accurate evaluation on Apple M1 Pro and Cortex-A72. *Electronics* **15**(7), 1456 (2026). <https://doi.org/10.3390/electronics15071456>
25. Use the ARMv8 AES instructions for AES_128_ECB with ENABLE_AESNEON. Pull request #12 to MAYO-C, GitHub (2026), <https://github.com/PQCMayo/MAYO-C/pull/12>, opened 24 September 2026; open when accessed 27 September 2026
26. Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.F.: Producing wrong data without doing anything obviously wrong! In: *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2009)*. pp. 265–276. ACM, Washington, DC, USA (Mar 2009). <https://doi.org/10.1145/1508244.1508275>
27. National Institute of Standards and Technology: SHA-3 standard: Permutation-based hash and extendable-output functions. *Federal Information Processing Standards Publication 202*, National Institute of Standards and Technology (Aug 2015). <https://doi.org/10.6028/NIST.FIPS.202>
28. National Institute of Standards and Technology: Call for additional digital signature schemes for the post-quantum cryptography standardization process (Sep 2022), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf>, submission deadline 1 June 2023
29. National Institute of Standards and Technology: Advanced encryption standard (AES). *Federal Information Processing Standards Publication 197-upd1*, National Institute of Standards and Technology (May 2023). <https://doi.org/10.6028/NIST.FIPS.197-upd1>
30. Reparaz, O., Balasch, J., Verbauwhede, I.: Dude, is my code constant time? In: *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*. pp. 1697–1702. IEEE (2017). <https://doi.org/10.23919/DATE.2017.7927267>
31. Sanchez Vicarte, J.R., Flanders, M., Paccagnella, R., Garrett-Grossman, G., Morrison, A., Fletcher, C.W., Kohlbrenner, D.: Augury: Using data memory-dependent prefetchers to leak data at rest. In: *2022 IEEE Symposium on Security and Privacy (SP)*. pp. 1491–1505. IEEE (2022). <https://doi.org/10.1109/SP46214.2022.9833570>
32. Sanchez Vicarte, J.R., Shome, P., Nayak, N., Trippel, C., Morrison, A., Kohlbrenner, D., Fletcher, C.W.: Opening Pandora’s box: A systematic study of new ways microarchitecture can leak private data. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. pp. 347–360. IEEE (2021). <https://doi.org/10.1109/ISCA52012.2021.00035>
33. Standaert, F.X.: How (not) to use Welch’s T-test in side-channel security evaluations. In: Bilgin, B., Fischer, J.B. (eds.) *Smart Card Research and Advanced Applications – CARDIS 2018. Lecture Notes in Computer Science*, vol. 11389, pp. 65–79. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-15462-2_5
34. The MAYO team: MAYO-C: C implementation of the MAYO signature scheme. GitHub repository (2026), <https://github.com/PQCMayo/MAYO-C>, Round 3 code at commit 74de637a7c4a7f9fafa888183bc05f5cf920cd2d (8 September 2026)

Table 7. Isolated cycles of all sixteen kernels, MAYO₁ and MAYO₂.

set	kernel	products	upstream	generated	change	% of model
MAYO ₁	P_1O (rmw)	2,073,600	46,718	31,918	−31.7 %	88.8 %
	P_1O , fused P_2 init	2,073,600	46,685	32,836	−29.7 %	86.3 %
	$O^\top(P_1O + P_2)$	409,600	9,091	6,214	−31.6 %	90.1 %
	$(P_1 + P_1^\top)O$ (rmw)	4,044,800	91,188	61,081	−33.0 %	90.5 %
	$(P_1 + P_1^\top)O$, fused init	4,044,800	91,146	61,196	−32.9 %	90.4 %
	$P_1V^\top$	2,592,000	57,042	37,844	−33.7 %	91.0 %
	$P_1^\top V^\top$	2,592,000	56,904	38,377	−32.6 %	89.7 %
	$U^\top O$	512,000	11,326	7,727	−31.8 %	90.6 %
	VL (rmw)	512,000	11,137	7,458	−33.0 %	91.2 %
	VL (store)	512,000	11,140	7,389	−33.7 %	92.0 %
	A_vO	51,200	1,251	891	−28.8 %	78.6 %
	VP_2 (packed P_2)	512,000	11,151	7,405	−33.6 %	91.8 %
	$VP_1V^\top$	640,000	13,893	9,206	−33.7 %	92.3 %
	$P_1S_1 + P_2S_2$	3,104,000	70,925	44,980	−36.6 %	91.7 %
	P_3S_2	28,800	832	673	−19.1 %	56.8 %
	$S_2^\top(PS)_2$	64,000	1,347	1,259	−6.5 %	67.5 %
MAYO ₂	P_1O (rmw)	2,247,232	43,172	33,485	−22.4 %	92.8 %
	P_1O , fused P_2 init	2,247,232	43,087	33,530	−22.2 %	92.6 %
	$O^\top(P_1O + P_2)$	789,568	15,022	11,356	−24.4 %	96.1 %
	$(P_1 + P_1^\top)O$ (rmw)	4,372,992	84,896	62,792	−26.0 %	96.3 %
	$(P_1 + P_1^\top)O$, fused init	4,372,992	84,910	63,067	−25.7 %	95.8 %
	$P_1V^\top$	864,320	20,137	15,588	−22.6 %	95.3 %
	$P_1^\top V^\top$	864,320	19,898	15,619	−21.5 %	95.1 %
	$U^\top O$	303,680	5,821	4,452	−23.5 %	94.3 %
	VL (rmw)	303,680	7,032	5,469	−22.2 %	95.4 %
	VL (store)	303,680	6,871	5,436	−20.9 %	96.0 %
	A_vO	60,736	1,271	991	−22.0 %	84.7 %
	VP_2 (packed P_2)	303,680	7,032	5,445	−22.6 %	95.9 %
	$VP_1V^\top$	116,800	2,782	2,179	−21.7 %	92.1 %
	$P_1S_1 + P_2S_2$	1,168,000	26,579	20,861	−21.5 %	96.2 %
	P_3S_2	29,120	854	715	−16.3 %	70.0 %
	$S_2^\top(PS)_2$	20,800	613	526	−14.2 %	68.0 %

35. Welch, B.L.: The generalization of ‘Student’s’ problem when several different population variances are involved. *Biometrika* **34**(1–2), 28–35 (1947). <https://doi.org/10.1093/biomet/34.1-2.28>

Appendix A Supplementary Tables

The tables of this appendix are generated from the session ledger and are not edited by hand. In the kernel names, rmw accumulates into existing data, store writes the first product, and fused init performs the packed- P_2 first touch (Section 4.3).

Table 8. Isolated cycles of all sixteen kernels, MAYO₃ and MAYO₅.

set	kernel	products	upstream	generated	change	% of model
MAYO ₃	$P_1 O$ (rmw)	6,356,880	149,032	96,270	−35.4 %	90.9 %
	$P_1 O$, fused P_2 init	6,356,880	148,935	97,902	−34.3 %	89.4 %
	$O^\top (P_1 O + P_2)$	1,166,400	26,388	17,057	−35.4 %	94.2 %
	$(P_1 + P_1^\top) O$ (rmw)	12,480,480	303,814	187,937	−38.1 %	91.5 %
	$(P_1 + P_1^\top) O$, fused init	12,480,480	303,979	189,269	−37.7 %	90.8 %
	$P_1 V^\top$	6,992,568	164,578	111,715	−32.1 %	92.2 %
	$P_1^\top V^\top$	6,992,568	184,105	111,734	−39.3 %	92.2 %
	$U^\top O$	1,283,040	29,747	18,752	−37.0 %	94.2 %
	VL (rmw)	1,283,040	29,769	19,953	−33.0 %	94.7 %
	VL (store)	1,283,040	29,732	19,841	−33.3 %	95.3 %
	$A_v O$	116,640	2,628	1,820	−30.7 %	88.3 %
	VP_2 (packed P_2)	1,283,040	29,900	19,965	−33.2 %	94.7 %
	$VP_1 V^\top$	1,411,344	32,892	21,810	−33.7 %	95.3 %
	$P_1 S_1 + P_2 S_2$	8,275,608	201,519	131,442	−34.8 %	92.7 %
	$P_3 S_2$	65,340	2,002	1,400	−30.1 %	68.8 %
	$S_2^\top (PS)_2$	130,680	3,514	2,519	−28.3 %	76.4 %
MAYO ₅	$P_1 O$ (rmw)	17,300,712	377,863	248,065	−34.4 %	92.1 %
	$P_1 O$, fused P_2 init	17,300,712	377,568	251,031	−33.5 %	91.0 %
	$O^\top (P_1 O + P_2)$	2,903,616	63,604	40,061	−37.0 %	95.7 %
	$(P_1 + P_1^\top) O$ (rmw)	34,117,488	721,745	483,587	−33.0 %	93.2 %
	$(P_1 + P_1^\top) O$, fused init	34,117,488	722,213	488,504	−32.4 %	92.2 %
	$P_1 V^\top$	17,300,712	378,286	245,799	−35.0 %	92.9 %
	$P_1^\top V^\top$	17,300,712	394,678	247,704	−37.2 %	92.2 %
	$U^\top O$	2,903,616	64,367	40,297	−37.4 %	95.1 %
	VL (rmw)	2,903,616	64,413	40,411	−37.3 %	94.9 %
	VL (store)	2,903,616	63,948	40,200	−37.1 %	95.4 %
	$A_v O$	241,968	5,306	3,481	−34.4 %	91.8 %
	VP_2 (packed P_2)	2,903,616	64,721	41,275	−36.2 %	92.9 %
	$VP_1 V^\top$	2,903,616	65,801	41,611	−36.8 %	92.1 %
	$P_1 S_1 + P_2 S_2$	20,204,328	427,270	285,326	−33.2 %	93.5 %
	$P_3 S_2$	132,912	3,682	2,382	−35.3 %	73.7 %
	$S_2^\top (PS)_2$	245,376	6,030	4,071	−32.5 %	79.6 %

Table 9. Generated against intrinsics tier, medians of seven placements.

set	operation	intrinsics	generated	change	spread intr.	spread gen.
MAYO ₁	keypair	88,536	70,940	−19.9 %	0.34 %	0.25 %
	expand sk	120,648	91,909	−23.8 %	0.70 %	0.58 %
	expand pk	28,484	28,476	−0.0 %	0.15 %	0.14 %
	sign	235,939	177,626	−24.7 %	0.18 %	0.37 %
	verify	121,464	92,355	−24.0 %	0.40 %	0.18 %
MAYO ₂	keypair	85,501	71,961	−15.8 %	0.21 %	0.15 %
	expand sk	124,731	87,687	−29.7 %	17.22 %	0.14 %
	expand pk	21,640	21,647	+0.0 %	0.18 %	0.21 %
	sign	116,315	103,701	−10.8 %	4.18 %	0.47 %
	verify	57,742	50,703	−12.2 %	0.29 %	0.11 %
MAYO ₃	keypair	250,176	192,319	−23.1 %	0.08 %	0.48 %
	expand sk	381,539	266,215	−30.2 %	0.16 %	0.22 %
	expand pk	68,436	68,434	−0.0 %	0.07 %	0.05 %
	sign	613,005	438,946	−28.4 %	0.46 %	0.27 %
	verify	328,731	237,236	−27.8 %	0.06 %	0.08 %
MAYO ₅	keypair	597,801	457,214	−23.5 %	0.12 %	0.19 %
	expand sk	899,893	655,344	−27.2 %	0.88 %	0.11 %
	expand pk	152,367	152,366	−0.0 %	0.04 %	0.04 %
	sign	1,293,513	937,302	−27.5 %	0.12 %	0.14 %
	verify	670,185	501,429	−25.2 %	0.09 %	0.12 %

Appendix B Supplementary Figures

Figure 5 plots the signing and verification rows of Table 5. Figure 6 plots the per-repetition statistics behind Table 6, with the conventional threshold $|t| = 4.5$ dotted; within a column, points are offset horizontally only to keep them countable, and the two echelon-form arms share one null.

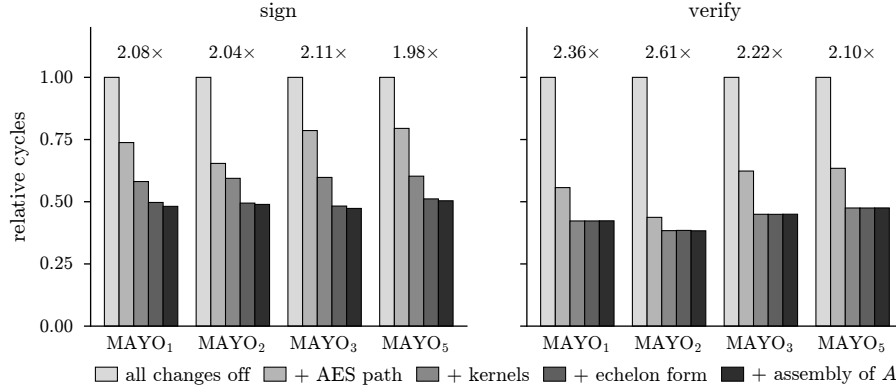


Fig. 5. Signing and verification with the changes switched on one at a time.

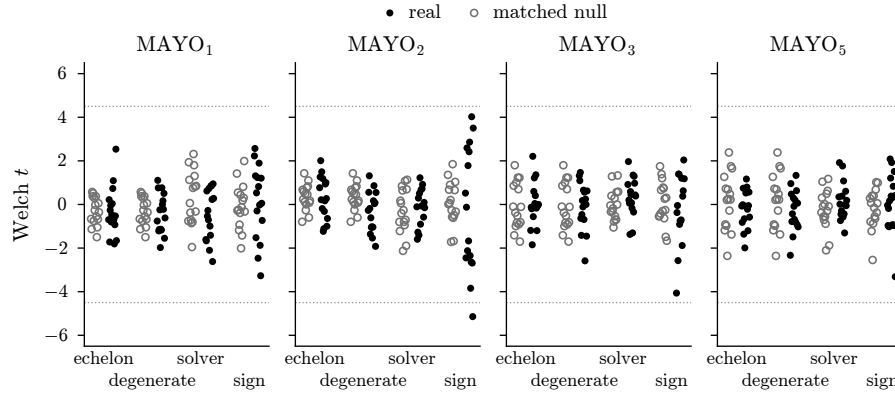


Fig. 6. Per-repetition Welch t of every arm and its matched null.