

Optimizing Montgomery Arithmetic for RSA on Cortex-M0+ and Cortex-M3

Minjoo Sim¹[0000-0001-5242-214X], Minwoo Lee¹[0000-0002-2356-3055],
Seungwon Lee¹[0009-0009-9583-1387], SuBeen Cho²[0009-0004-4352-8021],
Jiwon Bang²[0009-0006-5453-1722], and Hwajeong Seo²[0000-0003-0069-9061]*

¹ Department of Information Computer Engineering,
Hansung University, Seoul (02876), South Korea
{minjoos9797, minunejip}@gmail.com

² Department of Convergence Security,
Hansung University, Seoul (02876), South Korea
{dkajdfhd1, chosubin1208, bang050922, hwajeong84}@gmail.com

Abstract. Cryptographic tokens that retain RSA credentials for compatibility with existing authentication systems require efficient private-key operations on small processors. Cortex-M0+ (M0+) lacks native widening multiplication, whereas Cortex-M3 (M3) provides it with operand-dependent latency. We optimize Montgomery multiplication and squaring using 15-bit limbs and integrate the kernels into RSA-2048 and RSA-3072 private operations based on the Chinese remainder theorem (CRT). On M0+, a wrapped sum and accumulated block quotients enable exact carry recovery. The squaring schedule integrates square and reduction columns without storing the complete square. On M3, we retain the established half-diagonal representation and double each buffer limb when it is first incorporated into Montgomery reduction, eliminating a separate buffer traversal. We establish accumulator bounds for exact Montgomery digit and carry recovery in both schedules. At operand widths of 1024 and 1536 bits, the M0+ representation and register mapping reduce Montgomery multiplication and squaring cycles by 12.60% to 13.11% against blocked two-word controls. The M3 schedule reduces Montgomery squaring cycles by 2.12% to 3.08% with half-diagonal initialization and triangular accumulation held fixed. Kernel replacement in otherwise unchanged RSA implementations reduces warm CRT cycles by 12.59% to 12.80% on M0+ and by 1.69% to 2.39% on M3. Our M3 Montgomery multiplication also uses 31.47% fewer cycles than a reproduced number-theoretic transform (NTT) implementation for 2048-bit operands with matched integers and canonical outputs.

Keywords: RSA · Montgomery arithmetic · Cortex-M · Efficient implementation

* Corresponding author: hwajeong84@gmail.com.

1 Introduction

Existing and post quantum infrastructures are expected to coexist during migration [12]. Devices with provisioned RSA keys must continue to perform private operations because new signature algorithms alone do not replace existing credentials. Nitrokey Start supports RSA-2048 on a microcontroller [15].

RSA exponentiation repeatedly multiplies and squares integers wider than a processor word [13]. With equal prime widths the Chinese remainder theorem (CRT) uses 1024 and 1536 bit arithmetic for RSA-2048 and RSA-3072 respectively. Montgomery reduction replaces division by the modulus with products and shifts [11]. Carry handling and intermediate memory transfers still contribute to the arithmetic cost.

Cortex-M0+ (M0+) lacks native widening multiplication and restricts several Thumb operands to low registers. Cortex-M3 (M3) has widening multiplication with operand dependent latency [1,3]. The established radix 2^{15} keeps limb products within 32 bits and permits grouped accumulation [17].

Deferred doubling [20] and integration with Montgomery reduction [10,21] and half diagonal initialization [16] are established techniques. Number theoretic transforms (NTT) offer another approach to products at RSA widths [4]. We adapt accumulator states and update orders to each core while preserving exact cancellation residues and carry information. For squaring the schedules also preserve the distinct weights of diagonal and cross products. Bounds establish conditions for exact recovery.

1.1 Contributions

- **Montgomery accumulation and register mapping on Cortex-M0+.** Using 15 bit limbs we map a wrapped sum and block quotients to Thumb registers for multiplication and integrated squaring without storing the complete square. Against blocked two word accumulation under the same product and load schedule this representation and mapping reduce multiplication cycles by 12.60% and 13.05% and squaring cycles by 12.91% and 13.11% at 1024 and 1536 bits respectively. The squaring comparison retains the same generic integrated schedule.
- **Doubling and carry scheduling on Cortex-M3.** We retain half diagonal initialization and triangular accumulation while moving doubling to the first reduction read. This removes a separate buffer traversal. Untouched limbs are doubled while pending carries and updated limbs retain their weights. Blocks of four digits use shifted ADD and MUL/MLA. This placement reduces Montgomery squaring cycles by 3.08% and 2.12% at 1024 and 1536 bits respectively with initialization and triangular accumulation held fixed.
- **Controlled evaluation and RSA integration.** Kernel replacement within otherwise fixed RSA-2048 and RSA-3072 implementations measures the resulting reduction in warm CRT cycles. The M0+ comparison uses generic integrated squaring while M3 changes only doubling placement. Library comparisons follow the key preparation policies in Section 4. A

separate comparison of M3 Montgomery multiplication at 2048 bits saves 31.47% over reproduced NTT multiplication with matched integer values and canonical outputs. Three M0+ configurations compare execution time and code size.

2 Background

2.1 RSA Primitives and Limb Arithmetic

For distinct odd primes p and q RSA uses $N = pq$ and $\lambda(N) = \text{lcm}(p-1, q-1)$. The public exponent satisfies $\gcd(e, \lambda(N)) = 1$ and the private exponent satisfies $ed \equiv 1 \pmod{\lambda(N)}$. For $0 \leq x < N$ the public and private primitives compute $x^e \bmod N$ and $x^d \bmod N$ respectively. The public primitive supports encryption and verification. The private primitive supports decryption and signing [13].

CRT computes $m_p = x^{d_p} \bmod p$ and $m_q = x^{d_q} \bmod q$ with $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. Garner recombination returns $m = m_q + hq$ where $h = (m_p - m_q)q^{-1} \bmod p$ [8,13]. Equal prime widths roughly halve the arithmetic modulus length and reduce the cost of each exponentiation.

Window exponentiation groups exponent bits and uses precomputed powers to reduce multiplications. Section 3.3 gives our window widths and masked lookup.

Hereafter N denotes the current arithmetic modulus. With equal prime widths RSA-2048 and RSA-3072 use CRT moduli of 1024 and 1536 bits. Public operations use the full modulus. For radix $\beta = 2^{15}$ we write $a = \sum_{i=0}^{n-1} a_i \beta^i$ with $0 \leq a_i < \beta$ and use the same notation for b and N . The prime moduli require 69 and 103 limbs and the full moduli require 137 and 205 limbs respectively.

Schoolbook multiplication assigns n^2 products $a_i b_j$ to columns $k = i + j$. Normalization stores the residue modulo β and carries the quotient forward. Squaring uses $n(n+1)/2$ distinct products through

$$a^2 = \sum_i a_i^2 \beta^{2i} + 2 \sum_{i < j} a_i a_j \beta^{i+j}.$$

2.2 Montgomery Reduction

For odd N and $R = \beta^n > N$ Montgomery reduction chooses $M = -TN^{-1} \bmod R$ for $0 \leq T < NR$ so that

$$U = \frac{T + MN}{R} \equiv TR^{-1} \pmod{N} \quad 0 \leq U < 2N. \quad (1)$$

Since $M < R$ one conditional subtraction gives a canonical residue. For $0 \leq a, b < N$ the product $T = ab$ satisfies the input bound [11].

Define $\text{MontMul}(a, b) = abR^{-1} \bmod N$ with canonical output for $0 \leq a, b < N$ and $\text{MontSqr}(a) = \text{MontMul}(a, a)$. Conversion to and from Montgomery representation uses

$$\begin{aligned} \bar{a} &= \text{MontMul}(a, R^2 \bmod N) = aR \bmod N \\ a &= \text{MontMul}(\bar{a}, 1). \end{aligned}$$

Montgomery multiplication of $\bar{a}$ and $\bar{b}$ yields $abR \bmod N$. Exponentiation retains this representation until the final conversion.

After preceding cancellations and divisions by β let v_k denote the sum of all contributions to the current lowest column. With $N_0 = N \bmod \beta$ and $n'_0 = -N^{-1} \bmod \beta$ the cancellation digit is

$$m_k = (v_k n'_0) \bmod \beta \quad v_k + m_k N_0 \equiv 0 \pmod{\beta}. \quad (2)$$

Adding $m_k N$ and dividing by β advances to the next column. The digits form $M = \sum_{k=0}^{n-1} m_k \beta^k$. Higher columns may remain unnormalized if each cancellation residue is exact and deferred carry information is preserved.

2.3 Existing Accumulation Schedules

Separate routines store products before reduction. Coarsely integrated scanning alternates product and reduction rows while finely integrated operand scanning (FIOS) combines their inner traversal [9]. BearSSL i15 resolves carries after every two limb products while wolfSSL SP Thumb forms wide products from narrow multiplications [18,22]. Pornin bounds intermediate values for redundant Montgomery representations with signed reduction factors [19].

Sliding block doubling combines doubled triangular sums with diagonal products [20]. OpenJDK initializes half diagonals and accumulates cross products before a separate shift and Montgomery reduction [16]. Optimal prime field squaring on AVR and coarsely integrated lazy doubling on MIPS64 integrate doubling with reduction [10,21]. Becker et al. study NTT arithmetic and full modulus RSA on M3 and Cortex-M55 [4].

2.4 Arithmetic on Cortex-M0+ and Cortex-M3

On M0+ MULS returns the low 32 product bits. MULS and LDM and ADCS and SBSC use low registers while ADD can update high registers [2,3].

M3 supports shifted operands for ADD and provides MUL and MLA. Ordinary multiplication has fixed latency while widening multiplication has operand dependent latency [1]. Constant time (CT) execution requires instruction timing and control flow and memory access independent of secrets.

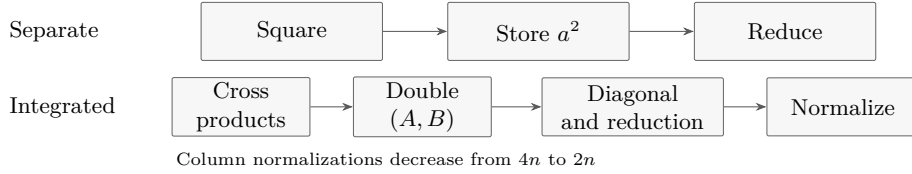
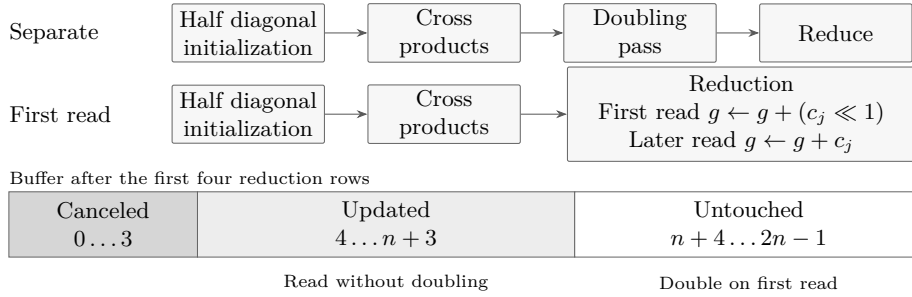
For $\beta = 2^{15}$ the largest limb product is $P = (\beta - 1)^2$. Four products fit in 32 bits because

$$4P = 2^{32} - 262\,140 < 2^{32} < 5P. \quad (3)$$

Stored limbs and incoming carries require additional headroom. Section 3 bounds the accumulators for exact digit recovery.

3 Montgomery Kernels and RSA Integration

Figure 1 contrasts integrated columns on M0+ and doubling placement on M3. Here g is the current accumulator and c_j is a loaded buffer limb.

(a) M0+ integrated columns**(b) M3 doubling on the first read****Fig. 1.** Integrated columns on M0+ and doubling placement on M3.**3.1 Split Accumulation on M0+**

Thumb 1 confines products and multiple loads to low registers. We accumulate unsigned product blocks in adjacent column states held in high registers and recover residue sums and outgoing carries during column normalization.

The update $\text{AddBlock}(A, B, S)$ accumulates up to four limb products whose sum S fits in 32 bits by Equation (3). It retains block quotients in B while A may wrap.

$$A \leftarrow (A + S) \bmod 2^{32} \quad B \leftarrow B + \lfloor S/\beta \rfloor.$$

Let L^* be the sum of the low parts of these blocks. The exact column value is $\beta B + L^*$ and

$$L^* = (A - \beta B) \bmod 2^{32} \quad L^* < 2^{32}. \quad (4)$$

Doubling both A and B doubles only the accumulated cross products. Adding diagonal and reduction terms afterwards avoids doubling those terms.

The example follows columns 2 and 3 for four limbs with incoming carry c . Here $F_j = \beta B_j + L_j^*$ is the value held by (A_j, B_j) .

$$\begin{array}{ll}
 1 \text{ Cross products} & F_2 \leftarrow a_0 a_2 \quad F_3 \leftarrow a_0 a_3 + a_1 a_2 \\
 2 \text{ Double and add} & F_2 \leftarrow 2F_2 + a_1^2 + m_0 N_2 + m_1 N_1 \\
 & F_3 \leftarrow 2F_3 + m_0 N_3 + m_1 N_2 \\
 3 \text{ Cancel even} & L \leftarrow L_2^* + c \quad m_2 \leftarrow L'_0 \bmod \beta \\
 & c \leftarrow B_2 + (L + m_2 N_0)/\beta \\
 4 \text{ Cancel odd} & L \leftarrow L_3^* + c + m_2 N_1 \quad m_3 \leftarrow L'_0 \bmod \beta \\
 & c \leftarrow B_3 + (L + m_3 N_0)/\beta
 \end{array}$$

For $k = 2h$ the square sweep groups at most four consecutive indices $0 \leq j < \min(h, n - 1 - h)$. They supply $a_{h-1-j}a_{h+1+j}$ and $a_{h-1-j}a_{h+2+j}$ to the even and odd columns. The central product $a_h a_{h+1}$ enters the split state before doubling. The diagonal product a_h^2 is computed early and added after doubling. Algorithm 1 uses this split update with missing terms set to zero. Only the first n columns create Montgomery digits. Later columns emit output limbs.

At the supported modulus widths $15n$ exceeds the modulus bit length by at least three bits. Hence $N < R/8$.

Algorithm 1 Montgomery squaring with split accumulation on M0+.

Input $0 \leq a < N < R/8$ and odd N with $2 \leq n \leq 205$ limbs

1. $c \leftarrow 0$
 2. **for** $k = 0, 2, \dots, 2n - 2$ **do**
 3. Initialize (A_k, B_k) and (A_{k+1}, B_{k+1}) to zero
 4. Use AddBlock to accumulate $a_i a_j$ with $i < j$ in columns $i + j \in \{k, k + 1\}$
 5. Double both split pairs
 6. Use AddBlock for $a_{k/2}^2$ in column k and earlier terms $m_i N_{k-i}$ and $m_i N_{k+1-i}$ in their columns for $i < \min(k, n)$
 7. **for** $j = k, k + 1$ in order **do**
 8. $L \leftarrow (A_j - \beta B_j) \bmod 2^{32} + c$
 9. Add $m_k N_1$ to L if $j = k + 1$ and $k < n$
 10. **if** $j < n$ **then**
 11. $m_j \leftarrow (L n'_0) \bmod \beta$
 12. $L \leftarrow L + m_j N_0$
 13. **else**
 14. $z_{j-n} \leftarrow L \bmod \beta$
 15. **end if**
 16. $c \leftarrow B_j + \lfloor L/\beta \rfloor$
 17. **end for**
 18. **end for**
 19. **return** masked selection of $z - N$ if $z \geq N$ and z otherwise
-

Let V_k denote the sum of square and earlier reduction terms in column $k < n$. Recovery gives $L \equiv V_k + c \pmod{\beta}$ and Equation (2) selects the same digit as a fully carried column. Adding $m_k N_0$ makes L divisible by β and yields the carry $B_k + L/\beta$. Repetition cancels the low half of $a^2 + MN$. The remaining value is below $2N < R$ so the final carry is zero and one masked subtraction returns a canonical result.

We bound deferred products with $g_{\text{sq}} = \lceil (n-1)/8 \rceil$ and $g_{\text{red}} = \lceil (n-1)/4 \rceil$. These bound the numbers of blocks per column for cross products and earlier reduction terms. An extra doubled single product covers the central cross product and the smaller diagonal contribution. Thus

$$\begin{aligned} L^* &\leq (2(g_{\text{sq}} + 1) + g_{\text{red}})(\beta - 1), \\ B &\leq (2g_{\text{sq}} + g_{\text{red}})\lfloor 4P/\beta \rfloor + 2\lfloor P/\beta \rfloor. \end{aligned}$$

The even column is normalized after adding $m_k N_0$. Its neighbor receives $m_k N_1$ and $m_{k+1} N_0$ which give the two products in the normalization bound.

Let $L_{\max}$ and $B_{\max}$ denote the respective bounds above. Let v be the largest intermediate value of L over all column normalizations. With initial carry zero every incoming carry satisfies $c \leq B_{\max} + \lfloor v/\beta \rfloor$. Combining this with $v \leq L_{\max} + c + 2P$ gives

$$v \leq \left\lfloor \frac{\beta(L_{\max} + B_{\max} + 2P)}{\beta - 1} \right\rfloor \leq 2\,164\,424\,289 < 2^{32}$$

for $n \leq 205$. Accumulating the central cross product $a_h a_{h+1}$ before doubling leaves at most two products for normalization.

Table 1 compares the representations under the same product and load schedule. Operand loads and products occupy the low registers while both representations retain their column states in high registers. The two word method reuses `r4` after its operand has been consumed. It captures one carry per block with `MOVS` and `ADCS` and adds this carry to H .

Table 1. M0+ accumulator states and normalization.

Role	Two word	Split
<code>r9</code> through <code>r12</code>	(L, H, L_1, H_1) $L \leftarrow t \bmod 2^{32}$	(A, B, A_1, B_1) $A \leftarrow (A + S) \bmod 2^{32}$
Block update	$H \leftarrow H + \lfloor t/2^{32} \rfloor$	$B \leftarrow B + \lfloor S/\beta \rfloor$
Normalization	$q = 2^{17}H + \lfloor L/\beta \rfloor$ $r = L \bmod \beta$	$L^* = (A - \beta B) \bmod 2^{32}$
Carry out	$q + \lfloor (r + c + \ell)/\beta \rfloor$	$B + \lfloor (L^* + c + \ell)/\beta \rfloor$

The two word state holds $V = 2^{32}H + L$. Here S is a sum of at most four products and $t = L + S$ before the update. The incoming carry is c and the late products sum to ℓ . The two word method extracts q and r before adding $c + \ell$ in one word. Accumulator updates for a pair of four product blocks take 13 instructions with two words and 7 with split states. Initial column recovery takes 7 and 4 instructions respectively before adding late products or selecting digits.

The separate square and reduction passes each normalize $2n$ columns. Integration normalizes only $2n$ columns without storing the complete square. These counts exclude quotient extraction within product blocks.

Multiplication uses the same split representation without doubling. Column k sums $a_i b_{k-i}$ for $\max(0, k - n + 1) \leq i \leq \min(k, n - 1)$. Each loaded b_i supplies $b_i a_{k-i}$ and $b_i a_{k+1-i}$ to adjacent columns. For $k = 2h \geq n$ with even $n - 1 - h$ we add $a_{n-1} b_{k-n+1}$ to the split pair and add $m_{k-n+1} N_{n-1}$ when column k is normalized. The bound conservatively allows $g_p = \lceil (n+1)/4 \rceil$ product blocks and two separate single product additions in upper columns. At most two products remain at normalization. Since $g_p + g_{\text{red}} \leq 103$ for $n \leq 205$ the same argument gives the global bound on v .

3.2 Doubling During Reduction on M3

We use half diagonal initialization and move doubling from a separate shift to the first reduction read. The representation follows the OpenJDK squaring method discussed in Section 2. The supported range is $8 \leq n \leq 205$.

Let $D = \sum_i a_i^2 \beta^{2i}$ and $T = \sum_{i < j} a_i a_j \beta^{i+j}$. For $z_i = a_i^2$ and $z_n = 0$ initialization stores $\lfloor D/2 \rfloor$ through the stored limbs c_j given below for $0 \leq i < n$.

$$\begin{aligned} c_{2i} &= \lfloor z_i/2 \rfloor \bmod \beta \\ c_{2i+1} &= \lfloor z_i/(2\beta) \rfloor + (z_{i+1} \bmod 2)\beta/2. \end{aligned}$$

The second term moves the next diagonal's low bit into bit 14 of the preceding limb. Disjoint fields need no carry. We retain $e_0 = a_0 \bmod 2$. Triangular accumulation gives $H = T + \lfloor D/2 \rfloor$ and therefore

$$2H + e_0 = a^2. \quad (5)$$

Triangular accumulation starts with $n \bmod 4$ single rows (K1) and continues with blocks of four rows (K4). A K1 row at index i stores its outgoing carry in column $i + n + 1$. A K4 block at index i stores it in column $i + n + 4$. The next row or block incorporates this carry when normalizing that column.

The last K4 block starts at $i = n - 4$ and normalizes through column $2n - 1$. Its remaining carry enters column $2n$ and is zero since $H < R^2/2$ by Equation (5). Thus all $2n$ stored limbs are normalized before reduction.

Let h_j be the normalized limbs of H before reduction. Reduction first cancels four digits at a time as K4 and then any remaining digits as K1. For $n = 9$ the first K4 block selects its four digits in order from

$$\begin{aligned} u_0 &= 2h_0 + e_0 \\ u_1 &= 2h_1 + \gamma_1 + m_0 N_1 \\ u_2 &= 2h_2 + \gamma_2 + m_0 N_2 + m_1 N_1 \\ u_3 &= 2h_3 + \gamma_3 + m_0 N_3 + m_1 N_2 + m_2 N_1 \\ m_i &= u_i n'_0 \bmod \beta \quad \gamma_{i+1} = (u_i + m_i N_0)/\beta \quad (0 \leq i < 4). \end{aligned}$$

The block then initializes columns 4 through 6 with $\gamma_4 + m_1 N_3 + m_2 N_2 + m_3 N_1$ and $m_2 N_3 + m_3 N_2$ and $m_3 N_3$. Each loaded N_j for $4 \leq j \leq 8$ supplies $m_0 N_j$ through $m_3 N_j$ to columns j through $j + 3$ before column j adds $2h_j$ and is normalized. Thus later columns already hold products when their stored limbs are read.

For $n = 4q + r$ with $0 \leq r < 4$ reduction performs q K4 blocks followed by r K1 digits. The first K4 doubles h_0 through h_{n+3} and adds e_0 at column 0. Each later block starting at d first reads h_{d+n} through $h_{d+n+s-1}$ where $s = 4$ for K4 and $s = 1$ for K1. Every block adds pending carry π once at column $d + n$ after doubling that original limb and retains the final quotient as pending carry for column $d + n + s$. Initially $\pi = 0$. Updated c_j and reduction products and carries are not doubled. The table specializes this order to $n = 9$.

Block	Updated reads	First reads	Pending column
K4 at $d = 0$	none	0...12	9 → 13
K4 at $d = 4$	4...12	13...16	13 → 17
K1 at $d = 8$	8...16	17	17 → 18

At each completed block boundary after the first K4 let d count the canceled digits and set $f = n + d$. The stored state satisfies

$$\begin{aligned}
 a^2 + N \sum_{i=0}^{d-1} m_i \beta^i &= \sum_{j=d}^{f-1} c_j \beta^j + \pi \beta^f \\
 &+ 2 \sum_{j=f}^{2n-1} h_j \beta^j.
 \end{aligned} \tag{6}$$

The initial state satisfies Equation (5). Column normalization stores the residue modulo β and passes the quotient to the next column while preserving this weighted value. The digit recurrence above instantiates Equation (2). Cancellation removes a zero low limb and passes its quotient forward.

After all n cancellations the untouched sum is empty and π is the carry into column $2n$. The result $Y = (a^2 + MN)/R < 2N < 2R$ therefore has top limb $\pi \in \{0, 1\}$. This limb participates in the final masked subtraction.

Carry and pending values start at zero. Assume an incoming carry $c_{\text{in}} \leq 2^{17} - 1$ and pending carry $\pi \leq 7$. A triangle limb is at most $(\beta - 1) + 7 = 2^{15} + 6$ and its column is bounded by $4P + (2^{15} + 6) + (2^{17} - 1) < 2^{32}$. The reduction bound is

$$4P + 2(2^{15} - 1) + (2^{17} - 1) + 7 = 2^{32} - 2^{16} + 8 < 2^{32}.$$

Thus each outgoing carry remains below 2^{17} . The last column without products is at most $c_{\text{in}} + \beta - 1 + 7$ in the triangle and $c_{\text{in}} + 2(\beta - 1) + \pi$ in reduction. Both are below 8β and leave at most seven carry units. These bounds are preserved inductively and ensure that the value preserving updates above fit in 32 bits.

Each four product update in K4 uses one MUL and three MLA instructions. A shifted ADD doubles first reads. The K1 evaluation groups four successive columns while retaining normalization after each column. The comparison method reconstructs $2H + e_0$ before reduction. Doubling during reduction removes the separate $O(n)$ doubling stage and retains $O(n^2)$ products.

Montgomery multiplication processes four operand rows followed by four cancellation rows. Four coefficients remain in registers so each loaded limb supplies four products. Four accumulators hold columns j through $j + 3$ and pass only the completed column carry forward. Section 4.2 evaluates the accumulation order and operand reuse through separate comparisons.

Multiplication has no triangle or doubling and each column is bounded by $4P + (2^{15} - 1) + (2^{17} - 1) < 2^{32}$. Define blocks of four digits by $A_i = \sum_{r=0}^3 a_{4i+r} \beta^r$ and $M_i = \sum_{r=0}^3 m_{4i+r} \beta^r$. Both are below β^4 and the initial value is $u = 0$. For $u < 2N$ and $b < N$ each block gives

$$u' = \frac{u + A_i b + M_i N}{\beta^4} < \frac{2N + 2(\beta^4 - 1)N}{\beta^4} = 2N.$$

Remaining K1 rows fuse $a_i b_j$ and $m_i N_j$ into one column and retain normalization after each column. Their range argument replaces β^4 with β . Both operations finish with one masked subtraction.

3.3 Exponentiation and Table Lookup

The 15 bit CRT implementation uses fixed window exponentiation with public parameters $(\ell, w) = (1024, 5)$ for RSA-2048 and $(\ell, w) = (1536, 6)$ for RSA-3072. The encoded exponent length ℓ fixes $t = \lceil \ell/w \rceil$ windows with a leading window of $f = \ell - w(t - 1)$ bits.

The power table starts with $G_0 = R \bmod N$ and $G_1 = \text{MontMul}(x, R^2 \bmod N)$. For $1 \leq i < 2^{w-1}$ we compute $G_{2i} = \text{MontSqr}(G_i)$ and $G_{2i+1} = \text{MontMul}(G_{2i}, G_1)$. A masked scan of the first 2^f entries initializes the result from the leading window. Each remaining window performs w squarings and one multiplication by a power selected through a masked scan of all 2^w entries. One final multiplication by 1 returns the ordinary representation.

Including table construction and conversion the numbers of Montgomery squarings and multiplications per prime exponentiation are

$$E_{\text{MS}} = w(t - 1) + 2^{w-1} - 1 \quad E_{\text{MM}} = t + 2^{w-1}.$$

Lookup addresses depend only on public sizes. Arithmetic and lookup use Thumb assembly while C controls exponentiation and CRT recombination.

3.4 CRT Input Reduction and Recombination

Let κ be the common prime bit length and split $x = x_H 2^\kappa + x_L$ once for both primes. For $\rho \in \{p, q\}$ put $R_\rho = 2^{15n}$ and $\delta = 15n - \kappa$. One Montgomery multiplication and a partial digit give

$$\begin{aligned} u &= \text{MontMul}(x_H, R_\rho^2 \bmod \rho) & \mu &= (-u\rho^{-1}) \bmod 2^\delta, \\ y &= (u + \mu\rho)/2^\delta \equiv x_H 2^\kappa \pmod{\rho}. \end{aligned}$$

The stored negative inverse modulo β supplies μ since δ is 11 or 9. The bounds $u < \rho$ and $\mu < 2^\delta$ give $y < \rho$ without subtraction. Since $x < pq$ we have $x_H < \rho$ and $x_L < 2\rho$. The sum $s = y + x_L < 3\rho$ is reduced by masked selection among s and $s - \rho$ and $s - 2\rho$.

Recombination forms $m_p + 2p - m_q$ and obtains $\Delta = (m_p - m_q) \bmod p$ by the same masked reduction. The key context stores $q^{-1}R_p \bmod p$ so one Montgomery

multiplication gives $h = q^{-1}\Delta \bmod p$ in the ordinary domain. Narrow products then form $m_q + hq < N$ without further reduction. The argument requires equal prime widths but no ordering between p and q . Key constants persist across calls while the input dependent power table is rebuilt for each exponentiation.

3.5 Specialization and Operation Scope

Compact combines split multiplication with general integrated squaring on M0+. Balanced specializes squaring for 69 limbs. Speed also pairs multiplication iterations and unrolls squaring while omitting padded products. The RSA-3072 configuration specializes 103 and 205 limbs and uses the general algorithm at other lengths. Selection depends on public lengths and each accumulator separates the quotient after at most four products.

Public exponentiation with $e = 65537$ uses sixteen squarings and one multiplication that also converts the result through $\text{MontMul}(x^{65536}R \bmod N, x)$. The auxiliary M3 implementation uses `UMULL` with 32 bit Karatsuba multiplication and recursive Hensel reduction [6]. It assumes a public modulus and representative because multiplication latency and carry termination depend on values. Its private timings are variable time references with $w = 5$ at both RSA sizes.

After checking the public length the private primitive scans the input and masks invalid representatives to zero for full CRT computation. A masked write returns a valid result or preserves the output on rejection. The evaluated primitives use initialized keys and valid nonaliasing buffers without blinding or fault detection.

4 Evaluation

4.1 Experimental Setup

Measurements use the STM32G0B1RE at 20 MHz and STM32F207ZG at 30 MHz with zero flash wait states and disabled prefetch and caches. The former executes `MULS` in one cycle. C code is compiled with GCC 13.2.1 at `O3`. Interrupts are masked during measurement. Timer 2 counts M0+ system clock cycles and the Data Watchpoint and Trace counter measures M3 cycles. Reported timings subtract read overheads of four and one cycles respectively.

Library comparisons and RSA kernel replacements share three independent RSA-2048 keys and two RSA-3072 keys with one input per key. Three sessions take two or three measurements per key for the library comparisons and two for kernel replacement. We take medians within sessions then across sessions and finally across keys. Kernel comparisons use a prime and the full modulus from one key per RSA size. Each shared input pair is measured 20 times in three sessions. Arithmetic controls use seven random and boundary inputs per modulus with the same repetitions.

Baselines are BearSSL i15 and the SP Thumb and SP Cortex M implementations of wolfSSL 5.9.2 [18,22]. SP Thumb uses `LARGE_CODE`. BearSSL uses

assembly on M0+ and C on M3. We reproduce Becker et al.’s NTT [5]. The unmodified FIOS failed private exponentiation with a full width modulus outside its original test range in QEMU and is omitted from comparisons. Window widths in Section 3.3 were selected using earlier kernels and retained for the reported measurements.

Independent integer arithmetic checks the measured operand widths. General control models cover $2 \leq n \leq 205$ on M0+ and $8 \leq n \leq 205$ on M3. ARM execution checks cover 28 selected M0+ lengths and 22 selected M3 lengths spanning all four classes of $n \bmod 4$. M0+ checks include general code and specializations at 69 and 137 limbs and at 103 and 205 limbs. Squaring inputs include 0 and 1 and $N - 1$ and $\lfloor N/3 \rfloor$.

ARM Valgrind applied TIMECOP style taint analysis to initialized 15 bit private operations for RSA-2048 and RSA-3072 [7,14]. Prime factors and their derived constants were secret along with exponents and the CRT coefficient and input representatives. Sizes and the full RSA modulus were public. Tested calls including rejection at N and $N + 1$ showed no secret dependent branches or addresses. Instruction inspection found no multiplication or division with operand dependent latency. M3 arithmetic and lookup kernels had no IT blocks. Other IT predicates were public or fixed by the input conditions. These checks concern software timing of initialized private primitives.

4.2 Arithmetic Performance

Table 2 compares two word and split states for M0+ Montgomery multiplication (MM) and squaring (MS). The comparisons retain the same two column scan and four product blocks and measure the combined effect of representation and register mapping. MS uses general integrated squaring. M3 control A adds doubling and diagonals after triangular accumulation. B and C share half diagonal initialization and triangular accumulation. B doubles separately while C doubles on the first reduction read.

Timings include initialization and final canonicalization under matched input ranges and fixed latency narrow products. Savings use $100(C_{\text{control}} - C_{\text{proposal}})/C_{\text{control}}$ with M3 control B as the denominator. M0+ retains the same product and memory transfer counts while M3 eliminates writes from the separate doubling stage. A separate M0+ comparison with split states fixed reduces MS cycles by 8.7% and 6.6% through square and reduction integration at the two widths.

Table 3 changes accumulation or loading while holding function addresses and product and write counts fixed. Each control removes the named choice from the optimized schedule. These timings use separate comparison implementations from the final specialized kernels in Table 4.

Savings use Control as the denominator. M0+ compares four product updates with individual product updates and batched loads with separate loads at equal transfer counts. M3 operand reuse reduces reads. Its accumulation comparison jointly changes carry handling and multiply instructions and initialization and

Table 2. Accumulator and doubling controls at CRT widths.

<i>(a) M0+ Montgomery multiplication</i>				
Bits	Two word	Split	Saving (%)	
1024	61 223	53 510	12.60	
1536	128 471	111 707	13.05	
<i>(b) M0+ Montgomery squaring</i>				
Bits	Two word	Split	Saving (%)	
1024	53 761	46 819	12.91	
1536	109 084	94 778	13.11	
<i>(c) M3 Montgomery squaring</i>				
Bits	A	B	C	Saving B to C (%)
1024	34 413	34 272	33 217	3.08
1536	74 344	74 135	72 561	2.12

Table 3. Multiplication controls at CRT widths.

<i>(a) 1024 bits</i>				
Core	Removed choice	Control	Optimized	Saving (%)
M0+	Four product split updates	64 381	53 509	16.89
M0+	Batched loads	63 173	53 509	15.30
M3	Accumulation schedule	57 015	40 457	29.04
M3	Operand reuse	53 923	40 457	24.97
<i>(b) 1536 bits</i>				
Core	Removed choice	Control	Optimized	Saving (%)
M0+	Four product split updates	135 808	111 706	17.75
M0+	Batched loads	133 130	111 706	16.09
M3	Accumulation schedule	125 404	88 729	29.25
M3	Operand reuse	118 532	88 729	25.14

remaining column computations. Instruction counts come from emulation and cycles from board measurements.

Table 4 reports final implementations with specialization and loop unrolling at CRT and full modulus widths. F and V denote fixed and variable multiplication latency. BearSSL uses $MM(a, a)$ for squaring. BearSSL and ours return values below N while wolfSSL may omit the final subtraction and return values below $R = 2^{\text{bits}}$. Our 15 bit squaring is faster than all measured fixed latency baselines.

A separate M3 comparison at 2048 bits takes 220 264 cycles for NTT and 150 957 for our multiplication with a reduction of 31.47%. It encodes identical integers under $R_{15} = 2^{2055}$ and $R_{\text{NTT}} = 2^{2112}$ and verifies decoded outputs. Timings include canonicalization below N but exclude context preparation and input encoding and output decoding.

Table 4. Montgomery kernel performance in cycles.

<i>(a) CRT widths</i>							
Core	Implementation	Lat.	1024 bits		1536 bits		
			MM	MS	MM	MS	
M0+	wolfSSL SP Thumb	F	59 066	51 824	128 287	112 641	
M0+	BearSSL i15 (asm)	F	94 304	94 304	203 667	203 667	
M0+	Ours (15 bits)	F	53 439	37 103	111 601	81 927	
M3	wolfSSL SP Cortex M	V	25 435	22 417	52 693	46 289	
M3	wolfSSL SP Thumb	F	57 991	51 378	126 409	112 243	
M3	BearSSL i15 (C code)	F	74 307	74 307	162 333	162 333	
M3	Ours (15 bits)	F	40 455	32 216	88 727	70 006	
M3	Ours (32 bits)	V	21 340	19 317	46 721	42 566	
<i>(b) Full modulus widths</i>							
Core	Implementation	Lat.	2048 bits		3072 bits		
			MM	MS	MM	MS	
M0+	wolfSSL SP Thumb	F	214 777	192 490	468 220	420 405	
M0+	BearSSL i15 (asm)	F	349 970	349 970	767 504	767 504	
M0+	Ours (15 bits)	F	190 876	141 063	412 700	318 981	
M3	wolfSSL SP Cortex M	V	92 504	82 748	193 115	172 882	
M3	wolfSSL SP Thumb	F	211 507	191 066	462 480	419 084	
M3	BearSSL i15 (C code)	F	282 435	282 435	624 665	624 665	
M3	Ours (15 bits)	F	150 955	115 635	331 716	250 496	
M3	Ours (32 bits)	V	70 642	64 712	153 036	140 177	

4.3 RSA Primitive Performance

Table 5 reports timings of initialized primitives with $e = 65537$ including input checks and byte conversion and CRT recombination without blinding.

Table 5. RSA primitive performance in millions of cycles.

Core	Implementation	Priv.	Public		Private CRT	
			RSA-2048	RSA-3072	RSA-2048	RSA-3072
Private implementations with profile C						
M0+	BearSSL i15 (asm)	C	15.762	42.154	268.06	959.30
M0+	Ours (15 bits)	C	2.643	5.935	103.46	330.36
M3	BearSSL i15 (C code)	C	12.522	33.868	208.62	758.09
M3	Ours (15 bits)	C	2.155	4.676	87.10	278.03
Private reference implementations with profiles A or V						
M0+	wolfSSL SP Thumb	A	3.541	7.709	133.74	433.05
M3	wolfSSL SP Cortex M	V	1.526	3.178	57.87	178.08
M3	wolfSSL SP Thumb	A	3.508	7.671	132.32	430.55
M3	Ours (32 bits)	V	1.176	2.558	51.78	166.68

Profile C requires fixed multiplication latency and branches and addresses independent of secrets. A denotes secret indexed addresses and V variable multiplication latency. Baselines assume public representatives while ours treats them as secret. Public timings require both the modulus and representative to be public.

Under these warm call conditions our RSA-2048 profile C operations use 61.4% fewer cycles on M0+ and 58.2% on M3 than BearSSL. Calls include power table construction but reuse our key constants while libraries recompute them per call. The gains combine assembly and exponentiation choices with this reuse.

Section 3.3 gives $2E_{MS}$ squarings and $2E_{MM} + 3$ multiplications for CRT including two input reductions and recombination. The totals are 2070 MS and 445 MM for RSA-2048 and 3122 MS and 579 MM for RSA-3072. Weighting Table 4 by these counts estimates 97.0% to 97.2% of measured 15 bit CRT cycles.

Table 6 fixes the remaining RSA code and window widths and table lookup and CRT processing while replacing kernels. M0+ uses general integrated squaring with W and S denoting two word and split states in MM/MS order. M3 fixes multiplication and replaces squaring B with C from Table 2.

Table 6. CRT kernel replacement in millions of cycles.

<i>(a) M0+ generic squaring</i>					
RSA bits	W/W	S/W	W/S	S/S	Saving (%) W/W to S/S
2048	141.404	137.972	127.034	123.602	12.59
3072	424.897	415.191	380.234	370.527	12.80
<i>(b) M3 doubling controls</i>					
RSA bits	B	C	Saving (%) B to C		
2048	91.351	89.167	2.39		
3072	290.908	285.994	1.69		

The measured CRT reductions match the savings in Table 2 weighted by the call counts above.

4.4 Resource Costs

Table 7 gives library code and working memory in bytes and CRT time in millions of cycles for profile C configurations. Code includes BearSSL padding and hashing [18] but excludes libc and measurement support. The optimized library also uses 332 bytes of read only data. Working memory combines observed stack use with required temporary storage including power tables reused across CRT branches. The optimized implementations allocate 16 480 bytes for temporary storage and a separate 4544 byte CRT key. The table reports each operation’s required storage with its observed stack use.

Table 7. Code and working memory of the 15-bit configurations.

Core	RSA bits	Configuration	Library .text	Stack + temporary storage	CRT
M0+	2048	Compact	16 768	9 812	123.86
		Balanced	35 912	9 812	113.95
		Speed	138 808	9 836	103.46
M3	3072	Optimized	131 168	19 416	330.36
	2048	Optimized	16 992	8 972	87.10
	3072	Optimized	16 992	18 024	278.03

M0+ Speed uses 8.28 times the Compact code size for 16.5% fewer RSA-2048 CRT cycles. Compact saves 53.8% against BearSSL under the same warm comparison policies.

5 Conclusion

We designed Montgomery states and processing orders for Cortex-M0+ and Cortex-M3. Split accumulation recovers carries from wrapped sums and integrates squaring with reduction on M0+. On M3 the first reduction read doubles original limbs while preserving updated values and pending carries. The bounds establish exact digit and carry recovery for both schedules.

Within otherwise unchanged RSA-2048 and RSA-3072 implementations the changes reduce warm CRT cycles by 12.59% to 12.80% on M0+ and 1.69% to 2.39% on M3. These results connect reduced accumulation and memory costs to repeated RSA arithmetic. Further M0+ specialization trades a larger code size for lower execution time.

References

1. ARM: Cortex-M3 Technical Reference Manual, Revision r2p0. ARM (2010), <https://documentation-service.arm.com/static/5e8e107f88295d1e18d34714>, ARM DDI 0337H. Table 3-1 and its footnotes a and b
2. ARM: Cortex-M0+ Devices Generic User Guide. ARM (2012), <https://documentation-service.arm.com/static/5f04abc8dbdee951c1cdc9f7>, DUI 0662B. Sections 1.1 and 3.5.6
3. ARM: Cortex-M0+ Technical Reference Manual, Revision r0p1. ARM (2012), <https://documentation-service.arm.com/static/60411750ee937942ba301773>, ARM DDI 0484C. Section 1.4.1 “Configurable multiplier”
4. Becker, H., Hwang, V., Kannwischer, M.J., Panny, L., Yang, B.Y.: Efficient multiplication of somewhat small integers using number-theoretic transforms. In: Cheng, C.M., Akiyama, M. (eds.) Advances in Information and Computer Security – IWSEC 2022. Lecture Notes in Computer Science, vol. 13504, pp. 3–23. Springer (2022). https://doi.org/10.1007/978-3-031-15255-9_1
5. Becker, H., Hwang, V., Kannwischer, M.J., Panny, L., Yang, B.Y.: ntt-int-mul-m3 source code. <https://github.com/ntt-int-mul/ntt-int-mul-m3> (2022), Revision 052f98c. Accessed 11 September 2026

6. Brent, R.P., Zimmermann, P.: Modern Computer Arithmetic. Cambridge University Press (2010). <https://doi.org/10.1017/CB09780511921698>
7. eBACS: Tips for software development. <https://bench.cr.yp.to/tips.html>, TIMECOP section. Accessed 26 September 2026
8. Garner, H.L.: The residue number system. In: Papers Presented at the March 3–5, 1959, Western Joint Computer Conference (IRE-AIEE-ACM '59 Western). pp. 146–153. ACM (1959). <https://doi.org/10.1145/1457838.1457864>
9. Koç, Ç.K., Acar, T., Kaliski, Jr., B.S.: Analyzing and comparing Montgomery multiplication algorithms. IEEE Micro **16**(3), 26–33 (1996). <https://doi.org/10.1109/40.502403>
10. Li, Y., Wang, J., Zeng, X., Ye, X.: Fast Montgomery modular multiplication and squaring on embedded processors. IEICE Transactions on Communications **E100-B**(5), 680–690 (2017). <https://doi.org/10.1587/transcom.2016EBP3189>
11. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation **44**(170), 519–521 (1985). <https://doi.org/10.1090/S0025-5718-1985-0777282-X>
12. Moody, D., Perlner, R., Regenscheid, A., Robinson, A., Cooper, D.: Transition to post-quantum cryptography standards. NIST IR 8547, National Institute of Standards and Technology (2024). <https://doi.org/10.6028/NIST.IR.8547.ipd>, Initial Public Draft. Section 3.2
13. Moriarty, K., Kaliski, B., Jonsson, J., Rusch, A.: PKCS #1: RSA cryptography specifications version 2.2. RFC 8017, RFC Editor (Nov 2016). <https://doi.org/10.17487/RFC8017>
14. Nethercote, N., Seward, J.: Valgrind: A framework for heavyweight dynamic binary instrumentation. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 89–100. ACM (2007). <https://doi.org/10.1145/1250734.1250746>, <https://valgrind.org/docs/valgrind2007.pdf>
15. Nitrokey: Nitrokey Start FAQ. <https://docs.nitrokey.com/nitrokeys/start/faq>, Accessed 27 September 2026
16. OpenJDK: BigInteger.java (2013), <https://github.com/openjdk/jdk8u/blob/4fe69c432ff11f5532ad5ed95eae09d53194fd4b/jdk/src/share/classes/java/math/BigInteger.java>, jdk8u commit 4fe69c432ff11f5532ad5ed95eae09d53194fd4b
17. Pornin, T.: BearSSL: Big integer design. <https://www.bearssl.org/bigint.html>, Accessed 11 September 2026
18. Pornin, T.: BearSSL source code. <https://www.bearssl.org/gitweb/> (2026), Revision 7bea48e. Accessed 11 September 2026
19. Pornin, T.: Montgomery Multiplication in Signed Redundant Representations. Cryptology ePrint Archive, Paper 2026/780 (2026), <https://eprint.iacr.org/2026/780>, version dated 19 April 2026
20. Seo, H., Liu, Z., Choi, J., Kim, H.: Multi-precision squaring for public-key cryptography on embedded microprocessors. In: Paul, G., Vaudenay, S. (eds.) Progress in Cryptology – INDOCRYPT 2013. Lecture Notes in Computer Science, vol. 8250, pp. 227–243. Springer (2013). https://doi.org/10.1007/978-3-319-03515-4_15
21. Seo, H., Liu, Z., Nogami, Y., Choi, J., Kim, H.: Montgomery multiplication and squaring for optimal prime fields. Computers & Security **52**, 276–291 (2015). <https://doi.org/10.1016/j.cose.2015.03.005>
22. wolfSSL Inc.: wolfSSL 5.9.2 source code. <https://github.com/wolfSSL/wolfssl> (2026), Revision ac01707. Accessed 11 September 2026