Improved Plantard Arithmetic for Lattice-based Cryptography

Junhao Huang[1], Jipeng Zhang[2], Haosong Zhao[1], Zhe Liu[2,3], Ray C. C. Cheung[4], Çetin Kaya Koç[5,2], Donglong Chen[1*]

[1]Guangdong Provincial Key Laboratory of Interdisciplinary Research and Application for Data Science, BNU-HKBU United International College
[2]Nanjing University of Aeronautics and Astronautics
[3]Zhejiang Lab
[4]City University of Hong Kong
[5]University of California Santa Barbara

July 25, 2022

CRYSTALS-Kyber

- One of the third-round KEM finalists **(The final KEM scheme to be standardized)**.
- Module-LWE problem $(\mathbf{A}, \mathbf{b} = \mathbf{A}^T\mathbf{s} + \mathbf{e})$.
- The IND-CCA secure KEM protocols are obtained from the IND-CPA secure PKE protocols using the Fujisaki-Okamoto transform.
- Parameters:

Schemes	n	k	q	η_1	η_2	(d_u, d_v)	δ
Kyber512	256	2	3329	3	2	$(10, 4)$	2^{-139}
Kyber768	256	3	3329	2	2	$(10, 4)$	2^{-164}
Kyber1024	256	4	3329	2	2	$(11, 5)$	2^{-174}

NTRU and NTTRU

NTRU:

- One of the third-round KEM finalists.
- The polynomial arithmetic operates in three polynomial rings $\mathbb{Z}_3[x]/\Phi_{\mathbf{n}}, \mathbb{Z}_q[x]/\Phi_{\mathbf{n}}$, and $\mathbb{Z}_q[x]/\left(\Phi_{\mathbf{1}} \cdot \Phi_{\mathbf{n}}\right)$ with $\Phi_1 = (x-1)$ and $\Phi_{\mathbf{n}} = \left(x^{n-1} + x^{n-2} + \cdots + 1\right)$.

NTTRU:

- An NTT-friendly variant of NTRU KEM scheme proposed in TCHES2019 [LS19].
- The KeyGen, Encaps and Decaps are $30\times, 5\times$, and $8\times$ faster than the respective procedures in the NTRU schemes.
- Parameters: $q = 7681, n = 768$.

Number Theoretic Transform (NTT)

- Kyber and NTTRU use 16-bit NTT for polynomial multiplication. Kyber: $\mathbb{Z}_{3329}[X]/(X^{256}+1)$, NTTRU: $\mathbb{Z}_{7681}[X]/(X^{768}-X^{384}+1)$.
- The polynomial ring $Z_q[X]/f(X)$ implemented with NTT factors the polynomial $f(X)$ as

$$f(X)=\prod_{i=0}^{n-1} f_i(X) (\bmod q),$$

where $f_i(X)$ are small degree polynomials like (X^2-r) and $(X^3 \pm r)$ in Kyber and NTTRU, respectively.

- For NTTRU, the polynomial $f(X)=X^{768}-X^{384}+1$ is initially split into $(X^{384}+684)(X^{384}-685)$, then all the way down to irreducible polynomials $X^3 \pm r$.

Montgomery and Barrett Arithmetic

State-of-the-art: Montgomery and Barrett arithmetic.

Algorithm 1 Signed Montgomery multiplication

Input: Constant $\beta = 2^l$ where l is the machine word size, odd q such that $0 < q < \frac{\beta}{2}$, and operand a, b such that $-\frac{\beta}{2}q \leq ab < \frac{\beta}{2}q$

Output: $r \equiv ab\beta^{-1} \bmod q, r \in (-q, q)$

1: $c = c_1\beta + c_0 = a \cdot b$
2: $m = c_0 \cdot q^{-1} \bmod^{\pm} \beta$
3: $r = c_1 - \lfloor m \cdot q/\beta \rfloor$
4: **return** r

Algorithm 2 Barrett multiplication

Input: Operand a, b such that $0 \leq a \cdot b < 2^{2l'+\gamma}$, the modulus q satisfying $2^{l'-1} < q < 2^{l'}$, and the precomputed constant $\lambda = \left\lfloor 2^{2l'+\gamma}/q \right\rfloor$

Output: $r \equiv a \cdot b \bmod q, r \in [0, q]$

1: $c = a \cdot b$
2: $t = \lfloor (c \cdot \lambda)/2^{2l'+\gamma} \rfloor$
3: $r = c - t \cdot q$
4: **return** r

Both Montgomery and Barrett multiplication:

- need 3 multiplications;
- use the product $c = a \cdot b$ twice;
- replace division with cheaper shift (non-word-size for Barrett's).

Plantard's Word Size Modular Multiplication

Plantard [Pla21] proposed a novel word size modular multiplication (Plantard multiplication). For simplicity, we denote $X \bmod 2^{l'}$ as $[X]_{l'}$, $X >> l'$ as $[X]^{l'}$ below.

Algorithm 3 Original Plantard Multiplication [Pla21]

Input: Unsigned integers $a, b \in [0, q]$, $q < \frac{2^l}{\phi}, \phi = \frac{1+\sqrt{5}}{2}, q' \equiv q^{-1} \bmod 2^{2l}$, where l is the machine word size

Output: $r \equiv ab(-2^{-2l}) \bmod q$ where $r \in [0, q]$

1: $r = \left[\left([[abq']_{2l}]^{l} + 1\right) q\right]^{l}$

2: **return** r

Plantard multiplication:

- also needs 3 multiplications;
- uses the product $a \cdot b$ once; saves one multiplication when one of the operands (b) is constant by precomputing $bq' \bmod 2^{2l}$;
- has many similarities with Montgomery arithmetic.

Motivations

Plantard multiplication has the following properties:

1. **Pros:** One multiplication can be saved when multiplying a constant; but it introduces an $l \times 2l$-bit multiplication.
2. **Cons:** The original Plantard multiplication only supports **unsigned integers**. In LBC schemes, this requires
    - an extra addition by a multiple of q during each butterfly unit;
    - expensive modular reduction after each layer of butterflies.

The state-of-the-art Montgomery multiplication:

- supports signed inputs in $-q2^{l-1} < ab < q2^{l-1}$;
- enables excellent lazy reduction strategy in NTT/INTT.

Motivations. We aim to support signed integers for Plantard multiplication, enlarge its input range, and utilize its efficient modular multiplication by a constant in LBC.

Improved Plantard Arithmetic

Observations:

- The original modulus restriction: $q < 2^{l-1} < \frac{2^l}{\phi}$.
- The moduli in LBC are much smaller, e.g., 12-bit modulus 3329 in Kyber, 13-bit modulus 7681 in NTTRU.

Trick 1. Stricter modulus restriction $q < 2^{l-\alpha-1} < \frac{2^{l-\alpha}}{\phi}$ by introducing a small positive integer α.

Algorithm 4 Improved Plantard multiplication

Input: Operands $a, b \in [-q2^{\alpha}, q2^{\alpha}], q < 2^{l-\alpha-1} < \frac{2^{l-\alpha}}{\phi}$, $q' = q^{-1} \bmod^{\pm} 2^{2l}$

Output: $r = ab(-2^{-2l}) \bmod^{\pm} q$ where $r \in (-\frac{q}{2}, \frac{q}{2})$

1: $r = \left[\left([[abq']_{2l}]^{l} + 2^{\alpha}\right) q\right]^{l}$

2: **return** r

Correctness Proof

Theorem (Correctness of Algorithm 4)

Let q be an odd modulus, l be the minimum word size (power of 2 number, e.g., 16, 32, and 64) such that $q < 2^{l-\alpha-1} < \frac{2^{l-\alpha}}{\phi}$, where $\alpha > 0$ and $\phi = \frac{1+\sqrt{5}}{2}$, then Algorithm 4 is correct for $-q2^{\alpha} \le a, b \le q2^{\alpha}$.

Proof of the above Theorem.

The main step of Algorithm 4 is $r = \left[\left([[abq']_{2l}]^{l} + 2^{\alpha}\right) q\right]^{l}$, namely:

$$r = \left\lfloor \frac{\left(\left\lfloor \frac{abq' \bmod^{\pm} 2^{2l}}{2^{l}} \right\rfloor + 2^{\alpha}\right) q}{2^{l}} \right\rfloor.$$

Correctness Proof: Step 1

We first check that $r \in (-\frac{q}{2}, \frac{q}{2})$. Since $\left\lfloor \frac{abq' \bmod^{\pm} 2^{2l}}{2^l} \right\rfloor \in [-2^{l-1}, 2^{l-1}-1]$, we have

$$\left\lceil \frac{(-2^{l-1}+2^{\alpha})q}{2^l} \right\rceil \leq r \leq \left\lfloor \frac{(2^{l-1}-1+2^{\alpha})q}{2^l} \right\rfloor$$
$$\left\lceil -\frac{q}{2} + \frac{q}{2^{l-\alpha}} \right\rceil \leq r \leq \left\lfloor \frac{q}{2} + \frac{(2^{\alpha}-1)q}{2^l} \right\rfloor .$$

Since $\frac{q}{2^{l-\alpha}} < \frac{1}{2}$, we can get $r > -\frac{q}{2}$ from the left-hand side of the inequation.

Correctness Proof: Step 1

We first check that $r \in (-\frac{q}{2}, \frac{q}{2})$. Since $\left\lfloor \frac{abq' \bmod^{\pm} 2^{2l}}{2^l} \right\rfloor \in [-2^{l-1}, 2^{l-1} - 1]$, we have

$$\left\lceil \frac{(-2^{l-1} + 2^{\alpha})q}{2^l} \right\rceil \leq r \leq \left\lfloor \frac{(2^{l-1} - 1 + 2^{\alpha})q}{2^l} \right\rfloor$$
$$\left\lceil -\frac{q}{2} + \frac{q}{2^{l-\alpha}} \right\rceil \leq r \leq \left\lfloor \frac{q}{2} + \frac{(2^{\alpha} - 1)q}{2^l} \right\rfloor.$$

Since $\frac{q}{2^{l-\alpha}} < \frac{1}{2}$, we can get $r > -\frac{q}{2}$ from the left-hand side of the inequation. Let's consider $\frac{q}{2} + \frac{(2^{\alpha}-1)q}{2^l}$ on the right-hand side; since $q < 2^{l-\alpha-1} < \frac{2^{l-\alpha}}{\phi}$, we obtain that

$$\frac{(2^{\alpha} - 1)q}{2^l} < \frac{q2^{\alpha}}{2^l} < \frac{2^{\alpha}2^{l-\alpha-1}}{2^l} = \frac{1}{2}.$$

Since q is an odd number, then

$$\left\lfloor \frac{q}{2} + \frac{(2^{\alpha} - 1)q}{2^l} \right\rfloor = \left\lfloor \frac{q}{2} \right\rfloor < \left\lfloor \frac{q+1}{2} \right\rfloor.$$

Therefore, the result r lies in $(-\frac{q}{2}, \frac{q}{2})$.

Correctness Proof: Step 2

Then, we check that $r = ab(-2^{-2l}) \bmod^{\pm} q$. Since $q < 2^{l-\alpha-1} < \frac{2^{l-\alpha}}{\phi}$ is an odd number, there exists a $2l$-bit number $p = abq^{-1} \bmod^{\pm} 2^{2l}$ so that

$$pq - ab \equiv \left(abq^{-1}\right) q - ab \bmod 2^{2l} \equiv ab - ab \bmod 2^{2l} \equiv 0 \bmod 2^{2l}.$$

Then, $pq - ab$ is divisible by 2^{2l}, so

$$ab\left(-2^{-2l}\right) \bmod q \equiv \frac{pq - ab}{2^{2l}}.$$

Correctness Proof: Step 2

Then, we check that $r = ab(-2^{-2l}) \bmod^{\pm} q$. Since $q < 2^{l-\alpha-1} < \frac{2^{l-\alpha}}{\phi}$ is an odd number, there exists a $2l$-bit number $p = abq^{-1} \bmod^{\pm} 2^{2l}$ so that

$$pq - ab \equiv \left(abq^{-1}\right) q - ab \bmod 2^{2l} \equiv ab - ab \bmod 2^{2l} \equiv 0 \bmod 2^{2l}.$$

Then, $pq - ab$ is divisible by 2^{2l}, so

$$ab\left(-2^{-2l}\right) \bmod q \equiv \frac{pq - ab}{2^{2l}}.$$

Let $p_1 = \left\lfloor \frac{p}{2^l} \right\rfloor$, $p_0 = p - p_1 2^l$ and $p_0 \in [0, 2^l)$.

Trick 2. Instead of analyzing $q2^l - p_0 q + ab$ in the original work, we slightly modify the equation to $q2^{l+\alpha} - p_0 q + ab$. The correctness of the original Plantard multiplication is based on the inequality: $0 < q2^l - p_0 q + ab < 2^{2l}$.

Correctness Proof: Step 2

We now check that our modified equation $q2^{l+\alpha} - p_0 q + ab$ also satisfies this inequality:

$$0 < q2^{l+\alpha} - p_0 q + ab < 2^{2l} \tag{1}$$

under the restrictions $q < 2^{l-\alpha-1} < \frac{2^{l-\alpha}}{\phi}$, $\alpha > 0$, and $-q2^{\alpha} \leq a, b \leq q2^{\alpha}$.

(1) First, as for two positive inputs or two negative inputs a, b, we have

$$q2^{l+\alpha} - p_0 q + ab < q2^{l+\alpha} + ab \leq q2^{l+\alpha} + (q2^{\alpha})^2$$
$$< \frac{2^{l-\alpha}}{\phi} \cdot 2^{l+\alpha} + (\frac{2^{l-\alpha}}{\phi} \cdot 2^{\alpha})^2 = \frac{2^{2l}}{\phi} + \frac{2^{2l}}{\phi^2} = 2^{2l} \cdot \frac{\phi + 1}{\phi^2}.$$

Since $\frac{\phi+1}{\phi^2} = 1$ according to [Pla21], we have $q2^{l+\alpha} - p_0 q + ab < 2^{2l}$. The proof of the right-hand side of Equation 1 ends.

Correctness Proof: Step 2

(2) As for one positive and one negative input such that $ab < 0$, we have

$$q2^{l+\alpha} - p_0 q + ab \geq q2^{l+\alpha} - q2^{l} - q^2 2^{2\alpha} = q\left(2^{l+\alpha} - 2^{l} - q2^{2\alpha}\right)$$
$$> q(2^{l+\alpha} - 2^{l} - 2^{l-\alpha-1}2^{2\alpha}) = q(2^{l+\alpha-1} - 2^{l}) \geq 0.$$

The proof of the left-hand side of Equation 1 ends. Therefore, we obtain

$$0 < \frac{q2^{l+\alpha} - p_0 q + ab}{2^{2l}} < 1.$$

Correctness Proof: Step 2

(2) As for one positive and one negative input such that $ab < 0$, we have

$$q2^{l+\alpha} - p_0 q + ab \geq q2^{l+\alpha} - q2^l - q^2 2^{2\alpha} = q\left(2^{l+\alpha} - 2^l - q2^{2\alpha}\right)$$
$$> q(2^{l+\alpha} - 2^l - 2^{l-\alpha-1}2^{2\alpha}) = q(2^{l+\alpha-1} - 2^l) \geq 0.$$

The proof of the left-hand side of Equation 1 ends. Therefore, we obtain

$$0 < \frac{q2^{l+\alpha} - p_0 q + ab}{2^{2l}} < 1.$$

Overall, we have the following equation:

$$ab\left(-2^{-2l}\right) \bmod q \equiv \frac{pq - ab}{2^{2l}} \equiv \left\lfloor \frac{pq - ab}{2^{2l}} + \frac{q2^{l+\alpha} - p_0 q + ab}{2^{2l}} \right\rfloor \equiv \left\lfloor \frac{qp_1 2^l + q2^{l+\alpha}}{2^{2l}} \right\rfloor$$
$$\equiv \left\lfloor \frac{q(p_1 + 2^\alpha)}{2^l} \right\rfloor \equiv \left\lfloor \frac{q\left(\left\lfloor \frac{abq^{-1} \bmod^{\pm} 2^{2l}}{2^l} \right\rfloor + 2^\alpha\right)}{2^l} \right\rfloor.$$

For signed inputs, we have $ab(-2^{-2l}) \bmod^{\pm} q = \left[\left([[abq']_{2l}]^l + 2^\alpha\right) q\right]^l = r$.

Comparisons

(1) versus Original Plantard multiplication.

- **Signed support.** Supports signed inputs and produces signed output in $(-\frac{q}{2}, \frac{q}{2})$.
- **Input range.** Extends the input range from $[0, q]$ up to $[-q2^{\alpha}, q2^{\alpha}]$. Eliminate the final correction step in the original version.

(2) versus Montgomery and Barrett arithmetic.

- **Efficiency.** The Plantard arithmetic saves one multiplication when multiplying a constant. Moreover, the Barrett arithmetic may require an explicit shift operation for a non-word-size offset.
- **Input range.** The Plantard reduction accepts input in $[-q^2 2^{2\alpha}, q^2 2^{2\alpha}]$, which is about 2^{α} times bigger than Montgomery reduction $[-q2^{l-1}, q2^{l-1}]$. Besides, the improved Plantard reduction can replace the Barrett reduction inside the NTT/INTT of Kyber and NTTRU.

Comparisons

- **Output range.** The output range of the improved algorithm is $1\times$ smaller than the Montgomery algorithm. Therefore, it halves or slows down the growing rate of the coefficient size in the NTT with CT butterflies or the INTT with GS butterflies, respectively.

(3) Weak Spots.

- **Special Multiplication.** The Plantard arithmetic introduces an $l \times 2l$-bit multiplication. We show that it is perfectly suitable on Cortex-M4/7 and some 32-bit microcontrollers when $l = 16$.
- **Load/Store Issue.** The precomputed twiddle-factors are double-size compared to the implementation with Montgomery arithmetic. It requires extra cycles to load/store the twiddle factors.

Target Platform: Cortex-M4

Cortex-M4:

- NIST's reference platform (the popular pqm4 repository: https://github.com/mupq/pqm4);
- 1MB flash, 192kB RAM.
- 14 32-bit usable general-purpose registers; 32 32-bit FP registers;
- SIMD extension: **uadd16, usub16** perform addition and subtraction for two packed 16-bit vectors; **smulw{b,t}** can efficiently compute the 16×32-bit multiplication in Plantard arithmetic.
- 1-cycle multiplication instruction: **smulw{b,t}**, **smul{b,t}{b,t}**
- Relative expensive load instructions, e.g., **ldr, ldrd, vldm**.

Efficient Plantard multiplication by a constant

(1) We set $l = 16$, $\alpha = 3$ in Kyber, $\alpha = 2$ in NTTRU s.t. $q < 2^{l-\alpha-1} < \frac{2^{l-\alpha}}{\phi}$.
(2) Efficient 2-cycle improved Plantard multiplication by a constant:

- reduce b down $[0, q)$; the input range of a is extended to $[-q2^{2\alpha}, q2^{2\alpha}]$.
- $\left[\left([[abq']_{2l}]^l + 2^\alpha\right) q\right]^l$ vs $\left[q\,[[abq']_{2l}]^l + q2^\alpha\right]^l$.

Algorithm 5 The 2-cycle improved Plantard multiplication by a constant on Cortex-M4

Input: An l-bit signed integer $a \in [-2^{l-1}, 2^{l-1})$, a precomputed $2l$-bit integer bq' where b is a constant and $q' = q^{-1} \bmod^{\pm} 2^{2l}$
Output: $r_{top} = ab(-2^{-2l}) \bmod^{\pm} q$, $r_{top} \in (-\frac{q}{2}, \frac{q}{2})$
1: $bq' \leftarrow bq^{-1} \bmod^{\pm} 2^{2l}$ ▷ precomputed
2: **smulwb** r, bq', a ▷ $r \leftarrow [[abq']_{2l}]^l$
3: **smlabb** $r, r, q, q2^\alpha$ ▷ $r_{\text{top}} \leftarrow [q[r]_l + q2^\alpha]^l$
4: **return** r_{top}

Algorithm 6 The 3-cycle Montgomery multiplication on Cortex-M4 [ABCG20]

Input: Two l-bit signed integers a, b such that $ab \in [-q2^{l-1}, q2^{l-1})$
Output: $r_{top} = ab2^{-l} \bmod^{\pm} q$, $r_{top} \in (-q, q)$
1: **mul** c, a, b
2: **smulbb** $r, c, -q^{-1}$ ▷ $r \leftarrow [c]_l \cdot \left(-q^{-1}\right)$
3: **smlabb** r, r, q, c ▷ $r_{\text{top}} \leftarrow [[r]_l \cdot q]^l + [c]^l$
4: **return** r_{top}

Efficient Plantard reduction

Plantard reduction for the modular multiplication of two variables.

- As efficient as the state-of-the-art Montgomery reduction;
- The input range is $c \in [-q^2 2^{2\alpha}, q^2 2^{2\alpha}]$, which is about 2^{α} times than Montgomery's $(-q2^{l-1}, q2^{l-1})$.

Algorithm 7 The 2-cycle improved Plantard reduction on Cortex-M4

Input: A $2l$-bit signed integer $c \in [-q^2 2^{2\alpha}, q^2 2^{2\alpha}]$
Output: $r_{top} = c(-2^{-2l}) \bmod^{\pm} q$, $r_{top} \in (-\frac{q}{2}, \frac{q}{2})$

1: $q' \leftarrow q^{-1} \bmod^{\pm} 2^{2l}$ ▷ precomputed
2: **mul** r, c, q'
3: **smlatb** $r, r, q, q2^{\alpha}$
4: **return** r_{top}

Butterfly unit

- Precompute twiddle factors as $\zeta = (\zeta \cdot (-2^{2l}) \bmod q) \cdot q^{-1} \bmod^{\pm} 2^{2l}$;
- **smulwb** and **smulwt** for $l \times 2l$-bit multiplication; reduce 2 cycles.

Algorithm 8 Double CT butterfly on Cortex-M4

Input: Two 32-bit packed signed integers a, b (each containing a pair of 16-bit signed coefficients), the 32-bit twiddle factor ζ

Output: $a = (a_{top} + b_{top}\zeta)||(a_{bottom} + b_{bottom}\zeta), b = (a_{top} - b_{top}\zeta)||(a_{bottom} - b_{bottom}\zeta)$

1: **smulwb** t, ζ, b
2: **smulwt** b, ζ, b
3: **smlabb** $t, t, q, q2^{\alpha}$
4: **smlabb** $b, b, q, q2^{\alpha}$
5: **pkhtb** $t, b, t,$ asr#16
6: **usub16** b, a, t
7: **uadd16** a, a, t
8: **return** a, b

Layer merging: 3-layer merging strategy

Using the improved Plantard arithmetic introduces the 32-bit twiddle factors, thus requiring extra loading cycles.

- Each iteration of each layer computes 8 butterflies over 16 coefficients at the cost of loading 1, 2, or 4 twiddle factors.
- Reduce 8 cycles at the cost of 0, 1, or 2 extra cycles for loading twiddle factors (**ldr,ldrd**) in each iteration of each layer.

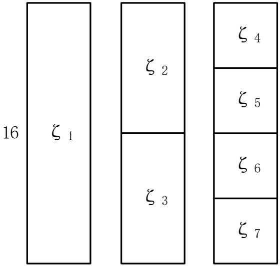

Figure 1: 3-layer merging CT butterfly

Layer merging: 4-layer merging strategy

- Each iteration of each layer computes 16 butterflies over 32 coefficients at the cost of loading 1,2,4, or 8 twiddle factors.
- The 4-layer merging strategy is used only when twiddle factors are reused multiple times.
- The Montgomery-based implementation loads all 15 twiddle factors into 8 FP registers with **vldm** instruction once and replaces the 2-cycle **ldrh,ldr** with the cheaper 1-cycle **vmov**.
- Instead of packing 15 16-bit twiddle factors into 8 32-bit FP registers, we need 15 32-bit FP registers. The **vldm** with 15 registers needs 7 more cycles than the one with 8 registers.
- Each iteration needs 7 extra **vmov** instructions to retrieve the twiddle factors from the FP registers to general registers, namely reduces 16×4 cycles with the cost of 7 extra cycles.

Better lazy reduction strategies

(1) Montgomery reduction:
input range: $[-q2^{l-1}, q2^{l-1}]$, output range: $(-q, q)$.
(2) Improved Plantard reduction:
input range: $[-q^2 2^{2\alpha}, q^2 2^{2\alpha}]$, output range: $(-\frac{q}{2}, \frac{q}{2})$.

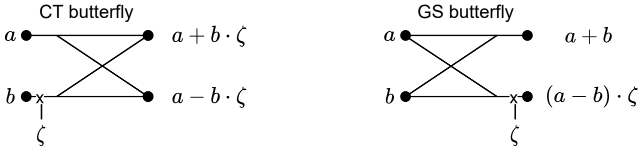

Figure 2: CT and GS butterflies

- **CT butterfly:** Coefficients grow by q or $\frac{q}{2}$ after each layer.
- **GS butterfly:** The first half of the coefficients double while the second half are reduced down to q or $\frac{q}{2}$ after each layer.

Better lazy reduction strategies: CT butterflies

Kyber: $q = 3329(9q < 2^{l-1} < 10q)$. The input of NTT is smaller than q.

- **Montgomery:** 7 layers of butterflies generates 256 coefficients by $7q$. Require 1 modular reduction for 256 coefficients since $8q$ is bigger than the input range of Montgomery multiplication, i.e., $[-\sqrt{q2^{l-1}}, \sqrt{q2^{l-1}}]$.
- **Plantard:** 7 layers of butterflies generates 256 coefficients by $3.5q$. $4.5q$ lies in the input range of Plantard multiplication, i.e., $[-q2^{\alpha}, q2^{\alpha}]$.

NTTRU: $q = 7681(4q < 2^{l-1} < 5q)$. The input of NTT is smaller than $0.5q$.

- **Montgomery:** We need two modular reductions after the 3rd and 6-th layer. The final two layers of butterflies generate coefficients smaller than $3q$, which is bigger than the input range of Montgomery multiplication; thus one more modular reduction for 768 coefficients is required.
- **Plantard:** Only needs one modular reduction after the 7-th layer. The final layer of butterflies generates coefficients smaller than $1q$, which lies in the input range of Plantard multiplication.

Better lazy reduction strategies: GS butterflies

Kyber: $q = 3329 (9q < 2^{l-1} < 10q)$. The advantages of applying the Plantard arithmetic are twofold in Kyber:

- Halve the matrix-vector product from kq to $\frac{kq}{2}$, $k = 2, 3, 4$ and have one-layer delay of the modular reduction. One modular reduction is required after the 2nd and 3rd layer when $k = 3, 4$ and $k = 2$.
- After one modular reduction, 4 layers of butterflies can be carried out instead of 3 with Montgomery arithmetic.

For Kyber768/Kyber1024, one modular reduction is needed after the 2nd layer. Then, after the 6th layer, 16 coefficients $(a_0 \sim a_7, a_{128} \sim a_{135})$ will grow to $8q$ and need to be reduced instead of 128 coefficients with Montgomery arithmetic.

NTTRU: $q = 7681 (4q < 2^{l-1} < 5q)$.

- After one modular reduction, 3 layers of butterflies can be carried out instead of 2 with Montgomery arithmetic.
- Only need two modular reductions for 384 coefficients instead of four with Montgomery arithmetic.

5-cycle double Plantard reduction inside NTT/INTT

- The Plantard reduction over a 16-bit signed integer can be viewed as a Plantard multiplication by the "Plantard" constant $-2^{2l} \bmod q$;
- 1-cycle/3-cycle faster than the 6-cycle/8-cycle double Barrett reduction with/without explicit shift operations in [AHKS22], and 2-cycle faster than the 7-cycle double Montgomery reduction in [ABCG20].

Algorithm 9 Double Plantard reduction for packed coefficients

Input: A 32-bit packed integers $a = a_{top}||a_{bottom}$ where a_{top}, a_{bottom} are two 16-bit signed coefficients

Output: $r = (a_{top} \bmod^{\pm} q)||(a_{bottom} \bmod^{\pm} q)$, $-q/2 < r_{top}, r_{bottom} < q/2$

1: $\mathrm{const} \leftarrow (-2^{2l} \bmod q) \cdot (q^{-1} \bmod^{\pm} 2^{2l}) \bmod^{\pm} 2^{2l}$ ▷ precomputed
2: **smulwb** t, const, a
3: **smulwt** a, const, a
4: **smlabt** $t, t, q, q2^{\alpha}$
5: **smlabt** $a, a, q, q2^{\alpha}$
6: **pkhtb** r, a, t, asr#16
7: **return** r

Extensibility on other 32-bit microcontrollers

- The improved Plantard arithmetic for 16-bit modulus on Cortex-M4 relies on the efficiency of the 16×32-bit multiplication instruction **smulwb**.
- The Plantard multiplication by a constant on other 32-bit microcontroller, like RISC-V, is shown in Algorithm 10. It reduces 1 multiplication and introduces 1 shift instruction compared to Montgomery's.

Algorithm 10 The improved Plantard multiplication by a constant on RISC-V

Input: A 32-bit signed integer $a \in [-q2^{2\alpha}, q2^{2\alpha}]$, a precomputed $2l$-bit integer bq' where b is a constant, $q' = q^{-1} \bmod^{\pm} 2^{2l}$

Output: $r = ab(-2^{-2l}) \bmod^{\pm} q$, $r \in (-\frac{q}{2}, \frac{q}{2})$

1: $bq' \leftarrow bq^{-1} \bmod^{\pm} 2^{2l}$ ▷ precomputed
2: **mul** r, a, bq' ▷ $r \leftarrow [abq']_{2l}$
3: **srai** $r, r, \#16$ ▷ $r \leftarrow [[abq']_{2l}]^{l}$
4: **mul** r, r, q
5: **add** $r, r, q2^{\alpha}$ ▷ $r \leftarrow q[[abq']_{2l}]^{l} + q2^{\alpha}$
6: **srai** $r, r, \#16$ ▷ $r \leftarrow [q[[abq']_{2l}]^{l} + q2^{\alpha}]^{l}$
7: **return** r

Performance of the Polynomial Arithmetic

Table 1: Cycle counts for the core polynomial arithmetic in Kyber and NTTRU on Cortex-M4, i.e., NTT, INTT, base multiplication, and base inversion.

Scheme	Implementation	NTT	INTT	Base Mult	Base Inv
Kyber	[ABCG20]	6 822	6 951	2 291	-
	This work[a]	5 441	5 775	2 421	-
	Speed-up	20.24%	16.92%	-5.67%	-
	Stack[AHKS22]	5 967	5 917	2 293	-
	Speed[AHKS22]	5 967	5 471	1 202	-
	This work[b]	4 474	4 684/4 819/4 854	2 422	-
	Speed-up (stack)	25.02%	20.84%/18.56%/17.97%	-5.58%	-
	Speed-up (speed)	25.02%	14.38%/11.92%/11.28%	-101.41%	-
NTTRU	[LS19]	102 881	97 986	44 703	100 249
	This work	17 274	20 931	10 550	40 763
	Speed-up	83.21%	78.64%	76.40%	59.34%

[a] Implementation based on [ABCG20], [b] Implementation based on the stack-friendly code of [AHKS22].

Performance of Schemes

Table 2: Cycle counts (cc) and stack usage (Bytes) for KeyGen, Encaps, and Decaps on Cortex-M4. k is the dimension of the underlying Module-LWE problem for Kyber. The first row of each entry indicates the cycle count and the second row refers to stack usage.

Scheme	Implementation	KeyGen			Encaps			Decaps		
		$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$
Kyber	[ABCG20]	454k 2 464	741k 2 696	1 177k 3 584	548k 2 168	893k 2 640	1 367k 3 208	506k 2 184	832k 2 656	1 287k 3 224
	This work[a]	446k 2 464	729k 2 696	1 162k 3 584	542k 2 168	885k 2 640	1 357k 3 208	497k 2 184	818k 2 656	1 270k 3 224
	Stack[AHKS22]	439k 2 608	717k 3 056	1 139k 3 576	534k 2 160	871k 2 660	1 329k 3 236	484k 2 176	797k 2 676	1 233k 3 252
	Speed[AHKS22]	438k 4 268	711k 6 732	1 129k 7 748	531k 5 252	864k 6 284	1 316k 7 292	479k 5 260	787k 6 308	1 217k 7 300
	This work[b]	430k 2 608	702k 3 056	1 119k 3 576	526k 2 160	861k 2 660	1 314k 3 236	472k 2 176	780k 2 676	1 211k 3 252
NTTRU	[LS19]	526k 9 384			431k 8 748			559k 10 324		
	This work	267k 9 372			237k 7 452			254k 8 816		

[a] Implementation based on [ABCG20], [b] Implementation based on the stack-friendly code of [AHKS22].

Conclusions and Future Work

Conclusions:

- An improved Plantard arithmetic taliored for Lattice-based cryptography.
- Excellent merits over the original Plantard, Montgomery, and Barrett arithmetic.
- Speed-ups for Kyber and NTTRU with 16-bit NTT on Cortex-M4.

Furture work:

- Application on other platforms like AVX2, NEON or other 32-bit microcontrollers.
- Application on other schemes with 32-bit NTT like Saber, NTRU, Dilithium.
- Application in other scenarios where modular multiplication by a constant is widely used.

References I

[ABCG20] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard.
Cortex-M4 optimizations for {R, M} LWE schemes.
IACR Trans. Cryptogr. Hardw. Embed. Syst., 2020(3):336–357, 2020.

[AHKS22] Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Daan Sprenkels.
Faster Kyber and Dilithium on the Cortex-M4.
In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 853–871. Springer, 2022.

[LS19] Vadim Lyubashevsky and Gregor Seiler.
NTTRU: Truly fast NTRU using NTT.
IACR Trans. Cryptogr. Hardw. Embed. Syst., 2019(3):180–201, 2019.

[Pla21] Thomas Plantard.
Efficient word size modular arithmetic.
IEEE Trans. Emerg. Top. Comput., 9(3):1506–1518, 2021.

Thanks!