Improved Plantard Arithmetic for Lattice-based Cryptography
In TCHES2022, Issue 4

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

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

September 19, 2022, Leuven
1 Introduction

2 Improved Plantard Arithmetic

3 Optimized Implementation on Cortex-M4

4 Results and Comparisons

5 Conclusions and Future Work
1 Introduction

Kyber and NTTRU

NTT and Modular Arithmetic

2 Improved Plantard Arithmetic

3 Optimized Implementation on Cortex-M4

4 Results and Comparisons

5 Conclusions and Future Work
Kyber and NTTRU

Kyber

- One of the third-round KEM finalists (The final KEM scheme to be standardized).
- Module-LWE problem \((A, b = A^T s + e)\).
- Parameters: \(n = 256, q = 3329, k = 2, 3, 4\).

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: \(n = 768, q = 7681\).
Improved Plantard Arithmetic

Optimized Implementation on Cortex-M4

Results

Conclusions

NTT and Modular Arithmetic

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 $\mathbb{Z}_q[X]/f(X)$ implemented with NTT factors the polynomial $f(X)$ as

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

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

**Figure 1:** CT and GS butterflies

Junhao Huang et. al

BNU-HKBU United International College

Improved Plantard Arithmetic for Lattice-based Cryptography, TCHES2022

5 / 27
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: $r \equiv ab\beta^{-1} \mod q, r \in (-q, q)$

1: $c = c_1\beta + c_0 = a \cdot b$
2: $m = c_0 \cdot q^{-1} \mod \pm \beta$
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 $\lambda = \lfloor 2^{2l'}+\gamma / q \rfloor$
Output: $r \equiv a \cdot b \mod q, r \in [0, q]$

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;
- support signed inputs in a large domain: "lazy reduction strategy".
Plantard’s Word Size Modular Multiplication

**Algorithm 3 Original Plantard Multiplication [Pla21]**

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

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

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

2: return $r$

Plantard multiplication [Pla21]:

- also needs 3 multiplications;
- uses the product $a \cdot b$ once; saves one multiplication when one of the operands (b) is constant by precomputing $bq' \mod 2^{2^l}$;
- only supports **unsigned integers** in a small domain $[0, q]$;
  - an extra addition by a multiple of $q$ during each butterfly unit;
  - expensive modular reduction after each layer of butterflies.
1 Introduction

2 Improved Plantard Arithmetic

3 Optimized Implementation on Cortex-M4

4 Results and Comparisons

5 Conclusions and Future Work
Improved Plantard multiplication

Observations:

- The original modulus restriction: \( q < \frac{2^l}{\phi}, \phi = \frac{1 + \sqrt{5}}{2} \).
- The moduli in Lattice-based Cryptography (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 integer \( \alpha \geq 0 \).

**Trick 2.** The correctness of the original Plantard multiplication is based on the inequality: \( 0 < q2^l - p_0 q + ab < 2^{2l} \). We modify and prove that \( 0 < q2^{l+\alpha} - p_0 q + ab < 2^{2l} \) for \( a, b \in [-q2^\alpha, q2^\alpha] \) under the new modulus restriction.
Improved Plantard multiplication

**Algorithm 4** Improved Plantard multiplication

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

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

1. $r = \left\lfloor \left( \lceil [abq']2^l \rceil + 2^\alpha \right) \frac{q}{2} \right\rfloor$
2. return $r$

**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 $q < 2^{l-\alpha-1}$, where $\alpha \geq 0$, then Algorithm 4 is correct for $-q2^\alpha \leq a, b \leq q2^\alpha$. 
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 \([-q^{2^\alpha}, q^{2^\alpha}]\). 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. Besides, the Barrett arithmetic may require an explicit shift operation for a non-word-size offset.

- **Input range.** The Plantard reduction accepts input in \([-q^{2^22^\alpha}, q^{2^22^\alpha}]\), which is about \(2^\alpha\) times bigger than Montgomery reduction \([-q^{2^{l-1}}, q^{2^{l-1}}]\).
Comparisons

- **Output range.** The output range of the improved algorithm \((-\frac{q}{2}, \frac{q}{2})\) is half of the Montgomery’s \((-q, q)\).

(3) **Weak Spots.**

- **Special Multiplication.** The Plantard arithmetic introduces an \(l \times 2l\)-bit multiplication. We show that it is 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 the twiddle factors.
Introduction

2 Improved Plantard Arithmetic

3 Optimized Implementation on Cortex-M4
   - Efficient Plantard Arithmetic for 16-bit Modulus on Cortex-M4
   - Efficient 16-bit NTT/INTT Implementation on Cortex-M4

4 Results and Comparisons

5 Conclusions and Future Work
Cortex-M4:

- NIST’s reference platform (the popular pqm4 repository: https://github.com/mupq/pqm4);
- SIMD extension: `uadd16`, `usub16` perform addition and subtraction for two packed 16-bit vectors; `smulw{b,t}` can efficiently compute the $16 \times 32$-bit multiplication in Plantard arithmetic.
- 1-cycle multiplication instruction: `smulw{b,t}`, `smul{b,t}{b,t}`
- Relative expensive load/store 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. $q < 2^{l-\alpha-1}$.

(2) Efficient 2-cycle improved Plantard multiplication by a constant:

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

**Algorithm 5** The 2-cycle 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 $q' = q^{-1} \mod \pm 2^{2l}$

*Output:* $r_{top} = ab(-2^{2l}) \mod \pm q$, $r_{top} \in (-\frac{q}{2}, \frac{q}{2})$

1: $bq' \leftarrow bq^{-1} \mod \pm 2^{2l}$ \hspace{1cm} $\triangleright$ precomputed
2: `smulwb` $r, bq'$, $a$ \hspace{0.5cm} $\triangleright$ $r \leftarrow [[abq']_{2l}]$
3: `smlabb` $r, r, q, q2^{\alpha}$ \hspace{0.5cm} $\triangleright$ $r_{top} \leftarrow [q[r]l + q2^{\alpha}]$
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:* $r_{top} = ab2^{-l} \mod \pm q$, $r_{top} \in (-q, q)$

1: `mul` $c, a, b$
2: `smulbb` $r, c, -q^{-1}$ \hspace{0.5cm} $\triangleright$ $r \leftarrow [c]_{l} \cdot (-q^{-1})$
3: `smlabb` $r, r, q, c$ \hspace{0.5cm} $\triangleright$ $r_{top} \leftarrow [[r]l \cdot q]l + [c]$
4: return $r_{top}$
Efficient Plantard Arithmetic

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 \( 2^\alpha \) times bigger than Montgomery's \((-q 2^{l-1}, q 2^{l-1})\).
- The output range is \((-\frac{q}{2}, \frac{q}{2})\), which is half of the Montgomery's \((-q, q)\).

**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:** \( r_{top} = c(-2^{-2l}) \mod^{\pm} q \), \( r_{top} \in (-\frac{q}{2}, \frac{q}{2}) \)

1: \( q' \leftarrow q^{-1} \mod^{\pm} 2^{2l} \) \quad \triangleright \text{precomputed}
2: \textbf{mul} \( r, c, q' \)
3: \textbf{smlatb} \( r, r, q, q2^{\alpha} \)
4: \textbf{return} \( r_{top} \)
Efficient 16-bit NTT/INTT Implementation on Cortex-M4

Butterfly unit

- Precompute twiddle factors as $\zeta = (\zeta \cdot (-2^{2l}) \mod q) \cdot q^{-1} \mod \pm 2^{2l}$;
- `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 $\zeta$

**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, \zeta, b`
2: `smulwt b, \zeta, 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: e.g., 3-layer merging strategy

Using the 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 ($\text{ldr}, \text{ldrd}$) in each iteration of each layer.

![Figure 2: 3-layer merging CT butterfly](image)
Better lazy reduction strategies

(1) **Montgomery reduction:**
input range: \([-q^{2^l-1}, q^{2^l-1}]\), output range: \((-q, q)\).

(2) **Improved Plantard reduction:**
input range: \([-q^22^{2\alpha}, q^22^{2\alpha}]\), output range: \((-\frac{q}{2}, \frac{q}{2})\).

\[
\begin{align*}
\text{CT butterfly:} & \quad a + b \cdot \zeta \\
\text{GS butterfly:} & \quad a + b
\end{align*}
\]

\[
\begin{align*}
\text{CT butterfly} & \quad a - b \cdot \zeta \\
\text{GS butterfly} & \quad (a - b) \cdot \zeta
\end{align*}
\]

**Figure 3:** 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.
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 $-2^{2l} \mod q$;
- 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 $a = a_{\text{top}} || a_{\text{bottom}}$ where $a_{\text{top}}$, $a_{\text{bottom}}$ are two 16-bit signed coefficients

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

1: const $\leftarrow (-2^{2l} \mod q) \cdot (q^{-1} \mod^{\pm} 2^{2l}) \mod^{\pm} 2^{2l}$ \(\triangleright \) 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 $r$
1 Introduction

2 Improved Plantard Arithmetic

3 Optimized Implementation on Cortex-M4

4 Results and Comparisons

5 Conclusions and Future Work
### 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.

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

\(^a\) 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.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Implementation</th>
<th>KeyGen</th>
<th>Encaps</th>
<th>Decaps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$k = 2$</td>
<td>$k = 3$</td>
<td>$k = 4$</td>
</tr>
<tr>
<td>Kyber</td>
<td>[ABC][20]</td>
<td>454k</td>
<td>741k</td>
<td>1177k</td>
</tr>
<tr>
<td></td>
<td>This work$^a$</td>
<td>2464</td>
<td>2696</td>
<td>3584</td>
</tr>
<tr>
<td></td>
<td>[AHKS][22]</td>
<td>446k</td>
<td>729k</td>
<td>1162k</td>
</tr>
<tr>
<td></td>
<td>This work$^b$</td>
<td>2464</td>
<td>2696</td>
<td>3584</td>
</tr>
<tr>
<td>NTTRU</td>
<td>[LS][19]</td>
<td>439k</td>
<td>717k</td>
<td>1139k</td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>2608</td>
<td>3056</td>
<td>3576</td>
</tr>
</tbody>
</table>

$^a$ Implementation based on [ABC][20], $^b$ Implementation based on the stack-friendly code of [AHKS][22].
1. Introduction

2. Improved Plantard Arithmetic

3. Optimized Implementation on Cortex-M4

4. Results and Comparisons

5. Conclusions and Future Work
Conclusions and Future Work

**Conclusions:**

- An improved Plantard arithmetic tailored 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.

**Future work:**

- Application on other platforms like AVX2, NEON or other 32-bit microcontrollers.
- Application on other schemes with 32-bit NTT like Saber [AMOT22], NTRU, Dilithium.

**Eprint paper and source code:**

https://eprint.iacr.org/2022/956.pdf
https://github.com/UIC-ESLAS/ImprovedPlantardArithmetic


NTTRU: Truly fast NTRU using NTT. 

Efficient word size modular arithmetic. 
Thanks!