Cortex-M4 optimizations for \( \{R,M\} \) LWE schemes

Erdem Alkım \(^1,2\)  
Yusuf Alper Bilgin \(^3,4\)  
Murat Cenk \(^4\)  
François Gérard \(^5\)

\(^1\)Department of Computer Engineering, Ondokuz Mayıs University, Turkey  
\(^2\)Fraunhofer SIT, Darmstadt, Germany  
\(^3\)Aselsan Inc., Turkey  
\(^4\)Institute of Applied Mathematics, Middle East Technical University, Turkey  
\(^5\)Université libre de Bruxelles, Brussels, Belgium

✉️ y.alperbilgin@gmail.com

September, 2020
1 Introduction

2 Implementation Details
   • Optimizations for Speed
   • Optimizations for Stack Usage
   • Optimizations of Secret-key Size

3 Results

4 Conclusion
1\textsuperscript{st}, 2\textsuperscript{nd}, and 3\textsuperscript{rd} round finalists including alternate candidates

<table>
<thead>
<tr>
<th></th>
<th>Signatures</th>
<th>KEM/Encryption</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1\textsuperscript{st}</td>
<td>2\textsuperscript{nd}</td>
<td>3\textsuperscript{rd}</td>
</tr>
<tr>
<td>Lattice-based</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Code-based</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Multi-variate</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Symmetric-based</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Other</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>19</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>
Target \{R,M\}LWE Schemes

- **Kyber**
  - One of the third round finalists,
  - Based on MLWE problem,
  - Using 7-level NTT with $\mathbb{Z}_{3329}[X]/(X^{256} + 1)$, and degree-2 schoolbook multiplications.
**Target \{R,M\}LWE Schemes**

- **Kyber**
  - One of the third round finalists,
  - Based on MLWE problem,
  - Using 7-level NTT with \( \mathbb{Z}_{3329}[X]/(X^{256} + 1) \), and degree-2 schoolbook multiplications.

- **NewHope**
  - Eliminated in the second round,
  - Based on RLWE problem,
  - Using 9-level or 10-level NTT with \( \mathbb{Z}_{12289}[X]/(X^{512} + 1) \) or \( \mathbb{Z}_{12289}[X]/(X^{1024} + 1) \).

---

1 E. Alkım, Y. A. Bilgin, M. Cenk, Compact and Simple RLWE Based Key Encapsulation Mechanism, Latincrypt2019
Target \{R,M\}LWE Schemes

- **Kyber**
  - One of the third round finalists,
  - Based on MLWE problem,
  - Using 7-level NTT with $\mathbb{Z}_{3329}[X]/(X^{256} + 1)$, and degree-2 schoolbook multiplications.

- **NewHope**
  - Eliminated in the second round,
  - Based on RLWE problem,
  - Using 9-level or 10-level NTT with $\mathbb{Z}_{12289}[X]/(X^{512} + 1)$ or $\mathbb{Z}_{12289}[X]/(X^{1024} + 1)$.

- **NewHope-Compact**
  - Faster and smaller variant of NewHope,
  - Based on RLWE problem,
  - Using 7-level NTT with $\mathbb{Z}_{3329}[X]/(X^{512} + 1)$, $\mathbb{Z}_{3329}[X]/(X^{728} - X^{384} + 1)$, $\mathbb{Z}_{3329}[X]/(X^{1024} + 1)$, and degree 4, 6 or 8 schoolbook multiplications.

---

1 E. Alkım, Y. A. Bilgin, M. Cenk, *Compact and Simple RLWE Based Key Encapsulation Mechanism*, LatinCrypt2019
NewHope: Algorithm Specifications and Supporting Documentation

Key Generation

Output: public key \( pk = (\hat{b}', \rho) \)
Output: secret key \( sk = \hat{s} \)

1: \( seed \leftarrow \{0, \ldots, 255\}^{32} \)
2: \( \rho, \sigma \leftarrow \text{SHAKE256}(64, seed) \)
3: \( \hat{a} \leftarrow \text{GenA}(\rho) \)
4: \( s \leftarrow \text{Sample}(\sigma, 0) \)
5: \( e \leftarrow \text{Sample}(\sigma, 1) \)
6: \( \hat{b} \leftarrow \hat{a} \circ \text{NTT}(s) + \text{NTT}(e) \)
7: \( \text{return } pk = (\hat{b}', \rho), sk = \hat{s} \)

Decryption

Input: ciphertext \( c = (\hat{u}, h) \)
Input: secret key \( sk = \hat{s} \)
Output: message \( \mu \in \{0, \ldots, 255\}^{32} \)

1: \( v' \leftarrow \text{Decompress}(h) \)
2: \( \text{return } \mu = \text{Decode}(v' - \text{NTT}^{-1}(\hat{u} \circ \hat{s})) \)

Encryption

Input: public key \( pk = (\hat{b}, \rho) \)
Input: message \( \mu \) encoded in \( \mathcal{R}_q \)
Input: seed \( \text{coin} \in \{0, \ldots, 255\}^{32} \)
Output: ciphertext \( c = (\hat{u}', h) \)

1: \( \hat{a} \leftarrow \text{GenA}(\rho) \)
2: \( s' \leftarrow \text{Sample}(\text{coin, 0}) \)
3: \( e' \leftarrow \text{Sample}(\text{coin, 1}) \)
4: \( e'' \leftarrow \text{Sample}(\text{coin, 2}) \)
5: \( \hat{t} \leftarrow \text{NTT}(s') \)
6: \( \hat{u} \leftarrow \hat{a} \circ \hat{t} + \text{NTT}(e') \)
7: \( v' \leftarrow \text{NTT}^{-1}(\hat{b} \circ \hat{t}) + e'' + \mu \)
8: \( \text{return } c = (\hat{u}', \text{Compress}(v')) \)
ARM Cortex-M4

- NIST recommended Cortex-M4 for PQC evaluation
- STM32F4DISCOVERY:
  - 32-bit, ARMv7E-M
  - Includes SIMD instructions
  - 1MB ROM, 192 KB RAM, 168 MHz
  - PQM4
  - 16 registers but only 14 available
Previous optimizations of **Kyber** on Cortex-M4

We also use them in our **NewHope** and **NewHope-Compact** implementations.

- Use signed representation
- Pack two coefficients into one register, utilize `uadd16` or `usub16` for parallel addition/subtraction
- All computations in Montgomery-domain
- Precompute twiddle factors - place them in Flash memory
- Enable link-time optimization (`flto`)

---

1 L. Botros, M. Kannwisher, P. Schwabe, *Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4*, Africacrypt2019
Montgomery Reduction

<table>
<thead>
<tr>
<th>Proposed by Botros et. al. ¹</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: <code>smulbb t, a, q⁻¹</code></td>
<td>1: <code>smulbb t, a, -q⁻¹</code></td>
</tr>
<tr>
<td>2: <code>smulbb t, t, q</code></td>
<td>2: <code>smlabb a, t, q, a</code></td>
</tr>
<tr>
<td>3: <code>usub16 a, a, t</code></td>
<td></td>
</tr>
</tbody>
</table>

- 3200 Montgomery reductions in 
  \((\text{NTT}^{-1}(\text{NTT} (a) \circ \text{NTT} (b))))\) where \(a\) and \(b\) ∈ \(\mathbb{Z}_{3329}[X]/(X^{256} + 1)\)

- Double Montgomery reduction on a packed argument
  - 1 cycle faster than double Barrett reduction

¹ L. Botros, M. Kannwisher, P. Schwabe, Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4, Africacrypt2019
More Aggressive Lazy Reduction

Lazy reductions after component-wise multiplication:

\[ c[1] \leftarrow (a[0] \cdot b[1]) \mod q + (a[1] \cdot b[0]) \mod q \]
\[ c[1] \leftarrow (a[0] \cdot b[1]) + a[1] \cdot b[0]) \mod q \]
More Aggresive Lazy Reduction

Lazy reductions after component-wise multiplication:

\[
\begin{align*}
    c[1] & \leftarrow (a[0] \cdot b[1]) \mod q + (a[1] \cdot b[0]) \mod q \\
    c[1] & \leftarrow (a[0] \cdot b[1] + a[1] \cdot b[0]) \mod q
\end{align*}
\]

- We save:
  - 128 reductions for \( \mathbb{Z}_{3329}[X]/(X^{256} + 1) \),
  - 1536 reductions for \( \mathbb{Z}_{3329}[X]/(X^{512} + 1) \),
  - 3840 reductions for \( \mathbb{Z}_{3457}[X]/(X^{768} - X^{384} + 1) \),
  - 7168 reductions for \( \mathbb{Z}_{3329}[X]/(X^{1024} + 1) \),

- Skip the reductions after the multiplications in the first layer of NTT
  - Inputs are small, sampled from the centered binomial distribution.
Merging NTT Layers

- 8 registers out of 14 reserved for the coefficients
  - Perform 3 or 4 layers of the NTT at a time
  - 3+3+1 for Kyber
  - 4+3+2 or 4+3+3 for NewHope
  - 3+4 for NewHope-Compact
Stack Optimizations

NTT is already stack friendly (entirely in-place).

Previous optimizations for Kyber on Cortex-M4:
- Inline comparison in CCA decapsulation,
- On-the-fly generation of matrix $A$ in matrix-vector multiplication.

In this work, these are also implemented for NewHope and NewHope-Compact.

1 L. Botros, M. Kannwisher, P. Schwabe, Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4, Africacrypt2019
Stack Optimizations

NTT is already stack friendly (entirely in-place).

Previous optimizations for Kyber on Cortex-M4\(^1\):

- Inline comparison in CCA decapsulation,
- On-the-fly generation of matrix \(A\) in matrix-vector multiplication.

In this work, these are also implemented for NewHope and NewHope-Compact.

\(^1\) L. Botros, M. Kannwisher, P. Schwabe, Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4, Africacrypt2019
On-the-fly error addition:

Instead of computing

$$\hat{b} \leftarrow \hat{a} \circ \text{NTT} \left( s \right) + \text{NTT} \left( e \right),$$

we compute

$$\hat{b} \leftarrow \text{NTT} \left( \text{NTT}^{-1} \left( \hat{a} \circ \text{NTT} \left( s \right) \right) + e \right)$$
On-the-fly error addition:

Instead of computing

\[ \hat{b} \leftarrow \hat{a} \circ \text{NTT} (s) + \text{NTT} (e), \]

we compute

\[ \hat{b} \leftarrow \text{NTT} (\text{NTT}^{-1}(\hat{a} \circ \text{NTT} (s)) + e) \]

At the cost of 1 \( \text{NTT}^{-1} \), the stack usage is decreased \( \approx 1 \) polynomial.
Secret-key Size Optimization

- Store secret-key in NTT domain
Secret-key Size Optimization

- Store secret-key in NTT domain
- Store only 32 byte seed, re-run KeyGen during Decaps
Secret-key Size Optimization

- Store secret-key in NTT domain
- Store only 32 byte seed, re-run KeyGen during Decaps
- Store secret-key in normal domain
Secret-key Size Optimization

- Store secret-key in NTT domain
- Store only 32 byte seed, re-run KeyGen during Decaps
- Store secret-key in normal domain
- Store 32 byte secret-key seed
Secret-key Size Optimization

- Store secret-key in NTT domain
- Store only 32 byte seed, re-run KeyGen during Decaps
- Store secret-key in normal domain
- Store 32 byte secret-key seed ✓
Secret-key Size Optimization

- Store secret-key in NTT domain
- Store only 32 byte seed, re-run KeyGen during Decaps
- Store secret-key in normal domain
- Store 32 byte secret-key seed

<table>
<thead>
<tr>
<th></th>
<th>Secret-key size</th>
<th>Decapsulation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber</td>
<td>-96%</td>
<td>+7%</td>
</tr>
<tr>
<td>NewHope</td>
<td>-96%</td>
<td>+18%</td>
</tr>
<tr>
<td>NewHope-Compact</td>
<td>-96%</td>
<td>+9%</td>
</tr>
<tr>
<td>Scheme</td>
<td>Previous work</td>
<td>This work Speed</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>NewHope</td>
<td>512</td>
<td>G: 588 683 (^a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E: 918 558 (^a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: 904 800 (^a)</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>G: 1 161 112 (^a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E: 1 777 918 (^a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: 1 760 470 (^a)</td>
</tr>
<tr>
<td>NH-Cmpct</td>
<td>512</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kyber</td>
<td>512</td>
<td>G: 514 291 (^b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E: 652 769 (^b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: 621 245 (^b)</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>G: 976 757 (^b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E: 1 146 556 (^b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: 1 094 849 (^b)</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>G: 1 575 052 (^b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E: 1 779 848 (^b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: 1 709 348 (^b)</td>
</tr>
</tbody>
</table>

\(^a\) PQM4, commit be0c421aaecaaad4443071bf62ad397d4f40832

\(^b\) L. Botros, M. Kannwisher, P. Schwabe, *Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4*, Africacrypt2019
## Stack Usage Comparison in bytes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>NewHope (This work)</th>
<th>NewHope [KRSS]</th>
<th>NH-Cmpct (This work)</th>
<th>Kyber (This work)</th>
<th>Kyber [BKS19]</th>
</tr>
</thead>
</table>

[KRSS] PQM4, commit be0c421aaeacaad4443071bfcf62ad397d4f40832.
[BKS19] L. Botros, M. Kannwisher, P. Schwabe, Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4, Africacrypt2019
## Cycle Count Comparison for Polynomial Multiplication Functions

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Dimension</th>
<th>NTT</th>
<th>NTT⁻¹</th>
<th>°</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NewHope</strong></td>
<td>512</td>
<td>28662</td>
<td>22836</td>
<td>4736</td>
</tr>
<tr>
<td></td>
<td>512 ([KRSS])</td>
<td>29767</td>
<td>35813</td>
<td>5459</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>63387</td>
<td>49880</td>
<td>9396</td>
</tr>
<tr>
<td></td>
<td>1024 ([KRSS])</td>
<td>59752</td>
<td>71942</td>
<td>10836</td>
</tr>
<tr>
<td></td>
<td>1024 ([AJS16])</td>
<td>86769</td>
<td>97340</td>
<td>14977</td>
</tr>
<tr>
<td><strong>NewHope-Compact</strong></td>
<td>512</td>
<td>12799</td>
<td>13052</td>
<td>7052</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>19647</td>
<td>21226</td>
<td>12749</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>25536</td>
<td>26039</td>
<td>18510</td>
</tr>
<tr>
<td><strong>Kyber</strong></td>
<td>256</td>
<td>6847</td>
<td>6975</td>
<td>2317</td>
</tr>
<tr>
<td></td>
<td>256 ([BKS19])</td>
<td>7754</td>
<td>9377</td>
<td>3076</td>
</tr>
</tbody>
</table>

[KRSS] PQM4, commit be0c421aaecada4443071bfcf62ad397d4f40832.
Conclusion

- We propose various optimizations:
  - More efficient modular reduction,
  - More aggressive layer merging,
  - More aggressive lazy reduction,
  - Optimized small-degree polynomial multiplications,
  - Reduce the stack usage of KeyGen by adding the error term on-the-fly,
  - Trade-off between secret-key size and speed.

- Unified framework to compare **Kyber**, **NewHope**, and **NewHope-Compact**.
Conclusion

- We propose various optimizations:
  - More efficient modular reduction,
  - More aggressive layer merging,
  - More aggressive lazy reduction,
  - Optimized small-degree polynomial multiplications,
  - Reduce the stack usage of KeyGen by adding the error term on-the-fly,
  - Trade-off between secret-key size and speed.

- Unified framework to compare Kyber, NewHope, and NewHope-Compact.

- Using the Gentleman-Sande butterfly in the NTT can be faster for NewHope?

- Using 32-bit signed integers instead of 16-bit can be faster?
Cortex-M4 optimizations for \( \{R,M\} \) LWE schemes

Erdem Alkım  Yusuf Alper Bilgin  Murat Cenk  François Gérard

Source code available online at
https://github.com/erdemalkim/NewHope-Compact-M4

✉️ y.alperbilgin@gmail.com

aselsan