Efficient Implementation of Kyber on Mobile Devices

Lirui Zhao¹, Jipeng Zhang¹, Junhao Huang², Zhe Liu¹³(✉), Gerhard Hancke⁴

¹Nanjing University of Aeronautics and Astronautics, Jiangsu, China
²Beijing Normal University-Hong Kong Baptist University United International College, Guangdong, China
³State Key Laboratory of Cryptology, Beijing, China
⁴City University of Hong Kong, Hong Kong, China

December 10, 2021
1 Short Overview

2 Kyber Scheme

3 Our Implementation

4 Implementation Results

5 Conclusion
Lattice-based Cryptography

- RSA and ECC: Discrete Logarithm and Integer Factorization Problems
  - Hard problems can be solved by Shor’s algorithm
- Lattice-based Cryptography: Hard for quantum computers
  - Kyber is a key encapsulation mechanism (Round 3 candidate): it has three parameter sets to scale security to different levels
- NIST Post-Quantum-Cryptography (PQC) Project
  - 2016, Formal call for proposals
  - 2017, Round 1 algorithms announced (69 submissions)
  - 2019, Round 2 algorithms announced (26 algorithms)
  - 2020, Round 3 algorithms announced (7 Finalists and 8 Alternates)
  - ......
Implementation Platform

- Raspberry Pi 3 (including ARM Cortex-A53 processor)
  - ARMv8-A is the first processor architecture of ARM that supports 64-bit instruction set.
  - The SIMD instruction set NEON in ARMv8-A is widely used to parallelize instructions.
Motivation & Contribution

Motivation
- The most important and time-consuming operation in Kyber scheme is polynomial multiplication.
- The Number Theoretic Transform (NTT) can greatly improve the performance of polynomial multiplication.

Contributions: Efficient implementation of Kyber
- Parallel design: NEON instruction set.
- Optimized Barrett and Montgomery modular reductions.
- Improved utilization of registers.
- The NTT layer merging technique and various strategies.
1 Short Overview

2 Kyber Scheme

3 Our Implementation

4 Implementation Results

5 Conclusion
Kyber scheme

- Kyber is an IND-CPA secure encryption scheme.
- It includes key-generation, encryption, and decryption.

**Algorithm 1 Kyber.CPAPKE.KeyGen**

1: \( \rho, \sigma \leftarrow \$ \{0, 1\}^{256} \times \{0, 1\}^{256} \)
2: \( \hat{A} \in \mathcal{R}_{q}^{k \times k} \leftarrow \text{SampleUniform (}\rho\text{)} \)
3: \( s, e \in \mathcal{R}_{q}^{k} \leftarrow \text{SampleCBD}(\sigma) \)
4: \( \hat{t} \leftarrow \hat{A} \circ \text{NTT}(s) + \text{NTT}(e) \)
5: \( \text{return } pk = (\rho, \hat{t}), sk = \hat{s} \)

**Algorithm 2 Kyber.CPAPKE.Enc**

**Require:** Public key \( pk = (\rho, \hat{t}) \)
**Require:** Message \( m \in \mathcal{R}_{q} \)
**Require:** Random coins \( r \in \{0, 1\}^{256} \)

**Ensure:** Ciphertext \( (u', v') \)
1: \( \hat{A} \in \mathcal{R}_{q}^{k \times k} \leftarrow \text{SampleUniform (}\rho\text{)} \)
2: \( r, e_{1} \in \mathcal{R}_{q}^{k} \leftarrow \text{SampleCBD (}\text{r\text{)}} \)
3: \( e_{2} \in \mathcal{R}_{q} \leftarrow \text{SampleCBD}(r) \)
4: \( \hat{r} \leftarrow \text{NTT}(r) \)
5: \( u \leftarrow \text{NTT}^{-1}(\hat{A}^{T} \circ \hat{r}) + e_{1} \)
6: \( v \leftarrow \text{NTT}^{-1}(\hat{t}^{T} \circ \hat{r}) + e_{2} + m \)
7: \( \text{return } (\text{Compress}(u), \text{Compress}(v)) \)

**Algorithm 3 Kyber.CPAPKE.Dec**

**Require:** Secret key \( sk = \hat{s} \)
**Require:** Compressed ciphertext \( (u', v') \)

**Ensure:** Message \( m \in \mathcal{R}_{q} \)
1: \( u \leftarrow \text{Decompress (}u'\text{)} \)
2: \( v \leftarrow \text{Decompress (}}v'\text{)} \)
3: \( \text{return } v - \text{NTT}^{-1}(\hat{s}^{T} \circ \text{NTT}(u)) \)
1 Short Overview

2 Kyber Scheme

3 Our Implementation

4 Implementation Results

5 Conclusion
Modular reduction is the core operation when computing NTT-based polynomial multiplication.

The traditional method of calculating modular reduction contains division, which is expensive and non-constant time.

The common optimizations: Barrett reduction and Montgomery reduction.

Algorithm 4 Signed Montgomery reduction [14]

Require: $0 < q < \frac{\beta}{2}$ odd, $-\frac{\beta}{2}q \leq a = a_1\beta + a_0 < \frac{\beta}{2}q$ where $0 \leq a_0 < \beta$, $\beta = 2^{16}$

Ensure: $r' \equiv \beta^{-1}a (\mod q)$, $-q < r' < q$

1: $m \leftarrow a_0q^{-1} \mod \beta$ \hspace{2cm} $\triangleright$ Only low-limb needed
2: $t_1 \leftarrow \left\lfloor \frac{mq}{\beta} \right\rfloor$ \hspace{2cm} $\triangleright$ Only high-limb needed
3: $r' \leftarrow a_1 - t_1$

Algorithm 5 Signed Barrett reduction for one word [14]

Require: $0 \leq q < \frac{\beta}{2}$, $-\frac{\beta}{2} \leq a < \frac{\beta}{2}$, $\beta = 2^{16}$

Ensure: $r \equiv a (\mod q)$ with $0 \leq r \leq q$

1: $v \leftarrow \left\lfloor \frac{a}{2^{\lceil \log(q) \rceil - 1}} \cdot \frac{q}{\beta} \right\rfloor$ \hspace{2cm} $\triangleright$ Only high-limb needed
2: $t \leftarrow \left\lfloor \frac{av}{2^{|\log(q)|} - 1} \cdot \frac{q}{\beta} \right\rfloor$ \hspace{2cm} $\triangleright$ Only low-limb needed
3: $t \leftarrow tq \mod \beta$ \hspace{2cm} $\triangleright$ Only low-limb needed
4: $r \leftarrow a - t$
Barrett Reduction

The Barrett reduction approximately represents $\frac{1}{q}$ by using multiplication and shift operations instead of division:

$$\frac{1}{q} \approx \frac{v}{2^k} \rightarrow v = \lfloor \frac{2^k}{q} \rfloor$$

(1)

The result of the Barrett reduction is computed by:

$$a \mod q = a - ((a \times v) \gg k) \cdot q$$

(2)

In Kyber, $q = 3329$, $k = 14$, $v = 5$. $a$ and $v$ are 16-bit integers, so the computing of $a \times v$ will produce a 32-bit integer. We propose an improved Barrett reduction as follows:

$$a \mod q = a - (((a \gg 3) \times v) \gg 11) \cdot q$$

(3)
Barrett Reduction

In our implementation, the multiplication \((a >> 3) \ast v\) doesn’t extend the data type to 32-bit, but only uses the 16-bit intermediate to make full use of the bandwidth advantage of the NEON registers, as given in Listing 1.

```
Listing 1 Barrett Reduction (BarR)

Input: va.8h = [a₀, a₁, ... , a₇]
Input: vq.h[0] = q = 3329
Input: vc.8h = 1 << 10
Input: vt1, vt2 is intermediate vector register
Output: va.8h = [a₀, a₁ ... , a₇]

1: sshr vt1.8h, va.8h, 3  ▷ t1 = a >> 3
2: shl vt2.8h, vt1.8h, 2   ▷ t2 = t1 << 2 = t1 \ast 4
3: add vt1.8h, vt1.8h, vt2.8h ▷ t1 = t1 \ast 5
4: add vt1.8h, vt1.8h, vc.8h ▷ t1+ = (1 << 10)
5: sshr vt1.8h, vt1.8h, 11  ▷ t1 = t1 >> 11
6: mls va.8h, vt1.8h, vq.8h ▷ a– = t \ast q
```
Lazy Reduction

In Kyber, the addition of polynomial coefficients when computing INTT (inverse of NTT) is followed by the Barrett reduction. Not all the results need to be reduced, so when the addition do not overflow, Barrett reduction can be removed (Lazy Reduction).

![Diagram showing the position of Barrett reduction in the INTT process.](image)

**Figure 1:** The position of Barrett reduction in the INTT
Montgomery Reduction

- Montgomery multiplication and Montgomery reduction:

<table>
<thead>
<tr>
<th>Listing 2 Montgomery Multiplication (MontM)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> va.8h = [a₀, a₁, ..., a₇]</td>
</tr>
<tr>
<td><strong>Input:</strong> vb.8h = [b₀, b₁, ..., b₇]</td>
</tr>
<tr>
<td><strong>Input:</strong> vt1, vt2 are intermediate vector registers</td>
</tr>
<tr>
<td><strong>Output:</strong> va.8h = [a₀, a₁, ..., a₇]</td>
</tr>
<tr>
<td>1: smull vt1.4s, va.4h, vb.4h ▶ t₁ = (L)a * b</td>
</tr>
<tr>
<td>2: smull2 va.4s, va.8h, vb.8h ▶ va = (H)a * b</td>
</tr>
<tr>
<td>3: MontR vt1.4s, va.4s, vt2 ▶ MontR(a * b)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 3 Montgomery Reduction (MontR)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> va.1.4s = [a₀, a₁, ..., a₃]</td>
</tr>
<tr>
<td><strong>Input:</strong> va.2.4s = [a₄, a₅, ..., a₇]</td>
</tr>
<tr>
<td><strong>Input:</strong> vq = q = 3329</td>
</tr>
<tr>
<td><strong>Input:</strong> vr.4s = [2₁⁶ - 1, ..., 2₁⁶ - 1]</td>
</tr>
<tr>
<td><strong>Input:</strong> vqp = q⁻¹ = 62209</td>
</tr>
<tr>
<td><strong>Input:</strong> vt is intermediate vector register</td>
</tr>
<tr>
<td><strong>Output:</strong> va.1.8h = [a₀, a₁, ..., a₇]</td>
</tr>
<tr>
<td>1: mul vt.4s, va.1.4s, vqp ▶ t = a₁ * q⁻¹</td>
</tr>
<tr>
<td>2: and vt.16b, vt.16b, vt.16b ▶ t = (LSB)t</td>
</tr>
<tr>
<td>3: mls va.1.4s, vt.4s, vq ▶ a₁⁻ = t * q</td>
</tr>
<tr>
<td>4: mul vt.4s, va.2.4s, vqp ▶ t = a₂ * q⁻¹</td>
</tr>
<tr>
<td>5: and vt.16b, vt.16b, vt.16b ▶ t = (LSB)t</td>
</tr>
<tr>
<td>6: mls va.2.4s, vt.4s, vq ▶ a₂⁻ = t * q</td>
</tr>
<tr>
<td>7: uzp2 va.1.8h, va.1.8h, va.2.8h ▶ a₁ = (MSB)(a₁, a₂)</td>
</tr>
</tbody>
</table>
NTT (Number Theoretic Transform)

- NTT is used to speed up polynomial multiplication.

\[ f \ast g = NTT^{-1}(NTT(f) \circ NTT(g)) \]  

where \( \circ \) denotes the coefficient-wise multiplication.

- Basic operation is the butterfly transform.

- There are two types of butterfly units, Cooley-Tukey(CT) and Gentleman-Sande(GS) below.

- \textbf{NTT}: CT butterfly unit; \textbf{INTT}: GS butterfly unit.

\begin{align*}
N &\rightarrow N + \gamma \cdot b \\
N &\rightarrow N - \gamma \cdot b
\end{align*}

\begin{align*}
A &\rightarrow A + B \\
A &\rightarrow (A - B) \cdot \gamma
\end{align*}
In NTT, the polynomial coefficients of each layer need to be loaded and stored.

The NTT using CT butterfly operations for $n = 16$:
For Kyber, \( n = 256 \), the 7-layer incomplete-NTT is available.

The 7-layer incomplete-NTT of Kyber has various layer merging strategies, such as 1+6, 2+5, 3+4 layer merging.

The 5-layer merging is more appropriate because we have enough registers to accommodate all values.

As a result, we adopted the 2+5 layer merging strategy on ARMv8-A to implement NTT/INTT.
1 Short Overview

2 Kyber Scheme

3 Our Implementation

4 Implementation Results

5 Conclusion
Implementation Results

- This table shows the results of optimized modules in Kyber512.
- Compared with pure C implementations, our Barrett and Montgomery reduction shows 8.52 and 8.89 times faster than the reference implementation.
- Our NTT and INTT achieved 11.89 and 13.45 times speedups compared with the reference implementation.

<table>
<thead>
<tr>
<th>Module</th>
<th>ref</th>
<th>Our work</th>
<th>ref / Our work</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarR</td>
<td>2675</td>
<td>314</td>
<td>8.52</td>
</tr>
<tr>
<td>MontR</td>
<td>3413</td>
<td>384</td>
<td>8.89</td>
</tr>
<tr>
<td>NTT</td>
<td>16575</td>
<td>1394</td>
<td>11.89</td>
</tr>
<tr>
<td>INTT</td>
<td>27284</td>
<td>2028</td>
<td>13.45</td>
</tr>
</tbody>
</table>

Table: Performance and Comparison of Kyber512
Implementation Results

- The key encapsulation mechanism (KEM) in Kyber has different implementations of three parameter sets.
- Our optimized software achieved $1.77 \times$, $1.85 \times$, and $2.16 \times$ speedups for key generation, encapsulation, and decapsulation compared with Kyber’s reference implementation.

<table>
<thead>
<tr>
<th>Schemes</th>
<th>ref</th>
<th>Our work</th>
<th>ref / Our work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber512</td>
<td>K</td>
<td>464238</td>
<td>262249</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>637189</td>
<td>343538</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>791471</td>
<td>367236</td>
</tr>
<tr>
<td>Kyber768</td>
<td>K</td>
<td>807544</td>
<td>484745</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>1030702</td>
<td>594449</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>1274856</td>
<td>641491</td>
</tr>
<tr>
<td>Kyber1024</td>
<td>K</td>
<td>1189371</td>
<td>783209</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>1491847</td>
<td>930112</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>1727240</td>
<td>1011992</td>
</tr>
</tbody>
</table>

Table II: Performance and Comparison of KEM
1 Short Overview

2 Kyber Scheme

3 Our Implementation

4 Implementation Results

5 Conclusion
Efficient Implementation of Kyber on Mobile Devices

- The optimized Barrett reduction increases the utilization of vector registers.
- Montgomery multiplication and reduction greatly improves the efficiency.
- The layer merging technique substantially accelerates the efficiency in NTT.

Our optimizations of modular reduction and NTT operations are useful for other works, such as NewHope.
Thanks for listening