Optimized Software Implementations of CRYSTALS-Kyber, NTRU, and Saber Using NEON-Based Special Instructions of ARMv8

Cryptographic Engineering Research Group @ George Mason University
Duc Tri Nguyen and Kris Gaj
- NEON is an alternative name for Advanced Single Instruction Multiple Data (ASIMD) extension to the ARM Instruction Set Architecture, mandatory since ARMv7-A.

- NEON provides 32x128-bit vector registers. Compared with Single Instruction Single Data (SISD), ASIMD can have ideal speed-up in the range 2..16 (for 64..8-bit operands).
- Most software implementations of PQC candidates on:
  - Intel/AMD (w/ AVX2 extension)
  - Cortex-M4 (w/ DSP extension)\(^1\)
- Lack of NEON implementations on ARMv7 and ARMv8 architectures

\(^1\) M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, pqm4 - Post-quantum crypto library for the ARM Cortex-M4, https://github.com/mupq/pqm4
- Our goal is to fill the gap between low-power embedded processors and high-performance x86-64 platforms.

- We developed constant-time, optimized ARMv8 implementations of 3 KEM finalists:
  - CRYSTALS-Kyber
  - NTRU
  - Saber
Polynomial Multiplication

Typically:

k=2: Karatsuba : $O(n^{1.58})$

k=3: Toom-3 : $O(n^{1.46})$

k=4: Toom-4 : $O(n^{1.40})$
Optimal Choice of Algorithms

Based on the analysis of algorithms, their parameters, and AVX2 implementations for the 3 lattice-based KEMs finalists.
5 Steps of Toom-4

1. Splitting

\[
A(x) = x^{\frac{3n}{4}} \sum_{i=\frac{3n}{4}}^{n-1} a_i x^{i-\frac{3n}{4}} + \cdots + x^{\frac{n}{4}} \sum_{i=\frac{n}{4}}^{\frac{2n}{4}-1} a_i x^{i-\frac{n}{4}} + \sum_{i=0}^{\frac{n}{4}-1} a_i x^i
\]

\[
= \alpha_3 \cdot x^{\frac{3n}{4}} + \alpha_2 \cdot x^{\frac{2n}{4}} + \alpha_1 \cdot x^{\frac{n}{4}} + \alpha_0
\]

2. Evaluation

\[
\begin{bmatrix}
A(0) \\
A(1) \\
A(-1) \\
A(\frac{1}{2}) \\
A(-\frac{1}{2}) \\
A(2) \\
A(\infty)
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 \\
-1 & 1 & -1 & 1 \\
\frac{1}{8} & \frac{1}{4} & \frac{1}{2} & 1 \\
-\frac{1}{8} & \frac{1}{4} & -\frac{1}{2} & 1 \\
8 & 4 & 2 & 1 \\
1 & 0 & 0 & 0
\end{bmatrix} \cdot \begin{bmatrix}
\alpha_3 \\
\alpha_2 \\
\alpha_1 \\
\alpha_0
\end{bmatrix}
\]

3. Pointwise multiplication

\[
\begin{bmatrix}
C(0) \\
C(1) \\
C(-1) \\
C(\frac{1}{2}) \\
C(-\frac{1}{2}) \\
C(2) \\
C(\infty)
\end{bmatrix} = \begin{bmatrix}
A(0) \\
A(1) \\
A(-1) \\
A(\frac{1}{2}) \\
A(-\frac{1}{2}) \\
A(2) \\
A(\infty)
\end{bmatrix} \cdot \begin{bmatrix}
B(0) \\
B(1) \\
B(-1) \\
B(\frac{1}{2}) \\
B(-\frac{1}{2}) \\
B(2) \\
B(\infty)
\end{bmatrix}
\]


4. Interpolation

\[
\begin{bmatrix}
\theta_0 \\
\theta_1 \\
\theta_2 \\
\theta_3 \\
\theta_4 \\
\theta_5 \\
\theta_6
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 \\
\frac{1}{64} & \frac{1}{32} & \frac{1}{16} & \frac{1}{8} & \frac{1}{4} & \frac{1}{2} & 1 \\
\frac{1}{64} & -\frac{1}{32} & \frac{1}{16} & -\frac{1}{8} & \frac{1}{4} & -\frac{1}{2} & 1 \\
64 & 32 & 16 & 8 & 4 & 2 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}^{-1} \begin{bmatrix}
C(0) \\
C(1) \\
C(-1) \\
C(\frac{1}{2}) \\
C(-\frac{1}{2}) \\
C(2) \\
C(\infty)
\end{bmatrix}
\]

where

\[
C(x) = \sum_{i=0}^{6} \theta_i x^i
\]

5. Merging

\[
C(x) = \sum_{i=0}^{2n-1} a_i x^i = \sum_{i=0}^{6} \theta_i x^{i n / 4}
\]
Toom-Cook: Splitting & Evaluation

- **Karatsuba**
  - 1 polynomial
  - N coefficients
  - 3 polynomials
  - N/2 coefficients each
  - \( 3 \cdot \frac{N}{2} \)

- **Toom-3**
  - 1 polynomial
  - N coefficients
  - 5 polynomials
  - N/3 coefficients each
  - \( 5 \cdot \frac{N}{3} \)

- **Toom-4**
  - 1 polynomial
  - N coefficients
  - 7 polynomials
  - N/4 coefficients each
  - \( 7 \cdot \frac{N}{4} \)
Toom-Cook Implementation: Saber

Schoolbook 16 x 16: Pointwise Multiplication

Figure 1: The Toom-Cook implementation strategy for Saber and NTRU-HPS821

Splitting & Evaluation

Interpolation & Merging

Repeated for A(x) & B(x)

C(x)
Toom-Cook Implementation: NTRU-HPS821

Splitting & Evaluation

Repeated for A(x) & B(x)

Interpolation & Merging

Schoolbook 16 x 16: Pointwise Multiplication
For multiple layers of Split-Evaluate/Interpolate-Merge:
• unroll these layers to save load/store instructions
In order to perform a **batch multiplication**, a matrix of $k$ polynomials with $k$ coefficients has to be transposed before and after the multiplication.

**Optimal value of $k$** was determined to be 16.
The transpose operation enables performing the same operation on the same coefficients of 8 polynomials in parallel.

The $8 \times 8$ matrix transpose requires 27 out of 32 NEON 128-bit registers.
The 16x16 Matrix Transpose Operation

- 16 x 16 matrix transpose requires memory
- To transpose 16 x 16 efficiently, transpose only 8 x 8 matrices and remember the location of each 8 x 8 block
Complete NTT

\[ C(x) = A(x) \times B(x) = \mathcal{NTT}^{-1}(C) = \mathcal{NTT}^{-1}(A \times B) = \mathcal{NTT}^{-1}(\mathcal{NTT}(A) \times \mathcal{NTT}(B)) \]

where \( A(x), B(x), C(x) \in \mathbb{Z}_q [x]/(x^n + 1) \) and \( q \equiv 1 \mod 2n \)
Number Theoretic Transform

Example of levels

Example of reordering indices between levels
NTT Implementation: CRYSTALS-Kyber and Saber

- Utilize Load and Interleave instructions for Level 0-1
- Use transpose instructions for Level 2-3
- Twist store registers in Level 4
NTT Implementation: CRYSTALS-Kyber and Saber
- Apply to NTT/FFT based submissions
- 16-bit coefficients can reach level 6
- 32-bit coefficients can reach level 5.
NTT Implementation: Multiply and Modular Reduction

**Algorithm 2**: Vectorized multiplication modulo a 16-bit $q$

**Input**: $B = (B_L, B_H), C = (C_L, C_H), R = 2^{16}$

**Output**: $A = B \cdot (CR) \mod q$

1. $T_0 \leftarrow \text{smull.s16}(B_L, C_L)$
2. $T_1 \leftarrow \text{smull.s16}(B_H, C_H)$
3. $T_2 \leftarrow \text{uzp1.s16}(T_0, T_1)$
4. $T_3 \leftarrow \text{uzp2.s16}(T_0, T_1)$
5. $(A_L, A_H) \leftarrow \text{mul.s16}(T_2, q^{-1})$
6. $T_1 \leftarrow \text{smull.s16}(A_L, q)$
7. $T_2 \leftarrow \text{smull.s16}(A_H, q)$
8. $T_0 \leftarrow \text{uzp2.s16}(T_1, T_2)$
9. $A \leftarrow T_3 - T_0$

**Algorithm 15**: Multiplication modulo 16-bit $q$

**Require**: $-2^{15} \leq a < 2^{15}, \quad \frac{q-1}{2} \leq b \leq \frac{q-1}{2}, \quad b' = bq^{-1} \mod 2^{16}$

**Ensure**: $r = 2^{16}ab \mod q$

1. $t_1 \leftarrow \left\lfloor \frac{ab}{2^{16}} \right\rfloor$
2. $t_0 \leftarrow ab' \mod 2^{16}$
3. $t_0 \leftarrow \left\lfloor \frac{t_0}{2^{16}} \right\rfloor$
4. $r \leftarrow (t_1 - t_0) \mod 2^{16}$

- NEON dependency chain: `vuzp` and `vmull` (vector unzip and vector multiply)
- Lack of an instruction similar to AVX2 `vpmulhw`:
  Multiply Packed Unsigned Word Integers and Store the high 16-bits of Result
- Compared to AVX2, our implementation uses additionally
  2 MUL and 3 UNZIP instructions
## Benchmarking Methodology

<table>
<thead>
<tr>
<th>Component</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apple M1 System on Chip</strong></td>
<td>Firestorm core, 3.2 GHz&lt;sup&gt;1&lt;/sup&gt;, MacBook Air</td>
</tr>
<tr>
<td><strong>Broadcom BCM2711 System on Chip</strong></td>
<td><strong>Cortex-A72</strong> core, 1.5 GHz, Raspberry Pi 4</td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td>MacOS 11.4, Arch Linux (March 25, 2021)</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td>clang 12.0 (MacBook Air), clang 11.1 (Raspberry Pi 4)</td>
</tr>
<tr>
<td><strong>Compiler Options</strong></td>
<td>-O3 -mtune=native -fomit-frame-pointer</td>
</tr>
<tr>
<td><strong>Cycles count on Cortex-A72</strong></td>
<td>PAPI&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>Cycles count on Apple M1</strong></td>
<td>Modified&lt;sup&gt;3&lt;/sup&gt; from Dougall Johnson’s work&lt;sup&gt;4&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>Iterations</strong></td>
<td>10,000,000 on Apple M1 to force CPU to run on high-performance “FireStorm” core; 1,000,000 otherwise</td>
</tr>
</tbody>
</table>

---

1. [https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested](https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested)
3. [https://github.com/GMUCERG/PQC_NEON/blob/main/neon/kyber/m1cycles.c](https://github.com/GMUCERG/PQC_NEON/blob/main/neon/kyber/m1cycles.c)
4. [https://github.com/dougallj](https://github.com/dougallj)
Toom-Cook vs. NTT for Saber

Dependencies degrade performance of NTT on high-performance processors.

On Apple M1, Toom-Cook better by 13-21%
On Cortex-A72 a tie.

We select Toom-Cook for the implementation of Saber.
### Encapsulation and Decapsulation ranking of baseline C implementations:

1. **Saber**
2. **CRYSTALS-Kyber**
3. **NTRU (Levels 1 & 3 only)**

Consistent between Cortex-A72 and Apple M1.
Decapsulation ranking of NEON implementations at Levels 1, 3 and 5

Encapsulation ranking of NEON implementations at Level 3 and 5:

1. CRYSTALS-Kyber
2. Saber
3. NTRU (Levels 1 & 3 only)

Consistent between Cortex-A72 and Apple M1.

<table>
<thead>
<tr>
<th>Rank</th>
<th>E</th>
<th>$\text{k}_c$</th>
<th>$\uparrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ntru-hrss701</td>
<td>93.6</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>kyber512</td>
<td>95.3</td>
<td>1.02</td>
</tr>
<tr>
<td>3</td>
<td>lightsaber</td>
<td>130.1</td>
<td>1.39</td>
</tr>
<tr>
<td>4</td>
<td>ntru-hps877</td>
<td>181.7</td>
<td>1.94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>D</th>
<th>$\text{k}_c$</th>
<th>$\uparrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>kyber768</td>
<td>151.0</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>saber</td>
<td>213.6</td>
<td>1.41</td>
</tr>
<tr>
<td>3</td>
<td>ntru-hps821</td>
<td>232.6</td>
<td>1.54</td>
</tr>
<tr>
<td>1</td>
<td>ntru-hrss701</td>
<td>220.7</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>firesaber</td>
<td>321.6</td>
<td>1.44</td>
</tr>
</tbody>
</table>

Exception: Encapsulation at Level 1

1. NTRU
2. CRYSTALS-Kyber
3. Saber
Why do the rankings of Saber and CRYSTALS-Kyber switch places between the baseline C and NEON implementations?

Answer: Performance of polynomial multiplication in vector by vector and matrix by vector multiplications
## Saber vs. Kyber

<table>
<thead>
<tr>
<th></th>
<th>Cortex-A72</th>
<th></th>
<th>Level 1 (kilocycles)</th>
<th></th>
<th>Level 3 (kilocycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1500 MHz</td>
<td></td>
<td>ref</td>
<td>neon</td>
<td>ref/neon</td>
</tr>
<tr>
<td>InnerProd</td>
<td></td>
<td>Saber: Toom-Cook</td>
<td>27.7</td>
<td>18.1</td>
<td>1.53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NTT</td>
<td>41.4</td>
<td>25.0</td>
<td>31.5</td>
</tr>
<tr>
<td>MatrixVectorMul</td>
<td></td>
<td></td>
<td>55.2</td>
<td>40.2</td>
<td>37.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Saber: Toom-Cook</td>
<td>125.7</td>
<td>81.0</td>
<td>71.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NTT</td>
<td>255.7</td>
<td>161.0</td>
<td>111.3</td>
</tr>
<tr>
<td>Kyber</td>
<td></td>
<td>VectorVectorMul</td>
<td>44.4</td>
<td>7.1</td>
<td>6.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>59.7</td>
<td>9.9</td>
<td>6.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MatrixVectorMul</td>
<td>68.1</td>
<td>10.7</td>
<td>6.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>117.5</td>
<td>19.3</td>
<td>6.1</td>
</tr>
</tbody>
</table>
## NEON vs. Baseline Speed-Up

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ref ((kc))</th>
<th>neon ((kc))</th>
<th>ref/neon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>lightsaber</td>
<td>50.9</td>
<td>54.9</td>
<td>37.2</td>
</tr>
<tr>
<td>kyber512</td>
<td>75.7</td>
<td>89.5</td>
<td>32.6</td>
</tr>
<tr>
<td>ntru-hps677</td>
<td>183.1</td>
<td>430.4</td>
<td>60.1</td>
</tr>
<tr>
<td>ntru-hrss701</td>
<td>152.4</td>
<td>439.9</td>
<td>22.8</td>
</tr>
<tr>
<td>saber</td>
<td>90.4</td>
<td>96.2</td>
<td>59.9</td>
</tr>
<tr>
<td>kyber768</td>
<td>119.8</td>
<td>137.8</td>
<td>49.2</td>
</tr>
<tr>
<td>ntru-hps821</td>
<td>245.3</td>
<td>586.5</td>
<td>75.7</td>
</tr>
<tr>
<td>firesaber</td>
<td>140.9</td>
<td>150.8</td>
<td>87.9</td>
</tr>
<tr>
<td>kyber1024</td>
<td>175.4</td>
<td>198.4</td>
<td>71.6</td>
</tr>
</tbody>
</table>
For **Decapsulation**, the rankings across all security levels are:
1. CRYSTALS-Kyber, 2. Saber, 3. NTRU (Levels 1 & 3 only)

For **Encapsulation**, at levels 1 and 3, the rankings are:
1. NTRU, 2. CRYSTALS-Kyber, 3. Saber

For **Encapsulation**, at level 5, the ranking is:
1. CRYSTALS-Kyber, 2. Saber
### NEON vs. AVX2 in Cycles

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>neon (kc)</th>
<th></th>
<th>AVX2 (kc)</th>
<th></th>
<th>AVX2/neon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>lightsaber</td>
<td>37.2</td>
<td>35.3</td>
<td>41.9</td>
<td>42.2</td>
<td>1.13</td>
</tr>
<tr>
<td>kyber512</td>
<td>32.6</td>
<td>29.4</td>
<td>28.4</td>
<td>22.6</td>
<td>0.87</td>
</tr>
<tr>
<td>ntru-hps677</td>
<td>60.1</td>
<td>54.6</td>
<td>26.0</td>
<td>45.7</td>
<td>0.43</td>
</tr>
<tr>
<td>ntru-hrss701</td>
<td>22.8</td>
<td>60.8</td>
<td>20.4</td>
<td>47.7</td>
<td>0.90</td>
</tr>
<tr>
<td>saber</td>
<td>59.9</td>
<td>58.0</td>
<td>70.9</td>
<td>70.7</td>
<td>1.18</td>
</tr>
<tr>
<td>kyber768</td>
<td>49.2</td>
<td>45.7</td>
<td>43.4</td>
<td>35.2</td>
<td>0.88</td>
</tr>
<tr>
<td>ntru-hps821</td>
<td>75.7</td>
<td>69.0</td>
<td>29.9</td>
<td>57.3</td>
<td>0.39</td>
</tr>
<tr>
<td>firesaber</td>
<td>87.9</td>
<td>86.7</td>
<td>103.3</td>
<td>103.7</td>
<td>1.18</td>
</tr>
<tr>
<td>kyber1024</td>
<td>71.6</td>
<td>67.1</td>
<td>63.0</td>
<td>53.1</td>
<td>0.88</td>
</tr>
</tbody>
</table>

Result for AVX2 AMD EPYC 7742 taken from supercop-20210125
Frequency Scaling Effect
Apple M1 @ 3.2 GHz versus Intel Core i7-8750H 4.1 GHz

<table>
<thead>
<tr>
<th>Apple M1 Core i7-8750H</th>
<th>ref (kc) E</th>
<th>D</th>
<th>neon (kc) E</th>
<th>D</th>
<th>AVX2 (kc) E</th>
<th>D</th>
<th>ref/neon E</th>
<th>D</th>
<th>AVX2/neon E</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTRU-HPS677</td>
<td>183.1</td>
<td>430.4</td>
<td>60.1</td>
<td>54.6</td>
<td>47.6</td>
<td>32.5</td>
<td>3.05</td>
<td>7.89</td>
<td>0.79</td>
<td>0.60</td>
</tr>
<tr>
<td>NTRU-HRSS701</td>
<td>152.4</td>
<td>439.9</td>
<td>22.8</td>
<td>60.8</td>
<td>28.8</td>
<td>33.9</td>
<td>6.68</td>
<td>7.24</td>
<td>1.26</td>
<td>0.56</td>
</tr>
<tr>
<td>LIGHTSABER</td>
<td>50.9</td>
<td>54.9</td>
<td>37.2</td>
<td>35.3</td>
<td>35.1</td>
<td>32.3</td>
<td>1.37</td>
<td>1.55</td>
<td>0.94</td>
<td>0.91</td>
</tr>
<tr>
<td>KYBER512</td>
<td>75.7</td>
<td>89.5</td>
<td>32.6</td>
<td>29.4</td>
<td>23.2</td>
<td>17.5</td>
<td>2.33</td>
<td>3.04</td>
<td>0.71</td>
<td>0.59</td>
</tr>
<tr>
<td>NTRU-HPS821</td>
<td>245.3</td>
<td>586.5</td>
<td>75.7</td>
<td>69.0</td>
<td>56.1</td>
<td>40.7</td>
<td>3.24</td>
<td>8.49</td>
<td>0.74</td>
<td>0.59</td>
</tr>
<tr>
<td>SABER</td>
<td>90.4</td>
<td>96.2</td>
<td>59.9</td>
<td>58.0</td>
<td>54.3</td>
<td>53.8</td>
<td>1.51</td>
<td>1.66</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>KYBER768</td>
<td>119.8</td>
<td>137.8</td>
<td>49.2</td>
<td>45.7</td>
<td>33.9</td>
<td>26.0</td>
<td>2.43</td>
<td>3.02</td>
<td>0.69</td>
<td>0.57</td>
</tr>
<tr>
<td>FIRESABER</td>
<td>140.9</td>
<td>150.8</td>
<td>87.9</td>
<td>86.7</td>
<td>78.9</td>
<td>78.1</td>
<td>1.60</td>
<td>1.74</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>KYBER1024</td>
<td>175.4</td>
<td>198.4</td>
<td>71.6</td>
<td>67.1</td>
<td>45.2</td>
<td>35.5</td>
<td>2.45</td>
<td>2.96</td>
<td>0.63</td>
<td>0.53</td>
</tr>
</tbody>
</table>
Frequency Scaling Effect
Apple M1 @ 3.2 GHz versus Intel Core i7-8750H 4.1 GHz

Saber: AVX2 vs. NEON

Level 1  Level 3  Level 5

AVX2-Saber-Encap: 15.1us 25.5us 37.1us
NEON-Saber-Encap: 11.6us 18.9us 28.0us
AVX2-Saber-Decap: 14.3us 24.2us 36.1us
NEON-Saber-Decap: 11.2us 18.4us 27.6us

Time measured with the ns accuracy using clock_gettime() on a MacBook Air and a PC laptop
Conclusions: Toom-Cook and NTT

- The polynomial multiplication performance affects the C baseline and NEON rankings in case of Saber and Kyber.

- Proposed optimal Toom-Cook strategy tailored for NTRU and Saber parameters.

- Missing instruction equivalent to AVX2 vpmulhw causes dependencies and worse performance
Conclusions

- First optimized implementation of CRYSTALS-Kyber, NTRU, and Saber targeting ARMv8.

- Largest speed-up for NTRU, followed by CRYSTALS-Kyber, and Saber

- The rankings of lattice-based PQC KEM finalists in terms of speed in software are similar for the NEON implementations and AVX2 implementations
  
  Decapsulation: 1. CRYSTALS-Kyber, 2. Saber, 3. NTRU (L1 & 3 only)
  Encapsulation: 1. NTRU (L1 & 3 only), 2. CRYSTALS-Kyber, 3. Saber
Thanks for your attention!

Our source code is available at:
https://github.com/GMUCERG/PQC_NEON