pqm4: NISTPQC Round 3 Results on the Cortex-M4

Matthias J. Kannwischer¹ and Richard Petri²
matthias@kannwischer.eu, rp@rpls.de

07 June, 2021, 3rd NIST PQC Standardization Conference

¹Max Planck Institute for Security and Privacy, Bochum, Germany → Academia Sinica, Taipei, Taiwan
²Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany
• [https://github.com/mupq/pqm4](https://github.com/mupq/pqm4)
• **pqm4** is a benchmarking and testing framework for NISTPQC candidates
  • Targeting the Cortex-M4 (with all options)
  • Our reference platform: STM32F407 (M4F, 128 KiB contiguous RAM, 1 MiB of flash)
    • You can get a discovery board for < $20
• **pqm4** is also a collection of the fastest (open source) implementations
  • If you have something faster, please open a pull request
• **pqm4** was presented at the 2nd NIST PQC standardization conference
  • Slides are available at [https://kannwischereu/talks/20190824_nistpqc.pdf](https://kannwischereu/talks/20190824_nistpqc.pdf)
  • This talk will focus on the changes since then
• Changed default AES implementation from t-table to bitsliced
  • Some Cortex-M4 platforms may have a cache → cache attacks possible
  • New bitsliced implementation by Adomnicai and Peyrin ([ia.cr/2020/1123](https://ia.cr/2020/1123))
  • Slows down HQC, NTRUPrime, Kyber-90s
  • Implementations can still use faster t-table implementation for \_publicinputs()
  • No change for FrodoKEM
• Multi-platform support
  • Improved and unified build system; fully based on make now; modular
  • Also supporting NUCLEO-L476R (STM32L476RG), ChipWhisperer LITEARM (STM32F303), QEMU simulator (ARM AN386)
    • Adding more hardware support should be easy
• Round 3 parameter changes for many of the candidates
• Various new record-breaking implementations merged
### Implementation updates: KEMs

<table>
<thead>
<tr>
<th></th>
<th>pqm4</th>
<th>M4 implementation</th>
<th>new since Aug 2019</th>
<th>implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic McEliece</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2021/492</td>
</tr>
<tr>
<td>KYBER</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2020/012</td>
</tr>
<tr>
<td>NTRU</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2020/1397</td>
</tr>
<tr>
<td>SABER</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2020/1397</td>
</tr>
<tr>
<td>BIKE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2021/493</td>
</tr>
<tr>
<td>FrodoKEM</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>ia.cr/2018/1116</td>
</tr>
<tr>
<td>HQC</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>ref</td>
</tr>
<tr>
<td>NTRU Prime</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2020/1216</td>
</tr>
<tr>
<td>SIKE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2021/115</td>
</tr>
</tbody>
</table>

✓: outdated implementation in pqm4
• Keys do not fit in RAM of our target (except the simulated platform)
• New implementation by Chen and Chou ([ia.cr/2021/492])
  • Write keys to flash instead
• Not suitable for pqm4
• Keys do not fit in RAM of our target (except the simulated platform)
• New implementation by Chen and Chou ([ia.cr/2021/492](https://ia.cr/2021/492))
  • Write keys to flash instead
• Not suitable for pqm4
- Jan 2020: Faster implementation by Alkim, Bilgin, Cenk, and Gérard (ia.cr/2020/012)
- Jul 2020: New round 3 parameter sets
CRYSTALS-KYBER (Round 2 vs. Round 3 parameters)
• Nov 2020: New implementation using NTTs by Chung, Hwang, Kannwischer, Seiler, Shih, Yang (ia.cr/2020/1397)
• Nov 2020: New implementation using NTTs by Chung, Hwang, Kannwischer, Seiler, Shih, Yang (ia.cr/2020/1397)
• New implementation by Chen and Chou (ia.cr/2021/493)
• No parameter and implementation changes
• Reference implementation from PQClean now working on the M4
• Jan 2019: Optimized implementation by Huang, Chen, and Yang (ia.cr/2019/100)
• Oct 2020: Faster implementation using NTTs by Alkim, Cheng, Chung, Evkan, Huang, Hwang, Li, Niederhagen, Shih, Wälde, and Yang (ia.cr/2020/1216)
• Apr 2020: M4 implementation by Seo, Anastasova, Jalali, and Azarderakhsh (ia.cr/2020/410)
• Jan 2021: Faster implementation by Anastasova, Azarderakhsh, and Mozaffari Kermani (ia.cr/2021/115)
• Only former has code available and integrated in pqm4
• Let’s compare some of the results
  • NIST security level 1
    • For NTRUPrime: M4 implementations are only available on level 3
  • Only CCA variants for KEMs
  • SHAKE/SHA-3 parameter sets in case there are multiple ones

• For the full, up-to-date, and unbiased results go to https://github.com/mupq/pqm4
Summary: KEMs in Round 3

The chart shows the performance of various KEMs in Round 3, measured in k cycles. The y-axis represents the number of k cycles, with a scale from 0 to 1e6. The x-axis lists the KEMs tested, including 'biket', 'frdokem40shake', 'hqcrns128', 'kyber512', 'lightning', 'mceliece345864f', 'ntrupps2048509', 'ntrups701', 'ntrulp761', 'sikep34', and 'suplep761'. The chart uses different colors to represent KeyGen, Encaps, and Decaps operations.
Summary: KEMs in Round 3

The diagram shows the comparison of Encaps and Decaps for different KEMs in Round 3. The x-axis represents the KEMs tested, and the y-axis represents the number of k cycles. One can observe the performance of each KEM in terms of encryption (Encaps) and decryption (Decaps) efficiency.
Summary: KEMs in Round 3 (Encapsulation)
Summary: KEMs in Round 3 (Encapsulation)
Summary: KEMs in Round 3 (Decapsulation)

[Bar chart showing performance of different KEMs during decapsulation process.]
Summary: KEMs in Round 3 (Decapsulation)
<table>
<thead>
<tr>
<th>Implementation</th>
<th>pqm4</th>
<th>M4 implementation</th>
<th>new since Aug 2019</th>
<th>implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DILITHIUM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2020/1278</td>
</tr>
<tr>
<td>Falcon</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>ia.cr/2019/893</td>
</tr>
<tr>
<td>Rainbow</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>ia.cr/2021/532</td>
</tr>
<tr>
<td>GeMSS</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Picnic</td>
<td>soon</td>
<td>✓</td>
<td>✓</td>
<td>unpublished ref</td>
</tr>
<tr>
<td>SPHINCS+</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
• New implementation by Greconici, Kannwischer, and Sprenkels (ia.cr/2020/1278)
• Significant parameter changes in Round 3
DILITHIUM (Round 2 vs. Round 3 parameters)
• No changes in Falcon
• Fastest implementation is still by Pornin (ia.cr/2019/893)
• Keys too large for our target platform
• New implementation by Chou, Kannwischer, and Yang targeting the EFM32GG11B (ia.cr/2021/532)
• Keys too large for our target platform
• New implementation by Chou, Kannwischer, and Yang targeting the EFM32GG11B (ia.cr/2021/532)
• Keys too large for our target platform
• No implementations available for other M4 platforms
• New M4 implementation by Aranha, Kales, Ramacher, and Zaverucha
• Not published yet, but soon to be merged into pqm4
SPHINCS+ (Round 2 vs. Round 3)
SPHINCS+ Verification (Round 2 vs. Round 3)
Summary: Signatures in Round 3

![Bar chart showing the performance of various signature schemes in Round 3. The x-axis represents different signature schemes, and the y-axis represents the number of cycles (k cycles). The schemes are color-coded: KeyGen (blue), Sign (orange), and Verify (green). The chart highlights the performance of the schemes with the 'sphincs-shake256-128s-simple' scheme requiring significantly more cycles compared to others.]
Summary: Signatures in Round 3

![Bar Chart]

- **Signatures in Round 3**
  - Signatures for different systems are represented.
  - The chart compares the processing time (k cycles) for signing and verifying operations.

- **Systems**:
  - Dilithium2
  - Falcon512
  - Picnic31
  - Picnic1fs
  - Picnic1full
  - RainbowClassic

- **Key Observations**:
  - Picnic31 shows the highest processing time for both signing and verifying operations.
  - Dilithium2 and Falcon512 have lower processing times, especially for signing.
  - Picnic1full and RainbowClassic have moderate processing times.

- **Legend**:
  - Orange: Sign
  - Green: Verify

- **Axes**:
  - **Y-axis**: k cycles
  - **X-axis**: Systems
Summary: Signatures in Round 3 (Sign)
Summary: Signatures in Round 3 (Sign)
Summary: Signatures in Round 3 (Sign)
Summary: Signatures in Round 3 (Verify)
Summary: Signatures in Round 3 (Verify)

- rainbowlassic
- falcon512
- dilithium2
Conclusion

- Significant progress in embedded PQC implementations in the last 2 years
- Still optimization work left; some candidates have received very little attention
- Memory consumption often considered now; code size usually ignored

- Fastest KEM (Encaps): **NTRU-HRSS-701**
- Fastest KEM (Decaps): **Lightsaber**
- Fastest KEM (Encaps+Decaps): **Lightsaber**
- Fastest Signature: **Rainbow**; followed by Dilithium (signing) or Falcon (verification)
• Implementations for the M4 can only get faster (and smaller)
• Cortex-M3
  • More limited instruction set
  • Long multiplications have data-dependent cycle count
    → compiling reference code often leads to vulnerable implementations
    • [https://github.com/mupq/pqm3](https://github.com/mupq/pqm3)
• Countermeasures against side-channel attacks and fault attacks
  • Only very few PQC schemes have (good) masked implementations
  • Ideally: Collection of masked implementations; unified framework for performing leakage evaluation and t-tests (new build system supports ChipWhisperer!)
• Vectorized ARM implementations
• RISC-V
Thank you very much for your attention

matthias@kannwischer.eu

https://github.com/mupq/pqm4