pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4

Matthias Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen
Radboud University, Nijmegen, The Netherlands
matthias@kannwischer.eu

Aug 24, 2019, 2nd PQC Standardization Conference, Santa Barbara
“Performance will play a larger role in the second round”
“Performance will play a larger role in the second round”

**Round 1:** Focus on Intel/AVX2 implementations
“Performance will play a larger role in the second round”

**Round 1:** Focus on Intel/AVX2 implementations

**But:** Majority of cryptographic devices is way smaller
“Performance will play a larger role in the second round”

**Round 1:** Focus on Intel/AVX2 implementations

**But:** Majority of cryptographic devices is way smaller
  - Limited RAM
Motivation

- “Performance will play a larger role in the second round”
- **Round 1:** Focus on Intel/AVX2 implementations
- **But:** Majority of cryptographic devices is way smaller
  - Limited RAM
  - No/limited vector instructions
“Performance will play a larger role in the second round”

**Round 1**: Focus on Intel/AVX2 implementations

**But**: Majority of cryptographic devices is way smaller

- Limited RAM
- No/limited vector instructions
- Side-channels?
“Performance will play a larger role in the second round”

**Round 1:** Focus on Intel/AVX2 implementations

**But:** Majority of cryptographic devices is way smaller
  - Limited RAM
  - No/limited vector instructions
  - Side-channels?

**Challenges**
“Performance will play a larger role in the second round”

**Round 1:** Focus on Intel/AVX2 implementations

**But:** Majority of cryptographic devices is way smaller
- Limited RAM
- No/limited vector instructions
- Side-channels?

**Challenges**
- Do schemes even fit in limited RAM + flash?
“Performance will play a larger role in the second round”

**Round 1:** Focus on Intel/AVX2 implementations

**But:** Majority of cryptographic devices is way smaller
  - Limited RAM
  - No/limited vector instructions
  - Side-channels?

**Challenges**
  - Do schemes even fit in limited RAM + flash?
  - Are schemes efficient on small ARMs?
“Performance will play a larger role in the second round”

**Round 1:** Focus on Intel/AVX2 implementations

**But:** Majority of cryptographic devices is way smaller
  - Limited RAM
  - No/limited vector instructions
  - Side-channels?

**Challenges**
  - Do schemes even fit in limited RAM + flash?
  - Are schemes efficient on small ARMs?
  - What is the overhead of masking?
“It’s big and it’s slow”
“It’s big and it’s slow”
– everyone, always
Post-quantum on small devices

“It’s big and it’s slow”
  – everyone, always

- STM32F4DISCOVERY
  - ARM Cortex-M4
  - 32-bit, ARMv7E-M
  - 192 KiB RAM, 168 MHz

- PQM4: test and optimize on the Cortex-M4
  - github.com/mupq/pqm4
They are cheap (< $30)
Rationale for using STM32F4DISCOVERY boards

- They are cheap (< $30)
- They are huge in terms of RAM and flash
Rationale for using STM32F4DISCOVERY boards

- They are cheap (< $30)
- They are huge in terms of RAM and flash
  - Great for PQC – many schemes fit
Rationale for using STM32F4DISCOVERY boards

- They are cheap (< $30)
- They are huge in terms of RAM and flash
  - Great for PQC – many schemes fit
  - Unfortunately, pqRSA did not terminate within round 1
Rationale for using STM32F4DISCOVERY boards

- They are cheap (< $30)
- They are huge in terms of RAM and flash
  - Great for PQC – many schemes fit
  - Unfortunately, pqRSA did not terminate within round 1
- ARMv7E-M more interesting for assembly optimizations
Rationale for using STM32F4DISCOVERY boards

- They are cheap (< $30)
- They are huge in terms of RAM and flash
  - Great for PQC – many schemes fit
  - Unfortunately, pqRSA did not terminate within round 1
- ARMv7E-M more interesting for assembly optimizations
- NIST recommended Cortex-M4 for PQC evaluation
Rationale for using STM32F4DISCOVERY boards

- They are cheap (< $30)
- They are huge in terms of RAM and flash
  - Great for PQC – many schemes fit
  - Unfortunately, pqRSA did not terminate within round 1
- ARMv7E-M more interesting for assembly optimizations
- NIST recommended Cortex-M4 for PQC evaluation
- We’re using it for teaching
Rationale for using STM32F4DISCOVERY boards

- They are cheap (< $30)
- They are huge in terms of RAM and flash
  - Great for PQC – many schemes fit
  - Unfortunately, pqRSA did not terminate within round 1
- ARMv7E-M more interesting for assembly optimizations
- NIST recommended Cortex-M4 for PQC evaluation
- We’re using it for teaching
  - We have dozens of them lying around
Rationale for using STM32F4DISCOVERY boards

- They are cheap (< $30)
- They are huge in terms of RAM and flash
  - Great for PQC – many schemes fit
  - Unfortunately, pqRSA did not terminate within round 1
- ARMv7E-M more interesting for assembly optimizations
- NIST recommended Cortex-M4 for PQC evaluation
- We’re using it for teaching
  - We have dozens of them lying around
  - Our students know how to work with them
Goals

1. Automate testing and benchmarking
2. Include as many schemes as possible

4 types of implementations

- **ref**: Reference C implementations from submission packages
- **clean**: Slightly modified reference implementations to satisfy basic code quality requirements
- **opt**: Optimized portable C implementations
- **m4**: Optimized using ARMv7E-M assembly

---

1. See [https://github.com/PQClean/PQClean](https://github.com/PQClean/PQClean)
Goals

- Framework that eases optimization for this platform

---

1see https://github.com/PQClean/PQClean
Goals

- Framework that eases optimization for this platform
- Automate testing and benchmarking

\(^1\text{see } \url{https://github.com/PQClean/PQClean}\)
Goals

- Framework that eases optimization for this platform
- Automate testing and benchmarking
- Include as many schemes as possible

\[\text{\textsuperscript{1}see https://github.com/PQClean/PQClean}\]
 Goals

- Framework that eases optimization for this platform
- Automate testing and benchmarking
- Include as many schemes as possible

4 types of implementations

\[\text{see https://github.com/PQClean/PQClean}\]
Goals

- Framework that eases optimization for this platform
- Automate testing and benchmarking
- Include as many schemes as possible

4 types of implementations

- ref: Reference C implementations from submission packages

\(^1\)see https://github.com/PQClean/PQClean
Goals

- Framework that eases optimization for this platform
- Automate testing and benchmarking
- Include as many schemes as possible

4 types of implementations

- `ref`: Reference C implementations from submission packages
- `clean`: Slightly modified reference implementations to satisfy basic code quality requirements

1 see https://github.com/PQClean/PQClean
Goals

- Framework that eases optimization for this platform
- Automate testing and benchmarking
- Include as many schemes as possible

4 types of implementations

- ref: Reference C implementations from submission packages
- clean: Slightly modified reference implementations to satisfy basic code quality requirements
- opt: Optimized portable C implementations

\[1\text{see https://github.com/PQClean/PQClean}\]
Goals

- Framework that eases optimization for this platform
- Automate testing and benchmarking
- Include as many schemes as possible

4 types of implementations

- ref: Reference C implementations from submission packages
- clean: Slightly modified reference implementations to satisfy basic code quality requirements
- opt: Optimized portable C implementations
- m4: Optimized using ARMv7E-M assembly

1See https://github.com/PQClean/PQClean
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Reference</th>
<th>Optimized</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIKE</td>
<td><img src="Lib" alt="X" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Classic McEliece</td>
<td><img src="Key" alt="X" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>CRYSTALS-Kyber</td>
<td><img src="Key" alt="✓" /></td>
<td><img src="Optimized" alt="✓" /></td>
<td>[BKS19]</td>
</tr>
<tr>
<td>Frodo-KEM</td>
<td><img src="Key" alt="✓" /></td>
<td><img src="Optimized" alt="✓" /></td>
<td>[BFM+18]</td>
</tr>
<tr>
<td>HQC</td>
<td><img src="Lib" alt="X" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>LAC</td>
<td><img src="Key" alt="✓" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>LEDAcrypt</td>
<td><img src="RAM" alt="X" /></td>
<td>WIP</td>
<td></td>
</tr>
<tr>
<td>NewHope</td>
<td><img src="Key" alt="✓" /></td>
<td><img src="Optimized" alt="✓" /></td>
<td>[AJS16]</td>
</tr>
<tr>
<td>NTRU</td>
<td><img src="Key" alt="✓" /></td>
<td><img src="Optimized" alt="✓" /></td>
<td>[KRS19]</td>
</tr>
<tr>
<td>NTRU Prime</td>
<td><img src="Key" alt="✓" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>NTS-KEM</td>
<td><img src="Key" alt="X" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>ROLLO</td>
<td><img src="Lib" alt="X" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Round5</td>
<td><img src="Key" alt="✓" /></td>
<td><img src="Optimized" alt="✓" /></td>
<td>Round5 team</td>
</tr>
<tr>
<td>RQC</td>
<td><img src="Lib" alt="X" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>SABER</td>
<td><img src="Key" alt="✓" /></td>
<td><img src="Optimized" alt="✓" /></td>
<td>[KRS19]</td>
</tr>
<tr>
<td>SIKE</td>
<td><img src="Key" alt="✓" /></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>ThreeBears</td>
<td><img src="Key" alt="✓" /></td>
<td><img src="Optimized" alt="✓" /></td>
<td>ThreeBears team</td>
</tr>
</tbody>
</table>

- **Key**: keys too large
- **RAM**: implementation uses too much RAM
- **Lib**: available implementations depend on external libraries
## Schemes included in pqm4—Signatures

<table>
<thead>
<tr>
<th>Scheme</th>
<th>reference</th>
<th>optimized</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYSRTALS-Dilithium</td>
<td>✅</td>
<td>✅</td>
<td>[GKOS18, RSGCB19]</td>
</tr>
<tr>
<td>FALCON</td>
<td>☒ RAM</td>
<td>✅</td>
<td>Falcon team</td>
</tr>
<tr>
<td>GeMSS</td>
<td>☒ Key</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>LUOV</td>
<td>✅</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>MQDSS</td>
<td>☒ RAM</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Picnic</td>
<td>☒ RAM</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>qTESLA</td>
<td>✅</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Rainbow</td>
<td>☒ Key</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>SPHINCS+</td>
<td>✅</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

- ☒ Key: keys too large
- ☒ RAM: implementation uses too much RAM
- ☒ Lib: available implementations depend on external libraries
Cycle counts

- We don't want to benchmark the memory controller.
- Downclock core to 24MHz with no wait states, allowing for comprehensible cycle counts.
- We use the hardware RNG of our platform.
- Most schemes only sample seed, so speed doesn't matter.
Cycle counts

- We don’t want to benchmark the memory controller.
Cycle counts

- We don’t want to benchmark the memory controller
  - Downclock core to 24MHz → no wait states
Cycle counts

- We don’t want to benchmark the memory controller
  - Downclock core to 24MHz → no wait states
  - Allows to have comprehensible cycle count
Benchmarking: Cycle Counts and RNG

- **Cycle counts**
  - We don’t want to benchmark the memory controller
    - Downclock core to 24MHz → no wait states
    - Allows to have comprehensible cycle count

- **randombytes**
Cycle counts

- We don’t want to benchmark the memory controller
  - Downclock core to 24MHz → no wait states
  - Allows to have comprehensible cycle count

randombytes

- We use the hardware RNG of our platform
Benchmarking: Cycle Counts and RNG

- **Cycle counts**
  - We don’t want to benchmark the memory controller
    - Downclock core to 24MHz → no wait states
    - Allows to have comprehensible cycle count

- **randombytes**
  - We use the hardware RNG of our platform
  - Most schemes only sample seed, so speed doesn’t matter
Submission packages often come with different implementations of SHA-2, SHA-3, or AES

1. `https://github.com/XKCP/XKCP`
2. `https://bench.cr.yp.to/supercop.html`
Submission packages often come with different implementations of SHA-2, SHA-3, or AES

- We don’t want to benchmark those

1. [https://github.com/XKCP/XKCP](https://github.com/XKCP/XKCP)
2. [https://bench.cr.yp.to/supercop.html](https://bench.cr.yp.to/supercop.html)
Submission packages often come with different implementations of SHA-2, SHA-3, or AES

- We don’t want to benchmark those

Our approach: Replace those with a single fast implementation to allow fair comparison

---

1. [https://github.com/XKCP/XKCP](https://github.com/XKCP/XKCP)
2. [https://bench.cr.yp.to/supercop.html](https://bench.cr.yp.to/supercop.html)
Submission packages often come with different implementations of SHA-2, SHA-3, or AES

- We don’t want to benchmark those

**Our approach:** Replace those with a single fast implementation to allow fair comparison

- SHA-3: ARMv7-M assembly implementation from XKCP
  
1. https://github.com/XKCP/XKCP

2. https://bench.cr.yp.to/supercop.html

Submission packages often come with different implementations of SHA-2, SHA-3, or AES
- We don’t want to benchmark those

Our approach: Replace those with a single fast implementation to allow fair comparison
- SHA-3: ARMv7-M assembly implementation from XKCP
- SHA-2: Fast C implementation from SUPERCOP

1https://github.com/XKCP/XKCP
2https://bench.cr.yp.to/supercop.html
3Schwabe and Stoffelen, SAC 2016
Submission packages often come with different implementations of SHA-2, SHA-3, or AES

- We don’t want to benchmark those

**Our approach:** Replace those with a single fast implementation to allow fair comparison

- SHA-3: ARMv7-M assembly implementation from XKCP
- SHA-2: Fast C implementation from SUPERCOP
- AES: ARMv7-M assembly implementation from [SS16]

---

1. [https://github.com/XKCP/XKCP](https://github.com/XKCP/XKCP)
2. [https://bench.cr.yp.to/supercop.html](https://bench.cr.yp.to/supercop.html)
Results

- Not all schemes have been optimized for this platform yet
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible

For the following results, we restrict to:
- Only schemes that have been optimized
- NIST security level 1
- For KEMs: CCA variants
- Only SHA-3/SHAKE variants

For the full results, see paper at https://ia.cr/2019/844
see https://github.com/mupq/pqm4
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something

For the following results, we restrict to
- Only schemes that have been optimized
- NIST security level 1
- For KEMs: CCA variants
- Only SHA-3/SHAKE variants

For the full results, see paper at https://ia.cr/2019/844
see https://github.com/mupq/pqm4
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us

For the following results, we restrict to
- Only schemes that have been optimized
- NIST security level 1
- For KEMs: CCA variants
- Only SHA-3/SHAKE variants

For the full results see paper at https://ia.cr/2019/844
see https://github.com/mupq/pqm4
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request

For the following results, we restrict to
- Only schemes that have been optimized
- NIST security level 1
- For KEMs: CCA variants
- Only SHA-3/SHAKE variants

For the full results, see paper at [https://ia.cr/2019/844](https://ia.cr/2019/844)

See [https://github.com/mupq/pqm4](https://github.com/mupq/pqm4)
Not all schemes have been optimized for this platform yet

We tried to collect as many implementations as possible

If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request

For the following results, we restrict to

- Only schemes that have been optimized
- NIST security level 1
- For KEMs: CCA variants
- Only SHA-3/SHAKE variants

For the full results, see paper at https://ia.cr/2019/844
Also see https://github.com/mupq/pqm4
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request
- For the following results, we restrict to
  - Only schemes that have been optimized

For the full results, see paper at [https://ia.cr/2019/844](https://ia.cr/2019/844)

See [https://github.com/mupq/pqm4](https://github.com/mupq/pqm4)
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request
- For the following results, we restrict to
  - Only schemes that have been optimized
  - NIST security level 1

[For the full results, see paper at](https://ia.cr/2019/844) [see](https://github.com/mupq/pqm4)
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request
- For the following results, we restrict to
  - Only schemes that have been optimized
  - NIST security level 1
  - For KEMs: CCA variants

For the full results, see paper at https://ia.cr/2019/844

See https://github.com/mupq/pqm4
- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request
- For the following results, we restrict to
  - Only schemes that have been optimized
  - NIST security level 1
  - For KEMs: CCA variants
  - Only SHA-3/SHAKE variants

For the full results see paper at https://ia.cr/2019/844 and see https://github.com/mupq/pqm4
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request
- For the following results, we restrict to
  - Only schemes that have been optimized
  - NIST security level 1
  - For KEMs: CCA variants
  - Only SHA-3/SHAKE variants
- For the full results

see paper at https://ia.cr/2019/844
see https://github.com/mupq/pqm4
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request
- For the following results, we restrict to
  - Only schemes that have been optimized
  - NIST security level 1
  - For KEMs: CCA variants
  - Only SHA-3/SHAKE variants
- For the full results
  - see paper at https://ia.cr/2019/844
Results

- Not all schemes have been optimized for this platform yet
- We tried to collect as many implementations as possible
- If we missed something
  - Send us an e-mail or talk to us
  - Open a pull request
- For the following results, we restrict to
  - Only schemes that have been optimized
  - NIST security level 1
  - For KEMs: CCA variants
  - Only SHA-3/SHAKE variants
- For the full results
  - see paper at https://ia.cr/2019/844
  - see https://github.com/mupq/pqm4
KEM Speed

- babybear-opt
- frodokem640shake-m4
- kyber512-m4
- lightsaber-m4
- ntruhps2048509-m4
- ntruhrss701-m4
- r5nd-1kemcca-0d-m4
- r5nd-1kemcca-5d-m4

KeyGen
Encaps
Decaps

k cycles

0
20000
40000
60000
80000
100000
120000
140000
160000
Signature Speed

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>KeyGen</th>
<th>Sign</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>dilithium2-m4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>falcon512-m4-ct</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The graph shows the comparison of KeyGen, Sign, and Verify times for the algorithms dilithium2-m4 and falcon512-m4-ct. The Y-axis represents the number of k cycles.
Signature Speed (2)

- dilithium2-m4
- falcon512-m4-ct

<table>
<thead>
<tr>
<th>k cycles</th>
<th>KeyGen</th>
<th>Sign</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
KEM RAM consumption

![Graph showing RAM consumption for various KEM algorithms]

- Babybear-opt
- FrodoKem640shake-m4
- Kyber512-m4
- LightSaber-m4
- NTRU+PS2048509-m4
- NTRU+RSS701-m4
- R5nd-1KemCCA-0d-m4
- R5nd-1KemCCA-5d-m4

Legend:
- KeyGen
- Encaps
- Decaps

Bytes range from 0 to 70000 in increments of 10000.
Signature RAM consumption

![Graph showing RAM consumption for different algorithms and operations.]

- **dilithium2-m4**
- **falcon512-m4-ct**

- **KeyGen**
- **Sign**
- **Verify**
Signature RAM consumption (2)

- dilithium2-m4
- falcon512-m4-ct

<table>
<thead>
<tr>
<th></th>
<th>KeyGen</th>
<th>Sign</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>1000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>2000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>3000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>4000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
</tbody>
</table>

Chart showing RAM consumption for KeyGen, Sign, and Verify for dilithium2-m4 and falcon512-m4-ct.
pqm4 currently includes

- Current implementations of Classic McEliece, LEDAcrypt, NTS-KEM, GeMSS, MQDSS, Picnic, and Rainbow consume more than 128 KiB of RAM—don’t fit

- BIKE, HQC, ROLLO, RQC use OpenSSL/NTL/GMP—needs to be replaced to make it work
pqm4 currently includes

- 10 KEMs (6 optimized)
pqt currently includes

- 10 KEMs (6 optimized)
- 5 signature schemes (2 optimized)
pqm4 currently includes
- 10 KEMs (6 optimized)
- 5 signature schemes (2 optimized)

Current implementations of Classic McEliece, LEDAcrypt, NTS-KEM, GeMSS, MQDSS, Picnic, and Rainbow consume more than 128 KiB of RAM
→ don’t fit
pqm4 currently includes

- 10 KEMs (6 optimized)
- 5 signature schemes (2 optimized)

Current implementations of Classic McEliece, LEDAcrypt, NTS-KEM, GeMSS, MQDSS, Picnic, and Rainbow consume more than 128 KiB of RAM
→ don’t fit

BIKE, HQC, ROLLO, RQC use OpenSSL/NTL/GMP
→ needs to be replaced to make it work
Still many schemes left to optimize → please PR
Conclusion

- Still many schemes left to optimize → please PR
- Level of optimization greatly differs
Conclusion

- Still many schemes left to optimize → please PR
- Level of optimization greatly differs
  - Most implementations don’t optimize RAM consumption
Still many schemes left to optimize → please PR

Level of optimization greatly differs

- Most implementations don’t optimize RAM consumption
- No implementations optimize code size
Conclusion

- Still many schemes left to optimize → please PR
- Level of optimization greatly differs
  - Most implementations don’t optimize RAM consumption
  - No implementations optimize code size
  - What does NIST care about?
Still many schemes left to optimize → please PR
Level of optimization greatly differs
- Most implementations don’t optimize RAM consumption
- No implementations optimize code size
- What does NIST care about?
Currently, Round5 seems to be the fastest on this platform
Conclusion

- Still many schemes left to optimize → please PR
- Level of optimization greatly differs
  - Most implementations don’t optimize RAM consumption
  - No implementations optimize code size
  - What does NIST care about?
- Currently, Round5 seems to be the fastest on this platform
  - But Kyber, NTRU, Saber, ThreeBears very close
https://github.com/mupq/pqm4

slides and paper available at kannwischer.eu

Thank you!
Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. A new hope on ARM Cortex-M.

Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam.
Fly, you fool! faster frodo for the ARM Cortex-M4.
Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. Memory-efficient high-speed implementation of Kyber on Cortex-M4.
