Radboud University

# 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


[pi kju ɛm fɔr]

# Motivation

- "Performance will play a larger role in the second round"

# Motivation

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

# 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

# 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

# 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

# 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
  - Side-channels?

# 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
  - Side-channels?
- **Challenges**

# 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
  - Side-channels?
- **Challenges**
  - Do schemes even fit in limited RAM + flash?

# 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
  - Side-channels?
- **Challenges**
  - Do schemes even fit in limited RAM + flash?
  - Are schemes efficient on small ARMs?

# 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
  - Side-channels?
- **Challenges**
  - Do schemes even fit in limited RAM + flash?
  - Are schemes efficient on small ARMs?
  - What is the overhead of masking?

# Post-quantum on small devices

*"It's big and it's slow"*

# Post-quantum on small devices

*"It's big and it's slow"*

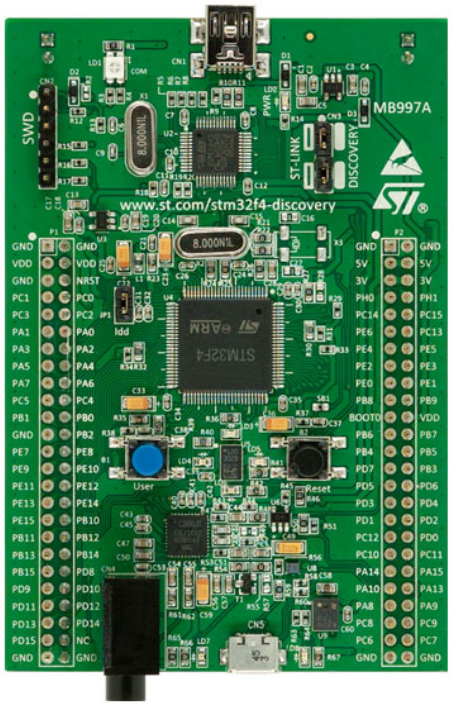
– everyone, always

# 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`



# Rationale for using STM32F4DISCOVERY boards

- 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] see https://github.com/PQClean/PQClean

- **Goals**
  - Framework that eases optimization for this platform

[1] see https://github.com/PQClean/PQClean

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

[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

[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**

[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

[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]

[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]
  - `opt`: Optimized portable C implementations

[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]
  - `opt`: Optimized portable C implementations
  - `m4`: Optimized using ARMv7E-M assembly

[1] see https://github.com/PQClean/PQClean

# Schemes included in pqm4– KEMs

| | reference | optimized | |
|---|---|---|---|
| BIKE | ✗$_{Lib}$ | — | |
| Classic McEliece | ✗$_{Key}$ | — | |
| CRYSTALS-Kyber | ✓ | ✓ | [BKS19] |
| Frodo-KEM | ✓ | ✓ | [BFM+18] |
| HQC | ✗$_{Lib}$ | — | |
| LAC | ✓ | — | |
| LEDAcrypt | ✗$_{RAM}$ | WIP | |
| NewHope | ✓ | ✓ | [AJS16] |
| NTRU | ✓ | ✓ | [KRS19] |
| NTRU Prime | ✓ | — | |
| NTS-KEM | ✗$_{Key}$ | — | |
| ROLLO | ✗$_{Lib}$ | — | |
| Round5 | ✓ | ✓ | Round5 team |
| RQC | ✗$_{Lib}$ | — | |
| SABER | ✓ | ✓ | [KRS19] |
| SIKE | ✓ | — | |
| ThreeBears | ✓ | ✓ | ThreeBears team |

✗$_{Key}$: keys too large ✗$_{RAM}$: implementation uses too much RAM

✗$_{Lib}$: available implementations depend on external libraries

# Schemes included in pqm4– Signatures

| | reference | optimized | |
|---|---|---|---|
| CRYSTALS-Dilithium | ✓ | ✓ | [GKOS18, RSGCB19] |
| FALCON | ✗$_{RAM}$ | ✓ | Falcon team |
| GeMSS | ✗$_{Key}$ | — | |
| LUOV | ✓ | — | |
| MQDSS | ✗$_{RAM}$ | — | |
| Picnic | ✗$_{RAM}$ | — | |
| qTESLA | ✓ | — | |
| Rainbow | ✗$_{Key}$ | — | |
| SPHINCS+ | ✓ | — | |

✗$_{Key}$: keys too large ✗$_{RAM}$: implementation uses too much RAM

✗$_{Lib}$: available implementations depend on external libraries

# Benchmarking: Cycle Counts and RNG

- **Cycle counts**

# Benchmarking: Cycle Counts and RNG

- **Cycle counts**
  - We don't want to benchmark the memory controller

# Benchmarking: Cycle Counts and RNG

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

# 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

# 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**

# 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

# 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

# Benchmarking: Fast Hashing

- 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
[3] Schwabe and Stoffelen, SAC 2016

# Benchmarking: Fast Hashing

- 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
[2] https://bench.cr.yp.to/supercop.html
[3] Schwabe and Stoffelen, SAC 2016

# Benchmarking: Fast Hashing

- 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
[2] https://bench.cr.yp.to/supercop.html
[3] Schwabe and Stoffelen, SAC 2016

# Benchmarking: Fast Hashing

- 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]

[1] https://github.com/XKCP/XKCP
[2] https://bench.cr.yp.to/supercop.html
[3] Schwabe and Stoffelen, SAC 2016

# Benchmarking: Fast Hashing

- 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]
- SHA-2: Fast C implementation from SUPERCOP [2]

[1] https://github.com/XKCP/XKCP
[2] https://bench.cr.yp.to/supercop.html
[3] Schwabe and Stoffelen, SAC 2016

# Benchmarking: Fast Hashing

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

[1] https://github.com/XKCP/XKCP
[2] https://bench.cr.yp.to/supercop.html
[3] Schwabe and Stoffelen, SAC 2016

# 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

# Results

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

# 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

# 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

# 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

# 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

# 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

# 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

# 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

# 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

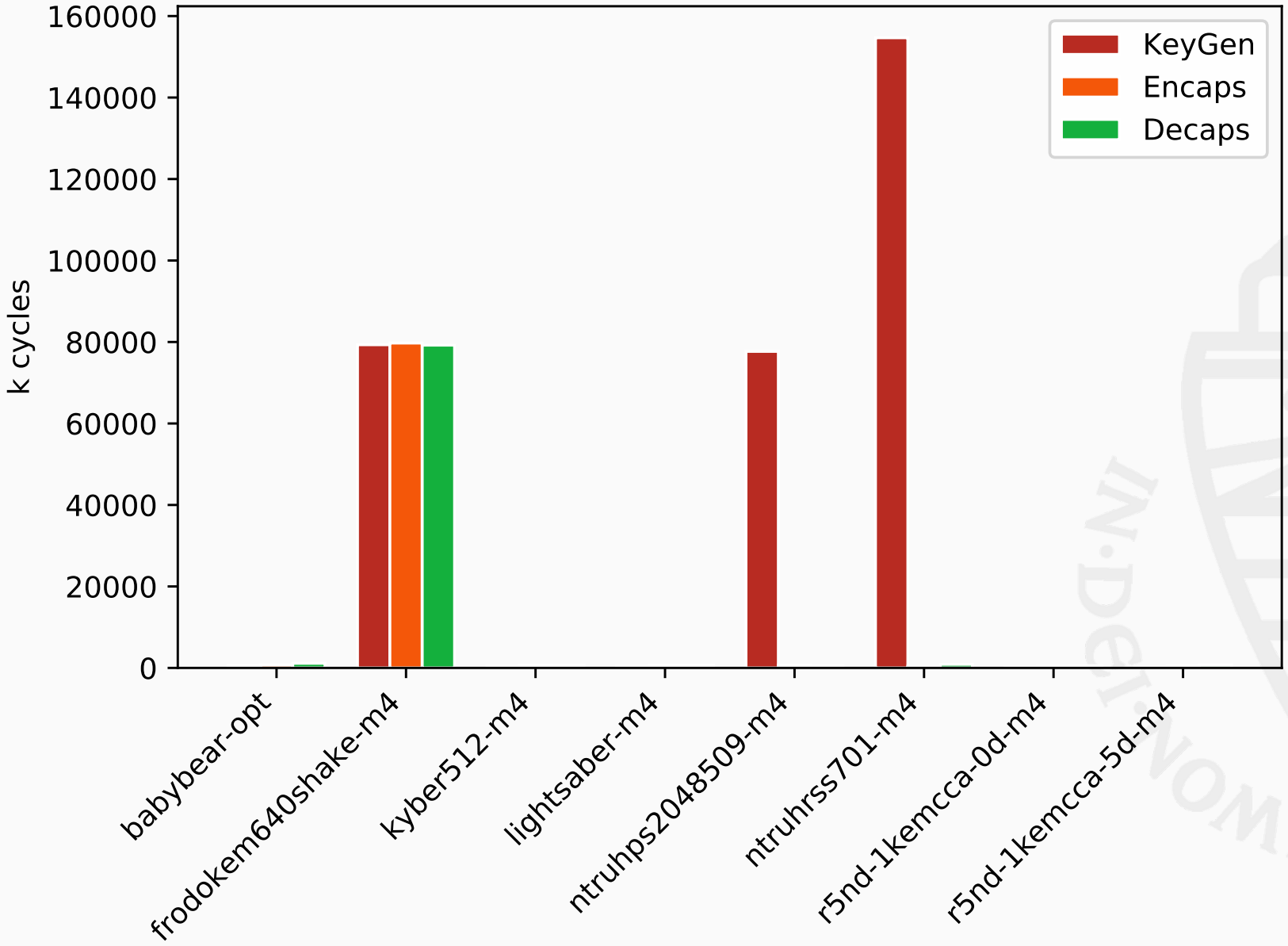
# Results

- 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`

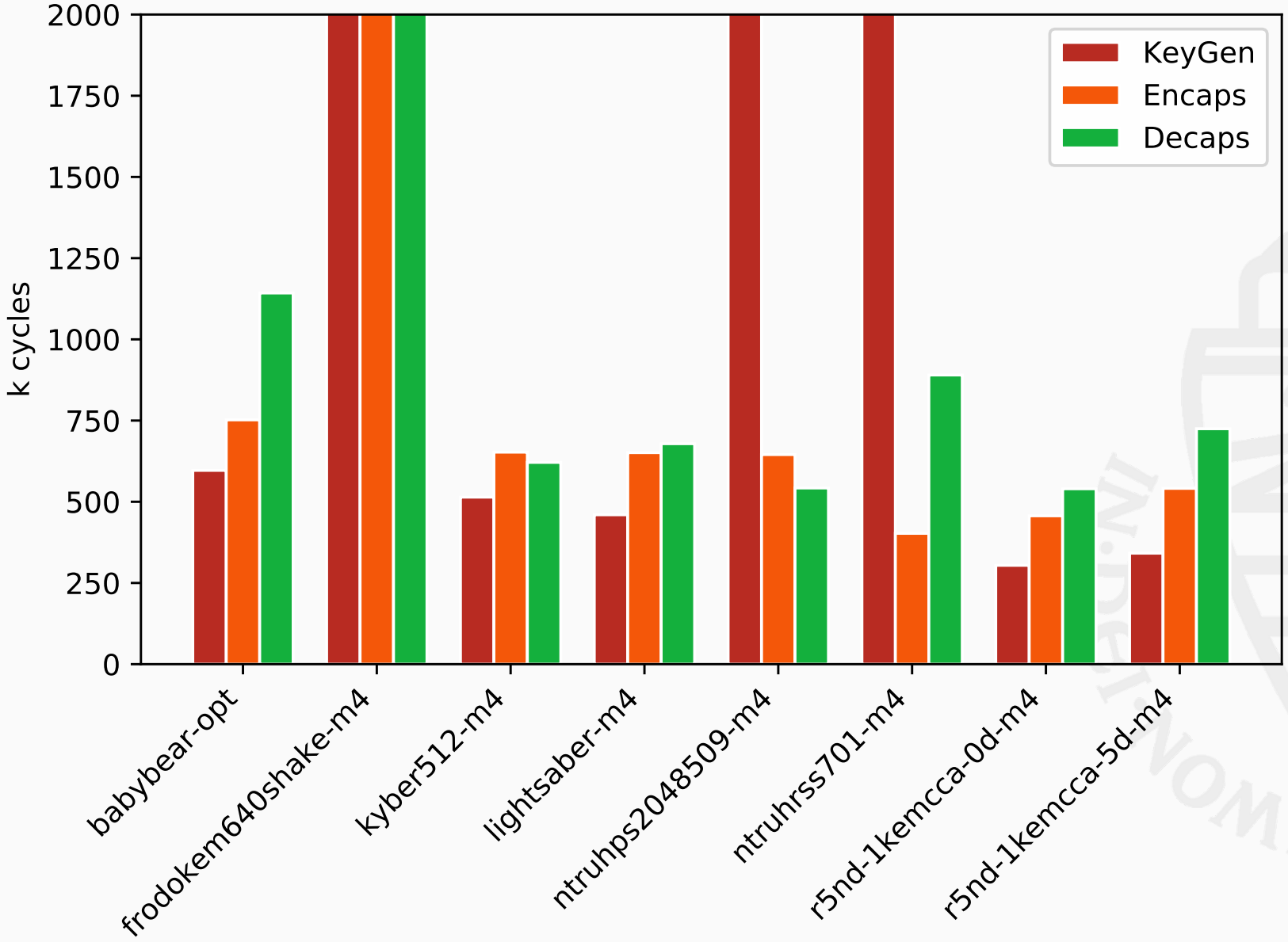
# Results

- 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`

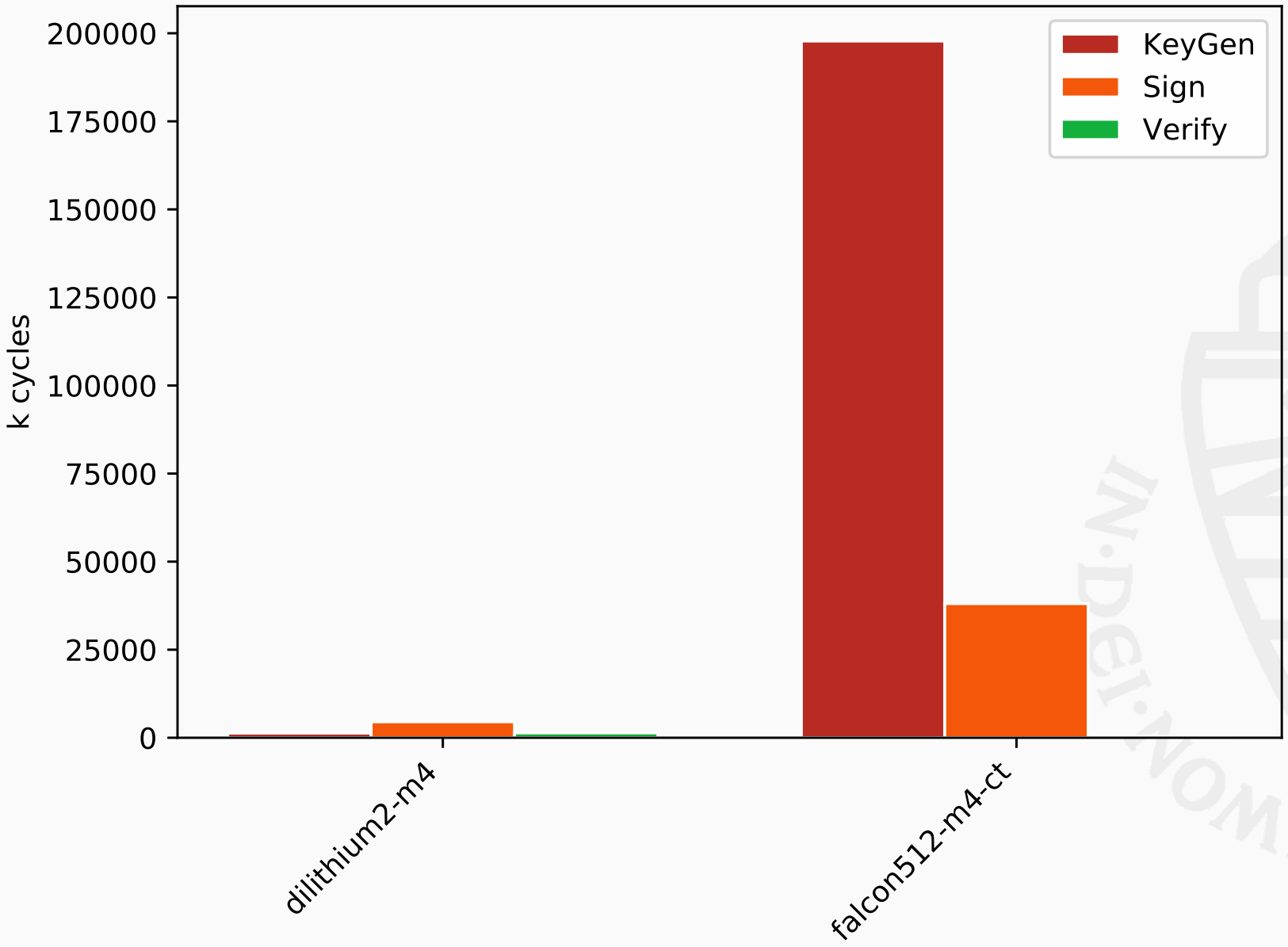
# KEM Speed


160000
140000
120000
100000
80000
60000
40000
20000
0
k cycles
KeyGen
Encaps
Decaps
babybear-opt
frodokem640shake-m4
kyber512-m4
lightsaber-m4
ntruhps2048509-m4
ntruhrss701-m4
r5nd-1kemcca-0d-m4
r5nd-1kemcca-5d-m4

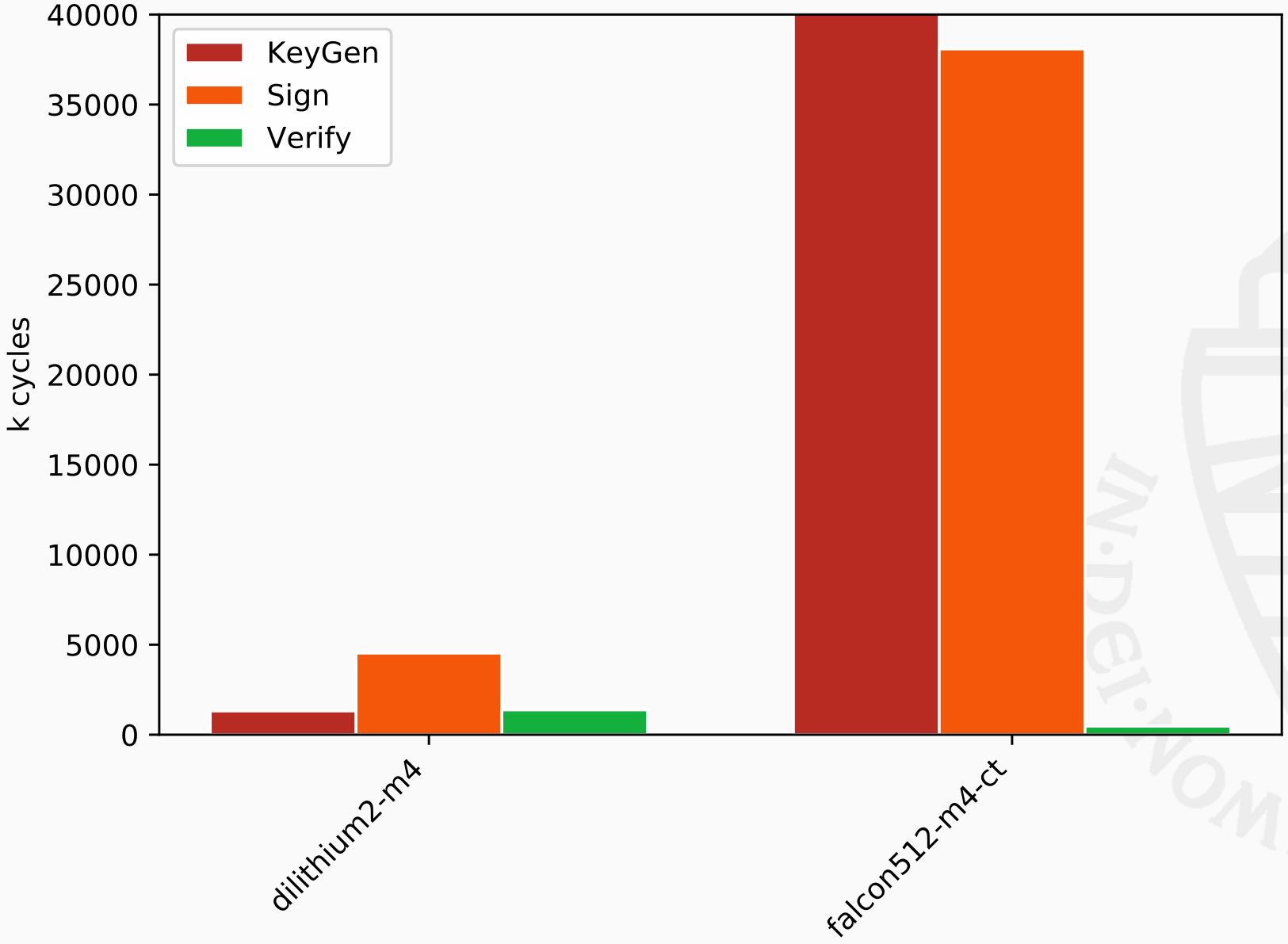
# KEM Speed (2)


2000
1750
1500
1250
1000
750
500
250
0
k cycles
KeyGen
Encaps
Decaps
babybear-opt
frodokem640shake-m4
kyber512-m4
lightsaber-m4
ntruhps2048509-m4
ntruhrss701-m4
r5nd-1kemcca-0d-m4
r5nd-1kemcca-5d-m4

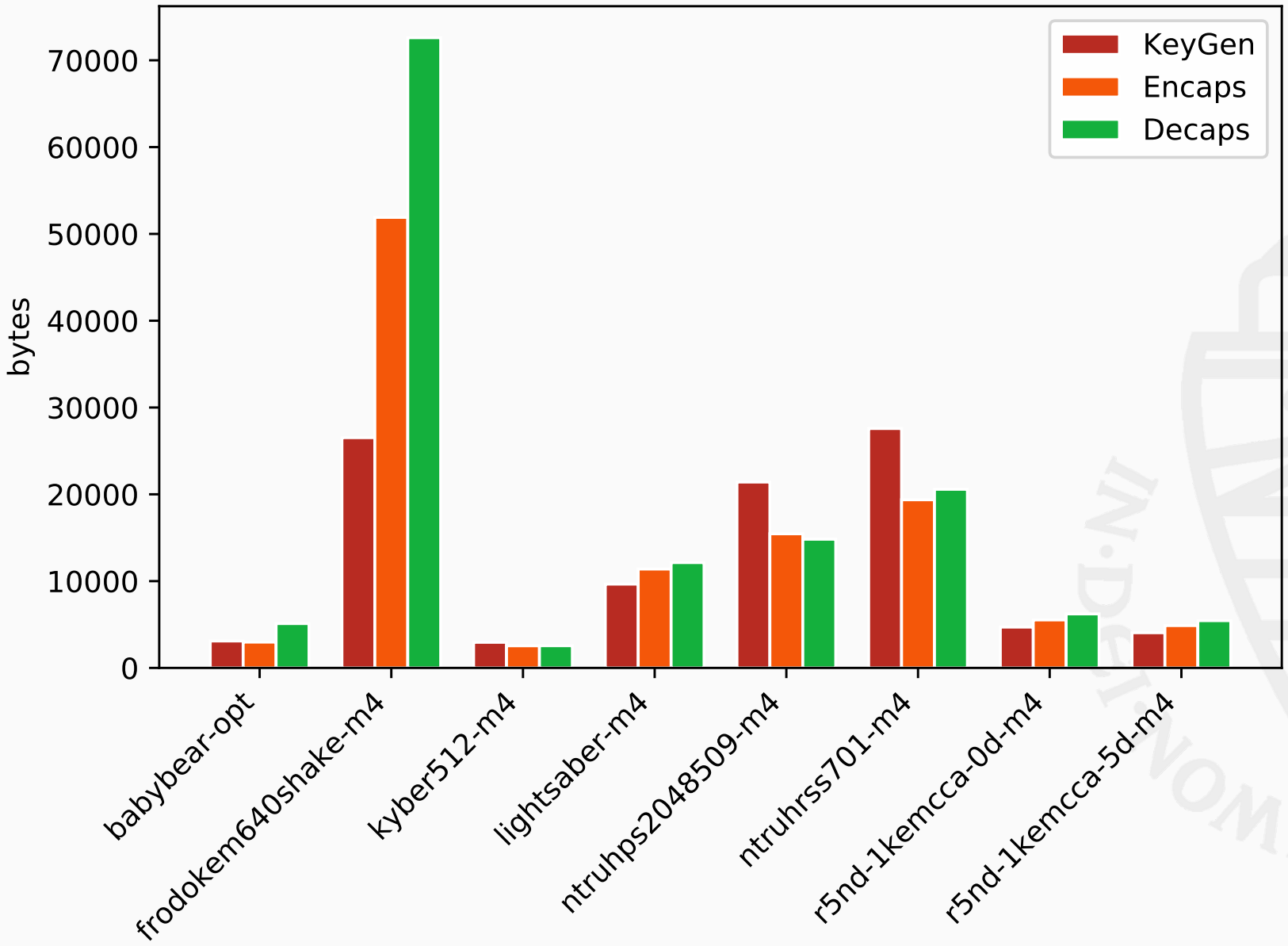
# Signature Speed



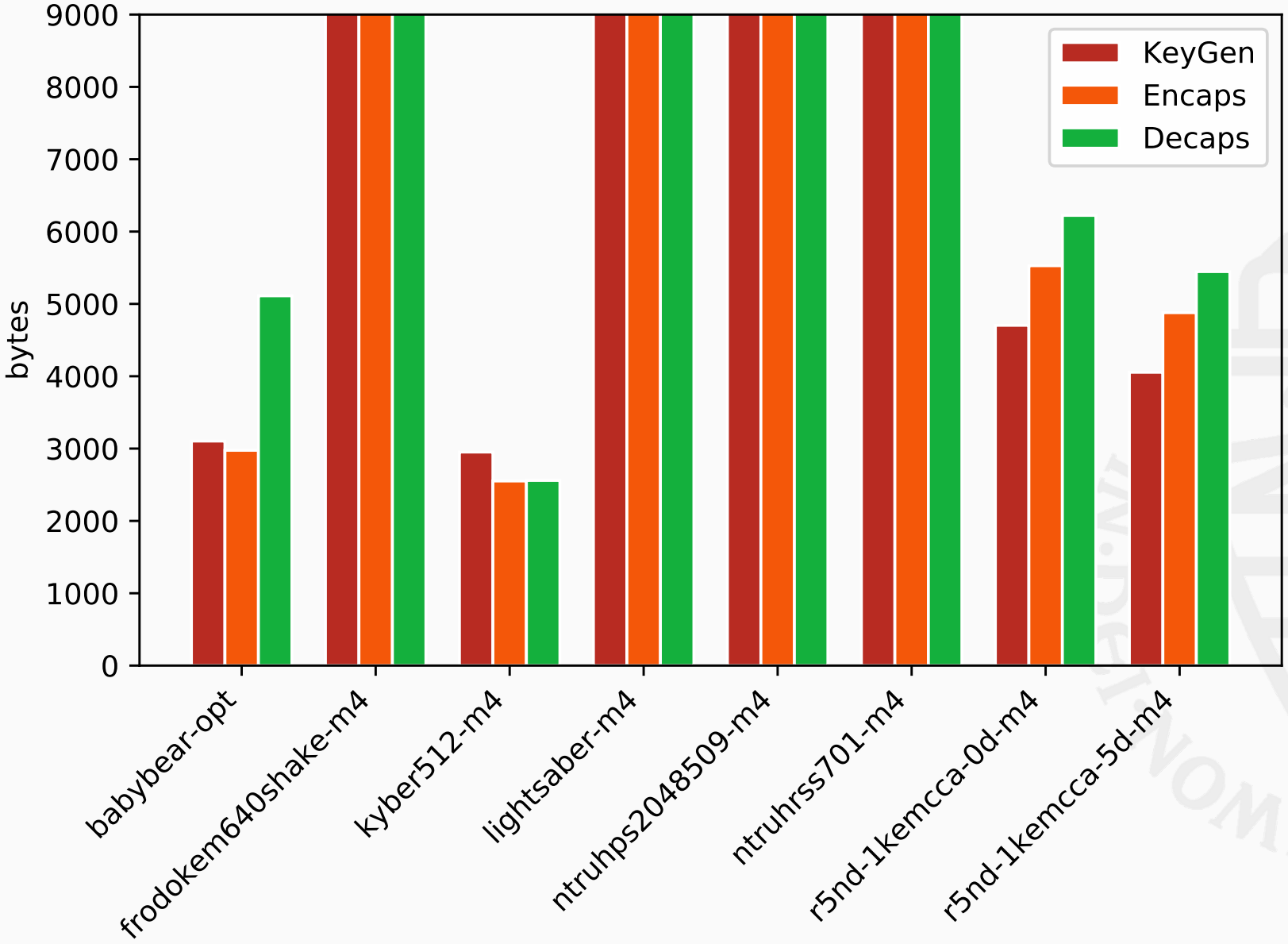
# Signature Speed (2)



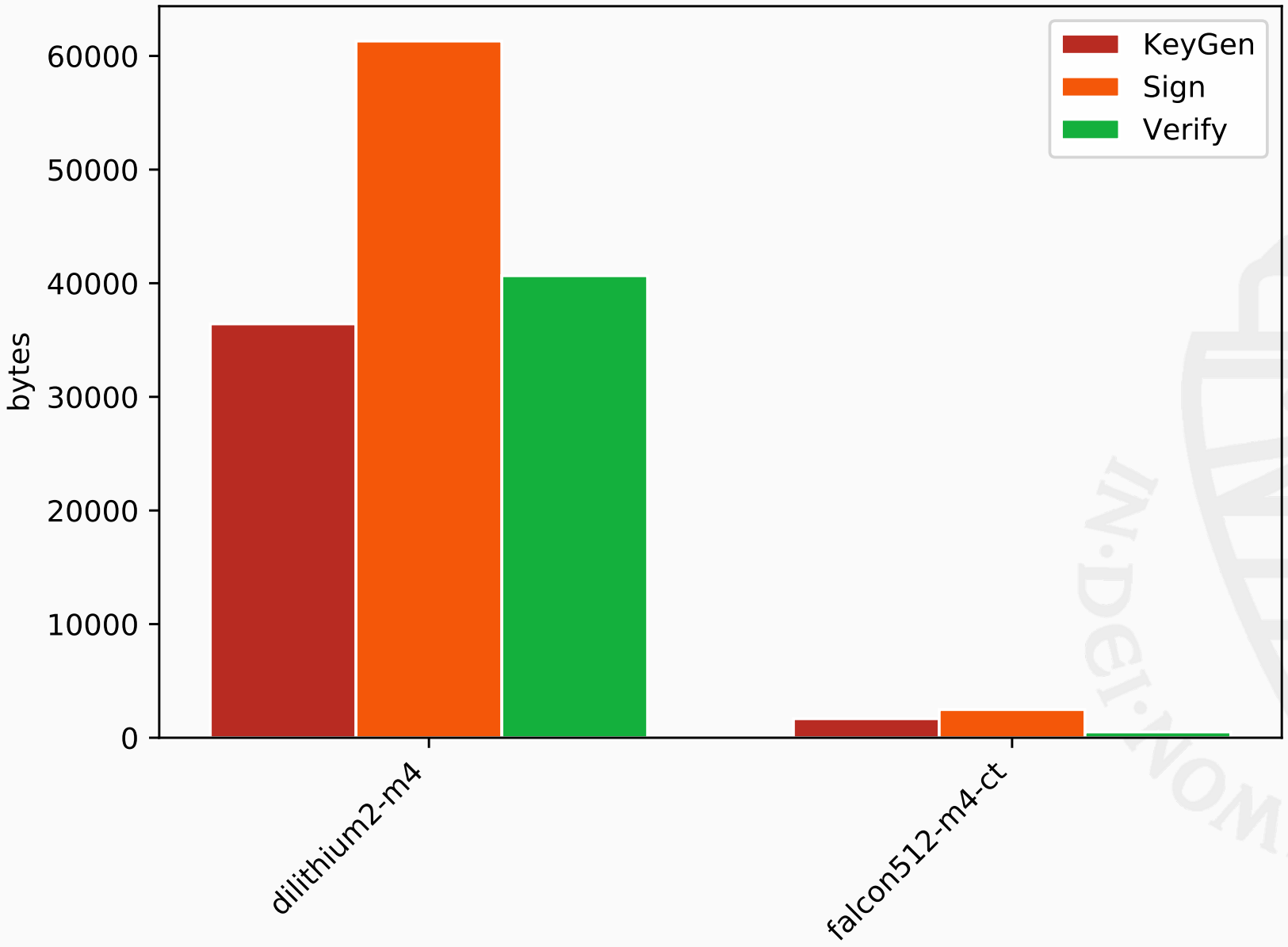
# KEM RAM consumption



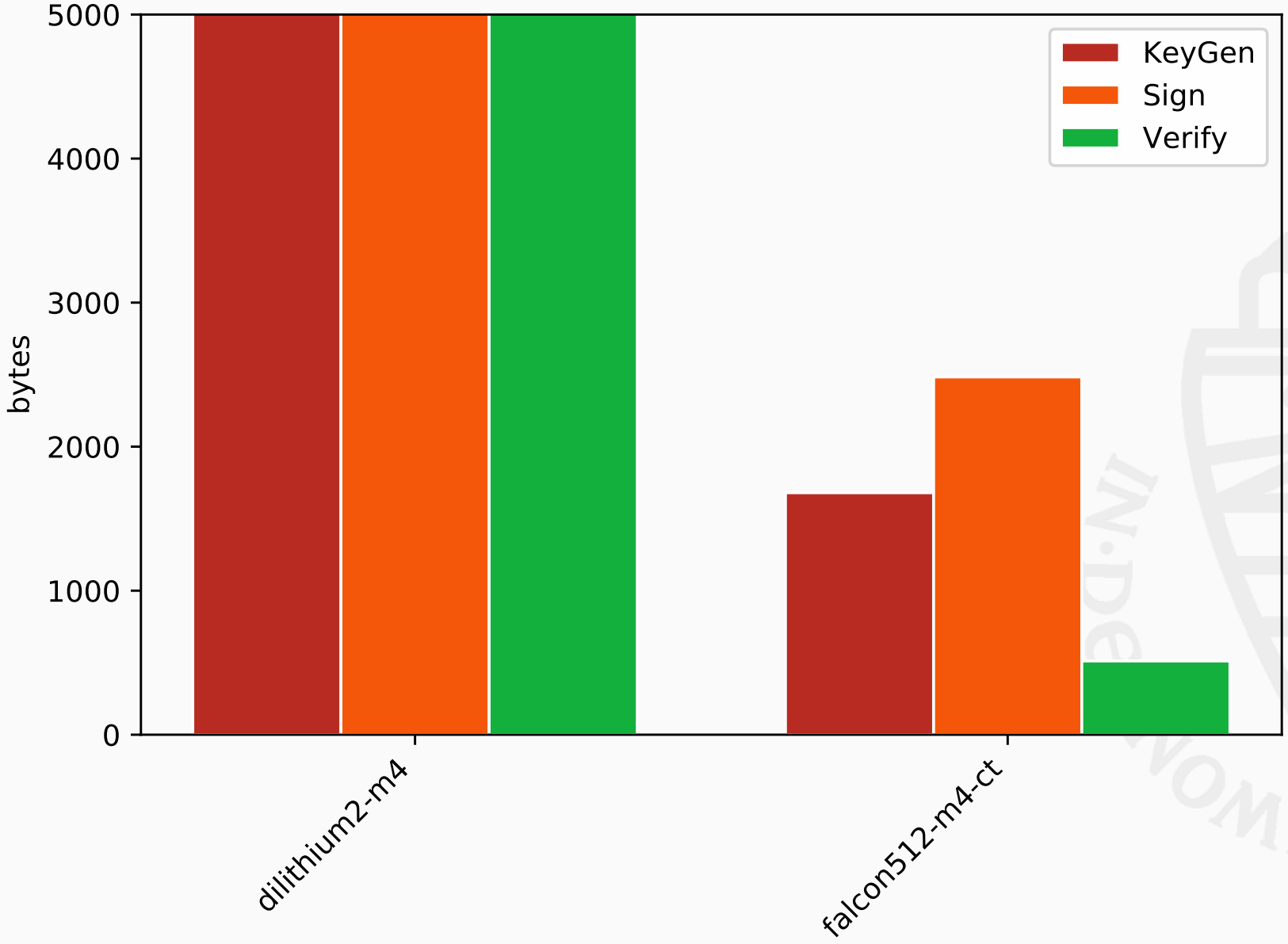
# KEM RAM consumption (2)



# Signature RAM consumption



# Signature RAM consumption (2)



# Conclusion

- `pqm4` currently includes

# Conclusion

- `pqm4` currently includes
  - 10 KEMs (6 optimized)

# Conclusion

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

# Conclusion

- `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
  $\rightarrow$ don't fit

# Conclusion

- `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
  $\rightarrow$ don't fit
- BIKE, HQC, ROLLO, RQC use OpenSSL/NTL/GMP
  $\rightarrow$ needs to be replaced to make it work

# Conclusion

- 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

# 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

# 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?

# 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

# 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!**

# References i


Erdem Alkim, Philipp Jakubeit, and Peter Schwabe.
**A new hope on ARM Cortex-M.**
In *Security, Privacy, and Advanced Cryptography Engineering*, volume 10076 of *Lecture Notes in Computer Science*, pages 332–349. Springer-Verlag Berlin Heidelberg, 2016.

Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam.
**Fly, you fool! faster frodo for the ARM Cortex-M4.**
Cryptology ePrint Archive, Report 2018/1116, 2018.
`https://eprint.iacr.org/2018/1116`.

# References ii


Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. **Memory-efficient high-speed implementation of Kyber on Cortex-M4.** In *Progress in Cryptology – Africacrypt 2019*, Lecture Notes in Computer Science, pages 209–228. Springer-Verlag Berlin Heidelberg, 2019.

Tim Güneysu, Markus Krausz, Tobias Oder, and Julian Speith. **Evaluation of lattice-based signature schemes in embedded systems.** In *25th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2018*, pages 385–388, 2018.

# References iii


Matthias J. Kannwischer, Joost Rijneveld, and Peter Schwabe.
**Faster multiplication in $\mathbf{Z}_{2^m}[x]$ on Cortex-M4 to speed up NIST PQC candidates.**
In *Applied Cryptography and Network Security*, Lecture Notes in Computer Science, pages 281–301. Springer-Verlag Berlin Heidelberg, 2019.

Prasanna Ravi, Sourav Sen Gupta, Anupam Chattopadhyay, and Shivam Bhasin.
**Improving speed of Dilithium's signing procedure.**
Cryptology ePrint Archive, Report 2019/420, 2019.
https://eprint.iacr.org/2019/420.