Radboud University

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

Matthias Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen

6 September 2019, Crypto Working Group

NIST PQC Recap

2012	NIST starts PQC project
Dec 2016	Call for proposals
Dec 2017	69 submissions “complete and proper”
Apr 2018	First PQC standardization conference
Jan 2019	26 submissions advance to round 2
Aug 2019	Second PQC standardization conference
2020/2021	Start round 3
2022/2024	Draft standards

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 memory?
  - Are schemes fast enough on small microprocessors?
  - What is the overhead of side-channel countermeasures?

Small Devices

RFC7228, Section 3

Name	data size (e.g., RAM)	code size (e.g., Flash)
Class 0, C0	<< 10 KiB	<< 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB

Table 1: Classes of Constrained Devices (KiB = 1024 bytes)

Post-Quantum on Small Devices

"It's big and it's slow"

— everyone, always

- STM32F4DISCOVERY
  - ARM Cortex-M4 @168 MHz
  - 32-bit, ARMv7E-M
  - 192 KiB RAM, 1 MiB flash
- PQM4: test and optimize on the Cortex-M4
  - `github.com/mupq/pqm4`

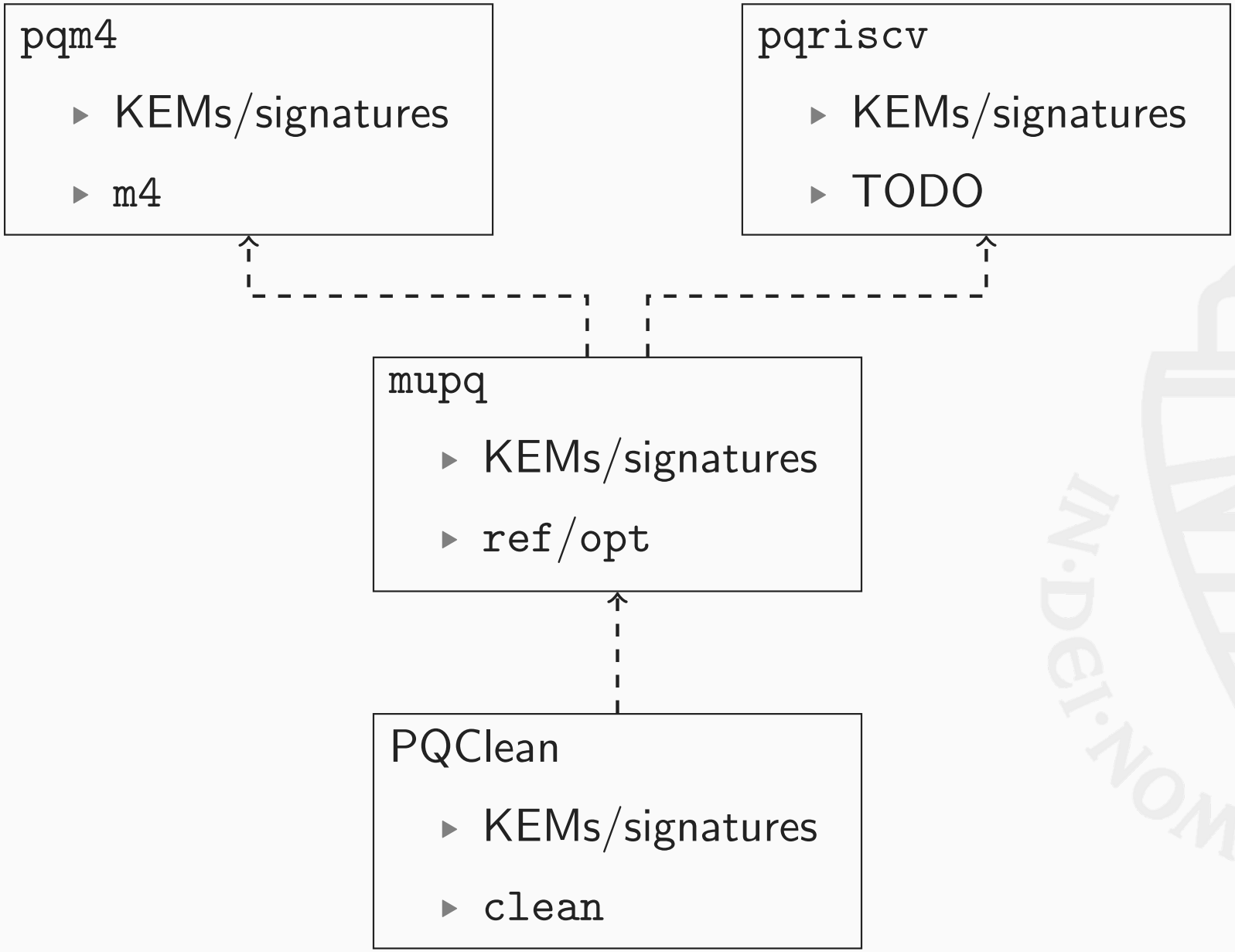
Why This Device

- They are cheap ($< €30$)
- They are huge in terms of RAM and flash memory
  - Great for PQC — many schemes fit
  - Unfortunately, not pqRSA
- ARMv7E-M more interesting for optimized assembly code
- Cortex-M4 widely deployed (billions)
- 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**
  - 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

pqm4 Structure



pqm4 Features

- `test.bin`:
  - For KEMs, Alice and Bob derive the same key
  - For signatures, valid signature can be verified
  - Wrong values should NOT work
  - Detect writing/reading out of bounds
- `testvectors.bin`:
  - Use a deterministic RNG
  - Cross-check result against other implementations
  - Cross-check result of M4 against result on host laptop
- `stack.bin`: Write stack canary, measure stack consumption
- `speed.bin`: Benchmark main functions
- `hashing.bin`: Measure time spent in SHA-2, SHA-3, AES
- Measure code size

Benchmarking: Cycle Counts and RNG

- **Cycle counts**
  - SysTick, not DWT `CYCCNT`
  - We don't want to benchmark the memory controller
    - Downclock core to 24MHz → no wait states
    - Allows to have comprehensible cycle count
- **Randomness**
  - Device has hardware RNG that we use
  - Most schemes only sample seed, so speed not significant
  - In practice: ~16 cycles/byte

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

Benchmarking: Other Challenges

- Fix `arm-none-eabi-gcc` version and flags
  - upgrading 5.4.1 to 9.2.0 can give 20% speed-up
- 192 KiB RAM = 64 KiB + 112 KiB + 16 KiB
  - CCM: 64 KiB mapped at `0x10000000`–`0x1000FFFF`
  - SRAM1: 112 KiB mapped at `0x20000000`–`0x2001BFFF`
  - SRAM2: 16 KiB mapped at `0x2001c000`–`0x2001FFFF`
- We can use SRAM1+SRAM2, but...
- Just using SRAM1 can turn out to be faster!

Schemes in Round 2

Biting the Bullet

Crystals-Kyber	Lattice	MLWE
KINDI	Lattice	MLWE
Saber	Lattice	MLWR
FrodoKEM	Lattice	LWE
Lotus	Lattice	LWE
Lizard	Lattice	LWE/RLWE
Emblem/R.emblem	Lattice	LWE/RLWE
KCL	Lattice	LWE/RLWE/LWR
Round 2	Lattice	LWR/RLWR
Hila5	Lattice	RLWE
Ding's key exchange	Lattice	RLWE
LAC	Lattice	RLWE
Lima	Lattice	RLWE
NewHope	Lattice	RLWE
Three Bears	Lattice	IMLWE
Mersenne-756839	Lattice	ILWE
Titanium	Lattice	MP-LWE
Ramstake	Lattice	LWE like
Odd Manhattan	Lattice	Generic
NTRU Encrypt	Lattice	NTRU
NTRU-HRSS-KEM	Lattice	NTRU
NTRUprime	Lattice	NTRU

Crystals-Kyber	Lattice	MLWE
Saber	Lattice	MLWR
FrodoKEM	Lattice	LWE
Round 5	Lattice	LWR/RLWR
LAC	Lattice	RLWE
NewHope	Lattice	RLWE
Three Bears	Lattice	IMLWE
NTRU	Lattice	NTRU
NTRUprime	Lattice	NTRU

Big Quake	Codes	Goppa
Classic McEliece	Codes	Goppa
NTS-KEM	Codes	Goppa
BIKE	Codes	short Hamming
HQC	Codes	short Hamming
LEDAkem	Codes	short Hamming
LEDApkc	Codes	short Hamming
QC-MDPC KEM	Codes	short Hamming
LAKE	Codes	low rank
LOCKER	Codes	low rank
Ouroboros-R	Codes	low rank
RQC	Codes	low rank
SIKE	Isogeny	Isogeny

Classic McEliece	Codes	Goppa
NTS-KEM	Codes	Goppa
BIKE	Codes	short Hamming
HQC	Codes	short Hamming
LEDAcrypt	Codes	short Hamming
Rollo	Codes	low rank
RQC	Codes	low rank
SIKE	Isogeny	Isogeny

Signatures		
CRYSTALS-Dilithium	Lattice	Fiat-Shamir
qTesla	Lattice	Fiat-Shamir
Falcon	Lattice	Hash then sign
pqNTRUSign	Lattice	Hash then sign
Gravity-SPHINCS	Symm	Hash
SPHINCS+	Symm	Hash
Picnic	Symm	ZKP
GeMMS	MultVar	HFE
Gui	MultVar	HFE
HiMQ-3	MultVar	UOV
LUOV	MultVar	UOV
Rainbow	MultVar	UOV
MQDSS	MultVar	Fiat-Shamir

Signatures		
CRYSTALS-Dilithium	Lattice	Fiat-Shamir
qTesla	Lattice	Fiat-Shamir
Falcon	Lattice	Hash then sign
SPHINCS+	Symm	Hash
Picnic	Symm	ZKP
GeMMS	MultVar	HFE
LUOV	MultVar	UOV
Rainbow	MultVar	UOV
MQDSS	MultVar	Fiat-Shamir

Schemes in Round 2

	KEMs/PKE	Signatures	Total
Lattices	9	3	12
Codes	7	0	7
Multivariate	0	4	4
Isogenies	1	0	1
Symmetric		2	2
Total	17	9	26

Schemes 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	TODO	—	
Round5	✓	✓	Round5 team
RQC	TODO	—	
SABER	✓	✓	[KRS19]
SIKE	✓	—	
ThreeBears	✓	✓	ThreeBears team

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

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

Schemes in pqm4 — Signatures

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

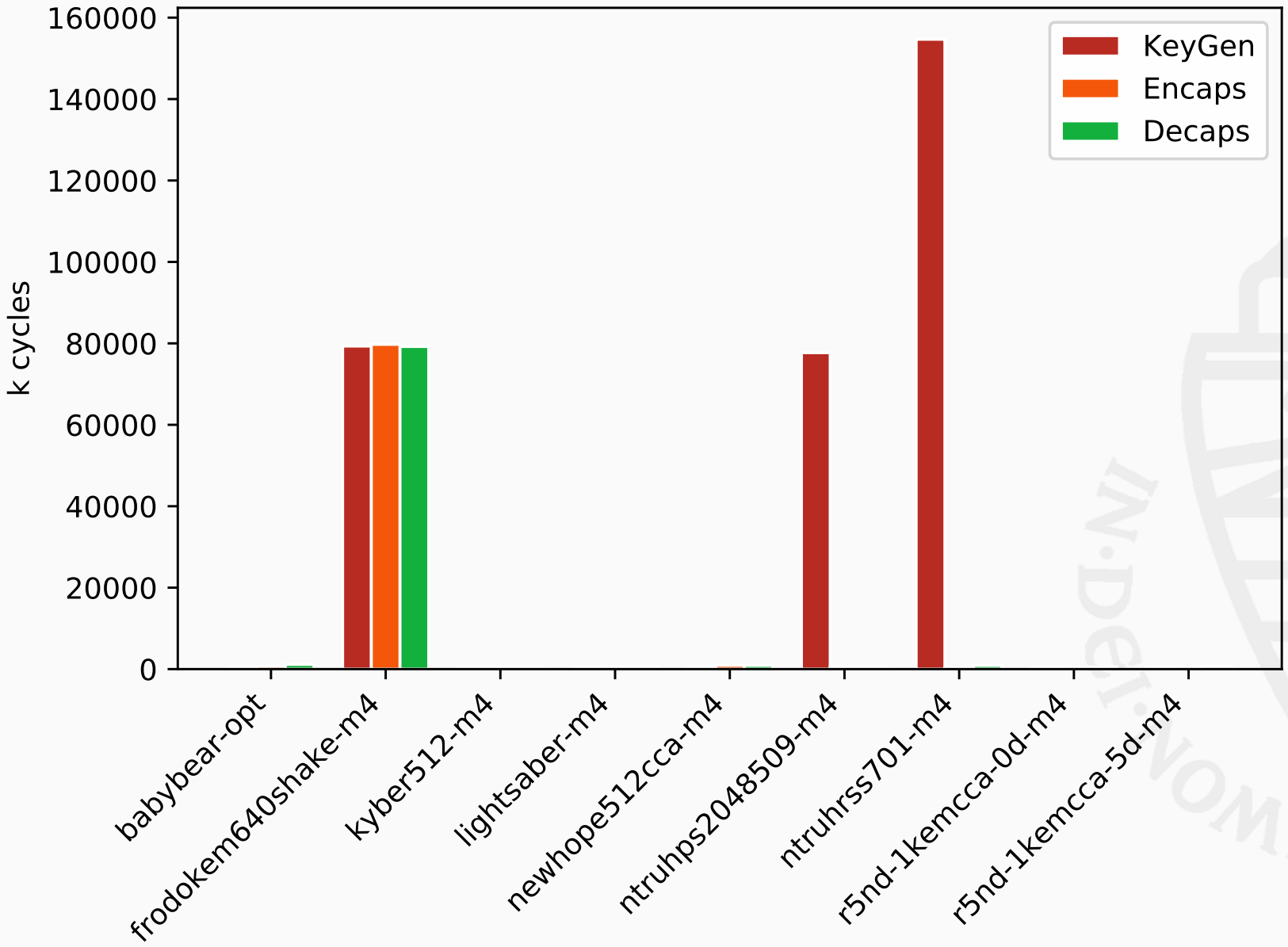
$\times_{Key}$: keys too large $\times_{RAM}$: implementation uses too much RAM

$\times_{Lib}$: available implementations depend on external libraries

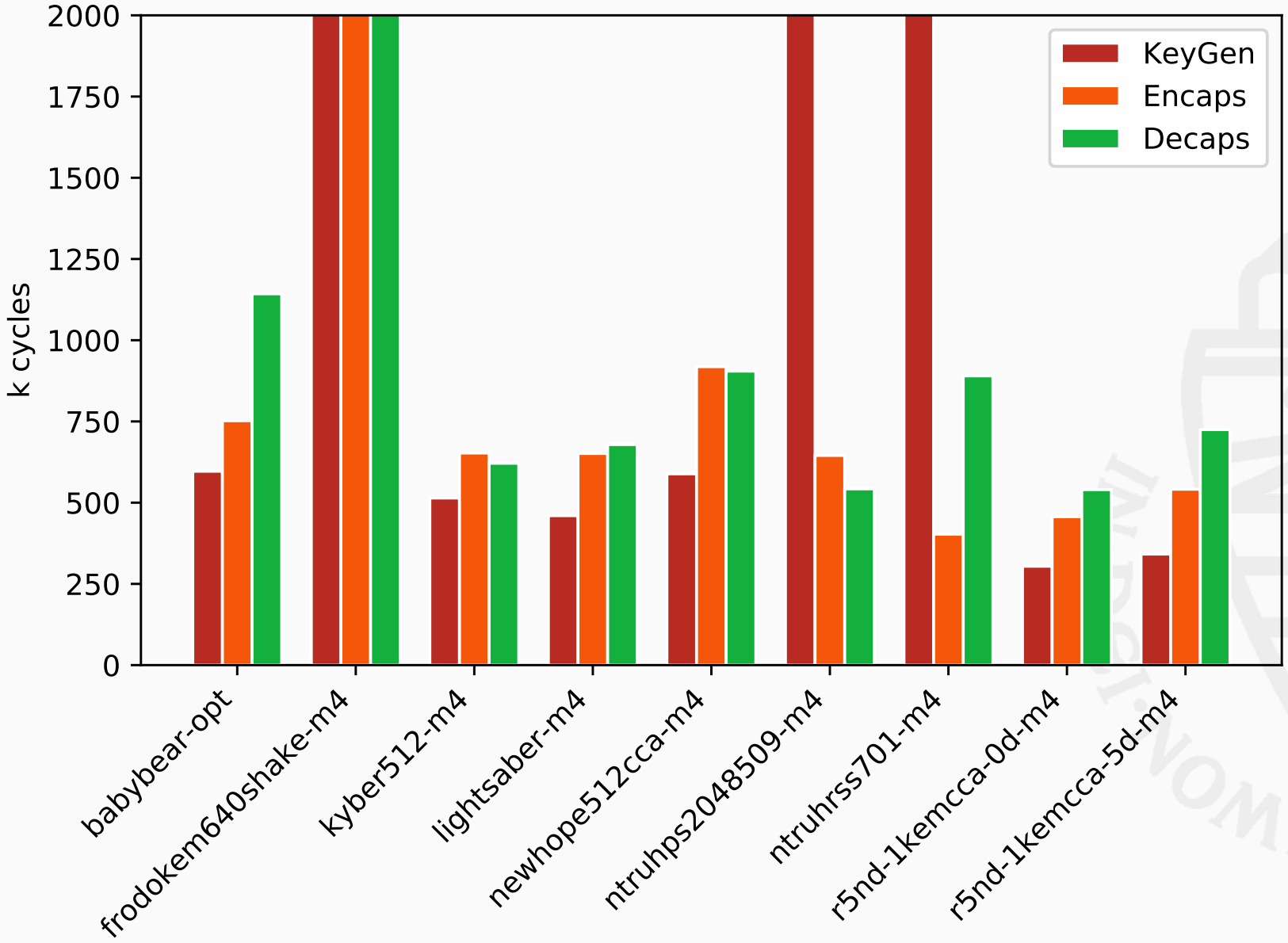
Results

- Not all schemes have been optimized for this platform yet
- 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`
  - $>$ 150 implementations!

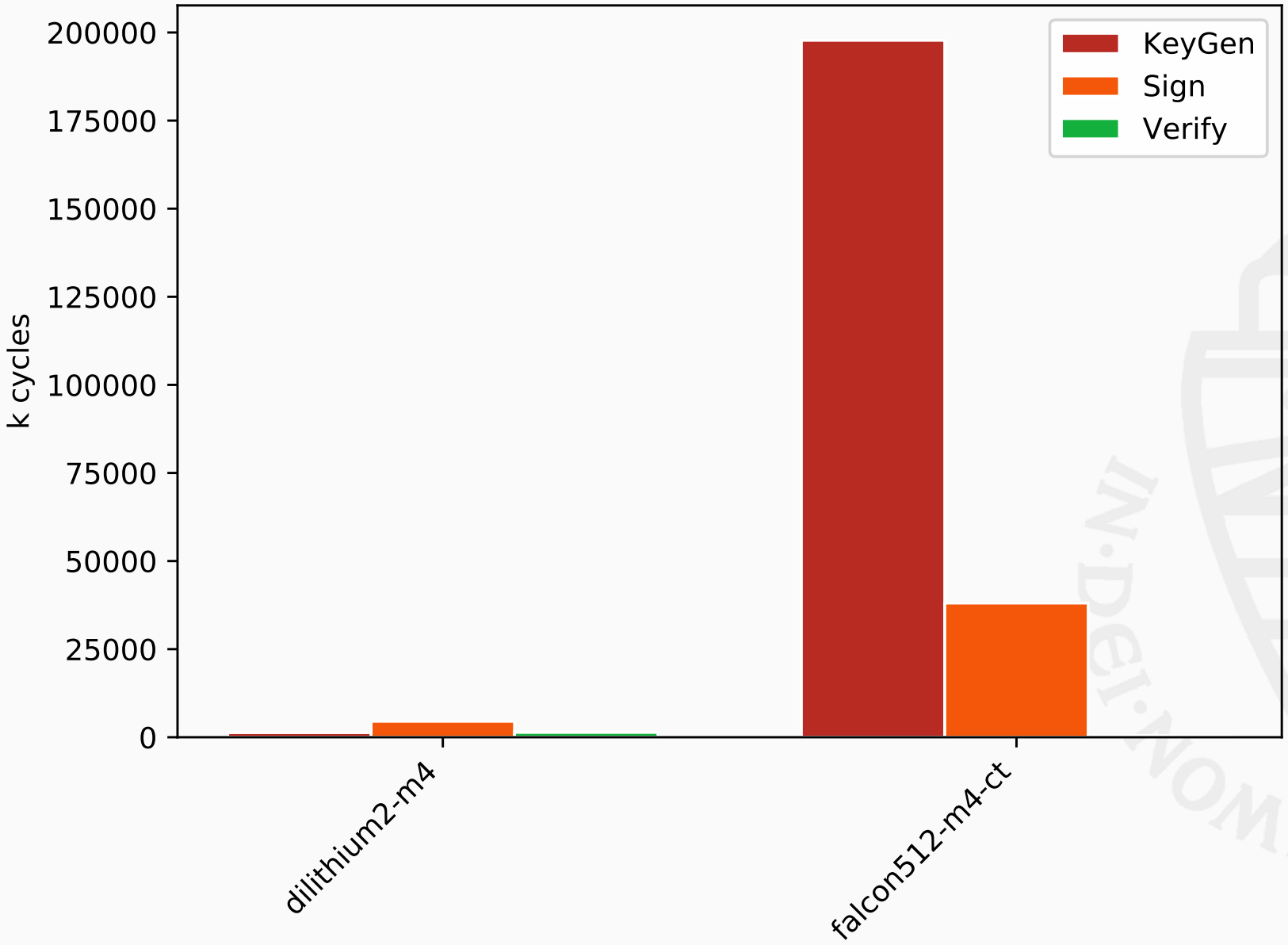
KEM Speed


160000
140000
120000
100000
80000
60000
40000
20000
0
k cycles
KeyGen
Encaps
Decaps
babybear-opt
frodokem640shake-m4
kyber512-m4
lightsaber-m4
newhope512cca-m4
ntruhps2048509-m4
ntruhrss701-m4
r5nd-1kemcca-0d-m4
r5nd-1kemcca-5d-m4

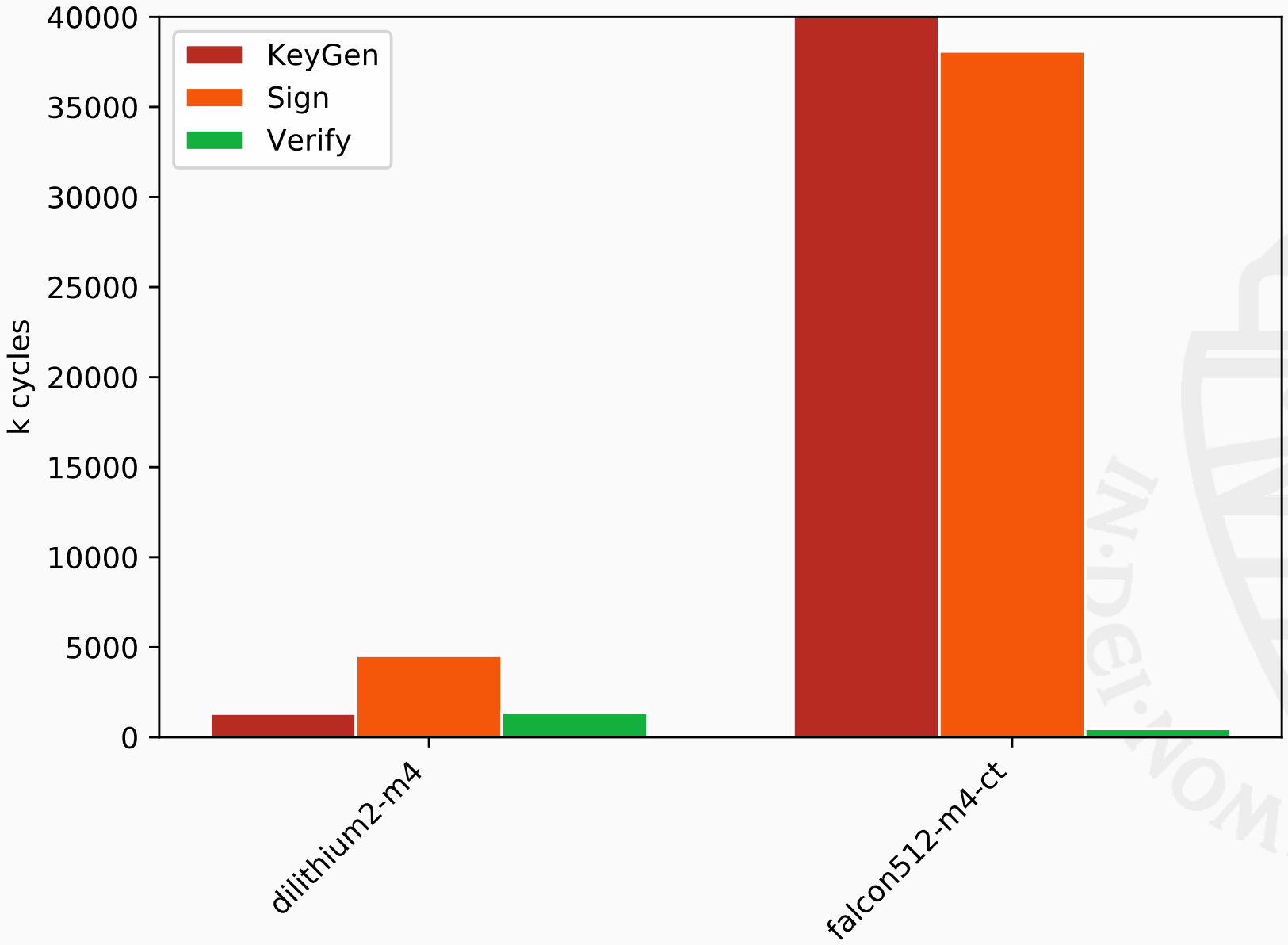
KEM Speed (2)


2000
1750
1500
1250
1000
750
500
250
0
k cycles
KeyGen
Encaps
Decaps
babybear-opt
frodokem640shake-m4
kyber512-m4
lightsaber-m4
newhope512cca-m4
ntruhps2048509-m4
ntruhrss701-m4
r5nd-1kemcca-0d-m4
r5nd-1kemcca-5d-m4

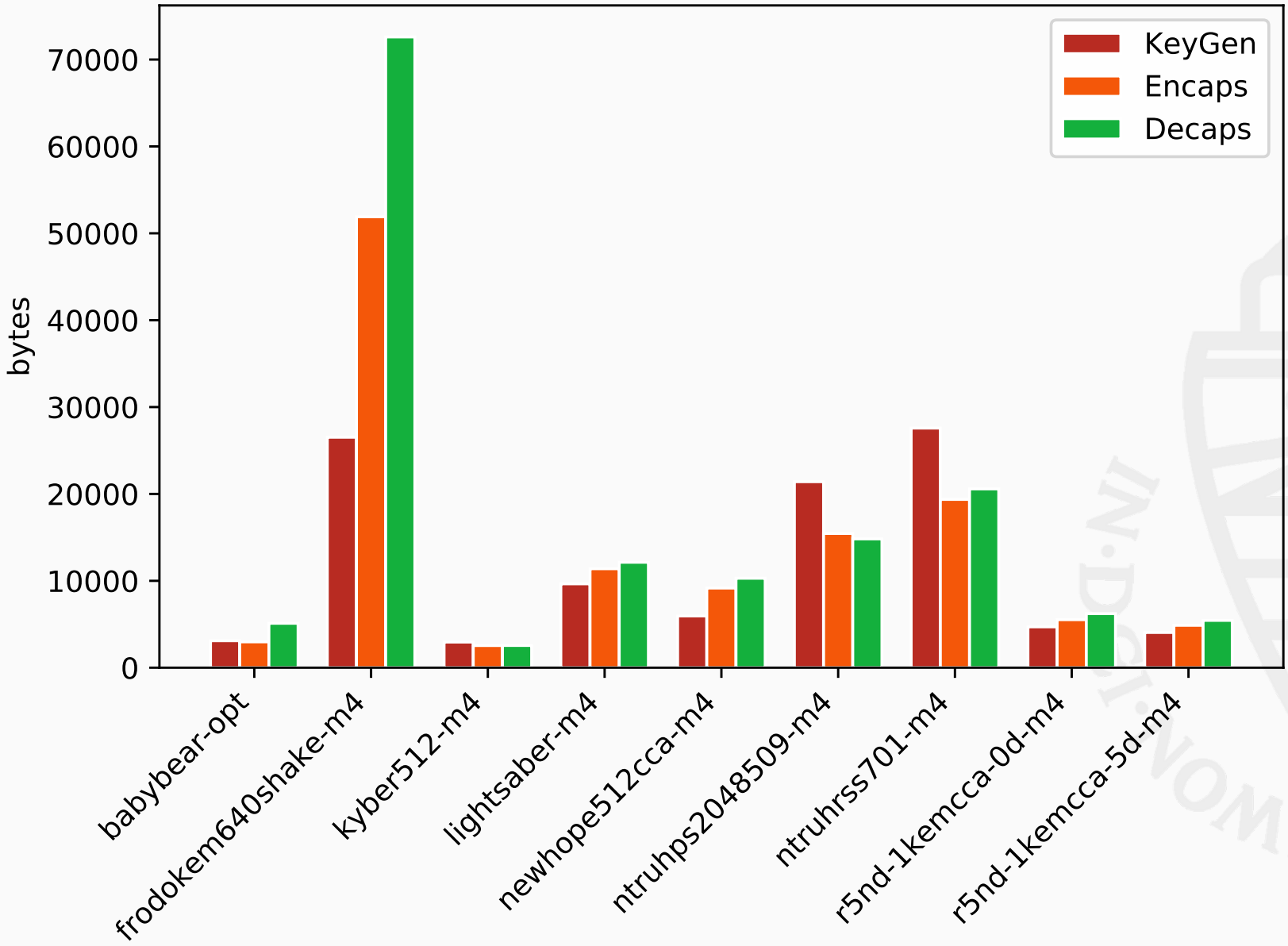
Signature Speed



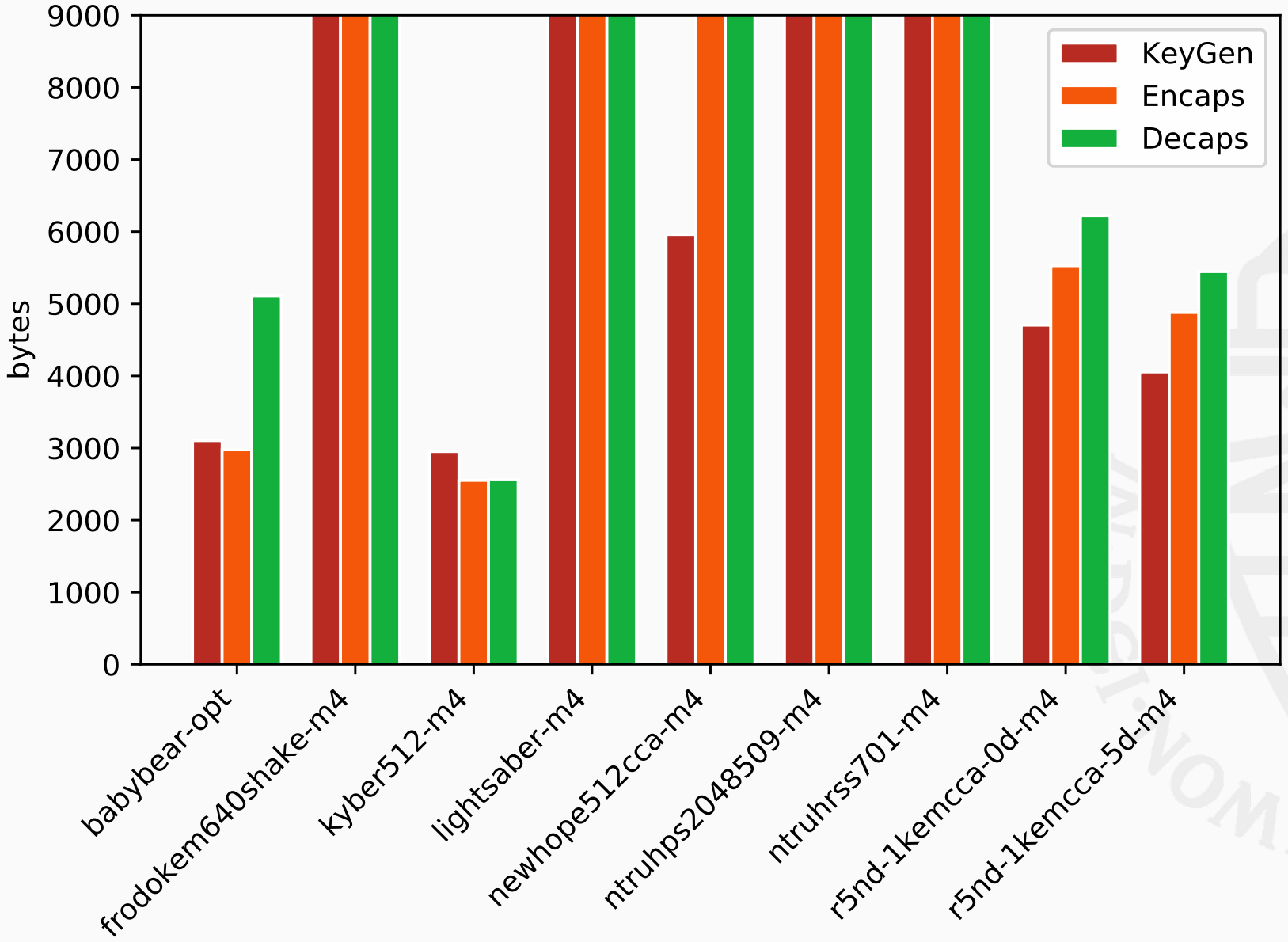
Signature Speed (2)



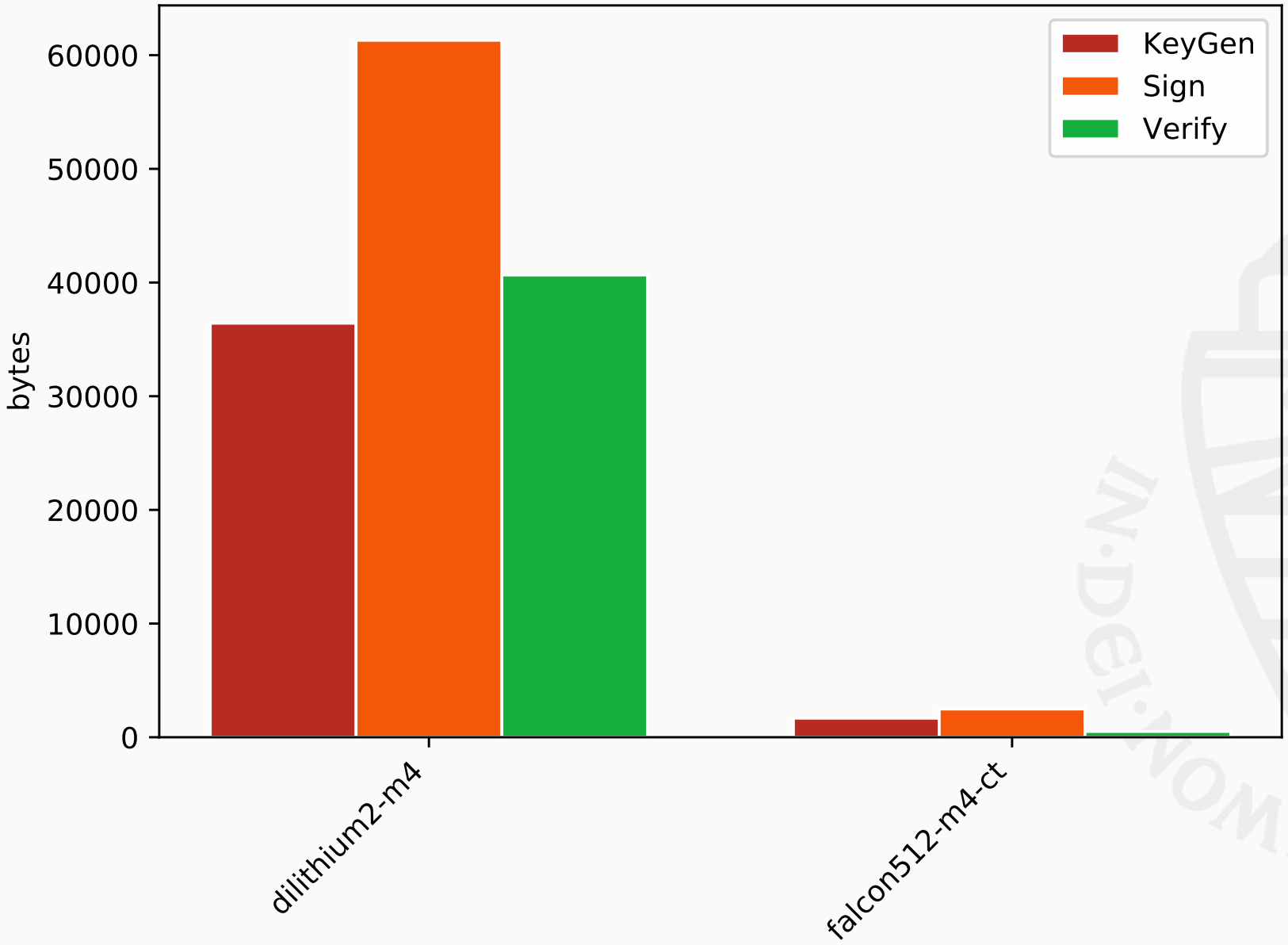
KEM RAM consumption



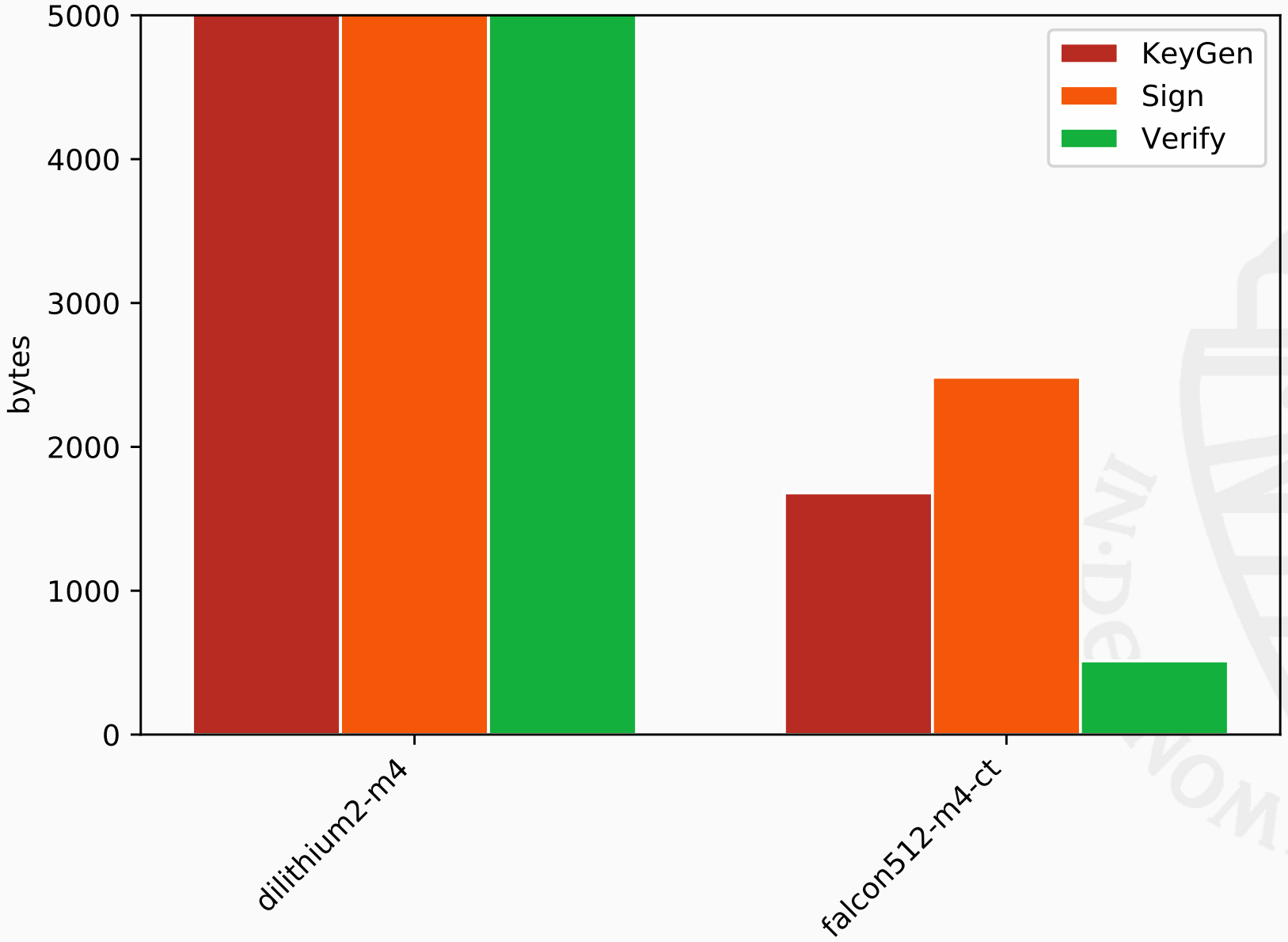
KEM RAM consumption (2)



Signature RAM consumption



Signature RAM consumption (2)



Conclusion

- `pqm4` currently includes
  - 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
- Level of optimization greatly differs
  - Most implementations don't optimize RAM consumption
  - No implementations optimize code size
- Currently, Round5 seems to be the fastest on this platform
  - But it looks like their parameters might have to be increased
  - Kyber, NTRU, Saber, ThreeBears all very close

https://github.com/mupq/pqm4

Slides at https://ko.stoffelen.nl

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`.