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

Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen*

Radboud University, Nijmegen, The Netherlands
matthias@kannwischer.eu, joost@joostrijneveld.nl, peter@cryptojedi.org, k.stoffelen@cs.ru.nl

Abstract. This paper presents pqm4 – a testing and benchmarking framework for the ARM Cortex-M4. It makes use of a widely available discovery board with 196 KiB of memory and 1 MiB flash ROM. It currently includes 10 key encapsulation mechanisms and 5 signature schemes of the NIST PQC competition. For the remaining 11 schemes, the available implementations do require more than the available memory or they depend on external libraries which makes them arguably unsuitable for embedded devices.

1 Introduction

Since NIST started its project to solicit, evaluate, and standardize post-quantum public-key cryptographic algorithms, many solutions have been proposed. One thing that these quite different approaches seem to have in common is that resistance against attacks with quantum computers appears to come at a cost, in particular when compared to traditional discrete logarithm-based elliptic-curve cryptography. For example, algorithms may have larger keys or may require more computational effort.

It is then interesting to wonder how feasible it is to deploy post-quantum cryptography (PQC) on small environments such as those of microcontrollers. Microcontrollers may be small, but they still may need to secure high-value assets for a long period of time, such that post-quantum cryptography is also relevant there. The pqm4 framework investigates the feasibility and performance of the proposed PQC solutions on microcontrollers by focusing on a specific scenario.

ARM Cortex-M4 and STM32F4DISCOVERY. We target the 32-bit ARM Cortex-M4 microcontroller which implements the ARMv7E-M instruction set. It comes with a floating point unit, but none of the cryptographic schemes currently in pqm4 make use of that. Since ARMv7E-M comes with a number of advanced instructions like the DSP instructions, the Cortex-M4 can be considered at the higher end of microcontrollers. Usually, widely available ARM Cortex-M4 development board are also equipped with a large RAM and flash memory. While there are clearly a number of use cases that target more constrained ARM architectures, targeting a higher-end microcontroller allows to run a larger number of NIST PQC schemes and to evaluate which ones may also work on devices which are more constrained in terms of RAM and code size. Therefore, we decided to use the ARM Cortex-M4 for pqm4. NIST in the meantime acknowledged the wide-spread use of the Cortex-M4 in academic post-quantum literature and recommended it to submission teams as an optimization target for the second round. In our experiments we use the widely and cheaply available discovery board STM32F4DISCOVERY which comes with 196 KiB of RAM and 1 MiB of flash ROM.

Availability of Software. We place all the code that was written for pqm4 in the public domain and make it available at https://github.com/mupq/pqm4. The results presented in this paper represent a snapshot of the results available in pqm4. For more recent results, we refer to the tables on the GitHub repository. Since pqm4 includes implementations of NIST PQC schemes as well as other third-party library code (e.g., for hashing), those are covered by their own licenses. We refer to the respective directories in pqm4 for licensing information.

Covered schemes. We require that all parts of the cryptographic scheme work on the target device. We do not implement partial schemes, i.e., schemes of which only a subset of routines can run on the device. We cover all round-2 candidates that can run on the device and for which there exists an implementation that can be integrated into pqm4. This means that it should not depend on many external libraries.

* This work has been supported by the European Commission through the H2020 program under project number ICT-645622 (PQCRYPTO) and the ERC Starting Grant 805031 (EPOQUE). Date: July 21, 2019

Related projects. pqm4 integrates pqclean which is a collection of C reference implementations satisfying a number of basic code quality requirements. Reference implementations in pqm4 are to be replaced by implementations in pqclean on the long term. Additionally, pqm4 is related to pqriscv which aims to provide similar functionality, but for RISC-V microcontrollers in the future. pqm4 and pqriscv share the same common code base, but a core difference is that pqm4 obtains performance results on a physical device whereas pqriscv uses a cycle-accurate simulator.

2 pqm4: The Framework

2.1 Repository Structure

The pqm4 is split into two parts. Everything that is not specific to the ARM Cortex-M4 is abstracted away into the mupq repository, which is a submodule of the pqm4 repository. mupq contains a simple hardware abstraction layer that declares functions to do some kind of setup, to somehow print text, and to somehow get a cycle count or time. mupq also defines an API to get random bytes and to use functions related to AES, SHA-2, SHA-3, SHAKE, and cSHAKE. Finally, mupq imports all implementations from the pqclean project by including it as a submodule. This means that schemes that are added to pqclean can be benchmarked automatically within pqm4.

Everything that is specific to the STM32F4 discovery board resides in the pqm4 repository. We use the libopencm3[1] library to make it easier to setup the board, to print text over a serial port, to read cycle counters, and to use the onboard hardware RNG to generate random bytes. pqm4 also contains optimized assembly implementations of AES and of the Keccak-$f[1600]$ permutation that is used in SHA-3, in SHAKE, and in cSHAKE. All schemes in the framework use these optimized implementations automatically. This includes the schemes that are imported from pqclean.

The pqriscv repository contains work in progress that uses the same mupq back-end for a benchmarking framework targeting the RISC-V architecture. It is not further discussed here.

2.2 Cryptographic Schemes

The pqm4 framework distinguishes between four types of implementations of cryptographic schemes.

- clean: a clean reference implementation from pqclean.
- ref: the reference implementation submitted to NIST. This should be an implementation in plain C. ref implementations reside in the shared mupq repository. If a scheme has a clean implementation, its ref counterpart gets removed.
- opt: an optimized implementation in plain C. For example, the optimized implementation submitted to NIST. opt implementations reside in the shared mupq repository.
- m4: an implementation optimized for the Cortex-M4. This typically means that it contains assembly code. m4 implementations reside the pqm4 repository.

At the moment only KEMs and signature schemes are supported. Naturally we use the same API as was adopted by NIST.

2.3 Features of pqm4

pqm4 provides simple Python scripts to interact with the implementations of cryptographic schemes. We discuss the most important features.

Testing. We perform basic functionality testing. For KEMs, we validate that both parties obtain the same shared secret. We also check that key exchanges where the private key of the initiator or the ciphertext are replaced by random bytes do not lead to the same shared secret. For signature schemes, we check that a signature validates under the correct public key and that it does not validate under a random public key.

In addition to these tests, we verify that the key and message sizes specified by an implementation are large enough by checking that the implementations do not write out of bounds. We do this by adding 8-byte canary values before and after the relevant variables and by checking that they are not overwritten. Of course, this is limited in that it only detects writes to memory addresses that are directly adjacent.

[1] https://libopencm3.org/

Test Vector Comparison. We generate reproducible test vectors for all implementations. We do this by replacing the function that is used to generate fresh random bytes by a deterministic variant. As that is the only source of entropy for the implementations, they should then be completely deterministic as well. We use the test vectors to see whether different implementations of the same instance of the same cryptographic scheme actually result in the same values.

For `clean`, `ref`, and `opt` implementations we generate test vectors on both the STM32F4 discovery board and on the host machine. This is useful to check whether an implementation produces the same results on a 32-bit machine and on a 64-bit machine, for example.

Speed Benchmarking. We make use of a built-in counter to report cycle counts for key generation, encapsulation, and decapsulation separately. Our platform comes with two cycle counters. Since we have schemes that require tens of billions of clock cycles to complete, the 32-bit DWT cycle counter is not sufficient and we have to use the 24-bit SysTick counter instead. The advantage of this smaller counter is that it can be set to trigger an interrupt when it overflows. Our interrupt handler keeps track of the number of times that an overflow occurs. The overhead that this adds is negligible and it allows us to easily compute the total cycle count.

All benchmarks are performed at 24 MHz such that the CPU can be set to have zero wait states when fetching instructions from the flash memory. This makes our benchmarks independent of caches and of the speed of the memory controller. For everything else the CPU is clocked to 168 MHz for convenience.

Stack Usage. We report the stack consumption for each of the primitives of each cryptographic scheme. Since `pqclean` disallows dynamic memory allocations and static variables, the stack usage for `clean` implementations corresponds to the total memory consumption of the cryptographic schemes. Arrays of constants are placed in flash memory and are, therefore, not included. Naturally, all key material and messages need to be stored on the stack, but since those need to be allocated outside of the actual scheme, we do not include them in the stack usage. We measure the stack usage by writing a random canary to the entire available stack space, then running the primitive, and finally checking which parts of the memory have been overwritten.

We cannot use the entire 196 KiB of RAM as stack, as it is divided over several distinct memories. There is 112 KiB of SRAM1, 16 KiB of SRAM2, 64 KiB of core-coupled memory, and 4 KiB of backup SRAM. While SRAM1 and SRAM2 can be used as contiguous block, we found that accessing both can cause wait cycles and hence we limited the amount of available memory to 112 KiB such that the complete stack can be mapped to SRAM1. `pqm4` supports custom linker scripts for implementations as a means to override this behavior. If these 16 KiB make the difference for a scheme, an implementation can specify to use 128 KiB of RAM in its linker script. Currently some parameter sets of FrodoKEM, Round5, and Dilithium do this to make it fit.

Hashing. Most NIST PQC schemes utilize hash function and block ciphers for various purposes throughout the algorithms. Usually, a large percentage of run-time is spent in hashing. For fair comparison, we use the same AES, SHA-2, SHA-3, SHAKE, and cSHAKE implementation for all schemes. Additionally, we report time spent in hashing for each scheme which gives an indication how much a certain scheme would benefit from a hardware-accelerated implementation of those primitives or the use of a more lightweight alternative.

Code Size. We report the code size for each particular implementation. We exclude the code that is the same for all schemes, i.e., hashing, calling code, standard library functions, and code that is required to set up the board and communicate with the host. The sizes of the `text`, `bss`, and `data` sections are reported separately.

3 Implementations currently in PQM4

3.1 Key Encapsulation Mechanisms

FrodoKEM. The reference implementation [NAB+19] of the smallest parameter set (frodokem640shake/ frodokem640aes) of FrodoKEM requires almost a megabyte of RAM (including messages and keys). This is mainly due to placing the entire matrix A in RAM. The larger parameter sets require even more memory. Therefore, the reference implementations are not suitable for the target platform of `pqm4`. The optimized implementations of the submission package [NAB+19] reduce memory consumption by sampling A on-the-fly.

Still the memory footprint remains large, and only frodokem640shake fits on our target platform consuming 117 KiB of RAM (including messages and keys)[2]. The implementations of the AES parameter sets of FrodoKEM use more memory than their SHAKE counterparts and frodokem640aes is already to large to fit in our available SRAM consuming 141 KiB. The larger parameter sets of FrodoKEM consume between 181 KiB (frodokem976shake) and 298 KiB (frodokem1344aes) and, thus, exceed our memory limits by far. `pqm4` also includes M4-optimized assembly implementations for frodokem640shake and frodokem640aes by Bos–Friedberger–Martinoli–Oswald–Stam [BFM+18].

Kyber. Kyber [SAB+19] specifies the three parameter sets kyber512, kyber768, and kyber1024. For all three parameter sets, `pqm4` includes the corresponding clean implementation from `pqclean` and the recent stack- and speed-optimized implementation by Botros–Kannwischer–Schwabe [BKS19]. It consists of a hand-written assembly implementation of the number-theoretic transform (NTT) and various changes to the reference C implementation to reduce the memory footprint. We focus on the schemes using SHAKE and leave the Kyber '90s' variants for future work.

LAC. `pqm4` integrates the reference implementations [LLJ+19] of lac128, lac192, and lac256.

NTRU. `pqm4` includes the four parameter sets of the merger NTRU: ntruhps2048509, ntruhps2048677, ntruhps4096821, and ntruhrss701. In addition to the reference implementations from [ZCH+19], we include optimized implementations by Kannwischer–Rijneveld–Schwabe [KRS19]. Those replace the slow schoolbook multiplication by optimized assembly polynomial multiplication. The remainder of the implementation is not modified, which, consequently, means that parts (especially in key generation) could be further optimized.

NTRU Prime. The submission document of NTRU Prime [BCLv19] contains three parameter sets for Streamlined NTRU Prime and three for NTRU LPRime. `pqm4` contains the reference implementations for all these parameter sets.

NewHope. For all parameter sets of NewHope, i.e., newhope512cpa, newhope512cca, newhope1024cpa, and newhope1024cca, we include the reference implementations of the submission package [PAA+19]. For the 1024-coefficient parameter sets, a hand-optimized assembly implementation of the NTT exists in previous work by Alkim–Jakubeit–Schwabe [AJS16] on the original NewHope key exchange exists. In `pqm4`, we use this implementation to replace the NTT in the reference implementation from the submission package [PAA+19]. Note that various other parts in the implementation of NewHope could benefit from assembly optimizations which needs to be considered when comparing NewHope's performance to other schemes. We leave the optimization of those to future work.

Round5. The Round5 implementations that are submitted to NIST [GZB+19] depend on the OpenSSL library, which makes them harder to integrate them into `pqm4`. Additional "portable, self-contained" implementations were later released on GitHub[3]. These implementations are optionally optimized with `ARMv7-M` assembly. We integrated these into `pqm4` as `opt` (disabling the assembly code) and `m4` (with the assembly code) implementations. Round5 provides CPA-secure KEMs and CCA-secure PKE algorithms. As `pqm4` only supports KEMs, one of the Round5 submitters suggested to use the CCA-secure KEM that underlies the CCA-secure PKE. This means that we can remove the DEM and the authentication tag in the ciphertext. We rename the parameter sets to `kemcca` to emphasize this technical difference with the schemes submitted to NIST. Round5 specifies 9 regular parameter sets and 3 more for specific use cases. `pqm4` does not include the parameter sets for special scenarios, but it does include the (unofficial) Round5 variants that use SNEIK instead of SHAKE [Saa19]. In our Cortex-M4 benchmarks the implementations perform slightly better than, but very close to, what the submitters already benchmarked themselves.

Saber. Reference implementations of all parameter sets of Saber (lightsaber, saber, and firesaber) have been integrated into `pqm4`. Since all three rely on arithmetic in the ring $\mathbb{Z}_{2^{13}}[X]/(X^{256}+1)$, we also include optimized M4 implementations for all parameter sets using optimized polynomial multiplication by Kannwischer–Rijneveld–Schwabe [KRS19]. Note that this work is partially based upon an earlier paper by Karmakar–Bermudo Mera–Sinha Roy–Verbauwhede [KBMSRV18] which also presents stack-optimized implementations which are not included in `pqm4`.

[2] Note that this slightly exceeds the size of SRAM1 of 112 KiB. Therefore, we use the concatenation of SRAM1 and SRAM2 which might impact comparability of clock cycles slightly.

[3] https://github.com/r5embed/r5embed

SIKE. The reference implementations of SIKE uses the GMP library for integer arithmetic and as such cannot simply be integrated into `pqm4`. However, the optimized C implementations that are also part of the submission package [JAC+19] re-implement this arithmetic using pure C and optimize for specific SIKE parameters. Those implementations of sikep434, sikep503, sikep610, and sikep751 have been integrated into `pqm4`. Additionally, recent work by Seo–Jalali–Azarderakhsh [SJA19] further optimizes arithmetic in those schemes using `ARMv7E-M` assembly. We will integrate this work into `pqm4` once the source code is available to us.

ThreeBears. ThreeBears specifies [Ham19] CPA-secure ephemeral and CCA-secure variants of 3 recommended parameter sets and of 2 additional sets with toy parameters that could be interesting for cryptanalysis. Both reference and optimized pure-C implementations of all recommended parameter sets are integrated into `pqm4`. In our Cortex-M4 benchmarks the implementations perform slightly better than, but very close to, what the submitter already described in the ThreeBears documentation [Ham19].

3.2 Signature Schemes

Dilithium. `pqm4` includes the reference implementations of dilithium2, dilithium3, and dilithium4. We do not include the "weak" parameter set diltihium1 since it is below NIST security level 1. In addition to the reference implementations, we integrate M4-optimized implementations by Güneysu–Krausz–Oder–Speith [GKOS18] which provides an assembly implementation of the NTT.

Falcon. The implementations of Falcon that are submitted to NIST [PFH+19] dynamically allocate a lot of memory. In particular, during the signing phase of Falcon1024 over 200 KiB get allocated. This causes those implementations not to fit into `pqm4`. Work by Oder–Speith–Höltgen–Güneysu [OSHG19] describes techniques to significantly reduce the memory consumption of Falcon. However, their implementation is not compatible with the second round version of Falcon. Recently, Thomas Pornin of the Falcon team provided us with new implementations of Falcon512 and Falcon1024 targeted at microcontrollers. These implementations have been integrated into `pqm4`. In addition, he provided variants Falcon512-Tree and Falcon1024-Tree in which the Falcon LDL tree is computed as part of the key generation and returned as private key. This speeds up signature generation, but also greatly enlarges the private key size. Unfortunately, Falcon1024-Tree does not fit into 128 KiB of RAM and only Falcon512-Tree could be added to `pqm4`. Since Falcon heavily relies on 64-bit floating point arithmetic which is not natively available on our platform, one has to emulate it using 32-bit arithmetic. When leaving this emulation to the compiler, the resulting implementation is heavily non constant-time. We include this implementation as "opt-leaktime". We also integrate constant-time variants which do the emulation in either plain C ("opt-ct") or assembly ("m4-ct").

LUOV. LUOV is a signature scheme with 6 different parameter sets [BPSV19], each with 2 PRNG choices. For LUOV-8-58-237, LUOV-48-43-222, and LUOV-64-61-302, `pqm4` contains the reference implementations submitted to NIST. The implementations of the remaining parameter sets require too much memory to fit into `pqm4`. We measured 155, 270, and 199 KiB for LUOV-8-82-323, LUOV-8-107-371, and LUOV-80-76-363, respectively. A different implementation might still make these parameter sets work.

qTESLA. The second round specification of qTesla [BAA+19] describes 7 different parameter sets: qTesla-I, qTesla-II, qTesla-III, qTesla-V, qTesla-V-size, qTesla-p-I, and qTesla-p-III. The reference implementations of the parameter sets qTesla-p-I and qTesla-p-III, which are claimed to be provably-secure, require 174 KiB and 403 KiB of RAM respectively and are, thus, not suitable for our target platform. The reference implementations of the remaining five parameter sets have been included in `pqm4`.

SPHINCS$^+$. All 36 parameter sets of SPHINCS$^+$ [HBD+19] have been integrated into `pqm4`. However, with signing times of 22 seconds to 88 minutes (at 24 MHz) those schemes are arguably not suitable for the target platform.

4 Remaining NIST PQC schemes

Several schemes are not yet or cannot be integrated into `pqm4`. One common reason is that the public key is too large by construction. These schemes are arguably unsuited for a microcontroller environment of this

size. Another common reason is that the implementation demands too much memory or depends on external libraries. In this case, a different implementation, written with microcontrollers in mind, might still rescue the scheme. Some schemes are simply still work in progress.

For each scheme that is still remaining, we briefly discuss the current status and its reason for not yet being included.

4.1 Key Encapsulation Mechanisms

BIKE. For BIKE it is not immediately obvious whether the scheme will fit into our platform. One complicating factor is that the current implementations [ABB+19] submitted to NIST depend on external libraries, such as OpenSSL and NTL, for multi-precision and finite-field arithmetic. This makes it more difficult to integrate the code into `pqm4`.

Classic McEliece. The specification of Classic McEliece [BCL+19] proposes 5 parameter sets and lists another 5 as possible future proposals. Their public key sizes range from 255 KiB to 1 326 KiB. These are too large to fit into the memory of our platform.

HQC. It is not immediately clear whether any parameter set of HQC would fit on our platform. The implementations provided by the submitters [AAB+19a] use the OpenSSL, NTL, and GMP external libraries, which makes it hard to integrate HQC into `pqm4`. There is work in progress on a standalone HQC implementation for `pqclean`[4]. We will integrate this into `pqm4` once it is there if possible.

LEDAcrypt. For LEDAcrypt it is not immediately clear whether the proposed parameter sets [BBC+19] would fit into `pqm4`. The reference implementations of the long-term CCA-secure version of LEDAcrypt have been integrated into `pqclean`. Their memory usage ranges from 853 KiB to 2 523 KiB, which is too much for `pqm4`. There is some work in progress to reduce the RAM usage, but at the time of writing we are not aware of any implementation fitting in 128 KiB of RAM.

NTS-KEM. The submitters of NTS-KEM recommend 3 parameter sets [ACP+19]. The public key sizes of NTS-KEM(12,64), NTS-KEM(13,80), and NTS-KEM(13,136) are 312.0, 908.0, and 1 386 KiB large, respectively. These are clearly too large for `pqm4`.

ROLLO. It is unclear to us whether any parameter set of ROLLO would fit into `pqm4`. The implementations that are submitted to NIST [ABD+19] are hard to integrate, because they depend on the OpenSSL, NTL, and GMP external libraries. Recently Lablanche–Mortajine–Benchaalal–Cayrel–El Mrabet posted a paper describing an implementation of ROLLO on an ARM SecurCore SC300 core [LMB+19]. The authors leverage the crypto co-processor implementing finite field arithmetic to speed-up ROLLO. Our platform does not feature a crypto co-processor, but it should be possible to adapt their implementation using alternative software-only multiplication.

RQC. What has been said for HQC and for ROLLO also holds for RQC. The implementations submitted to NIST [AAB+19b] use the OpenSSL, NTL, and GMP libraries, which makes them unsuited to integrate into `pqm4` to test if the scheme would fit at all.

4.2 Signature Schemes

GeMSS. The GeMSS specification [CFM+19] describes 9 parameter sets. Their public key sizes range from 343.9 KiB to 3 062.1 KiB. These are clearly too large to ever fit into the amount of memory that is available to us in `pqm4`.

MQDSS. For MQDSS, 2 recommended parameter sets are provided and several additional ones [SCH+19]. Only the recommended parameter sets are currently implemented. We measured the total stack usage of their reference implementations. MQDSS-31-48 takes 196.5 KiB and MQDSS-31-64 413.6 KiB. This is too much for `pqm4`.

[4] https://github.com/PQClean/PQClean/pull/202

Picnic. The submitters of Picnic [ZCD+19] used the Valgrind software to profile the static and dynamic memory usage of their implementations. They measured the peak total memory usage for both reference and optimized implementations for all 9 parameter sets. For signing a message, the total ranges from 133.0 KiB to 32.9 MiB. Therefore there currently exists no Picnic implementation that would fit into `pqm4`.

Rainbow. The Rainbow specification [DCP+19] proposes 3 Rainbow variants, each with 3 different parameter sets. For standard Rainbow Ia, IIIc, and Vc, the public keys are 145.5, 693.9, and 1 665.5 KiB large, respectively. These keys are too large to fit into the memory of this platform. For both cyclic and compressed Rainbow Ia, IIIc, and Vc, the public keys are 56.7, 201.9, and 480.3 KiB. Only the first parameter set could possible fit. However, together with a private key of 90.8 KiB and other variables, cyclic Rainbow Ia and its compressed variant still do not fit into memory.

5 Results

In this section we summarize the current results. It should be noted these numbers may change due to new improved implementations or due to improved new compiler versions. Hence we refer to the GitHub repository for the most recent numbers.

It should also be noted that these numbers should not be used directly for comparison, but merely to give an idea of an order of magnitude to see how practical a scheme could be in a microcontroller environment. There are many aspects that we do not consider that should be taken into account for a proper comparison, including, but not limited to, different NIST security levels, different computational assumptions, different security margins, constant-time versus non-constant-time implementations, and the fact that a reference implementation is explicitly not intended to be optimized.

5.1 Speed

Speed is measured in CPU clock cycles. We measure 100 executions per scheme and list the average value, the exceptions being the parameter sets of SIKE and SPHINCS+, for which we measured only a single execution for practical reasons. The GitHub repository also contains minimum and maximum values. For most schemes the cycle counts are very stable. Only the speed of schemes that use some form of rejection sampling varies widely.

Key Encapsulation Schemes

Scheme	Type	Key Generation	Encapsulation	Decapsulation
firesaber	m4	1 448 776	1 786 930	1 853 339
frodokem640aes	m4	47 050 559	45 883 334	45 366 065
frodokem640shake	m4	79 331 373	79 745 404	79 231 474
kyber1024	m4	1 575 052	1 779 848	1 709 348
kyber512	m4	514 291	652 769	621 245
kyber768	m4	976 757	1 146 556	1 094 849
lightsaber	m4	459 965	651 273	678 810
newhope1024cca	m4	1 219 908	1 903 231	1 927 505
newhope1024cpa	m4	1 034 955	1 495 457	206 112
ntruhps2048509	m4	77 698 713	645 329	542 439
ntruhps2048677	m4	144 383 491	955 902	836 959
ntruhps4096821	m4	211 758 452	1 205 662	1 066 879
ntruhrss701	m4	154 676 705	402 784	890 231
r5n1-1kemcca-0d	m4	5 553 096	4 437 003	5 279 762
r5n1-1kemcca-0d-sneik	m4	4 690 949	3 270 847	3 650 559
r5n1-3kemcca-0d	m4	8 891 809	7 424 266	8 449 118
r5n1-3kemcca-0d-sneik	m4	7 809 290	6 593 661	7 101 979

Scheme	Type	Key Generation	Encapsulation	Decapsulation
r5n1-5kemcca-0d	m4	32 784 752	21 838 175	25 468 525
r5n1-5kemcca-0d-sneik	m4	30 314 113	19 777 686	21 557 440
r5nd-1kemcca-0d	m4	304 296	456 780	540 076
r5nd-1kemcca-0d-sneik	m4	194 377	294 350	348 016
r5nd-1kemcca-5d	m4	341 725	541 305	724 861
r5nd-1kemcca-5d-sneik	m4	234 928	396 238	529 570
r5nd-3kemcca-0d	m4	1 014 871	1 443 502	1 876 446
r5nd-3kemcca-0d-sneik	m4	781 124	1 240 140	1 545 374
r5nd-3kemcca-5d	m4	674 743	1 011 757	1 303 079
r5nd-3kemcca-5d-sneik	m4	488 951	854 551	1 069 093
r5nd-5kemcca-0d	m4	1 356 508	1 945 123	2 498 516
r5nd-5kemcca-0d-sneik	m4	1 123 751	1 753 377	2 182 486
r5nd-5kemcca-5d	m4	1 228 319	1 782 156	2 332 469
r5nd-5kemcca-5d-sneik	m4	1 002 690	1 581 058	2 007 861
saber	m4	896 035	1 161 849	1 204 633
babybear	opt	596 665	752 117	1 142 773
babybear	ref	3 968 931	5 802 693	11 704 641
babybear-ephem	opt	596 664	767 860	231 728
babybear-ephem	ref	3 968 935	5 817 861	1 948 755
firesaber	ref	3 815 672	4 745 405	5 402 295
frodokem640shake	opt	91 087 806	105 377 455	105 133 550
kyber1024	clean	1 891 737	2 254 703	2 407 858
kyber512	clean	649 678	884 848	985 258
kyber768	clean	1 196 692	1 489 909	1 613 744
lac128	ref	2 266 368	3 979 851	6 303 717
lac192	ref	7 532 180	9 986 506	17 452 435
lac256	ref	7 665 769	13 533 851	21 125 257
lightsaber	ref	1 051 530	1 538 646	1 861 934
mamabear	opt	1 195 048	1 402 955	1 955 496
mamabear	ref	8 774 109	11 501 206	23 131 816
mamabear-ephem	opt	1 206 350	1 431 341	320 233
mamabear-ephem	ref	8 783 565	11 525 870	2 876 105
newhope1024cca	clean	1 460 167	2 264 773	2 410 906
newhope1024cpa	clean	1 274 813	1 857 055	327 778
newhope512cca	clean	719 785	1 134 083	1 192 460
newhope512cpa	clean	628 310	915 293	163 222
ntruhps2048509	clean	107 331 110	2 876 597	7 495 670
ntruhps2048677	clean	196 228 850	4 909 724	13 157 491
ntruhps4096821	clean	289 736 570	7 046 106	19 262 764
ntruhrss701	clean	208 992 073	4 656 818	14 142 725
ntrulpr653	ref	54 824 768	109 094 505	163 062 035
ntrulpr761	ref	74 265 583	147 846 761	221 088 122
ntrulpr857	ref	94 016 969	187 235 730	280 075 965
papabear	opt	2 014 216	2 276 138	3 000 239
papabear	ref	15 478 896	19 098 191	38 361 009
papabear-ephem	opt	2 029 284	2 308 148	411 623
papabear-ephem	ref	15 490 269	19 126 020	3 803 444
r5n1-1kemcca-0d	opt	5 388 610	5 692 748	6 535 000
r5n1-1kemcca-0d-sneik	opt	4 616 510	4 756 084	5 174 603
r5n1-3kemcca-0d	opt	8 614 772	9 366 074	10 390 737
r5n1-3kemcca-0d-sneik	opt	7 650 296	9 115 311	9 670 411

Scheme	Type	Key Generation	Encapsulation	Decapsulation
r5n1-5kemcca-0d	opt	31 668 323	30 323 237	33 960 444
r5n1-5kemcca-0d-sneik	opt	29 524 911	29 343 146	31 355 544
r5nd-1kemcca-0d	opt	385 047	553 492	652 222
r5nd-1kemcca-0d-sneik	opt	289 227	428 528	497 356
r5nd-1kemcca-5d	opt	441 336	715 940	974 817
r5nd-1kemcca-5d-sneik	opt	347 346	600 114	812 236
r5nd-3kemcca-0d	opt	1 390 259	2 014 897	2 643 688
r5nd-3kemcca-0d-sneik	opt	1 184 997	1 900 960	2 413 390
r5nd-3kemcca-5d	opt	903 289	1 390 057	1 830 592
r5nd-3kemcca-5d-sneik	opt	741 196	1 302 493	1 673 018
r5nd-5kemcca-0d	opt	1 903 143	2 782 242	3 627 662
r5nd-5kemcca-0d-sneik	opt	1 711 895	2 712 382	3 447 723
r5nd-5kemcca-5d	opt	1 710 318	2 571 737	3 426 802
r5nd-5kemcca-5d-sneik	opt	1 526 381	2 468 323	3 215 298
saber	ref	2 226 935	2 936 240	3 424 601
sikep434	opt	650 735 516	1 065 631 547	1 136 703 605
sikep503	opt	985 032 805	1 623 893 046	1 726 538 272
sikep610	opt	1 819 652 559	3 348 669 891	3 368 114 366
sikep751	opt	3 296 225 272	5 347 056 677	5 742 522 048
sntrup653	ref	566 054 965	54 942 173	166 481 625
sntrup761	ref	757 107 092	74 398 441	225 554 771
sntrup857	ref	961 965 366	94 154 502	286 203 168

Signature Schemes

Scheme	Type	Key Generation	Sign	Verify
dilithium2	m4	1 400 412	6 157 001	1 461 284
dilithium3	m4	2 282 485	9 289 499	2 228 898
dilithium4	m4	3 097 421	8 468 805	3 173 500
falcon1024	m4-ct	480 910 965	83 482 883	977 140
falcon512	m4-ct	197 793 925	38 090 446	474 052
falcon512-tree	m4-ct	201 459 670	17 181 744	475 278
dilithium2	clean	1 752 194	9 342 087	2 035 881
dilithium3	clean	2 733 423	14 885 750	2 946 998
dilithium4	clean	3 647 486	13 615 651	4 035 259
falcon1024	opt-ct	690 147 063	136 596 407	978 558
falcon1024	opt-leaktime	446 074 512	72 915 615	978 401
falcon512	opt-ct	229 088 624	62 225 400	473 964
falcon512	opt-leaktime	166 773 689	33 709 488	475 227
falcon512-tree	opt-ct	267 136 946	27 257 531	474 934
falcon512-tree	opt-leaktime	187 864 184	17 778 541	473 928
luov-48-43-222-chacha	ref	41 347 565	123 878 410	95 330 045
luov-48-43-222-keccak	ref	53 900 163	135 960 322	107 793 279
luov-64-61-302-chacha	ref	109 063 798	405 205 796	269 012 028
luov-64-61-302-keccak	ref	148 680 619	441 086 753	318 677 042
luov-8-58-237-chacha	ref	66 072 054	101 874 776	77 433 705
luov-8-58-237-keccak	ref	87 785 091	119 711 856	81 240 915
qTesla-I	ref	6 748 008	5 830 914	787 773

Scheme	Type	Key Generation	Sign	Verify
qTesla-II	ref	111 498 880	18 734 861	3 227 979
qTesla-III	ref	19 834 717	6 846 333	1 651 066
qTesla-V	ref	114 962 361	24 436 890	3 736 553
qTesla-V-size	ref	2 648 199 757	42 516 447	7 102 967
sphincs-haraka-128f-robust	clean	125 493 736	4 720 156 509	196 103 540
sphincs-haraka-128f-simple	clean	91 770 060	3 286 409 908	136 522 968
sphincs-haraka-128s-robust	clean	3 994 078 037	76 355 806 675	90 503 982
sphincs-haraka-128s-simple	clean	2 913 690 564	52 263 758 103	62 469 217
sphincs-haraka-192f-robust	clean	186 298 605	5 628 829 365	312 841 777
sphincs-haraka-192f-simple	clean	135 187 443	3 878 154 505	216 987 235
sphincs-haraka-192s-robust	clean	5 940 452 620	5 940 452 620	190 010 600 183
sphincs-haraka-192s-simple	clean	4 303 035 491	115 536 324 243	86 699 644
sphincs-haraka-256f-robust	clean	498 410 260	14 072 875 513	333 271 171
sphincs-haraka-256f-simple	clean	359 364 375	9 598 675 448	231 877 653
sphincs-haraka-256s-robust	clean	7 963 824 050	126 866 112 920	182 132 900
sphincs-haraka-256s-simple	clean	5 738 088 144	84 936 048 049	122 757 191
sphincs-sha256-128f-robust	clean	31 756 635	952 977 815	42 386 275
sphincs-sha256-128f-simple	clean	16 552 135	521 963 206	20 850 719
sphincs-sha256-128s-robust	clean	1 016 232 559	14 062 448 255	17 573 655
sphincs-sha256-128s-simple	clean	529 857 276	7 879 641 509	9 001 108
sphincs-sha256-192f-robust	clean	46 992 024	1 301 607 186	70 469 375
sphincs-sha256-192f-simple	clean	24 355 501	687 693 467	35 097 457
sphincs-sha256-192s-robust	clean	1 503 938 303	34 385 423 488	26 669 148
sphincs-sha256-192s-simple	clean	779 440 632	19 040 306 594	13 893 351
sphincs-sha256-256f-robust	clean	172 077 921	4 015 978 844	99 321 414
sphincs-sha256-256f-simple	clean	64 184 968	1 554 168 401	36 182 488
sphincs-sha256-256s-robust	clean	2 752 951 213	33 843 524 630	51 684 917
sphincs-sha256-256s-simple	clean	1 026 709 354	13 261 223 456	18 016 773
sphincs-shake256-128f-robust	clean	125 414 981	3 768 632 975	170 079 038
sphincs-shake256-128f-simple	clean	65 652 221	2 071 760 810	86 340 855
sphincs-shake256-128s-robust	clean	4 014 132 491	55 743 252 284	67 161 474
sphincs-shake256-128s-simple	clean	2 101 290 023	31 171 364 976	36 490 062
sphincs-shake256-192f-robust	clean	183 459 272	4 900 079 285	259 335 291
sphincs-shake256-192f-simple	clean	96 097 954	2 618 217 124	135 957 606
sphincs-shake256-192s-robust	clean	5 871 537 061	113 206 409 590	101 442 435
sphincs-shake256-192s-simple	clean	3 075 539 641	64 888 702 756	52 870 591
sphincs-shake256-256f-robust	clean	484 212 750	10 711 617 658	264 321 983
sphincs-shake256-256f-simple	clean	253 708 675	5 805 228 113	135 723 122
sphincs-shake256-256s-robust	clean	7 746 063 229	88 108 495 856	127 277 532
sphincs-shake256-256s-simple	clean	4 058 075 714	48 363 054 240	64 507 752

5.2 Stack Memory

Stack memory is measured in bytes and excludes the stack space required to store key material and messages as they are allocated outside of the implementations' code.

Key Encapsulation Schemes

Scheme	Type	Key Generation	Encapsulation	Decapsulation
firesaber	m4	20 144	23 008	24 592
frodokem640aes	m4	31 992	62 488	83 112
frodokem640shake	m4	26 528	51 904	72 600
kyber1024	m4	4 360	3 584	3 592
kyber512	m4	2 952	2 552	2 560
kyber768	m4	3 848	3 128	3 072
lightsaber	m4	9 656	11 392	12 136
newhope1024cca	m4	11 152	17 400	19 640
newhope1024cpa	m4	11 128	17 288	8 328
ntruhps2048509	m4	21 412	15 452	14 828
ntruhps2048677	m4	28 524	20 604	19 756
ntruhps4096821	m4	34 532	24 924	23 980
ntruhrss701	m4	27 580	19 372	20 580
r5n1-1kemcca-0d	m4	19 168	24 416	29 672
r5n1-1kemcca-0d-sneik	m4	18 704	24 096	29 352
r5n1-3kemcca-0d	m4	26 576	35 552	44 440
r5n1-3kemcca-0d-sneik	m4	26 128	35 224	44 112
r5n1-5kemcca-0d	m4	40 272	54 696	65 535
r5n1-5kemcca-0d-sneik	m4	39 808	54 368	65 535
r5nd-1kemcca-0d	m4	4 704	5 528	6 224
r5nd-1kemcca-0d-sneik	m4	4 208	5 096	5 792
r5nd-1kemcca-5d	m4	4 055	4 879	5 447
r5nd-1kemcca-5d-sneik	m4	3 560	4 520	5 088
r5nd-3kemcca-0d	m4	6 280	8 024	9 000
r5nd-3kemcca-0d-sneik	m4	5 792	7 640	8 648
r5nd-3kemcca-5d	m4	5 808	7 080	7 960
r5nd-3kemcca-5d-sneik	m4	5 320	6 736	7 616
r5nd-5kemcca-0d	m4	7 760	10 080	11 416
r5nd-5kemcca-0d-sneik	m4	7 272	9 728	11 032
r5nd-5kemcca-5d	m4	7 264	8 872	9 968
r5nd-5kemcca-5d-sneik	m4	6 776	8 520	9 616
saber	m4	13 256	15 544	16 640
babybear	opt	3 104	2 976	5 112
babybear	ref	6 168	6 088	10 192
babybear-ephem	opt	3 104	3 008	2 440
babybear-ephem	ref	6 168	6 088	4 864
firesaber	ref	20 144	23 008	24 488
frodokem640shake	opt	36 672	58 312	78 944
kyber1024	clean	15 224	18 928	20 496
kyber512	clean	6 480	9 168	9 904
kyber768	clean	10 576	13 776	14 864
lac128	ref	2 916	5 116	5 952
lac192	ref	4 344	7 464	8 664
lac256	ref	4 452	8 676	10 116
lightsaber	ref	10 024	11 672	12 504
mamabear	opt	3 592	3 464	6 072
mamabear	ref	6 792	6 712	11 600
mamabear-ephem	opt	3 592	3 488	2 920
mamabear-ephem	ref	6 792	6 712	4 864

Scheme	Type	Key Generation	Encapsulation	Decapsulation
newhope1024cca	clean	11 120	17 400	19 608
newhope1024cpa	clean	11 096	17 288	8 308
newhope512cca	clean	5 992	9 208	10 328
newhope512cpa	clean	5 968	9 128	4 248
ntruhps2048509	clean	11 784	6 896	5 192
ntruhps2048677	clean	15 608	9 104	6 840
ntruhps4096821	clean	18 880	10 992	8 344
ntruhrss701	clean	14 216	7 420	8 612
ntrulpr653	ref	12 204	19 652	23 068
ntrulpr761	ref	14 044	22 492	26 524
ntrulpr857	ref	15 692	25 068	29 628
papabear	opt	4 072	3 944	7 032
papabear	ref	7 416	7 344	13 008
papabear-ephem	opt	4 072	3 968	3 400
papabear-ephem	ref	7 416	7 336	4 864
r5n1-1kemcca-0d	opt	19 168	24 416	29 672
r5n1-1kemcca-0d-sneik	opt	18 856	24 248	29 504
r5n1-3kemcca-0d	opt	26 576	35 552	44 440
r5n1-3kemcca-0d-sneik	opt	26 280	35 376	44 264
r5n1-5kemcca-0d	opt	40 272	54 696	57 343
r5n1-5kemcca-0d-sneik	opt	39 960	54 520	68 840
r5nd-1kemcca-0d	opt	4 704	5 528	6 224
r5nd-1kemcca-0d-sneik	opt	4 360	5 248	5 944
r5nd-1kemcca-5d	opt	4 056	4 880	5 448
r5nd-1kemcca-5d-sneik	opt	3 712	4 672	5 240
r5nd-3kemcca-0d	opt	6 280	7 992	9 000
r5nd-3kemcca-0d-sneik	opt	5 944	7 792	8 800
r5nd-3kemcca-5d	opt	5 808	7 080	7 960
r5nd-3kemcca-5d-sneik	opt	5 472	6 888	7 768
r5nd-5kemcca-0d	opt	7 760	10 080	11 384
r5nd-5kemcca-0d-sneik	opt	7 424	9 880	11 184
r5nd-5kemcca-5d	opt	7 264	8 872	9 968
r5nd-5kemcca-5d-sneik	opt	6 928	8 672	9 768
saber	ref	13 624	15 912	17 008
sikep434	opt	6 776	7 088	7 424
sikep503	opt	6 848	7 232	7 600
sikep610	opt	10 080	10 512	10 968
sikep751	opt	11 624	11 768	12 328
sntrup653	ref	13 976	14 004	16 700
sntrup761	ref	16 248	16 028	19 156
sntrup857	ref	18 264	17 908	21 444

Signature Schemes

Scheme	Type	Key Generation	Sign	Verify
dilithium2	m4	36 424	61 312	40 664
dilithium3	m4	50 752	81 792	55 000
dilithium4	m4	67 136	104 408	71 472

Scheme	Type	Key Generation	Sign	Verify
falcon1024	m4-ct	1 680	2 680	512
falcon512	m4-ct	1 680	2 484	512
falcon512-tree	m4-ct	1 656	2 452	512
dilithium2	clean	36 424	61 312	40 664
dilithium3	clean	50 752	81 792	55 000
dilithium4	clean	67 136	104 408	71 472
falcon1024	opt-ct	1 680	2 612	512
falcon1024	opt-leaktime	1 632	2 716	512
falcon512	opt-ct	1 680	2 524	512
falcon512	opt-leaktime	1 640	2 728	512
falcon512-tree	opt-ct	1 664	2 492	512
falcon512-tree	opt-leaktime	1 632	2 776	512
luov-48-43-222-chacha	ref	2 968	4 720	2 732
luov-48-43-222-keccak	ref	3 240	4 872	3 120
luov-64-61-302-chacha	ref	3 736	6 896	4 504
luov-64-61-302-keccak	ref	4 152	7 144	4 928
luov-8-58-237-chacha	ref	3 216	3 224	1 440
luov-8-58-237-keccak	ref	3 632	3 216	2 144
qTesla-I	ref	17 632	19 224	15 024
qTesla-II	ref	19 080	34 792	26 240
qTesla-III	ref	30 200	37 376	29 080
qTesla-V	ref	55 744	77 240	60 848
qTesla-V-size	ref	47 552	68 928	52 096
sphincs-haraka-128f-robust	clean	2 424	2 400	2 772
sphincs-haraka-128f-simple	clean	2 316	2 400	2 772
sphincs-haraka-128s-robust	clean	2 608	2 568	2 092
sphincs-haraka-128s-simple	clean	2 608	2 568	2 120
sphincs-haraka-192f-robust	clean	3 768	3 872	4 128
sphincs-haraka-192f-simple	clean	3 768	3 872	4 100
sphincs-haraka-192s-robust	clean	4 112	4 008	3 472
sphincs-haraka-192s-simple	clean	4 112	4 008	3 472
sphincs-haraka-256f-robust	clean	5 752	5 792	5 364
sphincs-haraka-256f-simple	clean	5 832	5 792	5 392
sphincs-haraka-256s-robust	clean	6 104	5 936	5 336
sphincs-haraka-256s-simple	clean	6 104	5 936	5 416
sphincs-sha256-128f-robust	clean	2 248	2 408	2 704
sphincs-sha256-128f-simple	clean	2 192	2 248	2 544
sphincs-sha256-128s-robust	clean	2 464	2 496	2 024
sphincs-sha256-128s-simple	clean	2 304	2 336	1 872
sphincs-sha256-192f-robust	clean	3 784	3 808	4 040
sphincs-sha256-192f-simple	clean	3 512	3 640	3 872
sphincs-sha256-192s-robust	clean	4 048	3 944	3 384
sphincs-sha256-192s-simple	clean	3 880	3 776	3 216
sphincs-sha256-256f-robust	clean	5 776	5 736	5 360
sphincs-sha256-256f-simple	clean	5 600	5 560	5 184
sphincs-sha256-256s-robust	clean	6 048	5 880	5 360
sphincs-sha256-256s-simple	clean	5 872	5 704	5 080
sphincs-shake256-128f-robust	clean	2 272	2 368	2 664
sphincs-shake256-128f-simple	clean	2 208	2 368	2 664
sphincs-shake256-128s-robust	clean	2 424	2 456	2 000
sphincs-shake256-128s-simple	clean	2 424	2 456	2 000

Scheme	Type	Key Generation	Sign	Verify
sphincs-shake256-192f-robust	clean	3 744	3 768	4 000
sphincs-shake256-192f-simple	clean	3 640	3 768	3 999
sphincs-shake256-192s-robust	clean	4 008	3 904	3 448
sphincs-shake256-192s-simple	clean	4 008	3 904	3 448
sphincs-shake256-256f-robust	clean	5 736	5 696	5 320
sphincs-shake256-256f-simple	clean	5 736	5 696	5 320
sphincs-shake256-256s-robust	clean	6 008	5 840	5 320
sphincs-shake256-256s-simple	clean	6 008	5 840	5 320

5.3 Hashing

We measure the number of CPU clock cycles spent in AES, SHA-2, SHA-3, SHAKE, and cSHAKE as a percentage of the total number of CPU clock cycles.

Key Encapsulation Schemes

Scheme	Type	Key Generation [%]	Encapsulation [%]	Decapsulation [%]
firesaber	m4	49.4	49.3	40.0
frodokem640aes	m4	73.6	77.1	76.3
frodokem640shake	m4	85.5	83.4	86.1
kyber1024	m4	70.5	73.3	66.7
kyber512	m4	68.9	72.5	61.6
kyber768	m4	69.7	73.0	64.8
lightsaber	m4	56.5	55.7	42.0
newhope1024cca	m4	71.2	70.3	59.5
newhope1024cpa	m4	66.3	62.2	6.8
ntruhps2048509	m4	0.0	4.1	19.0
ntruhps2048677	m4	0.0	4.1	16.8
ntruhps4096821	m4	0.0	3.2	15.6
ntruhrss701	m4	0.0	9.7	17.3
r5n1-1kemcca-0d	m4	20.5	93.5	46.1
r5n1-1kemcca-0d-sneik	m4	0.0	0.0	0.0
r5n1-3kemcca-0d	m4	16.6	41.2	43.7
r5n1-3kemcca-0d-sneik	m4	0.0	0.0	0.0
r5n1-5kemcca-0d	m4	11.6	18.1	34.6
r5n1-5kemcca-0d-sneik	m4	0.0	0.0	0.0
r5nd-1kemcca-0d	m4	46.6	56.4	54.6
r5nd-1kemcca-0d-sneik	m4	0.0	0.0	0.0
r5nd-1kemcca-5d	m4	40.4	42.6	40.2
r5nd-1kemcca-5d-sneik	m4	0.0	0.0	0.0
r5nd-3kemcca-0d	m4	30.9	34.9	35.4
r5nd-3kemcca-0d-sneik	m4	0.0	0.0	0.0
r5nd-3kemcca-5d	m4	36.7	40.7	39.0
r5nd-3kemcca-5d-sneik	m4	0.0	0.0	0.0
r5nd-5kemcca-0d	m4	27.1	30.9	31.1
r5nd-5kemcca-0d-sneik	m4	0.0	0.0	0.0
r5nd-5kemcca-5d	m4	28.8	31.9	32.0
r5nd-5kemcca-5d-sneik	m4	0.0	0.0	0.0
saber	m4	53.0	52.9	41.5

Scheme	Type	Key Generation [%]	Encapsulation [%]	Decapsulation [%]
babybear	opt	55.2	47.5	41.9
babybear	ref	8.4	6.2	6.5
babybear-ephem	opt	55.2	48.4	35.8
babybear-ephem	ref	8.4	6.5	4.3
firesaber	ref	18.8	18.6	13.7
frodokem640shake	opt	74.9	65.4	64.9
kyber1024	clean	58.8	58.0	47.4
kyber512	clean	54.6	53.5	38.9
kyber768	clean	57.0	56.3	44.0
lac128	ref	6.1	4.6	2.9
lac192	ref	1.9	2.1	1.2
lac256	ref	3.4	2.5	1.6
lightsaber	ref	24.7	23.6	15.3
mamabear	opt	53.9	47.9	43.3
mamabear	ref	7.4	5.9	6.1
mamabear-ephem	opt	53.4	47.9	34.3
mamabear-ephem	ref	7.4	6.0	3.9
newhope1024cca	clean	59.5	59.1	47.6
newhope1024cpa	clean	53.9	50.2	4.3
newhope512cca	clean	61.4	61.5	49.9
newhope512cpa	clean	56.1	52.4	8.6
ntruhps2048509	clean	0.0	0.9	1.4
ntruhps2048677	clean	0.0	0.8	1.1
ntruhps4096821	clean	0.0	0.6	0.9
ntruhrss701	clean	0.0	0.8	1.1
ntrulpr653	ref	0.5	0.5	0.3
ntrulpr761	ref	0.4	0.4	0.2
ntrulpr857	ref	0.3	0.4	0.2
papabear	opt	52.7	47.9	44.0
papabear	ref	6.9	5.8	6.0
papabear-ephem	opt	52.3	47.9	33.2
papabear-ephem	ref	6.9	5.8	3.6
r5n1-1kemcca-0d	opt	21.1	33.6	37.8
r5n1-1kemcca-0d-sneik	opt	0.0	0.0	0.0
r5n1-3kemcca-0d	opt	17.0	32.9	35.9
r5n1-3kemcca-0d-sneik	opt	0.0	0.0	0.0
r5n1-5kemcca-0d	opt	12.0	20.9	26.5
r5n1-5kemcca-0d-sneik	opt	0.0	0.0	0.0
r5nd-1kemcca-0d	opt	36.9	46.8	45.6
r5nd-1kemcca-0d-sneik	opt	0.0	0.0	0.0
r5nd-1kemcca-5d	opt	31.6	32.5	30.3
r5nd-1kemcca-5d-sneik	opt	0.0	0.0	0.0
r5nd-3kemcca-0d	opt	22.8	25.2	25.5
r5nd-3kemcca-0d-sneik	opt	0.0	0.0	0.0
r5nd-3kemcca-5d	opt	27.7	29.9	28.1
r5nd-3kemcca-5d-sneik	opt	0.0	0.0	0.0
r5nd-5kemcca-0d	opt	19.6	22.1	22.0
r5nd-5kemcca-0d-sneik	opt	0.0	0.0	0.0
r5nd-5kemcca-5d	opt	21.1	22.4	22.2
r5nd-5kemcca-5d-sneik	opt	0.0	0.0	0.0
saber	ref	21.3	20.9	14.6

Scheme	Type	Key Generation [%]	Encapsulation [%]	Decapsulation [%]
sikep434	opt	0.0	0.0	0.0
sikep503	opt	0.0	0.0	0.0
sikep610	opt	0.0	0.0	0.0
sikep751	opt	0.0	0.0	0.0
sntrup653	ref	0.0	0.5	0.1
sntrup761	ref	0.0	0.5	0.1
sntrup857	ref	0.0	0.4	0.1

Signature Schemes

Scheme	Type	Key Generation [%]	Sign [%]	Verify [%]
dilithium2	m4	70.8	44.4	65.6
dilithium3	m4	73.5	45.6	68.0
dilithium4	m4	73.1	49.8	69.4
falcon1024	m4-ct	11.2	0.5	36.4
falcon512	m4-ct	15.3	0.5	35.9
falcon512-tree	m4-ct	12.8	1.2	35.9
dilithium2	clean	56.6	25.1	47.1
dilithium3	clean	61.4	26.9	51.4
dilithium4	clean	62.1	30.2	54.6
falcon1024	opt-ct	7.9	0.3	34.2
falcon1024	opt-leaktime	11.6	0.6	34.2
falcon512	opt-ct	13.0	0.4	40.4
falcon512	opt-leaktime	18.4	0.6	35.8
falcon512-tree	opt-ct	12.0	0.8	35.8
falcon512-tree	opt-leaktime	16.0	1.1	35.8
luov-48-43-222-chacha	ref	0.3	0.1	0.0
luov-48-43-222-keccak	ref	30.7	12.2	15.3
luov-64-61-302-chacha	ref	0.2	0.1	0.0
luov-64-61-302-keccak	ref	33.3	11.3	15.5
luov-8-58-237-chacha	ref	0.3	0.2	0.0
luov-8-58-237-keccak	ref	30.6	22.5	32.9
qTesla-I	ref	18.7	39.5	44.3
qTesla-II	ref	5.9	11.7	14.8
qTesla-III	ref	27.6	34.8	39.4
qTesla-V	ref	28.7	30.0	41.1
qTesla-V-size	ref	1.7	12.2	16.6
sphincs-haraka-128f-robust	clean	0.0	0.0	0.0
sphincs-haraka-128f-simple	clean	0.0	0.0	0.0
sphincs-haraka-128s-robust	clean	0.0	0.0	0.0
sphincs-haraka-128s-simple	clean	0.0	0.0	0.0
sphincs-haraka-192f-robust	clean	0.0	0.0	0.0
sphincs-haraka-192f-simple	clean	0.0	0.0	0.0
sphincs-haraka-192s-robust	clean	0.0	0.0	0.0
sphincs-haraka-192s-simple	clean	0.0	0.0	0.0
sphincs-haraka-256f-robust	clean	0.0	0.0	0.0
sphincs-haraka-256f-simple	clean	0.0	0.0	0.0
sphincs-haraka-256s-robust	clean	0.0	0.0	0.0

Scheme	Type	Key Generation [%]	Sign [%]	Verify [%]
sphincs-haraka-256s-simple	clean	0.0	0.0	0.0
sphincs-sha256-128f-robust	clean	89.7	89.1	89.8
sphincs-sha256-128f-simple	clean	88.2	88.6	88.2
sphincs-sha256-128s-robust	clean	89.0	89.2	89.7
sphincs-sha256-128s-simple	clean	88.2	87.3	88.1
sphincs-sha256-192f-robust	clean	89.3	89.4	86.3
sphincs-sha256-192f-simple	clean	88.0	89.1	88.2
sphincs-sha256-192s-robust	clean	89.3	89.9	89.8
sphincs-sha256-192s-simple	clean	87.7	88.3	88.5
sphincs-sha256-256f-robust	clean	92.5	92.7	92.8
sphincs-sha256-256f-simple	clean	87.8	87.9	88.1
sphincs-sha256-256s-robust	clean	92.4	92.5	93.0
sphincs-sha256-256s-simple	clean	87.4	88.1	88.3
sphincs-shake256-128f-robust	clean	95.8	97.5	97.8
sphincs-shake256-128f-simple	clean	96.4	96.2	90.6
sphincs-shake256-128s-robust	clean	97.6	97.5	97.8
sphincs-shake256-128s-simple	clean	96.3	96.2	96.3
sphincs-shake256-192f-robust	clean	96.4	97.5	97.7
sphincs-shake256-192f-simple	clean	96.3	96.2	96.3
sphincs-shake256-192s-robust	clean	97.7	97.4	97.7
sphincs-shake256-192s-simple	clean	96.2	96.0	96.3
sphincs-shake256-256f-robust	clean	97.6	97.6	97.6
sphincs-shake256-256f-simple	clean	96.3	96.2	96.2
sphincs-shake256-256s-robust	clean	97.6	97.5	97.6
sphincs-shake256-256s-simple	clean	96.3	96.1	96.2

5.4 Code Size

Code size is measured in bytes.

Key Encapsulation Schemes

Scheme	Type	.text	.data	.bss	Total
firesaber	m4	44 184	0	0	44 184
frodokem640aes	m4	8 496	0	0	8 496
frodokem640shake	m4	8 584	0	0	8 584
kyber1024	m4	12 424	0	0	12 424
kyber512	m4	11 000	0	0	11 000
kyber768	m4	11 400	0	0	11 400
lightsaber	m4	44 916	0	0	44 916
newhope1024cca	m4	12 176	0	0	12 176
newhope1024cpa	m4	11 780	0	0	11 780
ntruhps2048509	m4	89 140	0	0	89 140
ntruhps2048677	m4	129 504	0	0	129 504
ntruhps4096821	m4	154 460	0	0	154 460
ntruhrss701	m4	132 224	0	0	132 224
r5n1-1kemcca-0d	m4	3 297	0	0	3 297
r5n1-1kemcca-0d-sneik	m4	4 808	0	0	4 808

Scheme	Type	.text	.data	.bss	Total
r5n1-3kemcca-0d	m4	3 425	0	0	3 425
r5n1-3kemcca-0d-sneik	m4	4 916	0	0	4 916
r5n1-5kemcca-0d	m4	3 473	0	0	3 473
r5n1-5kemcca-0d-sneik	m4	4 972	0	0	4 972
r5nd-1kemcca-0d	m4	2 651	0	0	2 651
r5nd-1kemcca-0d-sneik	m4	4 146	0	0	4 146
r5nd-1kemcca-5d	m4	5 623	0	0	5 623
r5nd-1kemcca-5d-sneik	m4	7 114	0	0	7 114
r5nd-3kemcca-0d	m4	2 879	0	0	2 879
r5nd-3kemcca-0d-sneik	m4	4 374	0	0	4 374
r5nd-3kemcca-5d	m4	6 975	0	0	6 975
r5nd-3kemcca-5d-sneik	m4	8 470	0	0	8 470
r5nd-5kemcca-0d	m4	2 927	0	0	2 927
r5nd-5kemcca-0d-sneik	m4	4 410	0	0	4 410
r5nd-5kemcca-5d	m4	4 507	0	0	4 507
r5nd-5kemcca-5d-sneik	m4	6 002	0	0	6 002
saber	m4	44 468	0	0	44 468
babybear	opt	5 627	0	0	5 627
babybear	ref	4 922	0	0	4 922
babybear-ephem	opt	4 985	0	0	4 985
babybear-ephem	ref	4 892	0	0	4 892
firesaber	ref	12 244	0	0	12 244
frodokem640shake	opt	6 464	0	0	6 464
kyber1024	clean	5 352	512	0	5 864
kyber512	clean	4 444	512	0	4 956
kyber768	clean	4 600	512	0	5 112
lac128	ref	29 708	72	296	30 076
lac192	ref	21 196	72	152	21 420
lac256	ref	29 876	72	296	30 244
lightsaber	ref	12 976	0	0	12 976
mamabear	opt	5 623	0	0	5 623
mamabear	ref	5 414	0	0	5 414
mamabear-ephem	opt	4 917	0	0	4 917
mamabear-ephem	ref	4 900	0	0	4 900
newhope1024cca	clean	10 780	0	0	10 780
newhope1024cpa	clean	10 384	0	0	10 384
newhope512cca	clean	7 016	0	0	7 016
newhope512cpa	clean	6 620	0	0	6 620
ntruhps2048509	clean	6 916	0	0	6 916
ntruhps2048677	clean	6 960	0	0	6 960
ntruhps4096821	clean	6 684	0	0	6 684
ntruhrss701	clean	6 828	0	0	6 828
ntrulpr653	ref	4 516	0	0	4 516
ntrulpr761	ref	4 632	0	0	4 632
ntrulpr857	ref	4 696	0	0	4 696
papabear	opt	5 559	0	0	5 559
papabear	ref	5 398	0	0	5 398
papabear-ephem	opt	4 877	0	0	4 877
papabear-ephem	ref	4 908	0	0	4 908
r5n1-1kemcca-0d	opt	4 055	0	0	4 055
r5n1-1kemcca-0d-sneik	opt	6 591	0	0	6 591

Scheme	Type	.text	.data	.bss	Total
r5n1-3kemcca-0d	opt	4 539	0	0	4 539
r5n1-3kemcca-0d-sneik	opt	7 055	0	0	7 055
r5n1-5kemcca-0d	opt	4 615	0	0	4 615
r5n1-5kemcca-0d-sneik	opt	7 139	0	0	7 139
r5nd-1kemcca-0d	opt	2 685	0	0	2 685
r5nd-1kemcca-0d-sneik	opt	5 205	0	0	5 205
r5nd-1kemcca-5d	opt	5 713	0	0	5 713
r5nd-1kemcca-5d-sneik	opt	8 229	0	0	8 229
r5nd-3kemcca-0d	opt	2 961	0	0	2 961
r5nd-3kemcca-0d-sneik	opt	5 481	0	0	5 481
r5nd-3kemcca-5d	opt	7 005	0	0	7 005
r5nd-3kemcca-5d-sneik	opt	9 525	0	0	9 525
r5nd-5kemcca-0d	opt	3 005	0	0	3 005
r5nd-5kemcca-0d-sneik	opt	5 513	0	0	5 513
r5nd-5kemcca-5d	opt	4 545	0	0	4 545
r5nd-5kemcca-5d-sneik	opt	7 065	0	0	7 065
saber	ref	12 524	0	0	12 524
sikep434	opt	30 368	0	0	30 368
sikep503	opt	31 376	0	0	31 376
sikep610	opt	20 064	0	0	20 064
sikep751	opt	21 692	0	0	21 692
sntrup653	ref	6 028	0	0	6 028
sntrup761	ref	6 164	0	0	6 164
sntrup857	ref	6 148	0	0	6 148

Signature Schemes

Scheme	Type	.text	.data	.bss	Total
dilithium2	m4	13 948	0	0	13 948
dilithium3	m4	13 756	0	0	13 756
dilithium4	m4	13 852	0	0	13 852
falcon1024	m4-ct	77 335	0	79 872	157 207
falcon512	m4-ct	77 335	0	39 936	117 271
falcon512-tree	m4-ct	77 067	0	27 648	104 715
dilithium2	clean	11 336	0	0	11 336
dilithium3	clean	11 144	0	0	11 144
dilithium4	clean	11 240	0	0	11 240
falcon1024	opt-ct	76 203	0	79 872	156 075
falcon1024	opt-leaktime	71 947	0	79 872	151 819
falcon512	opt-ct	76 203	0	39 936	116 139
falcon512	opt-leaktime	71 947	0	39 936	111 883
falcon512-tree	opt-ct	75 935	0	27 648	103 583
falcon512-tree	opt-leaktime	71 679	0	27 648	99 327
luov-48-43-222-chacha	ref	405 722	36	0	405 758
luov-48-43-222-keccak	ref	405 730	36	0	405 766
luov-64-61-302-chacha	ref	405 412	36	0	405 448
luov-64-61-302-keccak	ref	405 452	36	0	405 488
luov-8-58-237-chacha	ref	404 920	36	0	404 956

Scheme	Type	.text	.data	.bss	Total
luov-8-58-237-keccak	ref	404 924	36	0	404 960
qTesla-I	ref	13 512	0	0	13 512
qTesla-II	ref	18 832	0	7 204	26 036
qTesla-III	ref	17 400	0	0	17 400
qTesla-V	ref	29 536	0	0	29 536
qTesla-V-size	ref	22 892	0	14 372	37 264
sphincs-haraka-128f-robust	clean	17 192	0	1 280	18 472
sphincs-haraka-128f-simple	clean	17 028	0	1 280	18 308
sphincs-haraka-128s-robust	clean	17 500	0	1 280	18 780
sphincs-haraka-128s-simple	clean	17 336	0	1 280	18 616
sphincs-haraka-192f-robust	clean	17 140	0	1 280	18 420
sphincs-haraka-192f-simple	clean	16 940	0	1 280	18 220
sphincs-haraka-192s-robust	clean	17 432	0	1 280	18 712
sphincs-haraka-192s-simple	clean	17 232	0	1 280	18 512
sphincs-haraka-256f-robust	clean	17 528	0	1 280	18 808
sphincs-haraka-256f-simple	clean	17 240	0	1 280	18 520
sphincs-haraka-256s-robust	clean	17 736	0	1 280	19 016
sphincs-haraka-256s-simple	clean	17 448	0	1 280	18 728
sphincs-sha256-128f-robust	clean	4 860	0	40	4 900
sphincs-sha256-128f-simple	clean	4 628	0	40	4 668
sphincs-sha256-128s-robust	clean	5 160	0	40	5 200
sphincs-sha256-128s-simple	clean	4 928	0	40	4 968
sphincs-sha256-192f-robust	clean	4 948	0	40	4 988
sphincs-sha256-192f-simple	clean	4 636	0	40	4 676
sphincs-sha256-192s-robust	clean	5 244	0	40	5 284
sphincs-sha256-192s-simple	clean	4 932	0	40	4 972
sphincs-sha256-256f-robust	clean	5 468	0	40	5 508
sphincs-sha256-256f-simple	clean	5 044	0	40	5 084
sphincs-sha256-256s-robust	clean	5 676	0	40	5 716
sphincs-sha256-256s-simple	clean	5 252	0	40	5 292
sphincs-shake256-128f-robust	clean	4 096	0	0	4 096
sphincs-shake256-128f-simple	clean	3 896	0	0	3 896
sphincs-shake256-128s-robust	clean	4 404	0	0	4 404
sphincs-shake256-128s-simple	clean	4 204	0	0	4 204
sphincs-shake256-192f-robust	clean	4 108	0	0	4 108
sphincs-shake256-192f-simple	clean	3 856	0	0	3 856
sphincs-shake256-192s-robust	clean	4 400	0	0	4 400
sphincs-shake256-192s-simple	clean	4 148	0	0	4 148
sphincs-shake256-256f-robust	clean	4 480	0	0	4 480
sphincs-shake256-256f-simple	clean	4 172	0	0	4 172
sphincs-shake256-256s-robust	clean	4 684	0	0	4 684
sphincs-shake256-256s-simple	clean	4 380	0	0	4 380

References

AAB+19a. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. HQC. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

AAB+19b. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillippe Gaborit, Gilles Zémor, Alain Couvreur, and Adrien Hauteville. RQC. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

ABB+19. Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, and Valentin Vasseur. BIKE. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

ABD+19. Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, Gilles Zémor, Carlos Aguilar Melchor, Slim Bettaieb, Loic Bidoux, Magali Bardet, and Ayoub Otmani. ROLLO. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

ACP+19. Martin Albrecht, Carlos Cid, Kenneth G. Paterson, Cen Jung Tjhai, and Martin Tomlinson. NTS-KEM. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

AJS16. 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. 4

BAA+19. Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. qTESLA. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 5

BBC+19. Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDAcrypt. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

BCL+19. Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

BCLv19. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 4

BFM+18. 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. 4

BKS19. 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. Springer-Verlag Berlin Heidelberg, 2019. (to appear). 4

BPSV19. Ward Beullens, Bart Preneel, Alan Szepieniec, and Frederik Vercauteren. LUOV. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 5

CFM+19. A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. GeMSS. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

DCP+19. Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang. Rainbow. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 7

GKOS18. 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. 5

GZB+19. Oscar Garcia-Morchon, Zhenfei Zhang, Sauvik Bhattacharya, Ronald Rietman, Ludo Tolhuizen, Jose-Luis Torre-Arce, Hayo Baan, Markku-Juhani O. Saarinen, Scott Fluhrer, Thijs Laarhoven, and Rachel Player. Round5. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 4

Ham19. Mike Hamburg. ThreeBears. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 5

HBD+19. Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Jean-Philippe Aumasson. SPHINCS+. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 5

JAC+19. David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, and Geovandro Pereira. SIKE. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 5

KBMSRV18. Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. Saber on ARM – CCA-secure module lattice-based key encapsulation on ARM. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):243–266, Aug. 2018. 4

KRS19. 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. Springer-Verlag Berlin Heidelberg, 2019. (to appear). 4

LLJ+19. Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, Zhenfei Zhang, Zhe Liu, Hao Yang, Bao Li, and Kunpeng Wang. LAC. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 4

LMB+19. Jérôme Lablanche, Lina Mortajine, Othman Benchaalal, Pierre-Louis Cayrel, and Nadia El Mrabet. Optimized implementation of the NIST PQC submission ROLLO on microcontroller. Cryptology ePrint Archive, Report 2019/787, 2019. https://eprint.iacr.org/2019/787. 6

NAB+19. Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 3

OSHG19. Tobias Oder, Julian Speith, Kira Höltgen, and Tim Güneysu. Towards practical microcontroller implementation of the signature scheme Falcon. In *Post-Quantum Cryptography – PQCrypto 2019*, Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 2019. (to appear). 5

PAA+19. Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. NewHope. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 4

PFH+19. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 5

Saa19. Markku-Juhani O. Saarinen. Exploring NIST LWC/PQC synergy with R5Sneik: How SNEIK 1.1 algorithms were designed to support Round5. Cryptology ePrint Archive, Report 2019/685, 2019. https://eprint.iacr.org/2019/685. 4

SAB+19. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 4

SCH+19. Simona Samardjiska, Ming-Shing Chen, Andreas Hulsing, Joost Rijneveld, and Peter Schwabe. MQDSS. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 6

SJA19. Hwajeong Seo, Amir Jalali, and Reza Azarderakhsh. SIKE round 2 speed record on ARM Cortex-M4. Cryptology ePrint Archive, Report 2019/535, 2019. https://eprint.iacr.org/2019/535. 5

ZCD+19. Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, and Vladmir Kolesnikov. Picnic. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 7

ZCH+19. Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, William Whyte, John M. Schanck, Andreas Hulsing, Joost Rijneveld, Peter Schwabe, and Oussama Danba. NTRUEncrypt. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 4