Evaluation of CRYSTALS-Kyber and Saber on the ARMv8 architecture

Jheyne N. Ortiz[1], **Félix Carvalho Rodrigues**[1], **Décio Gazzoni Filho**[1], **Caio Teixeira**[1], **Julio López**[1], **Ricardo Dahab**[1]

[1] Institute of Computing – University of Campinas (Unicamp)
Av. Albert Einstein, 1251 – 13083-852 – Campinas – SP – Brazil

{jheyne.ortiz,felix.rodrigues,decio.gazzoni}@ic.unicamp.br
{jlopez,rdahab}@ic.unicamp.br
caio.teixeira@students.ic.unicamp.br

Abstract. *This paper presents preliminary experimental results for the implementation of the third-round NIST finalists CRYSTALS-Kyber and Saber on the ARMv8 architecture. Our implementation uses NEON instructions to speed up key generation, encapsulation, and decapsulation of the reference codes. The benchmarks are performed on three devices: an Orange Pi WinPlus (Cortex-A53), a Raspberry Pi 4 (Cortex-A72), and a MacBook Air based on an Apple M1 chip. The experimental results show substantial improvements for Kyber and Saber, with speed-ups in the ranges* $1.16-1.38\times$ *and* $1.21-1.96\times$*, respectively. We focused on the most time-consuming operations of each cryptosystem; however, similar works suggest that more expressive speed-ups can be obtained by extending the use of NEON instructions to other primary sub-routines.*

1. Introduction

In 2016, the U.S. National Institute of Standards and Technology (NIST) initiated a process for the standardization of public-key quantum-resistant cryptographic algorithms. The Post-Quantum Cryptography Standardization project [NIST 2017] requested nominations of key-encapsulation mechanisms (KEMs), public-key encryption (PKE) algorithms, and digital signature schemes. This process drew strong interest from the academic community and the industry in the past few years to develop and implement post-quantum cryptosystems in various platforms, ranging from the popular ARM Cortex-M4 microcontroller to the Apple M1 chip and Intel CPUs. From the beginning of the standardization project, lattice-based cryptography has been the most prominent class of problems. During the third round, it represented five of the seven finalists. Particularly, among the lattice-based KEMs selected as third-round finalists were CRYSTALS-Kyber [Avanzi et al. 2019] and Saber [Basso et al. 2020]. On July, 2022, NIST announced the end of the third round and the standardization of Kyber KEM, along with CRYSTALS-Dilithium, Falcon, and SPHINCS+ for digital signatures. The fourth-round candidates, however, do not include Saber.

Our contributions. In this paper, we improve the running time of Kyber and Saber on the ARMv8 architecture, which embeds IoT devices, smartphones, and laptops. Particularly, the ARMv8-A architecture profile contains an Advanced SIMD (Single Instruction Multiple Data) architecture extension, named NEON, that provides additional instructions which performs mathematical operations in parallel on multiple data streams. In this

context, we first profiled Kyber's and Saber's most expensive operations, replacing the original portable-C implementation in the reference code with NEON-optimized counterparts. For Kyber, we focused on binomial sampling and computing the inner product in the module R^k. Our NEON implementation includes a vectorized version of Montgomery and Barrett reduction algorithms. Moreover, we designed an ARMv8-optimized polynomial multiplication for Saber combining the Schoolbook, Karatsuba, and Toom-Cook algorithms. For Kyber, our performance gains compared to the reference code for key generation, encapsulation, and decapsulation are $1.16 - 1.38\times$, $1.16 - 1.34\times$, and $1.19 - 1.35\times$. Moreover, our speed-ups for Saber in key generation, encapsulation, and decapsulation are in $1.21 - 1.83\times$, $1.28 - 1.86\times$, and $1.44 - 1.96\times$, respectively.

2. Background

In this section, we briefly present the third-round CRYSTALS-Kyber [Avanzi et al. 2019] and Saber [Basso et al. 2020] submissions, focusing on their main system parameters.

CRYSTALS-Kyber. CRYSTALS-Kyber is a key-encapsulation mechanism whose security is based on the hardness of solving the Mod-LWE problem. Kyber KEM is obtained by instantiating Kyber PKE and applying a variant of the Fujisaki-Okamoto transform. The PKE provides the core functions for the tuple of algorithms $(\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ which defines the KEM. The PKE scheme is mainly parameterized by the integers n, k, and q. The parameters n and q define the underlying polynomial ring as $R = \mathbb{Z}_q[x]/(x^n + 1)$, and k is the rank of the module R^k. Notice that n and q are always $n = 256$ and $q = 3329$; thus, the values for k, which are $\{2, 3, 4\}$, define the target security level and the parameter set as either Kyber-512, Kyber-768, or Kyber-1024.

Saber. Saber consists of a public-key encryption scheme and a key encapsulation mechanism that rely on the hardness of the Mod-LWR problem. Similarly to Kyber, a version of the Fujisaki-Okamoto transform is applied to an IND-CPA encryption scheme to obtain the IND-CCA2-secure key encapsulation scheme. Saber is mainly parameterized by the integers n, ℓ, q, p, and T. The parameters n and ℓ define the ring $R = \mathbb{Z}[x]/(x^n + 1)$ and the rank of the module $R^{\ell \times 1}$. In particular, the value of n is fixed as 256 and $\ell = \{2, 3, 4\}$ varies according to the parameter set, which is either LightSaber, Saber, or FireSaber. In turn, $q = 2^{\epsilon_q}$, $p = 2^{\epsilon_p}$, and $T = 2^{\epsilon_T}$ are the moduli, each chosen as a power of two such that $\epsilon_q > \epsilon_p > \epsilon_T$. Concretely, $q = 2^{13}$, $p = 2^{10}$, and $T = \{3, 4, 6\}$.

3. Performance Evaluation

We collected benchmarks for the main functions of Kyber and Saber in several devices, representing a range of possible ARMv8 processors. The performance of our implementations was measured using the Google Benchmark framework in three devices: an Orange Pi WinPlus (Cortex-A53), a Raspberry Pi 4 (Cortex-A72), and an Apple MacBook Air featuring the M1 chip. The OrangePi board has four in-order CPUs with maximum and minimum clock frequencies of 1152 MHz and 648 MHz. However, we performed our experiments at the constant frequency of 1008 MHz. Moreover, this board supports AES, SHA-1, and SHA-2 cryptographic instructions. The Raspberry Pi 4 contains four CPUs running at the maximum clock frequency of 1500 MHz and a minimum clock frequency of 600 MHz. The M1 device features eight cores (four performance and four efficiency cores) running at a maximum CPU clock rate of 3.2 GHz, and supports the ARMv8.4-A

instruction set, the aforementioned cryptographic instructions, and SHA-3 instructions. The binaries were compiled using the Clang compiler versions 10 (OrangePi) and 13 (Raspberry Pi 4 and M1) with the `-O3` optimization flag. Our cryptographic primitives core uses the ARMv8 AES and SHA-3 instructions to improve the performance of symmetric primitives used by both KEMs whenever possible. Moreover, it has an optimized implementation of Keccak, targeting the Cortex-A53 and A72 processors and accelerating SHA-3 and SHAKE functions.

CRYSTALS-Kyber. We optimized Kyber for the ARMv8 architecture providing a NEON implementation for its most time-consuming functions: sampling from the centered binomial distribution $\mathcal{B}$ parameterized by $\eta = 2$ and the inner product in the module R^k. Notice that Kyber requires samples from $\mathcal{B}_{\eta=2}$ and $\mathcal{B}_{\eta=3}$, but focusing on the first case impacts most of Kyber's parameter sets, as well as all three algorithms for key generation, encapsulation, and decapsulation. The inner product is invoked during matrix-vector multiplications and by the PKE scheme. Also, the point-wise product is computed as the product of degree-one polynomials instead of scalars. As a result, each point-wise product requires six multiplications modulo q followed by a Montgomery reduction. Therefore, NEON implementations of three functions were designed: multiplication modulo q followed by Montgomery reduction; point-wise multiplication of degree-one polynomials; and inner product of vectors of polynomials in $\mathbb{Z}_{3329}[x]/(x^{256}+1)$. The reduction modulo q at the end of the inner product was performed using a NEON implementation of the Barrett reduction algorithm. These optimizations were based on state-of-the-art implementations targeting the ARMv8 architecture [Sanal et al. 2021, Nguyen and Gaj 2021].

Saber. Profiling results for LightSaber, Saber, and FireSaber indicated that most of the running time in all variants is spent on polynomial multiplication. In this sense, we designed an ARMv8-optimized NEON implementation combining the Schoolbook, Karatsuba 2-way, Karatsuba 4-way, and Striding Toom-Cook algorithms for polynomial multiplication on inputs of sizes 8, 16, 64, and 256, respectively. Although this combination of algorithms for polynomial multiplication is new, it is very similar to the one presented by Nguyen and Gaj [Nguyen and Gaj 2021]. Nonetheless, the two formulations were developed independently.

3.1. Experimental Results

In this section, we present the running time, in microseconds, for the reference code and our ARMv8-optimized implementation in the Orange Pi (Cortex-A53), Raspberry Pi 4 (Cortex-A72), and M1 platforms. Tables 1 and 2 present experimental results for Kyber and Saber in all three security levels. For Kyber, the speed-ups range from 1.16 to $1.38\times$. The most modest results were obtained in the Cortex-A72 processor ($1.16 - 1.22\times$). On the other hand, for Cortex-A53 and M1 the speed-ups are $1.23-1.38\times$ and $1.27-1.34\times$, respectively. The most time-consuming operation of our implementation of Saber is the polynomial multiplication. By providing an ARMv8 implementation using variants of the Toom-Cook algorithm, we obtained speed-ups ranging from 1.21 to $1.96\times$. The results obtained in the Cortex-A72 processor were moderate, between 1.21 and $1.48\times$. More expressive gains were observed in the Cortex-A53, for which the minimum improvement was $1.69\times$. In the middle range, for the M1 chip, the results are in the interval $1.52 - 1.64\times$. Notice that the Raspberry Pi 4 platform does not provide the ARMv8 instructions for SHA-3 and AES, which can explain the modest speed-ups compared to A53 and M1.

Moreover, Kyber and Saber have taken advantage of the SHA-3 ARMv8 instructions present in the M1 chip for efficiently computing their symmetric primitives. However, we emphasize that better results could be achieved through parallelization of SHA-3/SHAKE calls.

Table 1. Benchmarking results, in microseconds, of the portable-C reference code and our ARMv8-optimized implementation of Kyber.

	Cortex-A53			Cortex-A72			M1		
Kyber-512	KGen	Encaps	Decaps	KGen	Encaps	Decaps	KGen	Encaps	Decaps
Reference	144	180	209	214	158	88.3	12.1	15.2	15.0
This work	107	140	156	184	136	73.7	9.54	11.4	11.7
Speedup	1.35	1.29	1.34	1.16	1.16	1.20	1.27	1.33	1.28
Kyber-768	KGen	Encaps	Decaps	KGen	Encaps	Decaps	KGen	Encaps	Decaps
Reference	252	297	338	262	213	145	21.2	23.8	23.9
This work	183	229	251	222	181	119	16.5	17.7	18.1
Speedup	1.38	1.30	1.35	1.18	1.18	1.22	1.28	1.34	1.32
Kyber-1024	KGen	Encaps	Decaps	KGen	Encaps	Decaps	KGen	Encaps	Decaps
Reference	381	432	483	325	282	216	33.8	34.4	35.8
This work	286	350	384	276	243	182	26.4	26.5	27.2
Speedup	1.33	1.23	1.26	1.18	1.16	1.19	1.28	1.30	1.32

Table 2. Benchmarking results, in microseconds, of the portable-C reference code and our ARMv8-optimized implementation of Saber.

	Cortex-A53			Cortex-A72			M1		
LightSaber	KGen	Encaps	Decaps	KGen	Encaps	Decaps	KGen	Encaps	Decaps
Reference	172	240	292	299	176	115	12.2	15.7	16.8
This work	102	138	154	248	138	79.6	7.56	10.3	10.6
Speedup	1.69	1.74	1.90	1.21	1.28	1.44	1.61	1.52	1.58
Saber	KGen	Encaps	Decaps	KGen	Encaps	Decaps	KGen	Encaps	Decaps
Reference	350	449	527	367	256	204	21.9	27.7	29.8
This work	197	246	270	297	195	138	14.0	17.5	18.3
Speedup	1.78	1.83	1.95	1.24	1.31	1.48	1.56	1.58	1.63
FireSaber	KGen	Encaps	Decaps	KGen	Encaps	Decaps	KGen	Encaps	Decaps
Reference	595	724	831	462	365	320	35.9	43.6	46.6
This work	326	389	423	361	268	216	22.5	27.3	28.4
Speedup	1.83	1.86	1.96	1.28	1.36	1.48	1.60	1.60	1.64

4. Comparison with State-of-the-art Implementations

Nguyen and Gaj [Nguyen and Gaj 2021] and Sanal et al. [Sanal et al. 2021] can be considered the first to provide a complete ARMv8-A implementation of the third-round finalist Kyber. Nguyen and Gaj also implement the third-round finalist Saber. Additionally, Becker et al. [Becker et al. 2021] present improved versions for the polynomial multiplication algorithms used in Saber and Kyber. Table 3 reports the speed-ups obtained by related works in the literature compared to the reference code. This table combines the updated data reported by Nguyen and Gaj on their public repository[1] and Becker et al. in their paper [Becker et al. 2021].

[1] https://github.com/GMUCERG/PQC_NEON.

Table 3. Speed-ups for Kyber-768 and Saber on Cortex-A72 and Apple M1 reported by state-of-the-art implementations.

	Cortex-A72			M1		
Kyber-768	KGen	Encaps	Decaps	KGen	Encaps	Decaps
[Becker et al. 2021]	2.37	2.33	2.88	4.42	3.33	4.48
[Nguyen and Gaj 2021]	2.16	2.13	2.56	3.08	2.59	3.34
[Sanal et al. 2021]	1.65	1.65	1.95	–	–	–
Saber	KGen	Encaps	Decaps	KGen	Encaps	Decaps
[Becker et al. 2021]	2.16	1.96	2.01	2.27	2.09	2.26
[Nguyen and Gaj 2021]	1.47	1.50	1.58	1.57	1.62	1.78

Nguyen and Gaj explored using Toom-Cook and NTT for polynomial multiplication on Saber. However, the replacement of Toom-Cook by NTT showed a slowdown in overall cryptosystem performance. Also, they present speed-ups for Kyber, reducing the number of calls to the Barrett reduction in the inverse NTT and vectorizing its implementation using NEON instructions. Sanal et al. focus on optimizing Kyber for 64-bit ARM Cortex-A processors, presenting experimental results on Cortex-A75 and Apple A12. Notice that the results reported in Table 3 on Cortex-A72 were further obtained by Becker et al.. In their work, the NTT operations are similar to those used by Nguyen and Gaj. In addition, the authors employ NEON instructions for implementing Barrett and Montgomery reductions and sampling from a centered binomial distribution. More recently, Becker et al. proposed a combination of Montgomery multiplications with Barrett reductions, resulting in a "Barrett multiplication". For Saber and Kyber, they treated matrix-vector multiplications such that not only the vector is cached but also parts of the product. Other improvements building upon Nguyen and Gaj's work include using 32-bit NTTs for Saber instead of the 16-bit version that proved inferior to the Toom-Cook approach. In this context, our preliminary results for Kyber-768 and Saber showed speed-ups in both Cortex-A72 ($1.18 - 1.48\times$) and Apple M1 ($1.28 - 1.63\times$) architectures. However, our implementations still require further optimizations to match the state-of-the-art. By implementing a NEON-optimized Toom-Cook-based approach for polynomial multiplication on Saber, we achieved results similar to those reported by Nguyen and Gaj. On the other hand, our implementation of Kyber still requires at least a NEON-optimized version of the NTT to obtain running times similar to those reported by Sanal et al. on the Cortex-A72.

5. Conclusion

This work presented preliminary optimization results for CRYSTALS-Kyber and Saber targeting the ARMv8 architecture. Our benchmarks were performed on the ARM Cortex-A53 and Cortex-A72 processors and the Apple M1 chip. We used NEON instructions to accelerate the most time-expensive routines and compared our implementations with the reference code submitted to the third round of NIST's standardization process, since our current results still do not outperform state-of-the-art ARMv8-A implementations. Table 4 compares our Kyber and Saber implementations on the M1 platform. Both KEMs perform similarly apart from the key-generation algorithm, although Saber has smaller system parameters. In most cases, our preliminary implementation of Kyber outperforms Saber on Cortex-A53 and A72. For further optimizations, we consider targeting Kyber's NTT transforms, as done in recent works in the litera-

ture [Sanal et al. 2021, Becker et al. 2021, Nguyen and Gaj 2021]. For Saber, implementations of matrix-vector multiplications of the form $\mathbf{A}^\top\mathbf{s}$ can explore the fact that rows are multiplied by the same vector $\mathbf{s}$ to reduce the number of memory accesses to elements of $\mathbf{s}$. Moreover, our Saber implementation can be improved by using NEON instructions to sample from a centered binomial distribution. Simpler optimizations such as vectorized polynomial addition are also possible. The incorporation of countermeasures against side-channel and fault attacks was left as future work, as the possibility of proposing a masked implementation for the decapsulation algorithm.

Table 4. Comparison of Kyber and Saber in terms of running time, in microseconds, and parameters' byte size in the Apple M1 chip.

Parameter Set	Running time (μs)			Parameter sizes (KB)		
	KGen	Encaps	Decaps	**Public Key**	**Secret Key**	**Ciphertext**
Kyber-512	9.54	11.4	11.7	800	1632	768
LightSaber	**7.56**	**10.3**	**10.6**	**672**	**1568**	**736**
Kyber-768	16.5	17.7	**18.1**	1184	2400	1088
Saber	**14.0**	**17.5**	18.3	**992**	**2304**	1088
Kyber-1024	26.4	**26.5**	**27.2**	1568	3168	1568
FireSaber	**22.5**	27.3	28.4	**1312**	**3040**	**1472**

Acknowledgements

Part of results presented in this work was obtained through the "Post-Quantum Cryptography" project, funded by Samsung Eletrônica da Amazônia Ltda., under the Brazilian Informatics Law 8.248/91.

References

Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2019). CRYSTALS – Kyber: Algorithm Specifications and Supporting Documentation. NIST Post-Quantum Cryptography Standardization Process. https://pq-crystals.org/kyber/.

Basso, A., Mera, J. M. B., D'Anvers, J.-P., Karmakar, A., Roy, S. S., Beirendonck, M. V., and Vercauteren, F. (2020). SABER: Mod-LWR based KEM (Round 3 Submission). Submission to the NIST Post-Quantum Cryptography Standardization Project. https://www.esat.kuleuven.be/cosic/pqcrypto/saber/index.html.

Becker, H., Hwang, V., Kannwischer, M. J., Yang, B.-Y., and Yang, S.-Y. (2021). Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1. Cryptology ePrint Archive, Report 2021/986. https://ia.cr/2021/986.

Nguyen, D. T. and Gaj, K. (2021). Optimized Software Implementations of CRYSTALS-Kyber, NTRU, and Saber Using NEON-Based Special Instructions of ARMv8. Third PQC Standardization Conference. https://csrc.nist.gov/Events/2021/third-pqc-standardization-conference.

NIST, N. (2017). Post-Quantum Cryptography. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

Sanal, P., Karagoz, E., Seo, H., Azarderakhsh, R., and Mozaffari-Kermani, M. (2021). Kyber on ARM64: Compact Implementations of Kyber on 64-bit ARM Cortex-A Processors. Cryptology ePrint Archive, Report 2021/561. https://ia.cr/2021/561.