Evaluating Kyber post-quantum KEM in a mobile application

José Paulo da Silva Lima[1], Leonardo A. D. S. Ribeiro[1], Ruy J. G. B. de Queiroz[1], Jonysberg P. Quintino[1], Fabio Q. B. da Silva[1], Andre L M Santos[1] and José Roberto[2]
[1]*Centro de Informática, Universidade Federal de Pernambuco, Recife - PE, Brazil*
[2]*Samsung Eletrônica da Amazônia LTDA, Campinas - SP, Brazil*
{jpsl3, ladsr, ruy, jpq, fabio, alms}@cin.ufpe.br, jose.junior@samsung.com

Keywords: Post quantum cryptography. Evaluating PQC algorithm. Mobile application. Kyber algorithm.

Abstract: The use of Post Quantum Cryptography algorithms has become a requirement to whomever is concerned with the security of digital information given the likelihood of the existence and wide availability of quantum computers, particularly in mobile devices. Here we present an evaluation of Kyber KEM algorithm running on an Android mobile application. The main objective is analysing if Kyber algorithm is efficient for this scenario.

1 INTRODUCTION

The advancement of quantum computing, as evidenced by the in progress standardization process proposed by NIST (PQC-NIST, 2017), evolves asymmetric key cryptography into the post-quantum cryptography phase. In fact, work in the area of PQC is already making progress as we already have a standardization process proposed by NIST well in its 3rd round.

Established PQC proposals include Public-key encryption, Key-establishment, and digital signature algorithms that require less computational resources than RSA or ECC (Saarinen, 2020), thus are suitable replacements for mobile devices, smart cards, and Internet of Things (IoT) applications. On the other hand, there are prohibitively "heavy" PQC proposals for any of these target environments, so it is necessary to understand the individual characteristics of each algorithm for the design of the system (Saarinen, 2020).

While the practical feasibility of quantum computers still arouses some mistrust among scientists, with the growing discoveries of quantum computing, many researchers are becoming increasingly positive about the future of large-scale quantum computers (Xu et al., 2018).

It is also important to highlight a popularization of mobile devices, especially smartphones. It is natural that we think about evaluating the performance of these post-quantum algorithms on smartphones (Hecht, 2017). Therefore, in this work, we compare the performance of Kyber algorithm, which is one of the candidates in NIST round 3 standardization process, both on a traditional computer and on a smartphone.

For evaluation on the smartphone, we inserted the Kyber algorithm into a mobile application prototype in which the Kyber algorithm was responsible for protecting the session key for this application.

2 PQC ALGORITHMS

Since 2017, NIST has been leading the process of standardizing post-quantum algorithms in the areas of key-establishment and digital signature algorithms. The competition is currently at the third round 3 of the selection process. From 69 participants at the beginning of the selection today we have only four candidates in dispute to be nominated as standard in KEM.

It is also essential to highlight that among these candidates of round 3, a notable presence is that three out of the four finalists are based on lattices.

Lattice-based encryption has strong security guarantees. The underlying difficult problems have been extensively studied for decades, but no efficient algorithm, whether classic or post-quantum, is known for these problems. In addition, lattice-based encryption enjoys a reduction from worst-case to medium-case.

Encryption inherently requires middle case intractability, considering the requirement for random keys. Reducing the complexity of the middle case to the complexity of the worst case essentially ensures that lattice-based encryption is secure on average, unless all instances of the underlying lattice problem are easy. From a practical point of view, this reduction in the worst case makes it much easier to select parameters and generate keys in lattice-based encryption (Khalid et al., 2019).

For example, the RSA encryption system is based on the hardness of factoring integers. But this is a worst-case problem. It is known that, if prime numbers have certain properties of number theory, the problem turns out to be essentially easy. Therefore, it is important to avoid these structures when generating keys for RSA (Dang et al., 2019).

The main factor that made us discard NTRU is that it "has a computationally expensive keypair generation which limits its usability." (Saarinen, 2020)

Among the candidates selected for Round 3, the idea was to make a choice based on performance of CPU usage, and taking into account the feasibility of building a prototype implementation. For this, it would be important to know the kind of license the reference implementation has, as well as the necessary crypto libraries code were under it. For example, unlike the other candidates, the candidate NTRU is not under a Public Domain License.

3 GATHERING CONCRETE DATA

The idea is to perform an analysis of NIST PQC Round 3 candidate Kyber performance in x64 and ARM architectures. The reference implementation of this KEM candidate was tested following a scenario where first a pair of public and private keys was generated and then data representing a session key was encrypted using the public key generated and decrypted using the corresponding private key. Algorithm performance were evaluated running these steps in each target architecture. Based on the collected data, the evaluation sought to check whether Kyber was suitable to be deployed in mobile devices. Bottlenecks were found in code according to performance analysis, and, as a consequence, some improvements were proposed to reference implementation code.

As noted by (Howe1 et al., 2019), "all real-world efforts to deploy post-quantum cryptography will have to contend with new, unique problems". By experimenting with the reference implementation of Kyber, we have tried to embark on a cryptographic engineering task which may bring insight into the real world deployment of such a candidate for standardization. Such an enterprise is likely to face peculiar circumstances whose evaluation may prove useful in the real world. Again, Howe et al. point out that specific cases "may require a diverse combination of computational assumptions woven together into a single hybrid scheme", as well as "special attention to physical management of sensitive state", not to mention "very unbalanced performance profiles, requiring distinct solutions for different application scenarios".

In this spirit, the intention here is to gather concrete performance data for real implementations in resource-constrained (memory, processing speed, energy consumption, etc.) devices, in particular, embedded devices and mobile application running in platforms such as iOS and Android (scope of cell phones and tablets, with resource restrictions, battery, memory, etc.), and perform code improvements wherever appropriate.

4 CODE PROFILING

Firstly, it is necessary to present the two environments where we perform this work: an x86 architecture based computer with Linux and an ARM architecture based smartphone with Android. It was necessary to search for an application that could help with the necessary data collection. However, the biggest challenge in this data collection was to find a tool that offers the same pattern or environment for two different architectures. These two different environments are necessary for understanding if each architecture interferes in the algorithm's performance.

The desktop computer had as hardware features an Intel (R) Core (TM) i7-6700 processor with 3.4GHz of clock and 8 GB of RAM, being controlled by the Ubuntu 20.04 LTS operating system. The smartphone had an Octa-core processor (two 2.73 GHz Mongoose M5 and two 2.60 GHz Cortex-A76 + four 2.0 GHz Cortex-A55), 8 GB of RAM and Android 10 installed.

Secondly, to run the Kyber algorithm, for analysis on a classic computer and smartphone, a small Java code was generated to communicate with the NIST reference C code. This Java code was implemented using JNI to interact with the selected algorithm.

KYBER code uses OpenSSL, so it was necessary to install it together with its libraries on Linux, and C code was compiled using GCC compiler once the JNI and the OpenSSL installation paths must be passed as parameters to compile the code. Java code was compiled using Maven. This environment was chosen to apply the algorithm in a "pure" state without modifications or other support applications.

Figure 1 depics a graphic representation of how the code was organized in the Linux computer.

It is important to highlight that keys are provided like arrays of bytes with no standard key storage component like key store, and this code already works

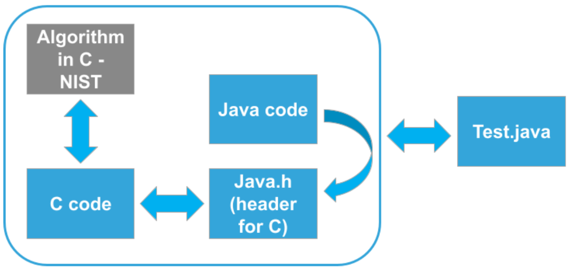

Figure 1: Scheme showing how the code was built.

with Serialized Objects in Java.

The OpenSSL library was required to build de code. Thus was necessary to compile it for the Android architecture. A compiled version of OpenSSL in Android architecture was downloaded to be used in Android code. OpenSSL headers code is used in Android to compile Kyber's code. C code was compiled with NDK compiler (version 21.3.6528147) and the Java code was generated using JNI to communicate with Kyber's code.

After analysing the code functions it was necessary to execute the algorithm and collect some data. This analysis may help one to understand where the algorithm presents more complexity to execute its functions. For that it was necessary profiling all code.

5 DATA COLLECTED

The goal is evaluating Kyber algorithm running in a mobile prototype application. In this app a session key is created as a plaintext, and will be protected with a pair of public and private keys generated according to Kyber algorithm. The session key from this app is encrypted using the public key and the encrypted text is decrypted using the private key. Key obtained is checked to be correct.

Kyber algorithm was tested in a total of 30 times with 1000 executions in loop each time. Each execution had the following requirements:

1. Different public and private keys are generated.
2. Plaintext is broken in blocks of size equals to algorithms block size.
3. Padding is used when necessary.

As one did not find an appropriate tool to generate a satisfactory analysis for these different scenarios, was decided to apply manually some time stamp into the code to collect each time spent on each function called into execution. This allowed one to verify both scenarios in a single way.

After defining how to collect the data, it was necessary to define a sample size for the study, and then choose 30 executions for each algorithm in each scenario. Generally, if the number of code runs collecting data is greater than 25, these approximations will be good (Hogg et al., 2015). Both algorithms are always running in equal conditions: generating a pair of keys, encrypting a key session and decrypting it. A key session was chosen given that it was an indispensable parameter, thus it was the object to which the cryptographic algorithms were applied. As we know these algorithms work in a cipher block scheme (size block = 32), and for this chosen object it will always be necessary seven blocks (seven), it was easy to see that in the execution round that encrypt and decrypt was called seven times each.

Finally, we present a data analysis for KYBER algorithms running in an x86 and Android environment. The time spending presented in graphs and tables are measured in microseconds.

6 DATA ANALYSIS

The first point that can be made in this analysis is the distribution of processing time in each cycle of execution of this study. It is well known that there is a greater expense in the action of encryption. The generation of keys is not as expensive, as it should be, and the decryption process ends up not being long compared to the encryption process.

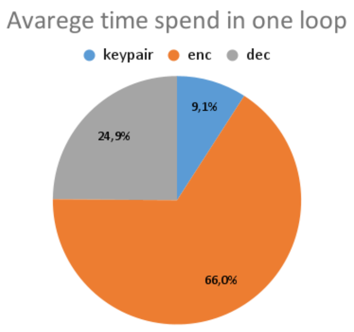

Figure 2: Avarege time spend in one loop

This greater work in encryption was already somewhat expected since it is a process, after the generation of keys, that builds the computational complexity necessary to protect confidentiality.

By analyzing both environments, it is observed that ARM architecture present in evaluated mobile can be faster than classical x86 computer architecture with code improvements. In general the decrypt action and the work with matrices in key pair generation and encrypt action in ARM are slower than x86.

In contrast, other functions in ARM are faster than x86. In the end, ARM is slower, but it shows room for improvement.

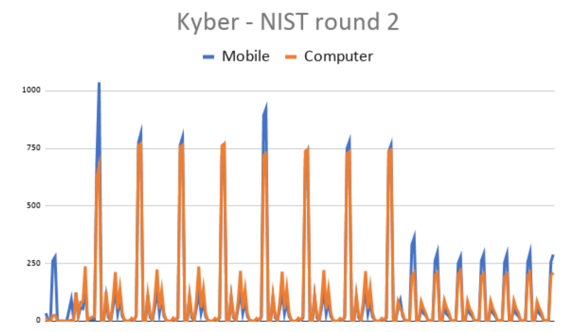

Figure 3: Data collected in Kyber NIST Round 2 version

Analysing the KYBER's code submitted in Round 2 (Figure 3) and Round 3 (Figure 4) was observed:

- Average execution time for Linux was increased;
- Average execution time for Android was reduced;
- Top spend actions were kept.

For Round 3 it is notorious how the time increased in Linux, so because of this it was necessary to compare if it was increased for smartphone too. However, we can identify that in ARM processors the time spent was reduced, which made us realize this algorithm was optimized for this processor family.

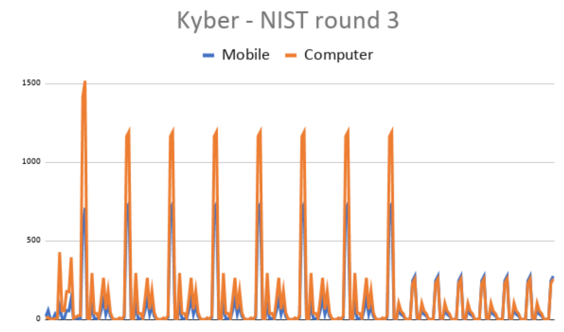

Figure 4: Data collected in Kyber NIST Round 3 version

As a conclusion stands out, KYBER was optimized for mobile evaluated processor architecture in the newest NIST submission. Average time spent was reduced by 10.67%.

Analyzing the algorithms that were submitted to NIST in round 2 and round 3, it is clear that there were average execution time for Linux was increased and in return average execution time for Android was reduced. But the top spend actions were kept, that is, the same actions that are more "expensive" in time spent continue to lead, only reaching much smaller numbers.

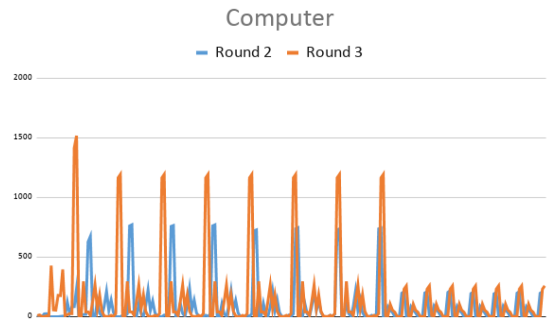

Figure 5: Comparing performance for Kyber Round 2 and Round 3 versions running in a computer

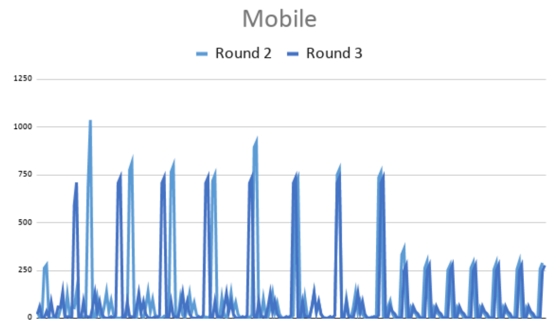

Figure 6: Comparing performance for Kyber Round 2 and Round 3 versions running in a mobile

7 CONCLUSION

Kyber was optimized for ARM processor architecture in the newest NIST submission, and its greatest improvement was be more efficient uniform sampling of the public matrix A. "Instead of sampling uniformly-random integers modulo 3329 by using rejection sampling on a 2-byte integer, we now use rejection sampling on a 12-bit integer. While the new rejection rate per coefficient is higher (i.e. $\approx$ 20%), the total number of required bits and the running time of key generation are noticeably smaller." (Avanzi et al., 2020, p. 2).

Kyber's Round 3 submission average time spent was reduced by 10,67%. It can be concluded then that the use of the Kyber algorithm has been optimized for use in the studied mobile device. In fact, we can highlight the positive use of this algorithm in smartphones.

REFERENCES

Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2020). Crystals-kyber: Algorithm specifications and supporting documentation. Technical report.

Dang, V. B., Farahmand, F., Andrzejczak, M., and Gaj, K. (2019). Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 206–214.

Hecht, P. (2017). Post-quantum cryptography(pqc): Generalized elgamal cipher over gf(251ˆ8). *CoRR*, abs/1702.03587.

Hogg, R. V., Tanis, E., and Zimmerman, D. (2015). *Probability and Statistical Inference*. Pearson, 9 edition.

Howe1, J., Prest1, T., , and Apon, D. (2019). Sok: How (not) to design and implement post-quantum cryptography. Cryptology ePrint Archive, Report 2021/462.

Khalid, A., McCarthy, S., Liu, W., and O'Neill, M. (2019). Lattice-based cryptography for iot in a quantum world: Are we ready? Cryptology ePrint Archive, Report 2019/681.

PQC-NIST (2017). *Post-Quantum Cryptography*. National Institute of Standards and Technology, Gaithersburg.

Saarinen, M.-J. O. (2020). Mobile energy requirements of the upcoming nist post-quantum cryptography standards.

Xu, R., Cheng, C., Qin, Y., and Jiang, T. (2018). Lighting the way to a smart world: Lattice-based cryptography for internet of things. *CoRR*, abs/1805.04880.