Performance Benchmarking of CRYSTALS-Kyber, a Quantum-Resistant Cryptography Algorithm

Denver Earl Paul S. Bugaoisan
Nuan Patricia S. Serrano
dsbugaoisan@up.edu.ph
nsserrano1@up.edu.ph
University of the Philippines - Diliman
Quezon City, Metro Manila, Philippines

ABSTRACT

With the possibility that commonly used cryptography algorithms can be solved quickly through the use of quantum computers following Shor's algorithm, the National Institute of Standards and Technology (NIST) held a competition for quantum-resistant algorithms in 2016. One of the finalists is CRYSTALS-Kyber, a key encapsulation mechanism(KEM) that is based on the Module Learning with Errors (MWLE) problem over lattices. This research aims to benchmark the performance of CRYSTALS-Kyber on multiple computing platforms, including an HPC, a desktop and laptops.

The study evaluates the number of iterations a CRYSTALS-Kyber's operation is executed over a given number of seconds ($n = 3$). The operations evaluated are key generation, encapsulation, decapsulation. Factors such as programming language, processor architecture, clock speed and memory capacity are considered to understand their effect on the algorithm's performance.

By benchmarking CRYSTALS-Kyber on various computing machines, the research provides insights into the feasibility, compatibility and potential bottlenecks of deploying quantum-resistant cryptography in different computing environments.

The results indicate that among the cryptographic operations tested, Key Generation stands out as the fastest. Furthermore, Kyber512 emerges as the quickest variant, while operations implemented in C demonstrate the highest programming language efficiency. Notably, the MacBook M1 Pro 2020 proves to be the fastest machine across the tested devices. Additionally, the findings emphasize a direct correlation between CPU clock rate and the speed of iterations per second, highlighting that higher clock rates enhance computational performance.

With that, the type of operation, Kyber variant, programming language, and hardware architecture influence the performance of the Kyber algorithm. The CPU architecture and CPU clock rate of a machine can also significantly impact runtime.

UPD DCS Computer Security Lab, June 21, 2024, Quezon City, Philippines

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

KEYWORDS

CRYSTALS-Kyber, Kyber, Benchmarking, Cryptography, Post-Quantum Algorithms, Quantum-Resistant Cryptographic Algorithms, Performance Evaluation, Lattice-Based Cryptography

1 INTRODUCTION

Cryptography is the study of securing messages and/or data through encryption and decryption. It is used to prevent unintended users to access information and maintain confidentiality. Encryption transforms the information into cipher text or unintelligible information for unauthorized users, while decryption reverses that process and makes the cipher text into ordinary text [9]. To do that, different kinds of cryptography algorithms are used.

There are two main types of cryptography based on the type of encoding used now, the secret key cryptography and public key cryptography. In secret key cryptography (SKC), the sender and the receiver share a key used to encrypt and decrypt the message. Some cryptography algorithms that do SKC are DES, AES and RC6. While in public key cryptography (PKC), two kinds of keys are used, the public key and the private key. The public key is used for encryption while the private key is for decryption. Some cryptography algorithms that do PKC are RSA, EES and Digital Secure [1][8]. Rivest-Shamir-Adleman (RSA) systems, a strong public key cryptography algorithm uses prime factorization as their method. The private key of a user are two big prime numbers, while their public key is the product of those two numbers. The algorithm is secure despite being simple because factoring the product of prime numbers takes computers a long time to compute.

However, in 1995, Peter Shor, an applied mathematics professor, was able to develop an algorithm called Shor's Algorithm, that can factor large numbers, solve discrete logarithm (DLP) and elliptic curve discrete logarithm (ECDLP) problems exponentially faster than classic algorithms using quantum mechanics[13]. These problems are used by a number of cryptography algorithms. Hence, if Shor's Algorithm is applied, the cryptography algorithm becomes easy to decipher [14][2].

Quantum mechanics is a branch of physics that study quantum particles and their properties[7]. While quantum computing is a type of computing based on quantum mechanics. Quantum computers are also based on this and use quantum bits (qubits). Unlike binary digits that can be either 1 or 0, qubits can be 1 and 0 at the same (by the principle of superposition)[2].

Shor's algorithm is a quantum algorithm, thus it uses quantum mechanics to solve problems very quickly, in polynomial time

$(O(n^2(\log n)(\log\log n)))$to be specific[13]. While Shor's algorithm has been proven to work, it has only been tested to solve a prime number equation. A quantum computer that can solve very large prime number equations has not been made yet as there are not enough stable qubits to be used. There has to be $(2n) + 3$ stable qubits, with n as the number of key bits, to crack the encryption. This means that 4099 qubits are needed to solve 2048-bit RSA key. As of now, there are less than 100 qubits that can be used[7]. However, now that the possibility that classic cryptography algorithms can be broken quickly exists, companies, organizations and different people around the world are looking for new algorithms that are quantum-resistant to prepare for the future.

With that, the National Institute of Standards and Technology (NIST) held a competition calling for post-quantum cryptography algorithms in 2016. In 2022, NIST announced the first four cryptography algorithm finalists that will be up for standardization after six years. These algorithms are based on math problems on structured lattices and hash functions. The algorithms for digital signatures are: CRYSTALS-Dilithium, SPHINCS+, and FALCON. While for general encryption, the algorithm selected is the CRYSTALS-Kyber[11].

The CRYSTALS-Kyber is a key encapsulation mechanism(KEM) that is based on Module Learning with Errors (MWLE) problem over lattices. The algorithm is also easily scalable when larger key sizes would be required. [5][7]. It is also main focus of this research project.

With that, this project aims to benchmark the CRYSTALS-Kyber cryptography algorithm (in its original implementation in C and Python implementation) and answer these questions:

(1) Is there a significant difference in speed between the C and Python implementation of CRYSTALS-Kyber?
(2) Is there a significant difference in speed when running the program on a machine with "better" specifications?

2 STATEMENT OF THE PROBLEM

2.1 Problem

The original implementation of the CRYSTALS-Kyber algorithm is in C and C is a lower level programming language than Python, does the language used affect the run time of the program?

A computer's performance depends on its specifications. With that, how significant is its effects to the run time of the program?

To summarize, how do factors like the language and the machine used affect Kyber run times?

2.2 Hypotheses

Null Hypothesis: Factors like the programming language, machine used and machine configuration have no significant effect on the run time of the Kyber algorithm.

Alternative Hypothesis: Factors like the programming language, machine used and machine configuration have a significant effect on the run time of the Kyber algorithm.

3 RELATED STUDIES

3.1 CRYSTALS-Kyber

The paper "CRYSTALS-Kyber algorithm specifications and supporting documentation (version 3.02)"[4] by Avanzi et al. provided detailed specifications and documentation for the CRYSTALS-Kyber algorithm. It discussed the improvements made to the Kyber algorithm with the aim of enhancing security and performance in quantum cryptography. The document also covered performance analysis, implementation details, and the impact of recent developments in lattice-based cryptography on security estimates. Additionally, it addressed the limitations of previous estimation techniques and the need for refined security analysis based on the latest developments in the field. Avanzi et al. also benchmarked their Intel Benchmarked Intel Core i7-4770K (Haswell) processor clocked at 3492 MHz and made specific optimization to their codebase for faster runtimes in the Haswell chip.

3.2 Open Quantum Safe Project

The paper "Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project"[15] by Douglas Stebila and Michele Mosca explores the development of quantum-resistant cryptographic systems and their implementation to secure internet communications against future quantum computer attacks. It highlights the importance of post-quantum cryptography and specifically discusses the Open Quantum Safe (OQS) project, an open-source initiative dedicated to prototyping and evaluating quantum-resistant cryptographic algorithms.

At the heart of the OQS project is `liboqs`, a C library designed to provide a comprehensive collection of quantum-resistant cryptographic algorithms. This library is a critical tool for testing and integrating these algorithms into existing cryptographic protocols and applications. `liboqs` stands out for its diverse algorithm collection, encompassing various quantum-resistant methods based on different hard mathematical problems, such as lattice-based, hash-based, code-based, and multivariate polynomial problems. The benchmark metric such as the iterations per second was derived in the `liboqs` library. Relevant registers to be observed were derived from `liboqs` library benchmark code.

3.3 Other Implementations

Moura in his paper, "High-Assurance, High-Speed Post-Quantum Cryptography in Safe Rust"[10], addressed the potential risks posed by quantum computers to current public-key cryptosystems, the challenges of flawed cryptographic software implementations, and the development of quantum-resistant algorithms. It introduced the basics of RSA and presents solutions for post-quantum cryptographic algorithms. The document discussed the Jasmin framework, the Rust programming language, and the implementation of RjKyber, a post-quantum cryptographic library. The benchmark results boasted faster runtimes than the compared Rust library counterpart, `pqc-library`. It is also worth noting that the Moura's implementation performed in microseconds signifying that a compiled language such as Rust performs significantly better than interpreted languages.

Viet et al.'s paper, "Implementation and Benchmarking of Round 2 Candidates in the NIST Post-Quantum Cryptography Standardization Process Using Hardware and Software/Hardware Co-design Approaches"[6], covered 26 candidates and highlighted six with the highest coverage in terms of implementations and related publications: NewHope, CRYSTALS-Kyber, FrodoKEM, Saber, Round5, and

SIKE. The authors noted that these candidates have both high-speed and lightweight implementations reported. It also emphasized the significant advantages of certain candidates over others in terms of execution times, power consumption, and energy usage. Kyber performed on the lower percentile based on the execution time benchmarks.

4 MATERIALS AND METHODS

4.1 Machines Utilized and their Specifications

The machines utilized in this study, along with their specifications, are outlined below. It is important to highlight that for the sake of consistency in results and accurate conversion from CPU cycles to CPU time in the C benchmark, specific adjustments were made to the CPU clock frequencies. The CPU clock of the Ryzen 3600 Desktop was overridden and set to 2.6 GHz, 3.2 GHz and 4.0 GHz for sensitivity analysis. The RAM Configuration was changed as well and is specified below. These deliberate modifications serve to maintain a standardized testing environment and to determine the machine components that affects the key generation, encapsulation and decapsulation performace, allowing for precise and equitable assessments of performance across the benchmarked devices.

(1) Desktop PC
- **CPU:** Ryzen 3600
- **Storage:** 1 TB NVMe SSD
- **RAM Configurations:**
  - 32 GB DDR4-3200, 16 GB DDR-3200, 8 GB DDR4-3200
  - 32 GB DDR4-3200, 32 GB DDR-2400, 32 GB DDR4-1600
- **OS:** Windows 11 (using WSL)
- **Clock Rate Configurations:** 2600 MHz, 3200 MHz, 4000 MHz

(2) Laptop
- **CPU:** Ryzen 3500U
- **Storage:** 500 GB NVMe SSD
- **RAM:** 10 GB DDR4-2400
- **OS:** Windows 11
- **Max Boost Clock:** 3700 MHz

(3) Macbook Pro M1 2020
- **CPU:** Apple M1 chip with 8-core CPU
- **Storage:** 256 GB SSD
- **RAM:** 16 GB unified memory
- **OS:** MacOS Sonoma 14.0
- **Max Boost Clock:** 3200 MHz

(4) DOST-ERDT High Performance Cluster
- **CPU:** 2x AMD EPYC 7513 32-Core Processor
- **RAM:** 16x 16GB DRAM
- **OS:** CentOS Linux
- **Clock Rate:** 2600 MHz
- **No. of Machines:** 16
- **Nodes Used:** 4

4.2 Programs Utilized

These two programs were utilized for benchmarking:

- Kyber Original C Implementation - Github link[3]
- Kyber Python Implementation - Github link[12]

It is worth noting that in the Kyber Python Implementation [3], the `ref` implementation which is the unoptimized version and closely tied with the algorithm in the CRYSTALS-Kyber is used in the benchmarking process.

4.3 Research Paradigm

(1) **Prepare Test Setup**
This involves setting up the necessary software environment for the benchmarking process. This ensures that the test environment is properly configured and ready for execution. Different applications that are irrelevant in the benchmark of each implementations were shutdown in order to avoid slowdowns and interruptions during each execution of relevant functions (i.e. Key Generation, Encapsulation, Decapsulation).

(2) **Develop and run the Benchmark Program**
A C program is designed to benchmark the performance of three cryptographic operations: key pair generation, encapsulation, and decapsulation. The program defines three arrays (`t1`, `t2`, and `t3`) to store the CPU cycle counts for each operation. It also defines several byte arrays to hold the public key (`pk`), secret key (`sk`), ciphertext (`ct`), and symmetric key (`key`).

In the `main` function, the program runs each of the three operations (key pair generation, encapsulation, and decapsulation) in a loop for a duration of 3 seconds each. The start time of each operation is recorded using the clock function, and the loop continues until 3 seconds have passed. Inside each loop, the current CPU cycle count is recorded using the `cpucycles` function before the operation is performed. The index variables `i`, `j`, and `k` are used to store the CPU cycle counts in the respective arrays and to keep track of the number of times each operation was performed. After each loop, the program prints the number of times or iterations the operation was performed.

A shell script was also developed in order to execute the C program for 30 iterations ensuring precise benchmark results.

(3) **Record the Output**
The number of iterations executed by the benchmark program within a specific number of seconds ($n = 3s$) are outputted in CSV format.

(4) **Remove Outliers with Z-Score**
Outliers for each Kyber level and configuration were removed using a threshold of 3 (`threshold_z = 3`) in Jupyter Notebook using the SciPy Python library.

(5) **Plot the Data**
The benchmark data were divided by 3 to calculate the iterations per second metric, and then the mean was computed. The obtained results were plotted in a bar graph to visualize and identify trends.

(6) **Interpret Results**
The final step involved drawing conclusions, comparisons, and insights from the benchmarking.

5 RESULTS AND DISCUSSION

5.1 Different Machines

5.1.1 C Implementation. Figure 1 illustrates a comparative analysis of the performance of various Kyber cryptographic operations—Key Generation (Keygen), Encapsulation (Enc), and Decapsulation (Dec)—across different devices and Kyber variants in the C Implementation. The devices evaluated include the M1 MacBook Pro, HPC @2.6 GHz, Ryzen 3500U, and Ryzen 3600 @2.6 GHz, while the Kyber variants tested are Kyber 512, Kyber 768, and Kyber 1024. Performance metrics are presented in terms of average iterations per second, where higher values indicate superior performance.

For Kyber 512, the M1 MacBook Pro exhibits the highest performance across all operations. Specifically, for Keygen, the M1 MacBook Pro achieves 50,681 iterations per second, followed by HPC @2.6 GHz with 31,798 iterations per second, Ryzen 3500U with 22,799 iterations per second, and Ryzen 3600 @2.6 GHz with 19,121 iterations per second. In encapsulation and decapsulation tasks, the M1 MacBook Pro similarly demonstrates the best performance, followed by HPC @2.6 GHz, Ryzen 3500U, and Ryzen 3600 @2.6 GHz, in descending order of performance.

Moving to Kyber 768, again, the M1 MacBook Pro maintains its leading position across keygen, encapsulation, and decapsulation operations. It achieves 30,119 iterations per second for Keygen, 26,214 iterations per second for Enc, and 23,387 iterations per second for Dec. The performance of other devices follows a similar pattern across these operations.

For Kyber 1024, the M1 MacBook Pro continues to outperform other devices with 19,252 iterations per second for Keygen, 17,604 iterations per second for Enc, and 16,037 iterations per second for Dec. HPC @2.6 GHz, Ryzen 3500U, and Ryzen 3600 @2.6 GHz show progressively lower performance across these operations.

5.1.2 Python Implementation. Figure 2 shows the results of the test for the Python implementation. In the case of Kyber 512, the M1 MacBook Pro significantly outperforms the other devices in all three operations. For key generation, it achieves 253 iterations per second, while the HPC @2.6 GHz, Ryzen 3500U, and Ryzen 3600 @2.6 GHz achieve 136, 73, and 111 respectively. Similarly, in encapsulation, the M1 MacBook Pro leads with 167 iterations per second, followed by HPC @2.6 GHz with 89, Ryzen 3500U with 49, and Ryzen 3600 @2.6 GHz with 76. For decapsulation, the M1 MacBook Pro reaches 106 iterations per second, with the other devices trailing at 56, 32, and 48 respectively.

The performance trend persists in the Kyber 768 variant. The M1 Macbook Pro performs key generation at 165 iterations per second, with HPC @2.6 GHz at 89, Ryzen 3500U at 50, and Ryzen 3600 @2.6 GHz at 74. For encapsulation, the M1 MacBook Pro achieves 112 iterations per second, while HPC @2.6 GHz, Ryzen 3500U, and Ryzen 3600 @2.6 GHz achieve 60, 34, and 51 respectively. Decapsulation on the M1 MacBook Pro reaches 72 iterations per second, compared to 39, 22, and 33 on the HPC, Ryzen 3500U, and Ryzen 3600 @2.6 GHz respectively.

For the Kyber 1024 variant, the M1 MacBook Pro continues to lead with 112 iterations per second in key generation, whereas HPC @2.6 GHz, Ryzen 3500U, and Ryzen 3600 @2.6 GHz achieve 61, 33, and 50 respectively. In encapsulation, the M1 MacBook Pro performs at 80 iterations per second, followed by HPC @2.6 GHz with 43, Ryzen 3500U with 24, and Ryzen 3600 @2.6 GHz with 37. decapsulation on the M1 MacBook Pro is recorded at 53 iterations per second, while the other devices lag at 29, 16, and 24 respectively.

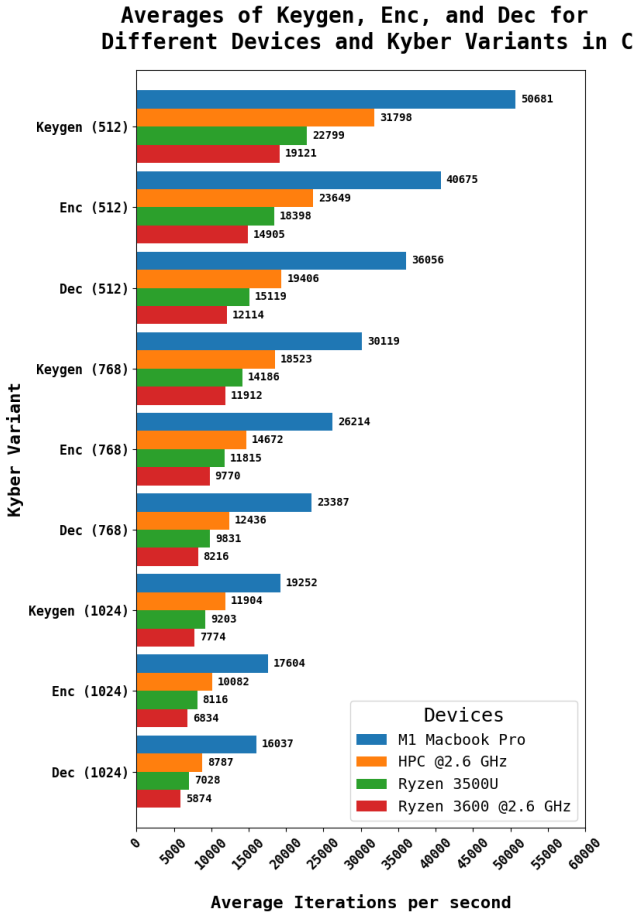

Figure 1: Benchmark scores across different machines using the C implementation.

5.2 Different Configurations

To assess the sensitivity of the Kyber implementation to machine configuration, a sensitivity analysis was conducted. The Ryzen 3600 desktop machine was selected for this analysis due to its high configurability compared to the other three machines tested. The components reconfigured in the test were RAM Clock Speed, RAM Capacity and CPU Clock Rate.

5.2.1 RAM Capacity. The Ryzen 3600 desktop machine was tested in different RAM capacities namely: 8 GB, 16 GB and 32 GB. The average iterations per second is highlighted in Figure 3. Across all Kyber variants and operations, there is minimal variation in performance across the different RAM Capacity. This suggests that the RAM speed has a negligible impact on the average iterations per second for these specific cryptographic operations.

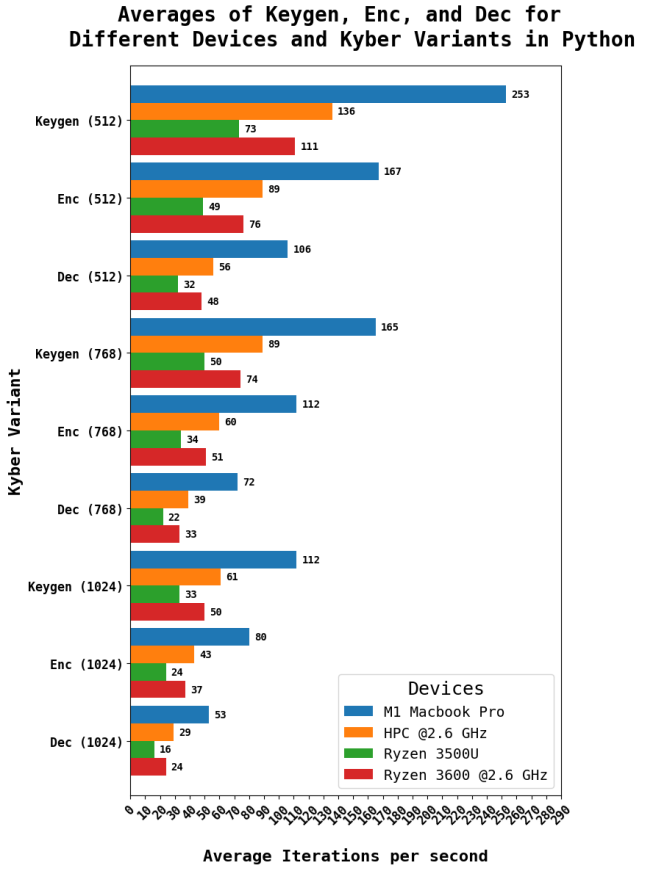

Figure 2: Benchmark scores across different machines using the Python implementation.

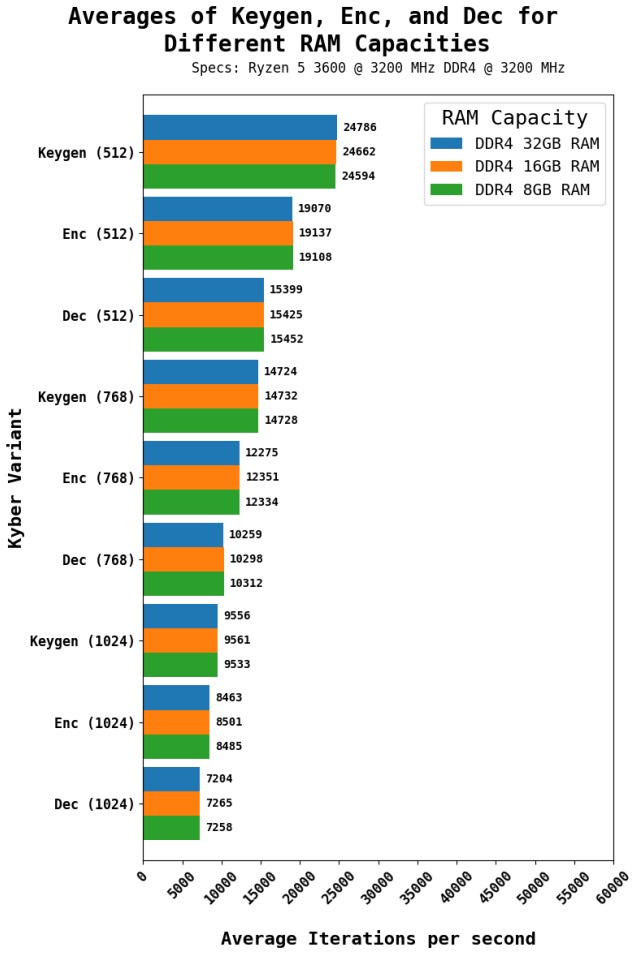

Figure 3: Benchmark scores in different RAM Capacities of the Ryzen 3600 desktop machine.

5.2.2 RAM Clock Speed. The Ryzen 3600 desktop machine was tested with RAM clock speeds of 1600 MHz, 2400 MHz, and 3200 MHz. The average iterations per second for each speed are shown in Figure 4. Similar to the results observed with varying RAM capacity, the graph indicates minimal variation in performance across different RAM clock speeds. This minimal sensitivity suggests that the primary limiting factor for performance in Kyber operations is likely the CPU speed or architecture, rather than the RAM speed.

5.2.3 CPU Clock Rate. The Ryzen 3600 desktop machine was tested with CPU clock rate of 2600 MHz, 3200 MHz, and 4000 MHz. The average iterations per second for each speed are shown in Figure 5. Across all Kyber variants and operations performance consistently improves as the CPU clock speed increases. This indicates that the Kyber implementation's performance is quite sensitive to the CPU clock speed. Higher clock speeds lead to better performance across all operations and security levels.

5.3 Discussion

Given the results of the benchmarking:

5.3.1 ***Fastest Programming Language:*** *C.* The analysis confirms the hypothesis that C implementations outperform Python implementations. C's direct hardware interaction translates to faster execution compared to Python's higher-level abstractions.

5.3.2 ***Fastest Operation:*** *Key Generation.* Key generation emerges as the fastest operation, followed by encapsulation and then decapsulation. This aligns with the inherent difficulty of each step, with key generation being a relatively straightforward process compared to the information exchange involved in encapsulation and decapsulation.

5.3.3 ***Fastest Variant:*** *Kyber512.* The Kyber512 variant exhibits the highest performance across all operations. This is attributed to its lower security level (NIST level 1) compared to Kyber1024 (NIST level 5). The increased security demands of Kyber1024 necessitate more complex computations, leading to a slower runtime.

5.3.4 ***Fastest Machine:*** *Macbook Pro M1 2020.* The M1 Macbook Pro showed the most number of iterations per second in terms of runtime across all Kyber variants and programming languages. This suggests that CPU architecture and operating system plays

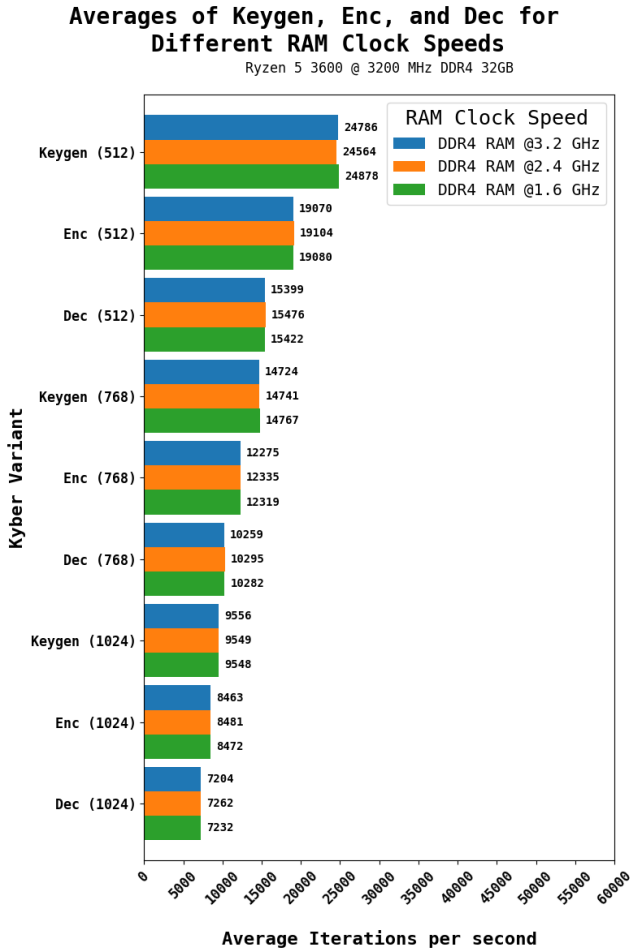

Figure 4: Benchmark scores in different RAM Clock Speeds of the Ryzen 3600 desktop machine.

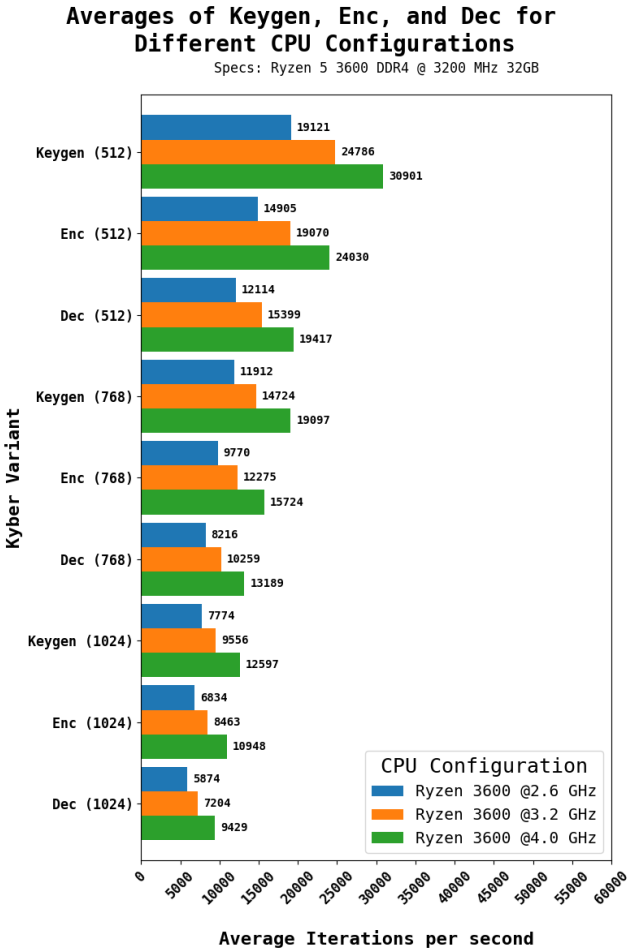

Figure 5: Benchmark scores in different CPU Clock Rate of the Ryzen 3600 desktop machine.

a significant role in performance alongside clock speed. Interestingly, the HPC cluster, despite boasting the highest core count, falls behind. This highlights the importance of factors beyond raw processing power, potentially indicating limitations in parallel computing frameworks for these Kyber Implementations.

5.3.5 ***Sensitivity Analysis of the Desktop Machine.*** The results provided demonstrate that the performance of the Kyber key encapsulation mechanism is significantly influenced by CPU clock speed but shows minimal variation with changes in RAM clock speed and capacity. Specifically, higher CPU clock speeds consistently result in improved performance across all Kyber variants and operations (Keygen, Enc, Dec), indicating a strong dependency on CPU capabilities. In contrast, both RAM speed and capacity have negligible impacts on performance, highlighting that Kyber operations are primarily CPU-bound and not significantly constrained by memory resources. Therefore, for optimal performance in Kyber implementations, focusing on enhancing CPU specifications is more beneficial than investing in faster or larger RAM.

6 SCOPE AND DELIMITATION

Due to limited resources, the researchers were able to utilize four machines to benchmark the Python and C implementation of the algorithm as listed in section 4. Materials and Methods. Additionally, the researchers focused on testing the speed of the programs and not its efficacy.

7 RECOMMENDATIONS FOR FURTHER WORK

This research project only tested the running time of the Kyber program in its C and Python implementations on a limited number of devices. The researchers recommend extending tests for a wider scope. This includes testing on other devices like microcontrollers and modify implementations that can utilize all CPU cores. They also further recommend to standardize testing by using uniform programs, dependency versions and equal configuration (i.e. CPU clock rate and operating system) across all machines that will be

tested. The researchers also recommend evaluating other KEM algorithms and comparing them with the Kyber algorithm. Additionally, testing the security of the algorithm besides its speed is also advised.

8 CONCLUSION

This analysis emphasizes how the type of operation, Kyber variant, programming language, and hardware architecture influence the performance of the Kyber algorithm. Factors like CPU architecture and CPU clock rate can also significantly impact runtime. Further research could focus on optimizing Kyber for HPC clusters to leverage their parallel processing capabilities. Additionally, studying the performance of alternative post-quantum cryptography schemes on various hardware platforms would help in selecting the most efficient solution for specific security needs.

9 APPENDIX

This github repository contains the codes and benchmarking program used by the researchers.

ACKNOWLEDGMENTS

We would like to extend our appreciation to our adviser, Ma'am Susan Pancho-Festin for her guidance and support towards our project. Thank you to the DOST-ERDT for providing access to an HPC. Lastly, thank you to our friends and family!

REFERENCES

[1] O. G. Abood and S. K. Guirguis. 2018. A Survey on Cryptography Algorithms. *International Journal of Scientific and Research Publications* (2018). https://doi.org/10.29322/IJSRP.8.7.2018.p7978

[2] J. Aumasson. 2017. *Serious Cryptography: A Practical Introduction to Modern Encryption.* No Starch Press.

[3] R. Avanzi et al. 2021. Kyber. https://github.com/pq-crystals/kyber.

[4] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2019. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round* 2, 4 (2019), 1–43.

[5] Cryptographic Suite for Algebraic Lattices (CRYSTALS). [n. d.]. *Kyber: Introduction.* https://pq-crystals.org/kyber/index.shtml

[6] Viet B Dang, Farnoud Farahmand, Michal Andrzejczak, Kamyar Mohajerani, Duc T Nguyen, and Kris Gaj. 2020. Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches. *Cryptology ePrint Archive: Report 2020/795* (2020).

[7] R. Grimes. 2020. *Cryptography Apocalypse: Preparing for the Day When Quantum Computing Breaks Today's Crypto.* John Wiley Sons, Inc.

[8] A. Gupta and N. Walia. 2014. Cryptography Algorithms: A Review. *International Journal of Engineering Development and Research* (2014).

[9] D. Liu. 2009. *Next Generation SSH2 Implementation: Securing Data in Motion.* 41–64 pages. https://doi.org/10.1016/B978-1-59749-283-6.00003-9

[10] Leonardo Fernandes Moura. 2022. High-Assurance, High-Speed Post-Quantum Cryptography in Safe Rust. (2022).

[11] National Institute for Science and Technology. [n. d.]. *NIST Announces First Four Quantum-Resistant Cryptographic Algorithms.* https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms

[12] G. Pope. 2023. CRYSTALS-Kyber Python Implementation. https://github.com/GiacomoPope/kyber-py.

[13] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (Oct. 1997), 1484–1509. https://doi.org/10.1137/s0097539795293172

[14] N. Singh. 2023. *Cracking Quantum Computing Interview: A Comprehensive Guide to Quantum Computing Interview Preparation.* self-published.

[15] Douglas Stebila and Michele Mosca. 2017. Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. In *Selected Areas in Cryptography – SAC 2016*, Roberto Avanzi and Howard Heys (Eds.). Springer International Publishing, Cham, 14–37.