



**QUEEN'S
UNIVERSITY
BELFAST**

Quantum-Light:

**A Light-weight Quantum-resistant Public
Key Encryption scheme on an ARM
processor.**

A dissertation submitted in partial fulfilment of the
requirements for the degree of
BACHELOR OF SCIENCE in Computer Science

In

The Queen's University of Belfast

by

Nathan Whaley

April 15th, 2024

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Nathan Whaley Student Number: 40284751
Project Title: Quantum-Light: A Light-weight Quantum-resistant Public Key
Encryption scheme on an ARM processor.
Supervisor: Dr Ayesha Khalid

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

By submitting your dissertation you declare that you have completed the tutorial on plagiarism at <http://www.qub.ac.uk/cite2write/introduction5.html> and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.

6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students.

Student's signature

Nathan Whaley

Date of submission

15/04/2024

Acknowledgements

First and foremost, I would like to thank God. He has given me strength and encouragement throughout all the ups and downs of completing this project and dissertation. I am truly grateful for His unconditional and endless love, mercy, and grace. I would also like to extend my gratitude to Queen's University for their support and resources, which have been essential to the success of this project.

Abstract

This dissertation explores the implementation of lightweight quantum-resistant encryption algorithms on ARM processors, addressing the threat posed by quantum computing to traditional encryption methods. When integrating variants of the lightweight Ascon hash into the Kyber-512 algorithm, when ran on a Nucleo L4R5ZI Arm M4 chip, a number of positive results were obtained with the target of increasing its suitability for lightweight and embedded applications. Significant improvements in performance were observed with as much of a ~25% reduction in clock cycle use compared to reference versions of Kyber. Improvements in memory usage were also found which with as much as ~8% in the most memory-optimised scheme, all while maintaining an improvement in performance. The findings illustrate the feasibility of lightweight, efficient encryption solutions for resource-constrained embedded systems, strengthening data security in the face of advancing technology. This research contributes insights into the optimization of encryption algorithms, highlighting the potential for practical deployment in low-power computing environments.

Contents

1. Introduction and Problem Area.....	6
2. Related Work.....	7
3. Solution Description and System Requirements.....	9
4. Design	15
5. Implementation	29
6. Testing.....	36
7. Experimental Results and Analysis	41
8. Conclusions and Future Work.....	46
9. References	49
10. Appendices.....	51

1. Introduction and Problem Area

The coming issue of the realisation of Quantum computers to a sufficient degree to pose a significant threat to all classical forms of encryption is a major threat in the field of cybersecurity. Traditionally, data was encrypted with algorithms that posed sufficient challenge that it would take a classical computer an extremely large amount of time to decipher encrypted data when the encryption method being used is not known. Quantum computers have the potential to break many of the most popular encryption methods used to secure data today. This is due to their ability to perform calculations differently and exponentially faster than classical computers. One example of this would be Shore's algorithm. [1]

To combat this future threat, a number of "post-quantum" encryption algorithms have been developed to be secure against both classical and quantum computers. The National Institute of Standards and Technology (NIST) held a post-quantum cryptography competition to standardise a number of quantum resistant algorithms. [2] The winner of this competition was announced in 2022 as "CRYSTALS-Kyber" and its open-source implementation made available on the NIST website in C. While Kyber may be fit for larger devices with a greater number of system resources available, the SHA-3 family of hashing algorithms it uses are too resource intensive to be suitable for many smaller devices such as embedded systems. This poses a problem as the current shift in the technological landscape away from PCs to smaller devices like smartphones and embedded systems demands the need for post-quantum encryption algorithms that are lightweight enough to be run on low-resource and low-power installations. [3]

The Kyber algorithm is a Key Encapsulation Mechanism (KEM) [4], comprising of 3 main components: key generation, encryption and decryption. Each aspect of the KEM is heavily reliant upon several of the SHA-3 family of hashing algorithms, namely SHA3-256, SHA3-512, SHAKE128 and SHAKE256. The aim of the project will be to determine if it is possible to implement the recently standardised CRYSTALS-Kyber post-quantum encryption algorithm on a lightweight device, specifically a NUCLEO L4R5ZI board with an ARM Cortex-M4 chip, by substituting the above hashing algorithms for the more lightweight ASCON [5] algorithm. The ASCON algorithm from the NIST Lightweight cryptography competition called the NIST LWC [6] is better suited for lower-resource applications such as embedded systems and IoT applications. The consequences of this replacement in terms of code/memory/stack sizes, throughput efficiency, security levels, etc. will be thoroughly investigated in this project. A fully working implementation of the Kyber encryption system with Key generation, encryption and decryption operations using a lightweight hashing algorithm on an ARM Cortex-M4 chip will be the goal. The ARM Cortex-M4 is an extremely widely used chip [7] so will be effective at demonstrating suitability of a lightweight post-quantum encryption algorithm for a wide range of embedded devices.

2. Related Work

The following section investigates the existing literature and research efforts in the field of lightweight post-quantum cryptography with a focus on cryptographic systems tailored for IoT devices.

Exploring NIST LWC/PQC Synergy with R5Sneik [8]

This study notes that to use any given NIST PQC algorithm; its internal symmetric components must also be implemented on all target platforms. This can be problematic for lightweight, embedded, and hardware implementations. The study details that it has been widely observed that current NIST-approved symmetric components (AES, GCM, SHA, SHAKE) form a major bottleneck on embedded and hardware implementation footprint and performance for many of the most efficient NIST PQC proposals. It then goes on to discuss R5Sneik, a variant of Round5 that internally uses SNEIK 1.1 permutation-based primitives instead of SHAKE and AES-GCM. They found that R5Sneik is up to 40% faster than Round5 for some parameter sets on ARM Cortex M4 and has substantially smaller implementation footprint. This study shows that there is lots of potential that exists for the optimisation of algorithms in the NIST PQC competition for more lightweight implementations.

Ring-LWE Ciphertext Compression and Error Correction [9]

The study explores lattice-based public key cryptography algorithms' ability to transform ciphertext between different lattice or ring representations efficiently, without needing knowledge of public and private keys. In the study it used ciphertext transformation to compress RingLWE ciphertexts, enabling lightweight implementations in IoT devices, Smart Cards, and RFID systems without modifying encryption procedures or security parameters. Through experimentation, it was demonstrated that this compression technique reduces ciphertext size by over 40% while maintaining equivalent security, enabling public key cryptography on previously inaccessible lightweight platforms, and outperforming RSA-based approaches at similar security levels. This study further shows that the potential for modifying the PQC schemes that already exist to make them more suitable for lightweight implementations is very real and doable via a number of approaches.

A post-quantum lattice based lightweight authentication and code-based hybrid encryption scheme for IoT devices [10]

This study introduces a lightweight post-quantum lattice-based authentication and code-based hybrid encryption scheme tailored for resource-constrained IoT devices. Similar to the previous study it utilised Ring-Learning with Errors (Ring-LWE) based authentication in addition to Bernstein reconstruction in polynomial multiplication, with the scheme achieving minimal computation cost on IoT devices. The total authentication delay of the proposed authentication scheme was 23% less than the authentication scheme that is considered conventional polynomial multiplication. In summary this study showed a further avenue of modification in PQC schemes, illustrating that there exists a wide variety of potential optimisations that can be done for lightweight PQC on IoT devices.

Summary:

In summary the above studies have shown that there exists a great potential for optimisation in many areas of PQC algorithms for lightweight implementations. One difference between this project

and those that currently are published is the use of Kyber, one of the algorithms selected for standardisation in the NIST PQC competition. In the first study detailed: “Exploring NIST LWC/PQC Synergy with R5Sneik”, A NIST round 2 candidate “ROUND5” was used, however this scheme was not selected to proceed to round 3 of the competition. [11] The aim of this project is to determine whether the Kyber scheme can be modified to utilise ASCON, another NIST standard scheme to be more suitable for lightweight implementations, creating a scheme utilising only NIST-standardised algorithms.

3. Solution Description and System Requirements

3.1 Introduction

To address the problem area stated before; the impending threat of quantum computers breaking most current forms of encryption and the unsuitability of most post-quantum encryption algorithms for smaller, lower resource applications, it was essential to set out the system requirements with clarity and precision. This section will detail the requirements for the system, both functional and non-functional, all objectives that need to be met and how each of them is to be met, with verifiable success criteria accompanying each objective. Further, this section will discuss the users of the new system being developed as well as any assumptions made and constraints that occurred.

With the problem area in mind, the aim of this project from the outset was to create a post-quantum encryption scheme that is less resource intensive than current schemes that exist, meaning that it will be better suited for running on lightweight IoT devices where resources are far more limited. As seen in Section 2, there are many possible methods of optimising PQC schemes for more lightweight targets. The angle taken on this project to attempt to solve the problem was to modify the hashing scheme used inside an existing PQC scheme. This was favoured over other approaches for a number of reasons, with the main being it allows for a more straightforward solution given the time constraints of this project. Modifying other aspects of a PQC scheme to improve efficiency on lightweight devices might involve complex changes to the algorithm itself necessitating a large time-sink to become familiar with the mathematics involved. Substituting the hash function for a more lightweight one is a more straightforward procedure, reducing development time and system complexity. Further, lightweight hashing algorithms that are part of established standards, such as those endorsed by NIST, ensures there is compatibility with existing cryptographic frameworks and libraries. This simplifies integration efforts and promotes interoperability across different platforms and implementations.

With the method of modification selected, the next stage in tackling the problem was to select a host scheme for the PQC algorithm and a suitable lightweight hashing algorithm to be used as a replacement for the internal hashing algorithm. It was decided to use two NIST standardised algorithms: CRYSTALS-Kyber and ASCON. The choice of both algorithms was strategic, Kyber was selected as the winner of the NIST post-quantum cryptography competition, gaining recognition and validation from the cryptographic community. Using a NIST-standardised algorithm ensures that the new scheme will adhere to cryptographic standards and is suitable for resistance against quantum computer-based attacks. The choice of ASCON as the hashing algorithm is also nuanced. ASCON is specifically designed for low-resource environments such as embedded systems and IoT devices and was also selected to be standardised in the NIST lightweight cryptography competition. The lightweight design of ASCON is complemented by the robust security of Kyber. With this choice of algorithms, it is possible to achieve a balance between security and efficiency by integrating Kyber with ASCON, guaranteeing that the final encryption system remains reliable while consuming the fewest resources possible.

With the schemes to be utilised selected, the next step in development was to select a target platform to use as a test bed for all implementations. The ARM Cortex-M4 chip was selected as the chip of choice due to its widespread adoption and popularity within the embedded systems industry.

The ARM Cortex-M4 is widely recognised for its widespread integration into a wide range of embedded devices in multiple industries, as mentioned in reference [7]. Making use of this widespread popularity, the ARM Cortex-M4 is the processor of choice for development of a lightweight post-quantum encryption algorithm as it would reflect the applicability and scalability of the proposed scheme across a multitude of embedded devices.

Detailed below is the section on the updated requirements definition. This section provides a thorough understanding of the evolution of the system requirements over the course of the project. Any changes and improvements that have taken place are detailed, with justification and clarification on all changes made based on gained knowledge and understanding. In addition, the system's overall high-level requirements are carefully outlined. These requirements form the basis of the functionality of the system. Each requirement detailed in this section is elaborated on in section 3.3 with a comprehensive explanation, where each component is covered in detail.

3.2 Updated Requirements Definition

The Kyber algorithm used in the development of the new scheme, is what is known as a key-encapsulation-mechanism (KEM). [12] A KEM is a type of cryptographic algorithm used to securely exchange secret keys between two parties over an insecure communication channel. In brief, a KEM combines the functionality of key generation and encryption into a single operation. The concept of a KEM will be elaborated on in sections 4 and 5, however for the purposes on the requirements definition section it should be detailed that each KEM consists of three components. Key generation, encapsulation and decapsulation.

Throughout the project's development, the overall requirements have undergone iterative refinement. Initially solely focused on core functionalities like compatibility with the ARM chip, modification of the hash used in key generation, encapsulation, decapsulation and the production of comparative benchmarks. However, as the project unfolded, it became evident that incorporating a graphical interface into the final system would significantly enhance usability. This addition would bolster results visualisation, facilitate ease-of benchmark comparison, and simplify results analysis and report writing through the creation of graphs and charts. While the overarching system requirements of modifying the 3 key parts of the Kyber KEM remained relatively consistent, the specifics of design and implementation underwent significant evolution over the system's lifecycle. Further elucidation on these changes is provided in *sections 4 and 5*.

Found below are the current high-level requirements of the system:

- *Compatibility of new scheme with the ARM Cortex-M4 Chip*
- *Implementation of ASCON hash into Kyber key generation stage*
- *Implementation of ASCON hash into Kyber encapsulation stage*
- *Implementation of ASCON hash into Kyber decapsulation stage*
 - *Production of comparative benchmarks*
- *Creation of a graphical interface to aid in analysis of benchmarks*

3.3 Software Requirements Specification

This section provides a comprehensive overview of the system's specific requirements. All previous requirements undergo in-depth analysis, categorising them into functional and non-functional requirements, each accompanied by specific success criteria to ensure verifiability. By compartmentalising requirements and defining use cases, the aim is to establish a robust framework that facilitates effective development, testing, and validation of the system's functionality and performance. Information such as assumptions made about the problem area and constraints that impacted the development of requirements are detailed below, along with user characteristics.

3.3.1 Functional Requirements

Compatibility of new scheme with the ARM Cortex-M4 Chip:

The integration of the Kyber and ASCON algorithms together as the new scheme must operate seamlessly on the ARM Cortex-M4 chip environment. This entails ensuring that the scheme operates flawlessly on the designated platform, without encountering any errors or compatibility issues.

Success Criteria:

The successful execution of the new scheme on the ARM Cortex-M4 chip platform should be validated through the demonstration of error-free operation under various scenarios.

Implementation of ASCON hash into Kyber key generation stage:

A complete overhaul is required in the Kyber key generation stage, replacing all instances of existing hashes being used with the ASCON counterparts.

Success Criteria:

Successful implementation will be evidenced by error-free compilation and robust testing, confirming the accurate generation of cryptographic keys without any encoding or decoding errors.

Implementation of ASCON hash into Kyber encapsulation stage:

Similar to the key generation stage, the Kyber encapsulation process must have all instances of existing hashes be replaced with the ASCON hash.

Success Criteria:

Successful integration will be validated through error-free compilation and robust testing, confirming the accurate execution of the encapsulation process without encountering encoding or decoding discrepancies.

Implementation of ASCON hash into Kyber decapsulation stage

As with the previous stages, the Kyber decapsulation process must have all instances of existing hashes be replaced with the ASCON hash.

Success Criteria:

The successful integration will be confirmed through error-free compilation and robust testing, validating the execution of the decapsulation process without any encoding or decoding errors.

Production of comparative benchmarks

To assess the performance and effectiveness of the new scheme, comprehensive benchmarks must be generated. These benchmarks should encompass key metrics such as clock cycles, memory usage, hash efficiency, and code size, providing insights into the scheme's suitability for lightweight platforms.

Success Criteria:

The production of concrete benchmarking data for multiple versions of the ASCON hash combined with the Kyber algorithm, each consisting of the defined metrics will be invaluable in providing insights into the schemes suitability for lightweight platforms.

Creation of a graphical interface to aid in analysis of benchmarks:

The graphical interface should allow the user to import benchmarks from various versions of the new scheme and combine them into an easy to read and analyse format. The interface should also allow the user to display a number of charts comparing the various versions of the new scheme across all recorded benchmark metrics to determine which versions of the scheme are best suited to particular applications.

Success Criteria:

The interface seamlessly integrates imported benchmark data, ensuring no errors during the import process and enabling comprehensive analysis. Clear and informative charts should be generated, accurately representing benchmark data for different versions of the new scheme, facilitating analysis and comparisons.

3.3.2 Non-functional Requirements

Performance:

The system: both interface and new scheme should demonstrate efficient performance in processing benchmark data, ensuring minimal latency in importing, analysing and visualising benchmarks. Response times for generating charts for benchmarks and should be within a reasonable timeframe to provide a smooth user experience.

Usability:

The interface should have an intuitive and user-friendly design, allowing users to navigate and utilise its features with minimal instruction. Clear and concise error messages should be provided to guide users in case of any issues during benchmark import, analysis, or visualisation.

Security:

The new scheme, integrating Kyber and ASCON algorithms, should maintain the same level of security as the original implementations of both algorithms.

Maintainability:

The system: both interface and new scheme, should be designed with modularity and extensibility in mind, allowing for easy maintenance and future enhancements. Code should be well-documented, with clear comments and structuring that facilitate ease of understanding and modification.

3.3.3 System Assumptions and Constraints***Assumptions:***

It was assumed that the decision to replace the existing hash in the Kyber scheme with ASCON was a reasonable goal within the project's timeframe. It was assumed that both algorithms' functionality would be modularised rather than tightly integrated, which would ensure that the integration of both schemes would be manageable within the project's timeframe.

Initially, it was assumed that the security level provided by ASCON would adequately align with the requirements of the Kyber algorithm so that the overall security level of the system would be maintained at the same level as before. This turned out to be true but at the project's inception, was not fully considered nor investigated.

Constraints:

The project timeframe, spanning from October to March, imposed constraints on the achievable scope within this period, limiting the extent of work that could be completed.

The complexity of the field of PQC and of the mathematics behind both algorithms was a limiting factor in potential optimisation as it meant with the given time period for project's development, a large part of both algorithms had to remain untouched leaving lots of room for further development.

The substantial learning curve associated with coding, comprehending the intricacies of each method within both algorithms, and becoming proficient with the toolkits tailored for the ARM M4 chip was significant. It required a considerable amount of time to acclimate to these complexities, as opposed to the relatively simpler process of writing and compiling code for conventional desktop applications.

3.3.4 User Characteristics

The user of the system, comprising both the new scheme, the obtaining of benchmarks for the various versions of the new scheme and the interface to browse benchmarks and view charts and analysis is assumed to be someone familiar with the basics of cryptography and key encapsulation mechanisms (KEMs). The user should have the technical ability required to be able to follow the detailed instructions on how to connect a physical device containing an ARM M4 chip or run an emulator to allow for the new scheme to be tested and benchmarked.

Typical users of the system would be as follows:

Cryptographers and Security Experts:

Users with expertise in cryptography and cybersecurity, including cryptographers, security researchers, and professionals, who require possess the skills required to evaluate post-quantum encryption algorithms like Kyber.

Software Developers and Engineers:

Users responsible for software development and engineering, particularly those involved in developing applications for resource-constrained devices such as embedded systems and IoT devices. They could utilise the system to assess the performance and suitability of the new scheme for implementation in such environments.

Researchers and Academics:

Users engaged in research and academic pursuits related to cryptography, cybersecurity, and post-quantum computing. They could utilise the system for experimentation, analysis, and validation of novel cryptographic techniques and algorithms.

3.3.5 Summary

In conclusion, the solution outlined in this section offers a thorough strategy for tackling the issues raised by the development of risks associated with quantum computing that arise from conventional encryption techniques. The project's goal is to offer a post-quantum encryption scheme that can be implemented in environments with limited resources, like embedded systems and Internet of Things devices, by combining the Kyber and ASCON algorithms. System limitations, such as the project timeline, the difficulty of post-quantum cryptography, and the learning curve for hardware platforms and cryptographic algorithms, have all been carefully taken into account during the development process. In spite of these obstacles, the system is made to accommodate the wide range of requirements of its users. Through an understanding of these limitations and user characteristics, the system aims to provide a practical and efficient means of protecting data against changing cybersecurity threats.

4. Design

The design phase is a crucial stage in the development lifecycle that establishes the functionality, performance, and maintainability of a system. Found in this section is a thorough explanation of the selected system design, covering system architecture, software system design, user interface design, and critical design decisions. Each element of the design is closely connected to the project specifications, ensuring the fulfilment of all designated requirements and features. Justifications for all design choices are detailed in the various sub-sections of the design section, along with visual aids in understanding the architecture and operation of the complete system.

4.1 Overview of system architecture

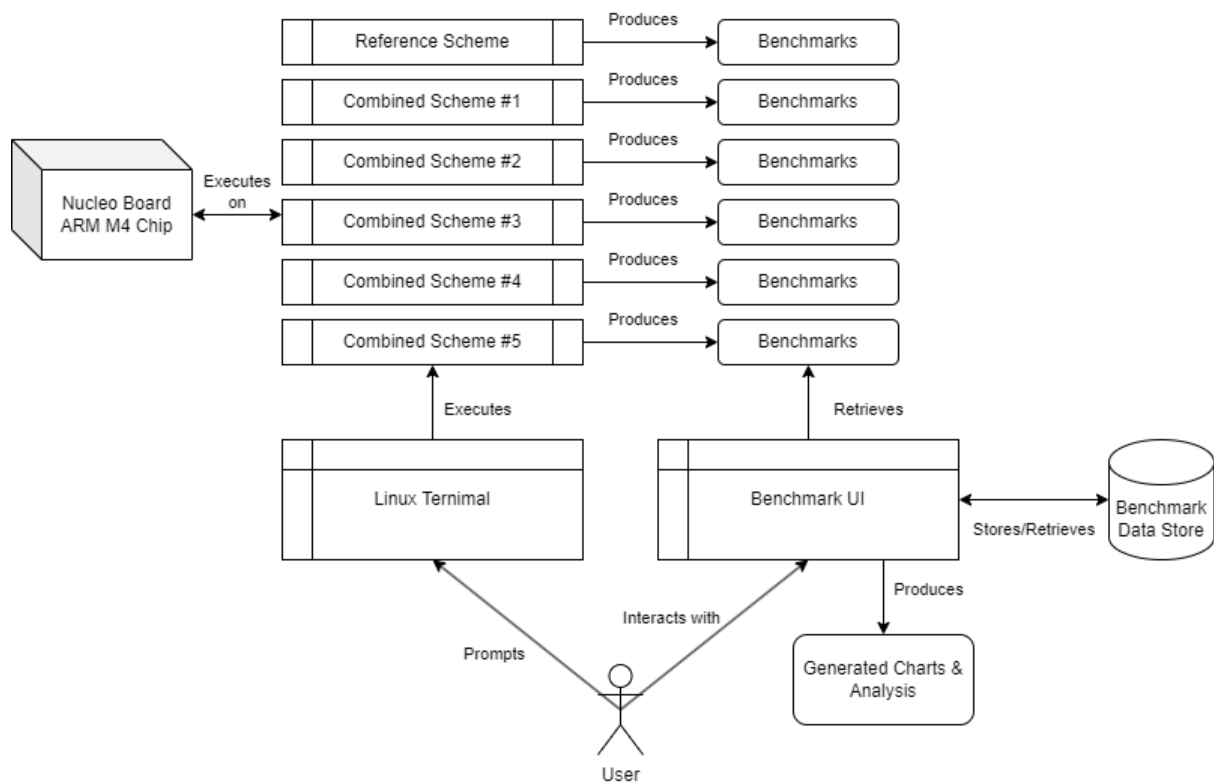


Figure (1) – System Architecture Overview

Illustrated in the diagram above is a high-level overview of the complete system architecture. The system consists of a number of individual components, each with their own functions, inputs, outputs and interactions with other components. To aid in the understand and description of the system, each component will first be investigated individually and its design characteristics analysed in depth. Further analysis of the design of the new schemes will then follow.

Nucleo Board:

Beginning with the foundation of the entire system – the Nucleo ARM M4 board. The specific details about the choice of board and specifications of the board will be discussed in the *Implementation* section.

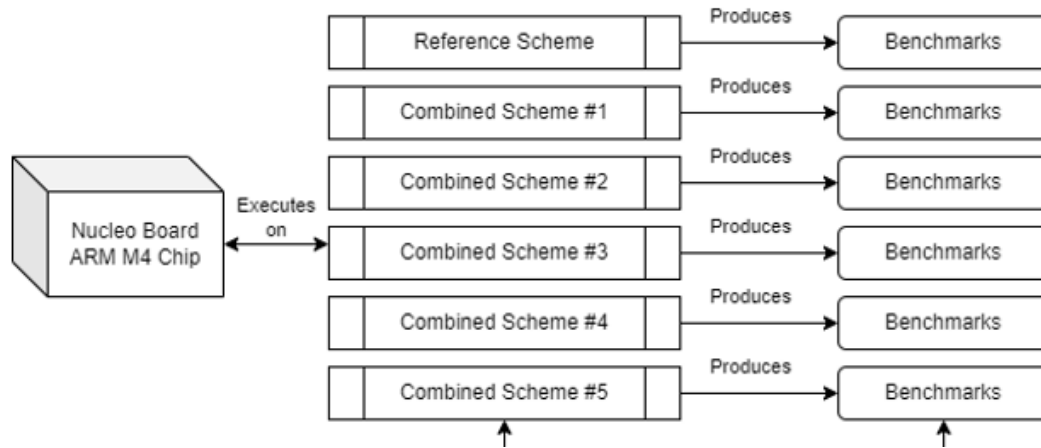


Figure (2) – Illustration of the Nucleo Board component of the System and its interactions.

This development board is the root of the system, all code is executed on this board and all benchmarks are obtained from the outputs of code executed on the board. The purpose of the board is to represent the signature of a typical lightweight, low-cost, low-power embedded device, such as an IoT device. It contains an ARM-Cortex-M4 chip so is an excellent choice for scalability purposes due to the previously discussed popularity of the M4 chip. In the context of the complete system, the board can either be physically plugged in to the host machine or emulated using software. It is important that the device the code will be ran on matches the profile of a typical lightweight device so the physical board, what was used in the creation of this project, is perfectly suited. One downside of using a physical board in the development of the project was that there comes the risk of damaging the board when it is being transported from location to location.

This risk was mitigated by ensuring the board was looked after carefully. If the board was damaged and rendered inoperable then testing and benchmarking would not be able to be carried out. When the compiled code from each scheme combination is executed by the user through the Linux terminal, the various tools used in connecting the host machine to the board flash the executable files to the board. A number of tests are then performed, and benchmarks obtained, which are passed back through the board's IO channels to the host machine where they are recorded in CSV format. The user can directly observe the output of the board through the Linux terminal via commands detailed in the *implementation* section. This includes any errors that may have occurred when tests were being performed or benchmarking was occurring.

New Combined Schemes:

Closely related to the board are the various combined schemes that are ran on the board. These schemes are the primary focus of the entire project. They are the combination of the Kyber and ASCON algorithms. Overall, there are 6 schemes that are contained within the GITLAB repository for the project. One reference Kyber scheme and 5 modified Kyber-ASCON merger schemes. Details on the nature of each of these merger schemes is found in the *implementation* section, however for now a high-level overview is that there is a single version of Kyber with 5 different versions of ASCON. Each scheme consists of 3 key parts, key generation, encapsulation and decapsulation. The

ASCON hash algorithm has been used to completely replace the existing SHA family of hashes that were previously used in Kyber. More detail to follow about the nature of this replacement in the next section. Each scheme is located within its own subdirectory and contains its own unique code, benchmark outputs and test outputs.

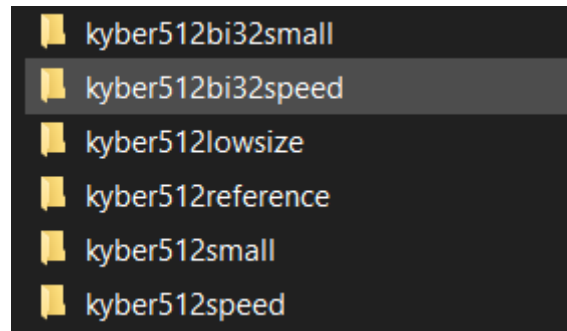


Figure (3) – The subdirectory structure of each combined scheme.

The user interacts with the Linux terminal to compile each scheme's code into a number of executable files that can be ran on the Nucleo board.

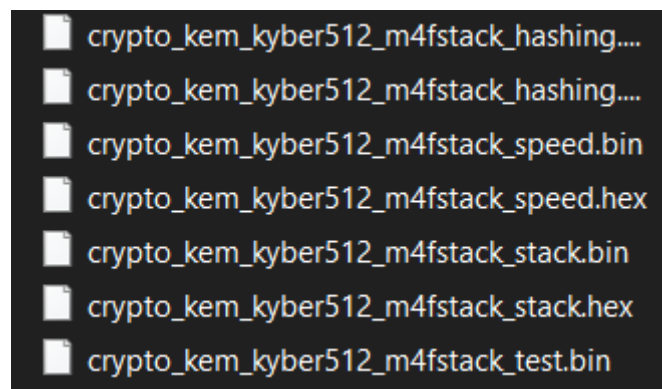


Figure (4) – Sample executable files generated for one combined scheme.

These executable files consist of test files, speed benchmark files, hashing benchmark files, stack test files and correctness checking test files. As mentioned before, the binary (.bin) versions of these files are flashed to the Nucleo board using the various build tools associated with the project and the board, and via the board's IO channels, outputs are returned to the host system. These outputs are stored in each schemes sub-directory, so are easy to locate and impossible to confuse with other scheme's benchmark/test results. Further detail on the design of each of the new schemes will be detailed later in this section.

Benchmarks:

The output of running the previously discussed new combined schemes on the Nucleo board; the benchmarks are the main output of the system as a whole. Though the architecture diagram is labelled as "benchmarks", this section will also discuss the tests which can be performed on the schemes in the same manner as the benchmarks. Beginning with the benchmarks, there are a number of different metrics that are recorded through the system's benchmarking suite. These are total clock cycles used, total amount of memory used, percentage of clock cycles spent in the

hashing algorithm in relation to the whole scheme and code size. These metrics allow for a complete and balanced comparison of schemes across a number of areas of interest in the PQC space. Benchmarks can be obtained through first compiling the code for the given scheme which produces the executable files shown in the previous image in the section above. Once the executable files have been produced, the user can then execute them on the Nucleo board through commands issued to the Linux terminal. The specific nature of these commands will be detailed in the *implementation* section. The outputs from all benchmarks obtained from the scheme under test are stored in a number of different files located within the subdirectory corresponding to the scheme as shown below.

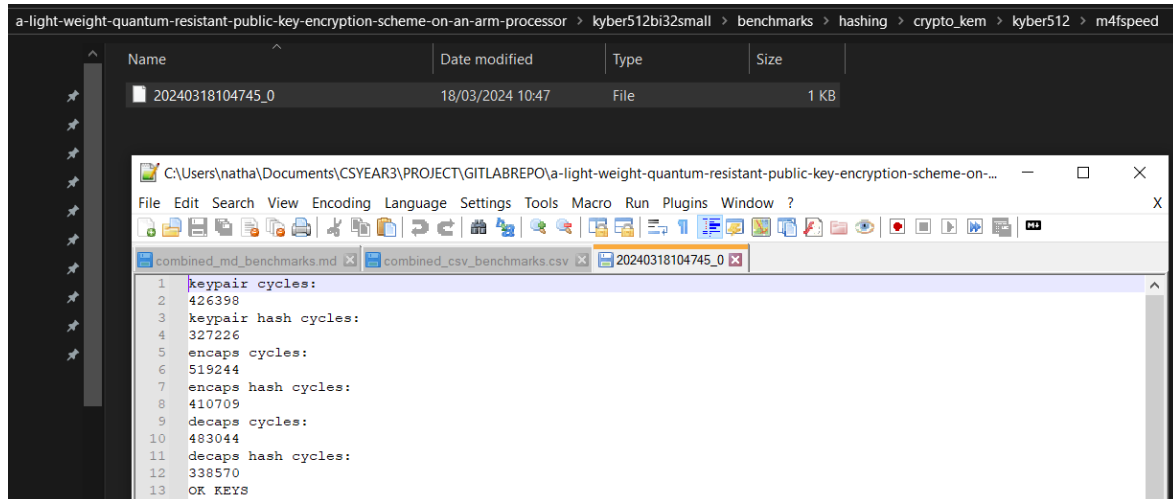


Figure (5) – Example hashing benchmarks taken from the Kyber 512 bi32 small scheme.

These segregated benchmarks can be combined into a single file in a number of formats for easier viewing. This is again done through the Linux terminal and can produce an output that contains either a CSV or MD file containing the total clock cycles used, total amount of memory used, percentage of clock cycles spent in the hashing algorithm in relation to the whole scheme and code size all in a single file. An example of the output from running this command is shown in the image below.

Speed Evaluation

Key Encapsulation Schemes

scheme	implementation	key generation [cycles]	encapsulation [cycles]	decapsulation [cycles]
kyber512 (1 executions) clean		AVG: 605,686	AVG: 802,369	AVG: 909,540
		MIN: 605,686	MIN: 802,369	MIN: 909,540
		MAX: 605,686	MAX: 802,369	MAX: 909,540
kyber512 (1 executions) m4fspeed		AVG: 422,805	AVG: 515,083	AVG: 479,159
		MIN: 422,805	MIN: 515,083	MIN: 479,159
		MAX: 422,805	MAX: 515,083	MAX: 479,159
kyber512 (1 executions) m4fstack		AVG: 422,376	AVG: 516,845	AVG: 480,982
		MIN: 422,376	MIN: 516,845	MIN: 480,982
		MAX: 422,376	MAX: 516,845	MAX: 480,982

Signature Schemes

scheme implementation key generation [cycles] sign [cycles] verify [cycles]

Memory Evaluation

Key Encapsulation Schemes

Scheme	Implementation	Key Generation [bytes]	Encapsulation [bytes]	Decapsulation [bytes]
kyber512 clean		5,984	8,640	9,408
kyber512 m4fspeed		4,172	5,276	5,284
kyber512 m4fstack		2,100	2,188	2,204

Figure (6) – Example benchmarks taken from the Kyber 512 bi32 small scheme in MD format.

To summarise what has been said so far about the benchmarks, there are 2 commands that can be run from the Linux terminal in relation to the benchmarks. One command runs the benchmarks on the Nucleo board for a given scheme, and the second command compiles the results into a single file for easier analysis. The combined benchmark files, both MD and CSV, are further used by the benchmark UI to import the combined data for all schemes into a single location and extract it for further analysis. In relation to the tests, it is even simpler. There are a number of tests related to the system which ensure that the scheme works as expected. Key generation, encapsulation and decapsulation are all tested, along with a number of failure cases checked also. It is important to have tests that cover the full range of operations of the scheme to ensure that the requirements detailed in the previous section can be effectively evaluated. The testing framework was essential to the development of the combined schemes as it allowed a detailed breakdown of where changes needed to be made to modify the scheme effectively. The nature of the tests are discussed further in the *testing* section, for now only a high-level overview of the tests are discussed. In similar fashion to the benchmarks, tests are ran through a single Linux command which flashes the corresponding test binary file to the board. The test output can then be observed directly from the board through the

use of the Linux Screen command, or with a summary of the output being displayed in the Linux terminal through which the tests were executed. An example of this is shown below:

```
INFO:Implementation:Building kyber1024 - m4fstack - test
make: 'bin/crypto_kem_kyber1024_m4fstack_test.hex' is up to date.
DEBUG:platform interface:Found start pattern: b'=====\\n'

INFO:BoardTestCase:Success
INFO:Implementation:Building kyber1024 - m4fspeed - test
make: 'bin/crypto_kem_kyber1024_m4fspeed_test.hex' is up to date.
DEBUG:platform interface:Found start pattern: b'=====\\n'

INFO:BoardTestCase:Success
INFO:Implementation:Building kyber1024 - clean - test
make: 'bin/mupq_pqc_clean_crypto_kem_kyber1024_clean_test.hex' is up to date.
DEBUG:platform interface:Found start pattern: b'=====\\n'

INFO:BoardTestCase:Success
```

Figure (7) – Sample high-level test output for each scheme variation of Kyber 1024.

Benchmark UI:

The benchmark UI is a core component to the visualisation and analysis of results in the system. It has a number of different interactions with other system components and produces the primary outputs of the system:

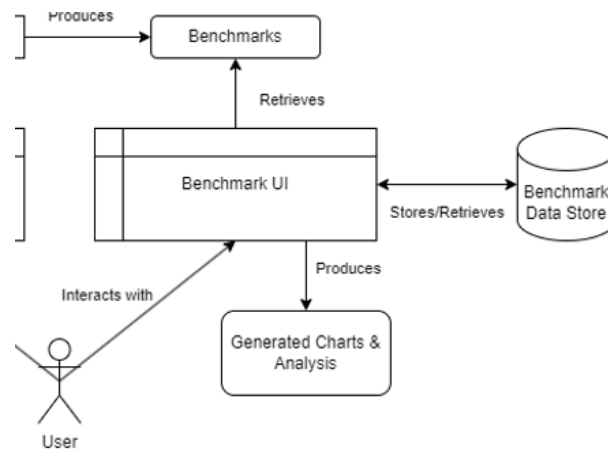


Figure (8) – The Benchmark UI and its interactions.

The Benchmark UI is one of two interfaces the user will use to interact with the system, the other being the Linux terminal. It is primarily responsible for combining the benchmarks from each individual scheme into one place and producing charts and analysis from the data. The graphical interface was designed to be as simplistic as possible, as the main attention of the project was given to the merger of the Kyber and Ascon schemes rather than significant time being invested in the GUI design. As such it features an uncomplicated yet modern design with minimal options barring the essential data the user would need to be able to perform further comparison and analysis of the results of the schemes' benchmarks. The interface remains relatively similar to the initial sketches/concept design that was produced at the start of the development phase.

Before diving into specifics, the general theme used in the design of the interface was simple and dark. This was deliberately chosen to keep unnecessary clutter to a minimum and allow for the easy understanding of important information related to the project results while also maintaining a clean, aesthetic appearance. All text is in white font with interactable information being displayed on grey buttons. The dark appearance of the interface is contrasted with the bright appearance of the charts, as shown later in this section. This is done to maximise the ease of readability the interface aims to deliver, and keep the focus on the important details, minimising unnecessary information. The key users of this system will be professionals such as academic researchers, software engineers or cryptographic experts. The interface is designed to easily deliver the necessary information in a manner suited for the given user base and as such does not contain lots of unnecessary options or customisations. Given a longer timeframe of development, these things could be added in a complimentary fashion.

The landing screen of the UI contains two grey buttons and a link to the project's README. The README contains all instructions related to the interface and information surrounding the entire project itself. The other two buttons navigate to the other screens of the interface.

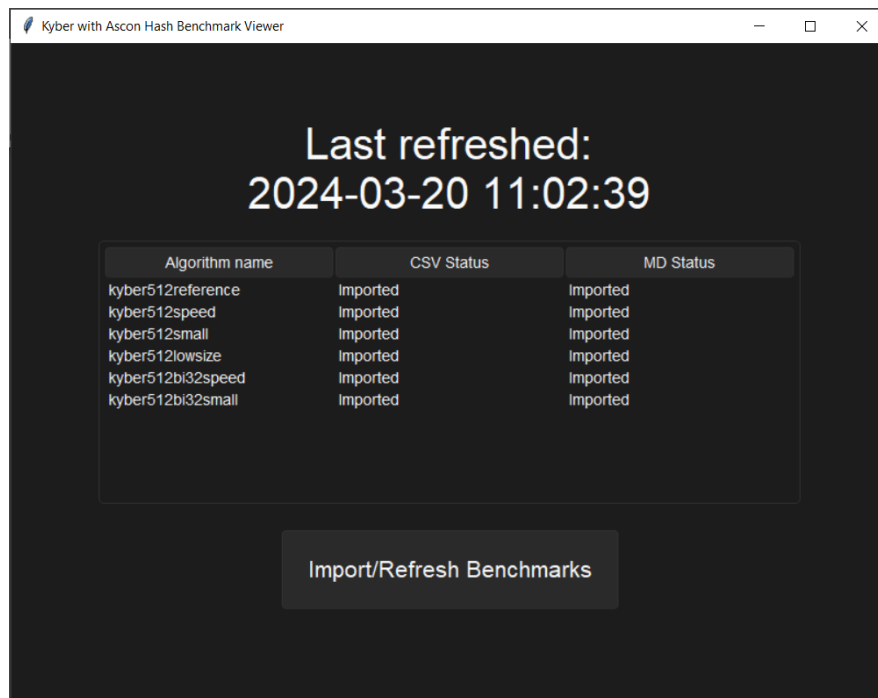


Figure (11) – The Benchmark Importer screen of the UI.

This screen contains all information relevant to importing the benchmark results from each individual scheme belonging to the project. The last time benchmarks have been imported into the system is displayed at the forefront of the screen. The addition of this information was to prevent the usage of old or outdated data in the generation of charts and analysis as the user will always be able to see what the timestamp of the data is that they are using. There is also a table containing the name of all schemes contained within the system and the status of each CSV and MD file. Clicking on the Import/Refresh Benchmarks button will instruct the system to search through the subdirectories of each of the schemes in an attempt to locate any CSV or MD files containing benchmarks. These files are then merged to produce two combined files. If any CSV or MD files cannot be found the status of the scheme on the table is updated accordingly.

kyber512lowsize	Error importing	Imported
-----------------	-----------------	----------

Figure (12) – The status of the scheme being updated when file is not found.

An example of the combined MD file is shown below. The system appends the individual scheme name to each version of the general scheme for better understanding of the data. The CSV file contains similar information in a similar format.

Speed Evaluation

scheme	implementation	key generation [cycles]	encapsulation [cycles]	decapsulation [cycles]
kyber512 reference (1 executions)	clean	AVG: 652,784	AVG: 863,593	AVG: 947,370
		MIN: 652,784	MIN: 863,593	MIN: 947,370
		MAX: 652,784	MAX: 863,593	MAX: 947,370
kyber512 reference (1 executions)	m4fspeed	AVG: 466,752	AVG: 573,165	AVG: 513,858
		MIN: 466,752	MIN: 573,165	MIN: 513,858
		MAX: 466,752	MAX: 573,165	MAX: 513,858
kyber512 reference (1 executions)	m4fstack	AVG: 466,621	AVG: 575,223	AVG: 515,982
		MIN: 466,621	MIN: 575,223	MIN: 515,982
		MAX: 466,621	MAX: 575,223	MAX: 515,982
kyber512 speed (1 executions)	clean	AVG: 613,885	AVG: 814,134	AVG: 918,211
		MIN: 613,885	MIN: 814,134	MIN: 918,211
		MAX: 613,885	MAX: 814,134	MAX: 918,211
kyber512 speed (1 executions)	m4fspeed	AVG: 431,339	AVG: 527,188	AVG: 488,173
		MIN: 431,339	MIN: 527,188	MIN: 488,173
		MAX: 431,339	MAX: 527,188	MAX: 488,173
kyber512 speed (1 executions)	m4fstack	AVG: 430,586	AVG: 528,625	AVG: 489,669
		MIN: 430,586	MIN: 528,625	MIN: 489,669
		MAX: 430,586	MAX: 528,625	MAX: 489,669
kyber512 small (1 executions)	clean	AVG: 638,735	AVG: 845,213	AVG: 943,794
		MIN: 638,735	MIN: 845,213	MIN: 943,794
		MAX: 638,735	MAX: 845,213	MAX: 943,794
kyber512 small (1 executions)	m4fspeed	AVG: 456,525	AVG: 558,605	AVG: 514,097
		MIN: 456,525	MIN: 558,605	MIN: 514,097
		MAX: 456,525	MAX: 558,605	MAX: 514,097

Figure (13) – Part of the speed evaluation section of the combined MD file.

Once the combined files have been created/updated with the individual benchmarks, the timestamp showing the last refreshed time is updated.

Moving on to the benchmark viewer screen. This screen consists of two sections, a navigation panel to view either the raw combined CSV or MD files or the bar charts from the data, and a brief analysis section. Clicking on either of the view raw data buttons will open the corresponding CSV or MD file of the combined results with whatever program the user has configured to view that file type. This feature was included as it was thought that there would be occasions where further analysis would be needed on the data and an easy access/navigation button would be useful.

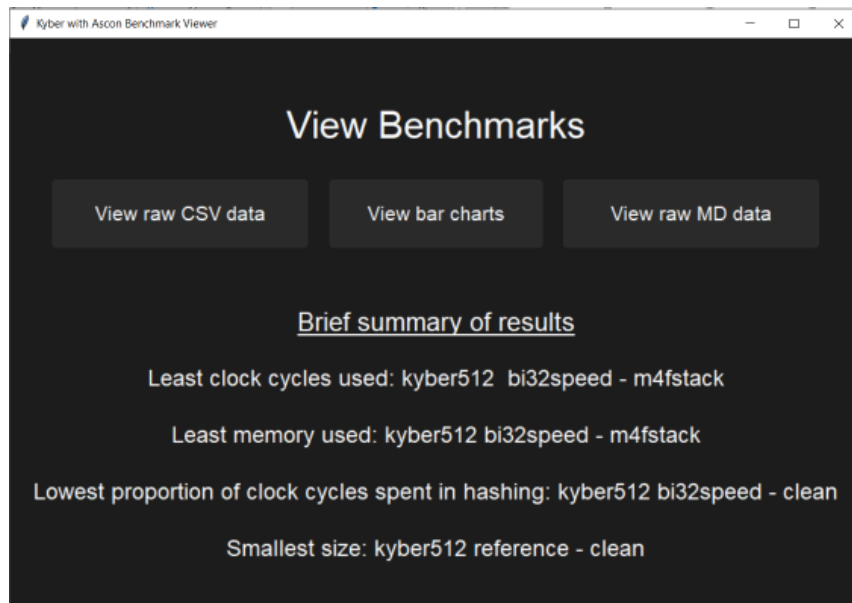


Figure (14) – The View Benchmarks Screen

Clicking on the view bar charts button navigates to a further menu where the metric for which to display charts for can be selected.

The brief analysis section contains an at-a-glance overview of the combined data. The scheme with the least number of clock cycles used, least memory used, lowest proportion of clock cycles spent in hashing and smallest code size are all displayed. This section was included in the final design as it was thought to provide a quick summary of results without the need to delve deeper into the statistics behind each category which can be done through the charts or analysis of the complete benchmark files.

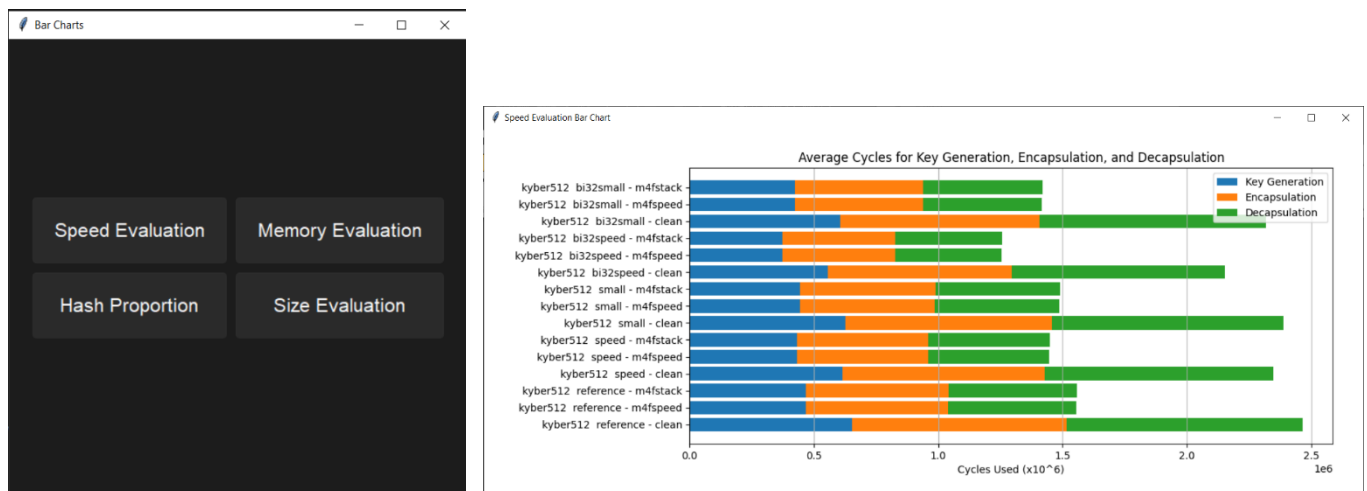


Figure (15) – The bar chart selector screen with accompanying bar chart.

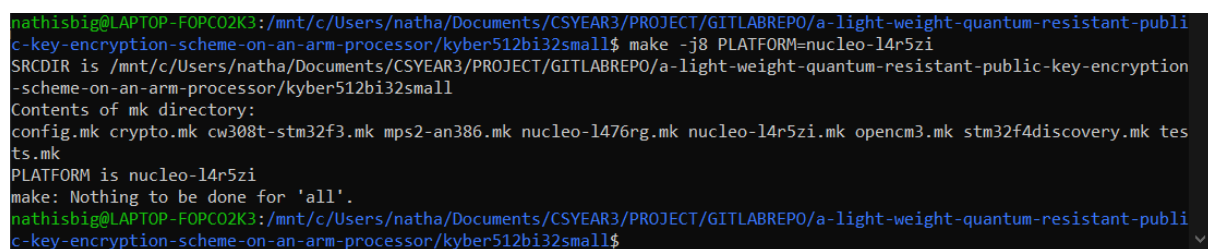
The bar chart selector screen consists of a simple layout of four buttons, one for each metric recorded in the benchmarks. Clicking on each button will open a window containing the corresponding chart. Each chart contains all variations of the combined scheme with a breakdown of information detailed on the chart legend. In the example above the key generation, encapsulation

and decapsulation cycles are illustrated in different colours to aid in the analysis of the chart. Each axis contains a clear and concise label and an appropriate title is given to each chart.

The design of the interface has both advantages and drawbacks, it is very simple and intuitive to use with no learning curve required, suiting the target userbase of professionals who are primarily interested in the data and would rather not spend time learning to use a complicated and nuanced interface. It is also sleek and modern being easy on the eye and emphasises key details such as discriminating groups on the charts. However, there are several disadvantages to the design of the interface. While it is simplistic, there is much left to be desired in terms of expansion of options/features that might be useful to an academic researcher or cryptographic expert including more filtering of the graphs, further methods of filtering data etc. Unfortunately, development time constraints limited how much could be included in the interface. The ideas behind how the interface could be expanded are detailed in the *future work* section of the report. A further drawback of the interface is that it does not run the benchmarks themselves, only compile them into a single location for analysis. This was too complex a task to implement given the time constraints of the project, given the complications that arise when considering the cross-platform Windows-Linux operations and USB interfacing with the Nucleo board. This is another idea that is discussed in the *future work* section of the report.

Linux Terminal:

A part of the system that is not original to the system is the Linux terminal. It is how the user compiles the code for each scheme, connects the board to their host pc and runs tests and benchmarks on the Nucleo board. Given the nature of this project having the domain of use aimed at professional/expert level users, the Linux terminal/command line method of issuing commands is more than suitable. The reason behind the use of Linux commands is based on the underlying library used in building the project heavily utilising them for compilation, testing and benchmarking. It was never planned as part of the project specification to remove the usage of the Linux terminal for compiling and executing code as it was set out from inception that the project would be aimed at advanced users who would possess the knowledge required to use a command-line-interface. Detailed instructions on how to perform operations such as linking the board to the host pc, compiling code and running tests and benchmarks can be found on the project README. A disadvantage of using this approach to interfacing with the system is that it requires additional time to become familiar with the specific commands and syntax required to use the system effectively. However, once a sufficient understanding is obtained, results can be obtained quickly and effectively using the terminal.



```
nathisbig@LAPTOP-FOPC02K3:/mnt/c/Users/natha/Documents/CSYEAR3/PROJECT/GITLABREPO/a-light-weight-quantum-resistant-public-key-encryption-scheme-on-an-arm-processor/kyber512bi32small$ make -j8 PLATFORM=nucleo-l4r5zi
SRCDIR is /mnt/c/Users/natha/Documents/CSYEAR3/PROJECT/GITLABREPO/a-light-weight-quantum-resistant-public-key-encryption-scheme-on-an-arm-processor/kyber512bi32small
Contents of mk directory:
config.mk crypto.mk cw308t-stm32f3.mk mps2-an386.mk nucleo-l476rg.mk nucleo-l4r5zi.mk opencm3.mk stm32f4discovery.mk tests.mk
PLATFORM is nucleo-l4r5zi
make: Nothing to be done for 'all'.
nathisbig@LAPTOP-FOPC02K3:/mnt/c/Users/natha/Documents/CSYEAR3/PROJECT/GITLABREPO/a-light-weight-quantum-resistant-public-key-encryption-scheme-on-an-arm-processor/kyber512bi32small$
```

Figure (16) – Sample Linux command to compile the code for the Kyber 512 bi32 small scheme.

4.2 Further detail of New Combined Schemes:

It is necessary to investigate the nature of the both the individual and new combined schemes in further detail to fully understand the design rationale behind the new schemes. Beginning with the Kyber scheme, Kyber is a post-quantum key encapsulation mechanism (KEM) designed to resist attacks from both classical and quantum computers. A KEM is a cryptographic tool used to securely exchange secret keys between parties over an insecure communications channel, it consists of three main areas: key generation, encapsulation and decapsulation. The Kyber algorithm is based on the principles of lattice-based cryptography, namely using a variant of the learning with errors (LWE) problem.

The learning with errors (LWE) problem is a complex challenge within lattice-based cryptography, serving as the basis for a number of cryptographic protocols. Fundamentally the LWE problem involves the endeavour to uncover a secret vector \mathbf{s} from pairs of linear equations affected with random noise. This mathematical challenge, although seemingly straightforward in its formulation, poses a formidable challenge to computational adversaries. Despite the simplicity of its structure, the LWE problem's inherent complexity renders it resistant to classical and quantum computing attacks, underpinning the security of lattice-based cryptographic schemes. Without diving into too much mathematics the LWE problem can be represented as follows:

Given a matrix \mathbf{A} and a secret vector \mathbf{s} , alongside a small random error vector \mathbf{e} the problem can be formally represented as:

$$(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$$

Where the aim is to deduce the secret vector \mathbf{s} from pairs of equations contaminated by random noise.

The LWE-based lattice cryptography backbone of Kyber lends it to pose a formidable challenge to both classical and quantum attacks. Kyber also makes use of a number of hashing algorithms in the key generation, encapsulation and decapsulation process, namely the Keccak family of cryptographic primitives. Various versions of SHA-3 and SHAKE are used in the Kyber algorithm to perform hashing operations such as hashing inputs and generating randomness to enhance the cryptographic integrity of the algorithm. The Kyber algorithm comes in a number of security levels to be well suited for a wide range of applications with differing security and size requirements. It offers three schemes: Kyber512, Kyber768 and Kyber1024 at NIST security levels 1, 3 and 5 respectively.

The Ascon hash algorithm is a lightweight cryptographic hash function designed for applications requiring a low resource usage such as embedded and IoT devices. It operates by using a sponge-like construction to work, absorbing input data into an internal state and producing output data by squeezing and absorbing iteratively. It uses the Ascon permutation, a special kind of permutation function designed to meet its needs and comprising several iterations of bitwise operations and modular addition. Ascon allows for customisation of security levels by changing variables like the quantity of rounds and internal state size. In contrast, the Keccak sponge construction is used by the SHA3 and SHAKE hash functions in the Kyber algorithm. Similar to Ascon, it also undergoes a squeezing phase to produce output data after absorbing input data into an internal state. SHA3 and SHAKE, on the other hand, use a different Keccak permutation that is based on a different design

philosophy and uses a different set of operations. Like the SHAKE algorithm, Ascon also features its own XOF or extendable output function allowing for variable length outputs.

Scheme	Security Level (Bits)	NIST Assigned Level
Kyber512	128	1
Kyber768	192	3
Kyber1024	256	5

Table detailing the security levels of each version of Kyber.

In the creation of the new combined Kyber-Ascon schemes, the version of Kyber that was suitable to be used in making the changes to the hashing algorithm was carefully considered. As Ascon offers a 128-bit security level, similar to that of Kyber512 it was chosen that Kyber512 would serve as the base of the new scheme. This was done to avoid an unnecessary reduction in the security level of a higher security and therefore more resource-intensive version of Kyber. An algorithm designed with 192-bit (Kyber768) or 256-bit (Kyber1024) security having its hashing functions replaced with a 128-bit security level would reduce the overall security level to that of the weakest link. For an algorithm to be entirely secure the hashing functions would also need to provide equivalent security to that of the LWE problem. It was with this thought that the Kyber512 version was chosen as it would match the security level of the Ascon algorithm, not requiring any unnecessary bottlenecks and be the most optimal scheme for lightweight, resource-constrained devices.

For the Ascon algorithm, there exist a number of different versions, much similar to the Kyber algorithm. For Kyber there exists reference implementations, speed-optimised versions and stack-optimised versions. Likewise, with Ascon there are five versions that are suitable to be ran on an ARM-M4 chip: low size, speed, small, 32-bit-interleaved speed and 32-bit-interleaved small. The nature of each of these schemes will be elaborated on in the *system evaluation and experimental results* section. Each of these schemes is merged together with the three versions of Kyber512 to produce a total 15 combinations of new combined schemes.

4.3 Critical Design Decisions and Patterns:

The final aspect of the design section, this sub-section details the miscellaneous parts of the design that have not been covered in previous sub-sections as well as a summary of the key critical design decisions and patterns that went into crafting the system.

The primary focus of the project was in ensuring that the new combined schemes functioned effectively, that they were able to perform key generation, encapsulation and decapsulation correctly with no errors of any sort. The merger of the two algorithms took priority over interface design and usability as the purpose of the project was mainly to combine the schemes, perform tests and obtain benchmarks. As such and as mentioned before, the interface has much room for improvement, however, does perform the tasks that are required to satisfy the requirements specification. The user base of the system are advanced users with assumed experience working with CLI interfaces like the Linux terminal. This enables a focus on the data itself rather than additional time spend both developing and using a graphical interface to perform operations like connecting the Nucleo board to the host pc and running tests and benchmarks. All files currently involved with the project sit

unencrypted and with combined benchmarks simply located within MD and CSV files rather than a database of any sorts. This was again due to the time constraints of the project with priority being given to ensuring the new schemes worked correctly. In future a database containing a number of tables corresponding to the metrics recorded during benchmarking of each scheme and all test results would be a worthwhile goal. As mentioned above, all files are unencrypted as there is no sensitive data associated with this project. Any sensitive communications associated with the schemes involved in this project will utilise the schemes produced to encrypt and decrypt data rather than having data associated with the algorithms themselves encrypted. Finally, with regards to error and exception handling strategies, there are error messages displayed when data cannot be pulled in from the CSV and MD files as shown in the image previously. With the Linux terminal, test failures and error messages are present for each command. Further there are built in safeguards into the benchmark interface that prevent the user from opening more than 1 window instance of the same window at a time, meaning importing benchmark data in an erroneous fashion is impossible.

5. Implementation

The implementation section of this report outlines the technical framework of the system, including language choices, software libraries used and key implementation decisions. The selection of implementation languages and environments are discussed along with the utilisation of software libraries and detailing of key functions in the system. Overall, this section aims to provide an in-depth analysis of each component's implementation in the system and all techniques and decisions that were involved in the implementation.

5.1 Implementation Decisions: Languages and Environments

New Combined Schemes:

Beginning with the hardware that forms the backbone of the system, where all new combined schemes are tested and benchmarked: the Nucleo board.

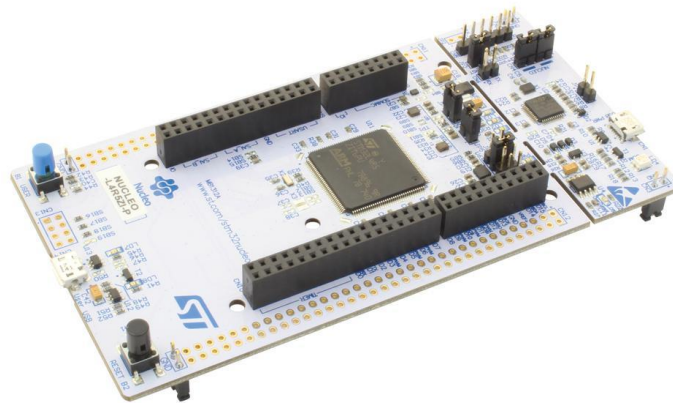


Figure (17) – The Nucleo-L4R5ZI board used in testing and benchmarking the new schemes.

As detailed in previous sections, the STMicroelectronics Nucleo-L4R5ZI board offers an immensely useful platform for performing testing and benchmarking. This is due to the ARM Cortex-M4 chip featured on the board, mimicking the most commonly used chip currently found in embedded devices. The low-resource, lightweight nature of the board coupled with the M4 chip installed makes the board a perfect testbed for all new schemes to evaluate their performance and suitability for embedded systems and IoT applications.

The manufacturers of the board, STMicroelectronics, also provide a number of tools that can be used to interface with the board, write code and flash binaries onto it for testing. This project made use of these development tools only in the initial understanding phase where experience and proficiency with the board was required before proceeding to attempt to implement any of the schemes onto the board. The tools used for this component of the project were STM32CubeProgrammer and STM32CubeIDE. Shown below is a screenshot taken from STM32CubeIDE detailing the pinout configuration of the board. Note that none of the STM32 tools are necessary for the operation of complete system, however they played an instrumental role in the initial phases of implementation of the system and as such are detailed here.

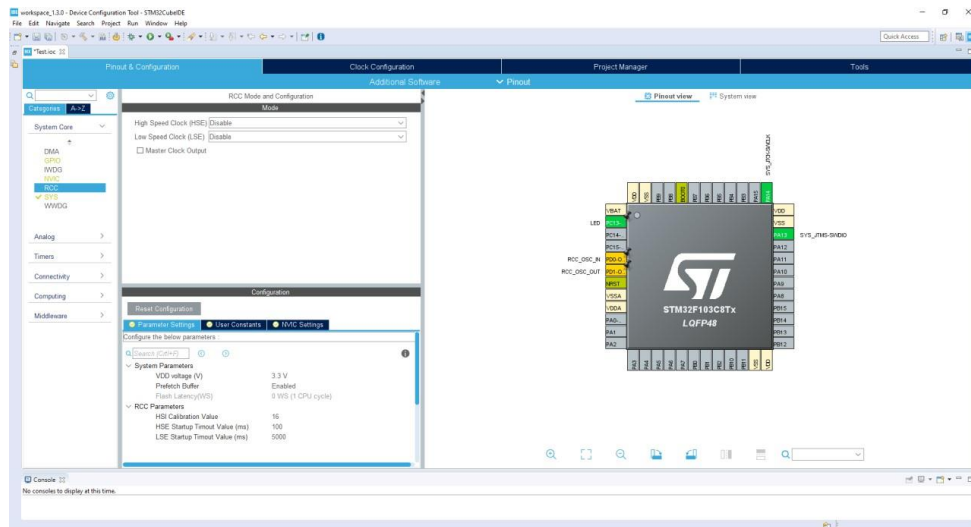


Figure (18) – The pinout interface in STM32CubeIDE.

After the initial understanding of how the board worked and how to flash test code onto it, focus turned to getting the reference versions of Kyber to run on the board.

The PQM4 library used in this project is fully detailed in the *software libraries* subsection, however for now it is important to note that the language used in this library is primarily C with large sections also written in ARM assembly. As such the majority of the project's development took place using the C language, with only minor parts taking place in assembly. The choice of language used to write the PQM4 library suggests that the authors attempted to strike a balance between the optimised performance of assembly with the ease of development of a higher-level language like C to allow for potential modifications while maintaining a solid performance on lightweight devices. These same languages are also used in the Ascon library containing all previously discussed implementations of the Ascon hashing algorithm.

The IDE of choice for the development of the project was VS Code, this was for several reasons. VS Code offers a number of features and extensions that aid in the understanding and development of a new system such as automatic syntax correction, code suggestions and a simple, easy to use interface. As such, VS Code was used in the understanding of both the PQM4 library containing the Kyber schemes and the Ascon library containing the Ascon hash algorithms.

For building, compiling and debugging code, the PQM4 library contains its own custom build system which makes use of two main tools: Make and Arm GCC Toolchain. The PQM4 library was created on Linux and as such requires a Linux terminal to run commands and perform debugging. This project was created on a Windows host machine and as such made use of a tool called Windows Subsystem for Linux (WSL). WSL is a compatibility layer developed by Microsoft that allows the running of Linux distributions natively on Windows 10 and later versions. It allowed for the development of this project to continue in a Linux environment but to be hosted on a Windows machine. The Make and Arm GCC toolchain development tools are packages in Linux that are used for compiling and debugging the C and ARM assembly code found within the PQM4 library and producing executable binaries that can be flashed to the board.

Running the appropriate make command on the root directory of the PQM4 library, links all files together and compiles them into a number of binaries:

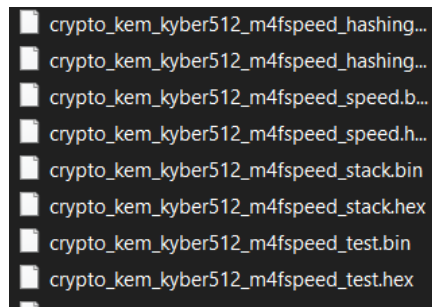


Figure (19) – Sample binaries produced by the PQM4 build system.

The binaries can then be flashed to the board using another Linux tool known as OpenOCD and testing and benchmarking performed using the Linux python-is-python3 package. The PQM4 library contains a testing and benchmarking framework written in Python that can be executed through the Linux terminal to flash the various test and benchmark binaries onto the board through the use of OpenOCD.

Benchmark Interface:

The benchmark interface was developed entirely using Python in the same VS Code environment as the PQM4 library was modified in. The choice of Python for the developmental language was for a number of reasons, mainly due to the speed and ease of which a graphical interface can be constructed within Python through the use of its built-in libraries. Python code is very simple to read and debug and as such allows for a rapid pace of development, well suited for the needs of this project where the merger of the Kyber and Ascon algorithms to produce new schemes was of paramount importance and interface design being a secondary priority. In addition to this, prior experience of using Python to develop graphical interfaces solidified it as the language of choice for the development of the benchmark interface.

5.2 Software Libraries

New Combined Schemes:

As touched on in the previous sub-section, the PQM4 library was used as the basis for this project. It contains everything needed to effectively evaluate the performance of post-quantum encryption schemes on an ARM M4 chip, including a build and debugging system as well as a testing and benchmarking framework. The PQM4 library, benchmarking and testing framework started as a result of the PQCRYPTO project funded by the European Commission in the H2020 program. It currently contains implementations post-quantum key-encapsulation mechanisms and post-quantum signature schemes targeting the ARM Cortex-M4 family [13]. It defines a clear format for each scheme in a logical structure and enables the easy integration of new schemes into the framework. It was selected for use in this project as it eliminates the task of porting a reference and desktop dependent version of Kyber to the Nucleo board as well as providing a robust framework for benchmarking and testing. As part of the KEMs contained within the PQM4 library, different versions of the Kyber512 algorithm are contained within it. These include a clean version written purely in C, a speed-optimised version with assembly language modifications and similarly a stack-optimised version. Having multiple versions of Kyber further serves this project's purpose of determining the effectiveness of replacing the hash function with Ascon by adding reliability to the results obtained

across not just one but three separate versions of Kyber. Further, it adds insight into the effect of modifying the hash function alone will have when compared to the effect of optimising some of the code using assembly language.

To implement the Ascon hash function into the PQM4 library, it was necessary to have an already existing implementation of Ascon written in C to allow for an easier merge of the two schemes. To solve this, the Ascon-C library was used. It consists of a number of different types of algorithms: Authenticated encryption schemes with associated data (AEAD), Hash functions (HASH) and extendible output functions (XOF). For this project, only the hash function and extendible output function was needed as Kyber makes use of the SHA-3 HASH and SHAKE XOF functions to perform these operations meaning this is all that needs to be replaced. For each type of algorithm in the Ascon library there are several versions that exist, some as pure C implementations and some optimised versions for the M4 chip. This project makes use of as many versions of Ascon that were compatible with the M4 chip and PQM4 library, namely Ascon lowsize, speed, size, 32-bit interleaved speed and 32-bit interleaved size versions. For Ascon hash and xof, there also exists a separate version known as Ascon hasha and Ascon Xofa, the difference between these “a” versions and their counterparts is the number of rounds used in performing the absorbing and squeezing operations. A high-level overview of absorbing and squeezing is that they are the cryptographic operations that occur when an input piece of data is absorbed into the hashing algorithm and when the hashed output is squeezed or extracted from the hash.

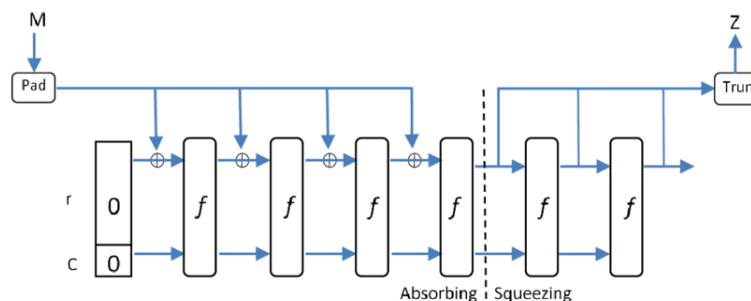


Figure (20) – The absorbing and squeezing phases of the sponge construction like that used in Ascon.

This change in the number of rounds has shown no notable change in the security margins of Ascon and yields a faster performance [ref to ascon changelog 14]. For this reason, the “a” versions of Ascon were selected to be used in the project as they would be even more suitable for lightweight devices.

How the merge of these two libraries was performed:

For the merger of these two libraries to create the various combinations of new Kyber-Ascon hybrid schemes there are several things which needed to be done. The Kyber code first needed to be examined in detail, with all references to any of the hashing algorithms (SHA-3/SHAKE) carefully studied to see how they are used and what would need to be modified to allow for their replacement. Once an understanding of the Kyber code and the workings of the PQM4 library in terms of building, testing and benchmarking is attained, work on beginning the merger of the two schemes can begin. The files relevant to the operation of the Ascon library need to be identified and those not relevant to the task removed. All relevant files need to be added to the build system’s build list and then all references to hashing algorithms made in the Kyber code need to be updated with

references to the corresponding functions in the Ascon library. Once all references are updated, the logic within some key Kyber functions needs to be updated to reflect the difference in the inputs and outputs of the two hashing schemes. An example of this is detailed in the following sub section. After extensive debugging, the merger is complete, and testing followed by benchmarks can be carried out.

Benchmark Interface:

To illustrate the many libraries used in the benchmark interface, a table is detailed below:

Packages and Libraries	Description/Reasoning
Tkinter	Used as a GUI framework for creating windows, widgets and managing interactions between all on-screen and off-screen elements. Features several extensions which were made use of to give the interface a clean and modern appearance.
Matplotlib	Used to produce the charts showing the comparison of performance of all combination of schemes. Allows for extensive customisation options like differentiating each scheme using a separate colour and is extremely widely documented.
OS	Used to obtain the file paths for benchmark files within the system so that the system could use relative path addresses and be usable across many devices.
WebBrowser	Used to open the project's README, which contains information regarding the project as a whole and instructions on how to use the system.
CSV	Used in the reading and writing of all CSV files involved in the system. Reads the individual benchmark CSV files from each scheme's subdirectory and writing these to the combined benchmarks CSV file.
Collections	Used in the updating of benchmark data by keeping track of each scheme's data before it is written to the combined CSV and MD files.
DateTime	Used in the updating of the "last refreshed" time stamp when the benchmarks have been imported/refreshed.
JSON	Used to cache the contents of the table containing the list of all combined schemes and their benchmark import status so that on window load this information can be retrieved and displayed to the user.
RE	Used in the parsing of information from the various CSV files to perform analysis on each scheme to determine the fastest, most memory efficient etc.

Numpy	Used to produce the colour scheme found on the size evaluation chart comparing the code size of each scheme.
-------	--

5.3 Important Functions

Discussed below are the key functions involved in the construction of the new combined schemes. An overview of each function, why it is significant in the system and details on the implementation of the function are elaborated on below.

Key Generation:

Beginning with key generation, it is the process through which keys used in the encapsulation and decapsulation process are obtained. A key pair consisting of a public key and corresponding private key is generated, the public key is used for encapsulation (encryption) operations and the private key is used for the decapsulation (decryption) of data. The public key is derived from the private key in a one-way manner, allowing it to be shared openly to be used in the encryption of data whereas the private key used in decryption is kept secret. The strength of the cryptographic scheme lies in the difficulty in deriving the private key from the public key. A number of parameters that are specified in the Kyber algorithm are used in the derivation of the key pair and the strength and randomness of the keys generated are essential to the security of the scheme.

Hashing is used in the key generation function to provide data integrity and deterministic randomness. This is accomplished through instances such as the public key being hashed for later verification of the public key's integrity by re-computing the hash and comparing it with the stored hash. A further example would be taking an input, applying a hash to it with the hashed output then being used as a seed for generating a matrix of polynomials. This ensures that for the same seed, the same matrix will always be generated, providing deterministic randomness.

The hashing function used for each of these examples was modified from SHA-3 to Ascon through updating references and including new files. One such example of the logic of the code needing to be modified to accommodate the new hash are shown in the images below.

```
void PQCLEAN_KYBER512_CLEAN_gen_matrix(polyvec *a, const uint8_t seed[KYBER_SYMBYTES], int
unsigned int ctr, i, j, k;
unsigned int buflen, off;
uint8_t buf[GEN_MATRIX_NBLOCKS * XOF_BLOCKBYTES + 2];
xof_state_t state;

for (i = 0; i < KYBER_K; i++) {
    for (j = 0; j < KYBER_K; j++) {
        if (transposed) {
            xof_absorb(&state, seed, (uint8_t)i, (uint8_t)j);
        } else {
            xof_absorb(&state, seed, (uint8_t)j, (uint8_t)i);
        }

        xof_squeezeblocks(buf, GEN_MATRIX_NBLOCKS, &state);
        buflen = GEN_MATRIX_NBLOCKS * XOF_BLOCKBYTES;
        ctr = rej_uniform(a[i].vec[j].coeffs, KYBER_N, buf, buflen);

        while (ctr < KYBER_N) {
            off = buflen % 3;
            for (k = 0; k < off; k++) {
                buf[k] = buf[buflen - off + k];
            }
            xof_squeezeblocks(buf + off, 1, &state);
            buflen = off + XOF_BLOCKBYTES;
            ctr += rej_uniform(a[i].vec[j].coeffs + ctr, KYBER_N - ctr, buf, buflen);
        }
    }
}
xof_ctx_release(&state);
```

```
void PQCLEAN_KYBER512_CLEAN_gen_matrix(polyvec *a, const uint8_t seed[KYBER_SYMBYTES], int
unsigned int ctr, i, j, k;
unsigned int buflen, off;
uint8_t buf[GEN_MATRIX_NBLOCKS * XOF_BLOCKBYTES + 2];
ascon_state_t state;

for (i = 0; i < KYBER_K; i++) {
    for (j = 0; j < KYBER_K; j++) {
        ascon_initialisation(&state);
        if (transposed) {
            xof_absorb(&state, seed, (uint8_t)i, (uint8_t)j);
        } else {
            xof_absorb(&state, seed, (uint8_t)j, (uint8_t)i);
        }

        xof_squeezeblocks(buf, GEN_MATRIX_NBLOCKS * XOF_BLOCKBYTES, &state);
        buflen = GEN_MATRIX_NBLOCKS * XOF_BLOCKBYTES;
        ctr = rej_uniform(a[i].vec[j].coeffs, KYBER_N, buf, buflen);

        while (ctr < KYBER_N) {
            off = buflen % 3;
            for (k = 0; k < off; k++) {
                buf[k] = buf[buflen - off + k];
            }
            xof_squeezeblocks(buf + off, XOF_BLOCKBYTES, &state);
            buflen = off + XOF_BLOCKBYTES;
            ctr += rej_uniform(a[i].vec[j].coeffs + ctr, KYBER_N - ctr, buf, buflen);
        }
    }
}
xof_ctx_release(&state);
```

Figures (21) and (22) – The before and after of the matrix generation function from the key generation part of Kyber.

As shown in the above images, Ascon requires a state initialisation to be done before any absorbing or squeezing operations can be performed. In addition, the squeeze function from the image on the left (SHAKE) requires that a number of blocks be specified rather than an output size in bytes. This is due to the method through which the SHAKE algorithm squeezes its outputs. SHAKE works with the concept of blocks fixed at a certain size, 168 bytes in this instance. The matrix generation function shown in both images is written using the concept of blocks as it was originally intended for the SHAKE algorithm, so when replacing it with Ascon a conversion was needed from blocks to bytes. Though it seems trivial, understanding why and where to make these changes was a significant task in modifying the scheme to use Ascon.

Encapsulation and Decapsulation:

The encapsulation and decapsulation operations are what allow for the secure key exchange to occur between parties using an insecure communications channel. Encapsulation consists of encrypting a randomly generated shared secret key using the recipient's public key. This produces a ciphertext which then is transmitted to the recipient. Decapsulation consists of the recipient using their private key to decrypt the received ciphertext, therein recovering the shared secret key. This then enables encrypted communications to take place between the two parties using each other's public and private keys.

Hashing is used extensively through this process. As mentioned in the key generation stage, hashing is also used in encapsulation and decapsulation to provide data integrity and deterministic randomness. It is additionally used here as a pseudorandom number generator for noise generation and for key derivation. The pseudorandom function, as shown in the below image, is used to generate pseudorandom bytes that are used to construct a polynomial with a specific distribution. These randomly distributed polynomials are used both in the key pair generation phase of key generation as well as the encapsulation function. The key derivation function is also illustrated in the image below. A key derivation function is used to derive one or more secret keys from a secret value such as a master key, a password, or a passphrase using a pseudo-random function. It is used in the Kyber encapsulation function to ensure that the shared secret object is of a suitable length and format for the encryption algorithm that will use it.

```
#define xof_absorb(STATE, SEED, X, Y) PQCLEAN_KYBER512_CLEAN_kyber_ascon_absorb(STATE, SEED, X, Y)
#define xof_squeezeblocks(OUT, OUTLENGTH, STATE) ascon_squeeze_interface(OUT, OUTLENGTH, STATE)
#define xof_ctx_release(STATE) ascon_ctx_release(STATE)
#define ascon_initialisation(STATE) ascon_init(STATE);
#define prf(OUT, OUTBYTES, KEY, NONCE) PQCLEAN_KYBER512_CLEAN_kyber_ascon_prf(OUT, OUTBYTES, KEY, NONCE)
#define kdf(OUT, IN, INBYTES) ascon_var(OUT, KYBER_SSBYTES, IN, INBYTES)
```

Figure (23) – A selection of functions that have been replaced with custom Ascon counterparts, including the key derivation function and pseudorandom function.

6. Testing

This section contains details on all forms of testing used in the system. It outlines the test approach taken and provides justifications for the decisions made and evidence of comprehensive testing having taken place. Test cases are detailed for all aspects of the system, both front-end and the back-end portions. A description of how to run the automated back-end tests is also mentioned and where documentation can be located is also detailed.

6.1 Back-End Testing:

The back-end portion of the system consists of the new combined schemes, so testing the correctness and completeness of these schemes is the end goal of all back-end testing in the project. The PQM4 library used as the basis in the creation of the combined schemes very handily features its own built-in automated functional testing framework for all KEM schemes included in the library. This functional testing framework was used extensively throughout the development of the project, with each replacement of a SHA/SHAKE hash function in the Kyber code being followed by running the functional testing suite on the modified scheme to ensure that the intended operations of the scheme were not altered. Ensuring the reliability and security of KEMs is essential to the operation of the cryptographic schemes so it is important to have a robust and effective testing framework to add validation to the security and operation of the new schemes.

Throughout the description of all tests involved in the back end of the system, the names Alice and Bob are used frequently. In the fields of computer science and cryptography, Alice and Bob are commonly used as stand-in names for the two parties in a communication scenario or cryptographic protocol. Bob usually represents the receiving party, whereas Alice usually represents the party starting the conversation or carrying out an action. By using these names, examples and explanations become more comprehensible and straightforward without requiring the use of difficult or specialised language.

There are three test cases that are covered as part of the PQM4 functional testing framework.

Key testing:

This test case scrutinises the entire lifecycle of a KEM operation, including key generation, encapsulation, and decapsulation. By generating public and secret keys, creating a response, and verifying the derived secret key between Alice and Bob, this test ensures the consistency and correctness of the KEM implementation.

Invalid secret key testing:

This test case tests the scenario where an invalid secret key is provided during the decapsulation process. By replacing the secret key with random values, the test assesses the system's ability to handle erroneous and unexpected inputs and maintain security integrity despite anomalies being present.

Invalid ciphertext testing:

This test case focuses on evaluating the system's resilience against modified/tampered ciphertexts. By altering the ciphertext to a random value, the test investigates if the system can detect and

mitigate potential attacks, ensuring that the derived secret key remains consistent between Alice and Bob.

These test cases were carefully chosen to cover important aspects of KEM security and functionality. The key testing case ensures the accuracy and cohesion of the KEM by thoroughly evaluating the processes of key generation, encapsulation, and decapsulation. As well as this, the invalid secret key and invalid ciphertext scenarios assess the system's resistance to potential attacks and real-world threats by simulating adversarial scenarios.

The above test cases are included in the evaluation process for a number of important reasons. Firstly, they assist in identifying possible gaps and vulnerabilities in the KEM implementation, allowing for the implementation of preventative measures to improve security and resilience. Further, a thorough assessment of the system's essential features guarantees that it functions as intended, boosting confidence in its dependability and credibility. Lastly, by creating adversarial scenarios, the tests help strengthen the system's defences against possible attacks.

As mentioned previously, the correctness of all new scheme combinations was verified using the above test cases. If at any point during the new schemes' development any of the test cases failed, something was known to be wrong in the code and a change had to be made.

Running these tests on the Nucleo board for any of the schemes will yield the following output, given the correct commands found in the documentation were used to produce the output.

```
INFO:Implementation:Building kyber512 - m4fspeed - test
SRCDIR is /mnt/c/Users/natha/Documents/CSYEAR3/PROJECT/GITLABREPO/a-light-weight-kem
Contents of mk directory:
config.mk crypto.mk cw308t-stm32f3.mk mps2-an386.mk nucleo-l476rg.mk nucleo-l476zg.mk
PLATFORM is nucleo-l4r5zi
OBJCOPY bin/crypto_kem_kyber512_m4fspeed_test.hex
DEBUG:platform interface:Found start pattern: b'=====\\n'

INFO:BoardTestCase:Success
INFO:Implementation:Building kyber512 - m4fstack - test
SRCDIR is /mnt/c/Users/natha/Documents/CSYEAR3/PROJECT/GITLABREPO/a-light-weight-kem
Contents of mk directory:
config.mk crypto.mk cw308t-stm32f3.mk mps2-an386.mk nucleo-l476rg.mk nucleo-l476zg.mk
PLATFORM is nucleo-l4r5zi
OBJCOPY bin/crypto_kem_kyber512_m4fstack_test.hex
DEBUG:platform interface:Found start pattern: b'=====\\n'

INFO:BoardTestCase:Success
INFO:Implementation:Building kyber512 - clean - test
SRCDIR is /mnt/c/Users/natha/Documents/CSYEAR3/PROJECT/GITLABREPO/a-light-weight-kem
Contents of mk directory:
config.mk crypto.mk cw308t-stm32f3.mk mps2-an386.mk nucleo-l476rg.mk nucleo-l476zg.mk
PLATFORM is nucleo-l4r5zi
OBJCOPY bin/mupq_pqclean_crypto_kem_kyber512_clean_test.hex
DEBUG:platform interface:Found start pattern: b'=====\\n'
```

Figure (24) – Linux terminal output showing the successful testing of all variations of the Kyber 512/Ascon low size scheme.

Direct output from the board itself can be obtained through use of the “screen” command, also documented on the project’s README:



```
DONE key pair generation!
DONE encapsulation!
DONE decapsulation!
OK KEYS
OK invalid sk_a
OK invalid sk_a
OK invalid sk_a
OK invalid sk_a
OK invalid sk_a
OK
OK invalid sk_a
OK invalid sk_a
OK invalid ciphertext
OK invalid ciphertext
OK invalid ciphertext
OK invalid ciphertext
OK invalid ciphertext
OK invalid ciphertext
```

Figure (25) – Linux terminal screen command output showing the successful testing of all variations of the Kyber 512/Ascon low size scheme.

Note that the above screen output is direct from the Nucleo board so is not optimised for readability, however this output proved to be essential to the development of the new schemes as it would show specifically which part of the test case failed for which scheme.

6.2 Front-End Testing

To perform testing on the front end, manual testing was employed. This was done for several reasons. Firstly, due to time constraints in developing the complete system, the priority of the project was the development and verification of the new combined schemes. This meant that the majority of development time was spent on the creation and testing of the new schemes, leaving the interface development as a secondary priority. This was detailed in previous sections but also extends to testing the interface as there was not sufficient time to develop an automated test suite to cover all use cases of the benchmark viewer interface. However, due to the small size of the interface and its limited functionality, manual testing of a number of predetermined test cases can serve to comprehensively verify the correct functionality of the interface. Discussed below are a number of advantages and disadvantages of manual testing in the context of the project.

Simplicity: manual testing is straightforward and requires minimal setup like creating test classes, learning how to utilise frameworks and concepts surrounding different types of testing, making it suitable for small interfaces with limited functionality, like the benchmark viewer interface.

Flexibility: manual testing allows testers to quickly adapt test cases and scenarios to accommodate changes in requirements or design iterations. This was particularly advantageous in the development of the interface, due to its short timeframe and rapidly evolving requirements.

Subjectivity: Test results in manual testing may vary depending on the tester's interpretation and judgment, leading to subjective assessments and potential bias. All testing of the interface in this project was done by the developer so there exists a potential for incompleteness in some parts of the interface or sufficient testing depth not being achieved.

Lack of repeatability: Manual tests may not be easily repeatable or reproducible, as they depend on the tester's actions and may vary between test executions. Again, all testing was done by the developer, meaning that if the system is to be re-tested, the outcomes may differ from what is written in this report.

With the advantages and disadvantages of manual testing considered, manual testing does appear to be suitable for the small-scale interface used in this project given the time constraints on development.

Test Case	Expected Output	Actual Output	Passed/Failed
TC1 – Import/Refresh Benchmarks button opens correct window.	The import/refresh benchmarks window is opened.	The import/refresh benchmarks window is opened.	Passed
TC2 – View Benchmarks button opens correct window.	View benchmarks window is opened.	View benchmarks window is opened.	Passed
TC3 – Last refreshed time and date displayed correctly on load.	Last refreshed time and date displayed correctly on window load.	Last refreshed time and date displayed correctly on window load.	Passed
TC4 – Scheme names with associated imported/error status displayed correctly.	Imported/error importing displayed correctly beside each scheme.	Imported/error importing displayed correctly beside each scheme.	Passed
TC5 – Refresh benchmark button imports schemes, updating relevant fields.	On clicking refresh button schemes are imported and last updated time updated.	On clicking refresh button schemes are imported and last updated time updated.	Passed
TC6 – Brief summary of results panel displays results correctly.	Summary panel displays selected schemes correctly.	Summary panel displays selected schemes correctly.	Passed
TC7 – Brief summary of results panel displays error message when results cannot be imported.	Summary panel displays error message when schemes cannot be imported.	Summary panel displays error message when schemes cannot be imported.	Passed
TC8 – View raw CSV data button opens combined CSV file when present.	Clicking view raw CSV button opens combined CSV file.	Clicking view raw CSV button opens combined CSV file.	Passed
TC9 – View raw CSV data button displays error message when file not present.	Clicking view raw CSV button displays error message.	Clicking view raw CSV button displays error message.	Passed
TC10 – View raw MD data button opens combined MD file when present.	Clicking view raw MD button opens combined MD file.	Clicking view raw MD button opens combined MD file.	Passed
TC11 – View raw MD data button displays error message when file not present.	Clicking view raw MD button displays error message.	Clicking view raw MD button displays error message.	Passed

TC12 – View bar charts button opens bar chart selector window.	Clicking on view bar charts button opens the bar chart selector window.	Clicking on view bar charts button opens the bar chart selector window.	Passed
TC13 – Speed evaluation button opens window containing speed evaluation chart.	Clicking on speed evaluation button opens speed evaluation chart.	Clicking on speed evaluation button opens speed evaluation chart.	Passed
TC14 – Speed evaluation button opens window containing error message when data not available.	Clicking on speed evaluation button displays an error message.	Clicking on speed evaluation button displays an error message.	Passed
TC15 – Memory evaluation button opens window containing memory evaluation chart.	Clicking on memory evaluation button opens memory evaluation chart.	Clicking on memory evaluation button opens memory evaluation chart.	Passed
TC16 – Memory evaluation button opens window containing error message when data not available.	Clicking on memory evaluation button displays an error message.	Clicking on memory evaluation button displays an error message.	Passed
TC17 – Hash proportion button opens window containing hash proportion chart.	Clicking on hash proportion button opens hash proportion chart.	Clicking on hash proportion button opens hash proportion chart.	Passed
TC18 - Hash proportion button opens window containing error message when data not available.	Clicking on hash proportion button displays an error message.	Clicking on hash proportion button displays an error message.	Passed
TC19 - Size evaluation button opens window containing size evaluation chart.	Clicking on size evaluation button opens size evaluation chart.	Clicking on size evaluation button opens size evaluation chart.	Passed
TC20 - Size evaluation button opens window containing error message when data not available.	Clicking on size evaluation button displays an error message.	Clicking on size evaluation button displays an error message.	Passed

7. Experimental Results and Analysis

This section consists of a thorough evaluation of the project's outcomes and experimental results. By systematically analysing the data gathered through experimentation, the goal is to assess the success of the project in achieving its objectives and contributing to the field of post-quantum cryptography. Topics discussed in this section include the experimental methodology employed, procedures involved, and results observed. Further, the obtained results are then analysed, drawing comparisons with existing work where applicable.

7.1 Experimental Methodology

Beginning with the experimental methodology employed in the obtaining of results from the system, the approach was as follows. It was immensely important to ensure that the tenets of the scientific method were used as a foundation for all results obtained in this project, meaning that all observations must be empirical, all results must be reproducible, and any hypotheses generated must be logical and objective.

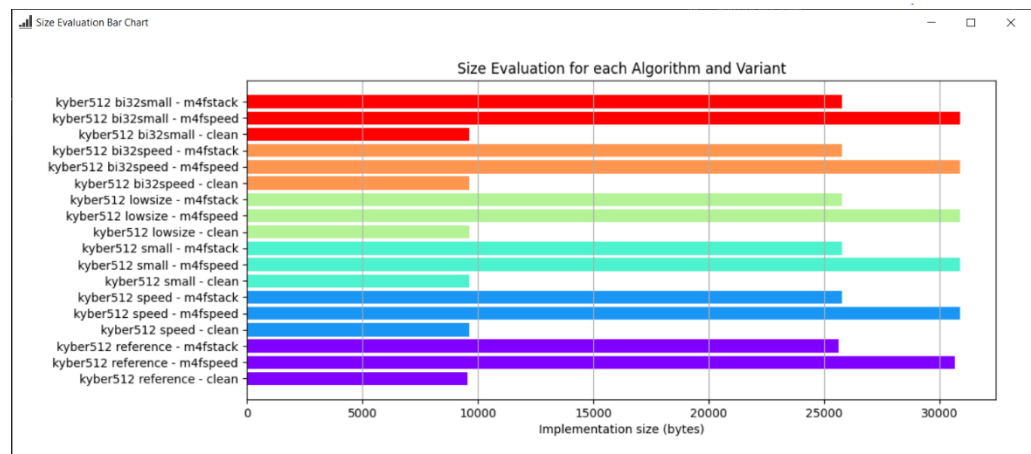
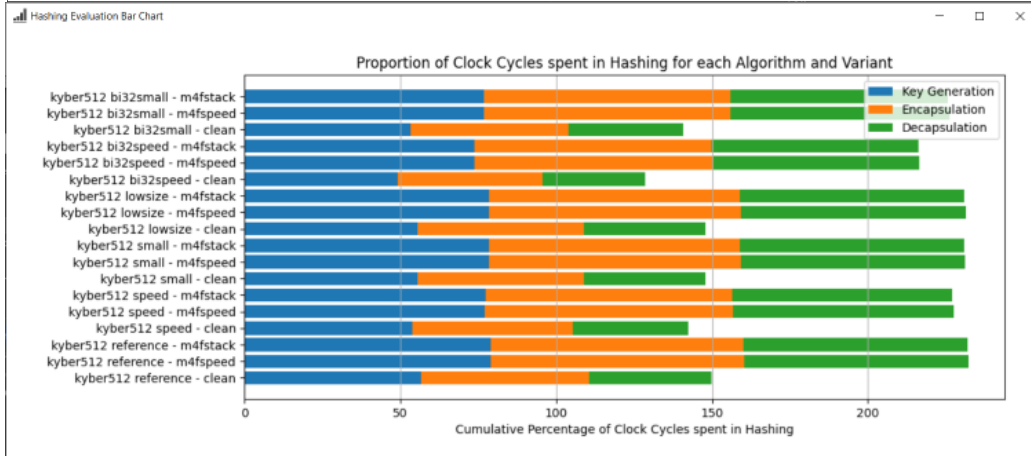
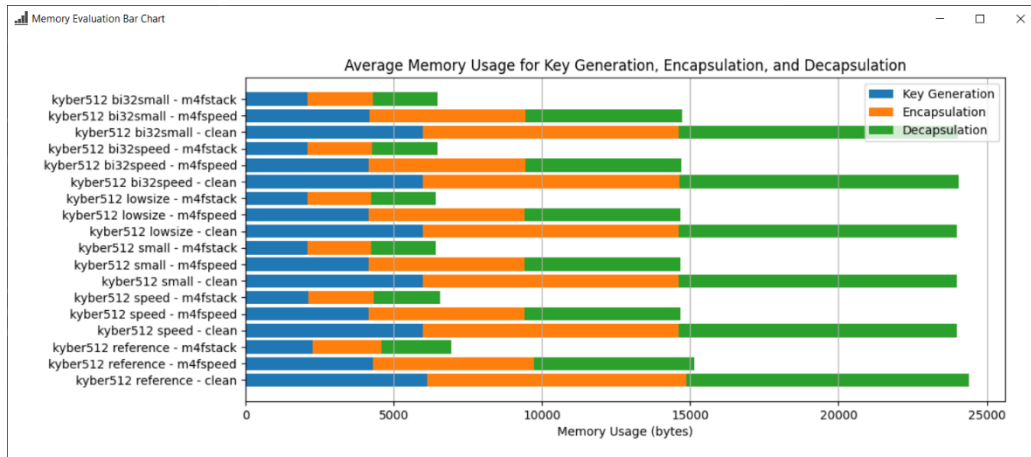
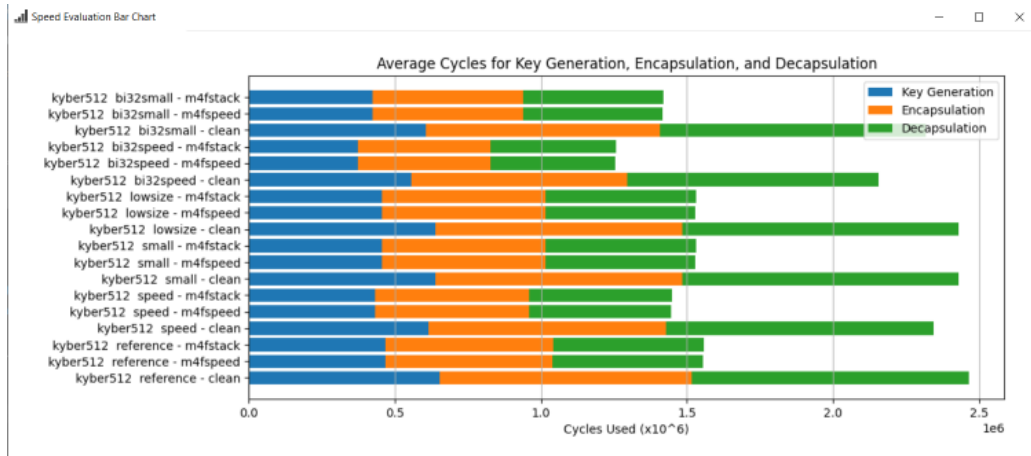
In the production of benchmarks for the new combined schemes, it was important to make note of what variables were involved in the experiment. There is the independent variable (the new combined schemes), which is the variable that is manipulated in order to observe its effect on the dependent variable. There is also the dependent variable (the recorded benchmarks of cycles used, memory used, code size and hash proportion) which is the variable that is measured to determine the effect of the independent variable. Finally, there is the control variable (the Nucleo board) which is the variable that is kept as a constant to prevent any interference in the measurement of the dependent variable.

It was important throughout the obtaining of results of the new combined schemes that the conditions under which the schemes were benchmarked remained consistent throughout. As such a number of precautions taken are detailed below:

- The same Nucleo L4R5ZI board was used in all benchmarking.
- The same USB connector cable was used in all benchmarking.
- Consistent environmental conditions were present in all benchmarking.
- The same base Kyber 512 codebase was used in all benchmarking.
- The same benchmarking suite was used in the obtaining of all results.
- All benchmarking was ran 10 times to ensure reliability of results.

7.2 Experimental Results

With the experimental methodology used in the obtaining of results detailed, the results obtained from the benchmarking of each of the 5 Kyber512-Ascon combinations is shown below, with accompanying reference Kyber512 data:



Figures (26-29) – The speed evaluation, size evaluation, memory evaluation and hash proportion charts as output from the system for 10 executions.

As shown in the above files, there is a marked difference between each Kyber-Ascon combination and the Kyber reference scheme across all benchmark metrics.

Further shown in the appendix are the combined MD files, illustrating the obtained results in greater detail.

7.3 Analysis of Results

To proceed with analysing the results as a whole, there are 4 recorded metrics for each version of the new combined scheme. These are as follows:

The average number of clock cycles used in key generation, encapsulation and decapsulation:

Beginning with the average number of clock cycles used. At a high level there is a notable difference in the number of clock cycles used between all Ascon version of the scheme as compared to the reference implementation. Even the most stack optimised versions of Ascon (small and low size) showed a faster execution time with lower clock cycles used than the reference version. A trend that is observed throughout all implementations is the clean variations of Kyber being the slowest followed by the stack optimised variations and then the speed optimised variations which are roughly equivalent to each other in terms of speed. This is explained with the speed and stack optimised versions of Kyber having a large amount of functionality written using assembly language, lending to greatly improved efficiency of the implementations. The improvements in clock cycle usage observed over all versions of the Kyber-Ascon hybrid schemes appear to be roughly equal over the key generation, encapsulation and decapsulation phases, with no particular phase showing greater improvement than the others. This will be examined further in the hash proportion analysis section.

The scheme that overall used the least number of cycles on across key generation, encapsulation and decapsulation was the 32-bit-interleaved-speed Ascon combined with the speed optimised version of Kyber. The reference version of Kyber speed-optimised, across 10 iterations took a calculated average of 1,554,485 cycles in total, whereas the 32-bit-interleaved-speed version of Ascon/Kyber took a calculated average of 1,253,655 cycles. This showed an ~24% lower cycle usage for the 32-bit-interleaved-speed version of the new combined scheme. This improvement of speed is attributed to the bit-interleaving technique that was employed in the Ascon scheme, improving efficiency and maximising the utilisation of available ARM M4 chip resources.

The average memory usage for key generation, encapsulation and decapsulation:

Investigating the average memory usage for each of the new combined schemes, it does not yield a great improvement over the reference versions of Kyber. There is a small improvement observed for all Ascon versions of the new combined scheme over the reference versions of Kyber with each combination involving ascon being between approximately 2% at minimum to 8% at maximum more memory efficient than the reference versions. A trend observed over all combinations of Kyber and Ascon is the memory usage of speed and stack optimised versions of Kyber are significantly less and

in some cases almost a quarter of what the clean (pure C) implementations of Kyber are. This is again due to large portions of the speed and stack optimised versions being written in assembly language, meaning that faster and more efficient execution is possible. It is interesting to note that even the speed optimised versions of Kyber are almost 50% more memory efficient than the clean versions of Kyber, despite not being written specifically for stack optimisation. This showcases the potential that writing pure assembly implementations of post-quantum cryptographic schemes has in the space of performance optimisation.

The scheme that overall used the least amount of memory was the small version of Ascon, combined with the stack optimised version of Kyber. The reference version of stack optimised Kyber used 6932 bytes throughout key generation, encapsulation and decapsulation, whereas the small Ascon/Kyber stack hybrid scheme used 6432 bytes, showing ~8% less memory usage while simultaneously using less clock cycles.

The proportion of clock cycles spent in hashing functions for key generation, encapsulation and decapsulation:

The proportion of clock cycles taken in hashing reflects how much the amount of clock cycles required to perform the hashing in each stage of the benchmark has improved compared to the reference versions of Kyber. As can be seen in the chart, there is significant improvement in the proportion of hashing across that mirrors the improvement seen in the number of clock cycles used. The number of clock cycles spent in hashing is most reduced in the clean versions of Kyber, where a general trend can be observed across all versions, both reference and modified implementations, that the proportion of clock cycles spent in hashing is significantly less than the stack and speed optimised versions. This could be due to a number of reasons, one such explanation is that the stack and speed optimised versions of Kyber have optimised the non-hashing aspects of each implementation, leading to a larger proportion of cycles being spent in the non-optimised hashing parts of the code. Compared to the reference versions of Kyber, the modified schemes show a wide range of differing amounts of time spent in hashing, with as little as a 1.5% less clock cycle change in the low size version of Ascon/Kyber to a significant 7% in the 32-bit-interleaved version of Ascon/Kyber. The observed differences in hashing proportion in the modified schemes compared to the reference implementations is evidence that the replacement of the SHA/SHAKE family of hashes with ASCON has successfully reduced the number of resources used in the hash portion of the scheme, as the amount of time spent in hashing over each stage of the scheme has reduced, meaning that it now takes less time. As for the specific variations in each Ascon/Kyber hybrid, the differences are minimal between combined scheme versions and are explained with the specific goal of each version of Ascon. E.g. speed optimised requiring less time etc.

The size of each implementation:

Finally, analysing the results of the size evaluation of each implementation. Overall, the size of each implementation is very similar, with all Ascon/Kyber hybrid schemes possessing the exact same size, which itself is only slightly larger than the reference Kyber implementations. The average difference between the Ascon/Kyber versions and the reference Kyber versions is between ~0.5 to 0.8%. This small increase in size is a result of the Ascon hash scheme having a slightly larger code size than the SHA/SHAKE schemes used in the reference implementations of Kyber. All ascon versions have the same size due to the hash function within each scheme remaining fundamentally the same. It is the

parameters that change within each version such as number of rounds and block size which influence the performance of the algorithm which do not affect the code size. Although the new combined schemes are slightly larger than the reference versions of Kyber, as was revealed the difference is less than 1% with this difference being compensated by having a better memory efficiency and significantly improved performance over the reference versions of Kyber.

8. Conclusions and Future Work

To conclude the report, this section contains a comprehensive overview of the project's outcomes, discussing its achievements, societal implications, strengths, weaknesses, and avenues for future research and development. Detailed here is a complete evaluation of the project's goals, assessing the extent to which they have been met, highlighting key findings and insights derived from the experimental results. Furthermore, reflection on the broader societal implications of the project and its potential impact on post-quantum cryptography for lightweight devices is also considered. An analysis is conducted on the project's strengths and weaknesses, identifying areas of success and areas for improvement. Finally, potential directions for future work, including areas of further investigation and extension are investigated.

8.1 Summary of findings:

Overall, it was observed that replacing the hashing scheme used in the Kyber 512 algorithm as a part of the PQM4 library with the Ascon hash yielded a number of positive results for the target of increasing its suitability for lightweight and embedded applications. Significant improvements in performance were observed with as much as a ~25% reduction in clock cycle use compared to reference versions of Kyber. Improvements in memory usage were also found which with as much as ~8% in the most memory-optimised scheme, all while maintaining an improvement in performance. The amount of clock cycles spent in hashing decreased for key generation, encapsulation and decapsulation while implementation size increased by less than 1%.

These findings are very promising, demonstrating that substituting the hashing algorithm used in a post-quantum cryptographic scheme alone is already sufficient to yield substantial improvements in suitability for lightweight applications, with a reduced memory usage and improved performance being observed.

8.2 Evaluation of project's goals:

The overarching aim of this project was to attempt to produce a post-quantum cryptographic scheme that would be more suitable for lightweight/embedded applications, such as IoT devices. This was intended to combat the future problem of classical cryptographic algorithms being compromised by quantum computers and unsuitability of most current post-quantum cryptographic schemes for lightweight devices. It is the belief of the writer that this aim was met, with the substitution of the hashing algorithm used in the Kyber 512 algorithm for the more lightweight Ascon hash yielding a number of performance improvements while also reducing resource usage. The new combined schemes that have been produced in this project serve as implementations suitable for various scenarios. Ranging from more performance-critical scenarios, using the speed-optimised versions of Kyber/Ascon to more memory-critical scenarios, using the more size-optimised versions of Kyber/Ascon. The Ascon hash has completely replaced all usages of the SHA/SHAKE algorithms throughout Kyber and an interface has been produced to allow for the comparison of results of benchmarking of the new schemes. With the project's outputs noted, it is the writer's belief that all project goals have been satisfied.

8.3 Strengths and weaknesses:

Throughout the development of the combined schemes and interfaces associated with the project, there have been a number of successes, which are also accompanied by a number of setbacks. Some of the key strengths of the project have been the results produced by the combined schemes with the limited amount of optimisation that has taken place. There exists a vast amount of possibility to further tailor the Ascon algorithm to be particularly suitable to Kyber, only a basic implementation was done in this project. Despite this, the results produced are very promising, with significant performance increases being observed with simultaneously less memory usage and a similar implementation size. A working implementation of Kyber with a number of versions of Ascon has been produced during this project, lending itself to further study and a fresh avenue of investigation for future optimisations. The biggest weakness of this project was its development time leading to a reduced scope. The number of things that could be achieved through this project were limited significantly by its development time. The development of the project was part-time, with other university modules competing for development time, meaning that the scope of what could realistically be achieved in the project was limited. The interface through which the combined schemes are compiled, tested and benchmark is not streamlined and not intuitive. This among other usability and documentation issues would be included in the scope of the project, given a larger development timeframe was available.

8.4 Societal implications and future work:

The results obtained from the combined Ascon/Kyber schemes produced in this project have a number of implications on the field of post-quantum cryptography, namely lightweight PQC. The findings show promise that replacing the hashing algorithm used in a cryptographic scheme is a viable method in improving the performance and modifying other characteristics of a scheme. They open a new avenue for tailoring and optimising the Ascon scheme for further suitability to the Kyber algorithm. Primarily, the results obtained indicate that the Kyber algorithm with the Ascon hash incorporated is more suitable for lightweight devices due to its lower memory usage, similar size signature and significantly improved performance. There exists a vast potential for this project to be expanded upon for future research, detailed below are a number of directions that the project could be taken.

Further optimisation to assembly language:

As was observed in the results of the benchmarking of the combined schemes, the speed and stack optimised versions of Kyber were significantly faster than the pure C implementations. The difference in performance caused by the re-writing of Kyber code into assembly language far exceeds that caused by integrating Ascon into the scheme. This shows tremendous potential for expansion of the combined Kyber/Ascon scheme into an assembly project. This comes with the downside of being platform-dependent.

Tuning of Ascon hash to Kyber:

There are a number of parameters that can be configured in the Ascon algorithm, such as number of rounds, block size and key size. There might exist a combination of these parameters that is optimal for synergy with the Kyber algorithm that would yield even greater compatibility and potentially improved performance. A full investigation into tuning the Ascon algorithm for Kyber could be undertaken.

Improvement of benchmarking suite:

The current benchmarking suite, while comprehensive as is, could be expanded to include other metrics such as a security evaluation against different types of attack, power consumption and resistance to faults. Further charts and metrics, showing detailed analysis of patterns and trends observed from the various combinations of the schemes could also be produced.

Replacement of Ascon with alternative hash:

It has been demonstrated in this project that the replacement of the SHA/SHAKE hashing functions with Ascon has had a number of impacts on the different performance metrics of the Kyber scheme. Further research into alternative hashing algorithms that could be used in place of Ascon could be conducted and merged with Kyber to evaluate the effectiveness of these alternative schemes.

Replacement of Kyber with alternative scheme:

The Kyber algorithm was chosen as the algorithm of choice for this project due to its standardisation by NIST in its post-quantum cryptography competition. There exist a large number of alternative PQC schemes that could be suitable for experimentation with modification to suit more lightweight applications.

9. References

- [1] Monz, T., Nigg, D., Martinez, E.A., Brandl, M.F., Schindler, P., Rines, R., Wang, S.X., Chuang, I.L. and Blatt, R., 2016. Realization of a scalable Shor algorithm. *Science*, 351(6277), pp.1068-1070.
- [2] National Institute of Standards and Technology, “Post-Quantum Cryptography” nist.gov, para 1, Jan. 3, 2017 [Online]. Available: <https://csrc.nist.gov/projects/post-quantum-cryptography>. [Accessed Sept 20, 2023]
- [3] F. Richter, “The Post-PC Era Has Arrived,” statista.com, chart 1, May. 7, 2012. [Online]. Available: <https://www.statista.com/chart/276/global-tablet-smartphone-and-pc-shipments-from-2010-to-2016/>. [Accessed Sept 20, 2023]
- [4] PQ-Crystals, “Cryptographic Suite for Algebraic Lattices” pq-crystals.org, para 1, Dec 23, 2020 [Online]. Available: <https://pq-crystals.org/kyber/> [Accessed Jan 10, 2024]
- [5] C. D. Schl  ffer Maria Eichlseder, Florian Mendel, Martin, “Ascon – Authenticated Encryption and Hashing,” ascon.iaik.tugraz.at. <https://ascon.iaik.tugraz.at/> [Accessed April 14, 2024]
- [6] National Institute of Standards and Technology, “Lightweight Cryptography” nist.gov, para 1, Jan. 3, 2017 [Online]. Available: <https://csrc.nist.gov/projects/lightweight-cryptography>. [Accessed Sept 20, 2023]
- [7] S. Segars, “Arm Partners Have Shipped 200 Billion Chips,” arm.com, para 1, Oct. 18, 2021. [Online]. Available: <https://www.arm.com/blogs/blueprint/200bn-arm-chips> [Accessed Sept 20, 2023]
- [8] M.-J. Saarinen, “Exploring NIST LWC/PQC Synergy with R5Sneik How SNEIK 1.1 Algorithms were Designed to Support Round5.” Available: https://csrc.nist.gov/CSRC/media/Events/Second-PQC-Standardization-Conference/documents/accepted-papers/saarinen_r5sneiktxt.pdf [Accessed April 14, 2024]
- [9] M.-J. O. Saarinen, “Ring-LWE Ciphertext Compression and Error Correction,” Apr. 2017, Available: <https://doi.org/10.1145/3055245.3055254>. [Accessed April 14, 2024]
- [10] S. Kumari, M. Singh, R. Singh, and H. Tewari, “A post-quantum lattice based lightweight authentication and code-based hybrid encryption scheme for IoT devices,” *Computer Networks*, vol. 217, p. 109327, Nov. 2022, Available: <https://doi.org/10.1016/j.comnet.2022.109327>. [Accessed April 14, 2024]
- [11] D. Moody et al., “Status report on the second round of the NIST post-quantum cryptography standardization process,” Page 4, Jul. 2020, Available: <https://doi.org/10.6028/nist.ir.8309>. [Accessed April 14, 2024]
- [12] “Kyber - How does it work?,” *Approachable Cryptography*, Sep. 14, 2021. Available: <https://cryptopedia.dev/posts/kyber/> [Accessed April 14, 2024]

- [13] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, “pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4,” ePrint IACR, 2019. Available <https://eprint.iacr.org/2019/844> [Accessed April 14, 2024]

10. Appendices

10.1 The complete results MD file speed evaluation section:

Speed Evaluation

scheme	implementation	key generation [cycles]	encapsulation [cycles]	decapsulation [cycles]
kyber512 reference (10 executions)	clean	AVG: 652,873	AVG: 863,682	AVG: 947,459
		MIN: 652,784	MIN: 863,593	MIN: 947,370
		MAX: 652,924	MAX: 863,733	MAX: 947,510
kyber512 reference (10 executions)	m4fspeed	AVG: 466,988	AVG: 573,402	AVG: 514,095
		MIN: 466,752	MIN: 573,165	MIN: 513,858
		MAX: 467,164	MAX: 573,577	MAX: 514,271
kyber512 reference (10 executions)	m4fstack	AVG: 466,758	AVG: 575,359	AVG: 516,118
		MIN: 466,621	MIN: 575,223	MIN: 515,982
		MAX: 466,933	MAX: 575,534	MAX: 516,293
kyber512 speed (10 executions)	clean	AVG: 613,817	AVG: 814,066	AVG: 918,143
		MIN: 613,697	MIN: 813,946	MIN: 918,023
		MAX: 613,885	MAX: 814,134	MAX: 918,211
kyber512 speed (10 executions)	m4fspeed	AVG: 431,293	AVG: 527,142	AVG: 488,127
		MIN: 431,129	MIN: 526,980	MIN: 487,964
		MAX: 431,411	MAX: 527,260	MAX: 488,245
kyber512 speed (10 executions)	m4fstack	AVG: 430,893	AVG: 528,931	AVG: 489,975
		MIN: 430,586	MIN: 528,625	MIN: 489,669
		MAX: 431,111	MAX: 529,148	MAX: 490,192
kyber512 small (10 executions)	clean	AVG: 639,016	AVG: 845,494	AVG: 944,075
		MIN: 638,735	MIN: 845,213	MIN: 943,794
		MAX: 639,184	MAX: 845,662	MAX: 944,243
kyber512 small (10 executions)	m4fspeed	AVG: 456,436	AVG: 558,515	AVG: 514,007
		MIN: 456,262	MIN: 558,341	MIN: 513,834
		MAX: 456,525	MAX: 558,605	MAX: 514,097
kyber512 small (10 executions)	m4fstack	AVG: 455,902	AVG: 560,174	AVG: 515,720
		MIN: 455,788	MIN: 560,060	MIN: 515,606
		MAX: 455,966	MAX: 560,239	MAX: 515,785
kyber512 lowsize (10 executions)	clean	AVG: 639,021	AVG: 845,498	AVG: 944,079
		MIN: 638,965	MIN: 845,443	MIN: 944,024
		MAX: 639,076	MAX: 845,553	MAX: 944,134
kyber512 lowsize (10 executions)	m4fspeed	AVG: 456,499	AVG: 558,578	AVG: 514,071
		MIN: 456,276	MIN: 558,354	MIN: 513,847
		MAX: 456,934	MAX: 559,013	MAX: 514,505
kyber512 lowsize (10 executions)	m4fstack	AVG: 456,008	AVG: 560,280	AVG: 515,826
		MIN: 455,891	MIN: 560,161	MIN: 515,707
		MAX: 456,099	MAX: 560,370	MAX: 515,916
kyber512 bi32speed (10 executions)	clean	AVG: 556,047	AVG: 740,344	AVG: 858,333
		MIN: 555,788	MIN: 740,085	MIN: 858,074
		MAX: 556,207	MAX: 740,504	MAX: 858,493
kyber512 bi32speed (10 executions)	m4fspeed	AVG: 372,989	AVG: 452,886	AVG: 427,780
		MIN: 372,895	MIN: 452,792	MIN: 427,686
		MAX: 373,087	MAX: 452,983	MAX: 427,877
kyber512 bi32speed (10 executions)	m4fstack	AVG: 372,467	AVG: 454,548	AVG: 429,502
		MIN: 372,197	MIN: 454,278	MIN: 429,232
		MAX: 372,612	MAX: 454,693	MAX: 429,647
kyber512 bi32small (10 executions)	clean	AVG: 605,807	AVG: 802,490	AVG: 909,661
		MIN: 605,686	MIN: 802,369	MIN: 909,540
		MAX: 605,944	MAX: 802,627	MAX: 909,798
kyber512 bi32small (10 executions)	m4fspeed	AVG: 422,704	AVG: 514,981	AVG: 479,057
		MIN: 422,560	MIN: 514,836	MIN: 478,911
		MAX: 422,805	MAX: 515,083	MAX: 479,159
kyber512 bi32small (10 executions)	m4fstack	AVG: 422,180	AVG: 516,649	AVG: 480,786
		MIN: 421,990	MIN: 516,458	MIN: 480,595
		MAX: 422,376	MAX: 516,845	MAX: 480,982

10.2 The complete results MD file memory evaluation section.

Memory Evaluation

Scheme	Implementation	Key Generation [bytes]	Encapsulation [bytes]	Decapsulation [bytes]
kyber512 reference	clean	6,116	8,764	9,532
kyber512 reference	m4fspeed	4,312	5,408	5,416
kyber512 reference	m4fstack	2,252	2,332	2,348
kyber512 speed	clean	5,976	8,632	9,400
kyber512 speed	m4fspeed	4,152	5,256	5,264
kyber512 speed	m4fstack	2,120	2,208	2,224
kyber512 small	clean	5,976	8,632	9,400
kyber512 small	m4fspeed	4,152	5,256	5,264
kyber512 small	m4fstack	2,080	2,168	2,184
kyber512 lowsize	clean	5,976	8,632	9,400
kyber512 lowsize	m4fspeed	4,152	5,256	5,264
kyber512 lowsize	m4fstack	2,080	2,168	2,184
kyber512 bi32speed	clean	5,988	8,644	9,412
kyber512 bi32speed	m4fspeed	4,164	5,268	5,276
kyber512 bi32speed	m4fstack	2,092	2,180	2,196
kyber512 bi32small	clean	5,984	8,640	9,408
kyber512 bi32small	m4fspeed	4,172	5,276	5,284
kyber512 bi32small	m4fstack	2,100	2,188	2,204

10.3 The complete results MD file hashing proportion section.

Hashing Evaluation

Scheme	Implementation	Key Generation [%]	Encapsulation [%]	Decapsulation [%]
kyber512 reference	clean	56.6%	54.0%	39.2%
kyber512 reference	m4fspeed	79.0%	81.2%	72.1%
kyber512 reference	m4fstack	79.0%	81.0%	71.8%
kyber512 speed	clean	53.9%	51.3%	37.3%
kyber512 speed	m4fspeed	77.2%	79.6%	70.6%
kyber512 speed	m4fstack	77.3%	79.3%	70.4%
kyber512 small	clean	55.7%	53.1%	39.1%
kyber512 small	m4fspeed	78.4%	80.7%	72.1%
kyber512 small	m4fstack	78.5%	80.4%	71.8%
kyber512 lowsize	clean	55.7%	53.1%	39.1%
kyber512 lowsize	m4fspeed	78.5%	80.8%	72.1%
kyber512 lowsize	m4fstack	78.5%	80.5%	71.9%
kyber512 bi32speed	clean	49.1%	46.5%	33.0%
kyber512 bi32speed	m4fspeed	73.7%	76.3%	66.5%
kyber512 bi32speed	m4fstack	73.8%	76.0%	66.3%
kyber512 bi32small	clean	53.3%	50.6%	36.8%
kyber512 bi32small	m4fspeed	76.8%	79.1%	70.1%
kyber512 bi32small	m4fstack	76.9%	78.9%	69.9%

10.4 The complete results MD file size evaluation section.

Size Evaluation

Scheme	Implementation	.text [bytes]	.data [bytes]	.bss [bytes]	Total [bytes]
kyber512 reference	clean	4,772	0	0	4,772
kyber512 reference	m4fspeed	15,320	0	0	15,320
kyber512 reference	m4fstack	12,816	0	0	12,816
kyber512 speed	clean	4,812	0	0	4,812
kyber512 speed	m4fspeed	15,444	0	0	15,444
kyber512 speed	m4fstack	12,880	0	0	12,880
kyber512 small	clean	4,812	0	0	4,812
kyber512 small	m4fspeed	15,444	0	0	15,444
kyber512 small	m4fstack	12,880	0	0	12,880
kyber512 lowsize	clean	4,812	0	0	4,812
kyber512 lowsize	m4fspeed	15,444	0	0	15,444
kyber512 lowsize	m4fstack	12,880	0	0	12,880
kyber512 bi32speed	clean	4,812	0	0	4,812
kyber512 bi32speed	m4fspeed	15,444	0	0	15,444
kyber512 bi32speed	m4fstack	12,880	0	0	12,880
kyber512 bi32small	clean	4,812	0	0	4,812
kyber512 bi32small	m4fspeed	15,444	0	0	15,444
kyber512 bi32small	m4fstack	12,880	0	0	12,880