Integration of ASCON's Hashing Scheme with CRYSTALS-Kyber

By

Sahar Shehzadi
(Registration No: 00000400914)

Department of Information Security

Military College of Signals

National University of Sciences and Technology (NUST)

Rawalpindi, Pakistan

(2024)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Mr/MS **Sahar Shehzadi**, Registration No. 00000400914, of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.

Signature: [illegible]

Name of Supervisor: Dr. Abdul Ghafoor, PhD

Date: 11/10/24

Signature (HoD): [illegible]

HoD
Information Security
Military College of Sigs

Date: 11/10/24

Signature (Dean/Principal): [illegible]

Brig
Dean, MCS (NUST)
(Asif Masood, PhD)

Date: 18/10/24

Form TH-4

NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY

MASTER THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by **Sahar Shehzadi MSIS-21 Course** Regn No **00000400914** Titled: **"Integration of ASCON's Hashing Scheme with CRYSTALS-Kyber"** be accepted in partial fulfillment of the requirements for the award of **MS Information Security** degree.

Examination Committee Members

1. Name : **Dr. Shahzaib Tahir** Signature: [illegible]

2. Name: **Dr. Faiz Ul Islam** Signature: [illegible]

Co-Supervisor's Name: **Dr. Sadiqa Arshad** Signature: [illegible]

Supervisor's Name: **Dr. Abdul Ghafoor** Signature: [illegible]

Date: ________

[illegible]
HoD
Information Security
Military College of Sigs
Head of Department

11/10/24
Date

COUNTERSIGNED

Date: 18/10/24

[illegible]
Brig
Dean, MCS (NUST)
(Asif Masood, Phd)
Dean

CERTIFICATE OF APPROVAL

This is to certify that the research work presented in this thesis, entitled "**Integration of ASCON's Hashing Scheme with CRYSTALS-Kyber** " was conducted by **Sahar Shehzadi** under the supervision of **Dr Abdul Ghafoor** No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the **Military College of Signals, National University of Science & Technology Information Security Department** in partial fulfillment of the requirements for the degree of Master of Science in Field of **Information Security** Department of information security National University of Sciences and Technology, Islamabad.

Student Name: Sahar Shehzadi Signature: [illegible]

Examination Committee:

a) External Examiner 1: Name Dr Shahzaib Tahir (MCS) Signature: [illegible]

b) External Examiner 2: Name Dr. Faiz Ul Islam (MCS) Signature [illegible]

Name of Supervisor: Dr. Abdul Ghafoor Signature: [illegible]

Name of Dean/HOD. Dr Muhammad Faisal Amjad Signature: [illegible]

HoD
Information Security
Military College of Sigs

AUTHOR'S DECLARATION

I **Sahar Shehzadi** hereby state that my MS thesis titled **Integration of ASCON's Hashing Scheme with CRYSTALS-Kyber** is my own work and has not been submitted previously by me for taking any degree from National University of Sciences and Technology, Islamabad or anywhere else in the country/ world. At any time if my statement is found to be incorrect even after I graduate, the university has the right to withdraw my MS degree.

Student Signature: [illegible]

Name: Sahar Shehzadi

Date: 27/9/24

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled **A Integration of ASCON's Hashing Scheme with CRYSTALS-Kyber** is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and National University of Sciences and Technology (NUST), Islamabad towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the University reserves the rights to withdraw/revoke my MS degree and that HEC and NUST, Islamabad has the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized thesis.

Student Signature: [illegible]

Name: Sahar Shehzadi

Date: 27/9/24

DEDICATION

This thesis is dedicated to the pursuit of knowledge and to the resilient spirit of the Palestinian people, who inspire hope and strength in the face of adversity.

ACKNOWLEDGEMENTS

I am grateful to Allah Almighty for giving me the strength to successfully complete my thesis. I would like to express my sincere gratitude to all those who have supported me throughout this academic journey. First and foremost, I would like to extend my deepest appreciation to my supervisor, Dr. Abdul Ghafoor, for his invaluable guidance, insightful feedback, and constant encouragement. He guided me through every tough situation, offering support that was crucial in helping me navigate challenges. His unwavering dedication has been key in the completion of this thesis.

I would also like to thank Dr. Ayesha Khalid, whose mentorship and advice significantly shaped my approach to this work. Her sweet and positive nature had a wonderful effect on my progress, giving me the boost I needed during difficult times.

I am equally grateful to my co-supervisor, Dr. Sadiqa Arshad, whose motivation and encouragement were pivotal throughout this work. Her belief in my abilities provided me with the confidence to push forward.

Lastly, I am grateful to my family and friends for their continuous support, patience, and encouragement throughout this journey. Without all of you, this achievement would not have been possible.

Contents

LIST OF TABLES XIII

LIST OF FIGURES XIV

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS XV

ABSTRACT XVII

1 Introduction 1

1.1 Background 1

1.1.1 The NIST PQC Standardization Project 2

1.1.2 NIST's Lightweight Cryptography Competition 2

1.1.3 CRYSTALS-Kyber_Key Encapsulation Mechanism 3

1.1.4 The Cryptograghic Hash Functions 4

1.2 Motivation 5

1.3 Problem Statement 6

1.4 Research Objective 6

1.5 Thesis Structure 6

2 Literature Review 8

3 Preliminaries 14

3.1 Overview 14

3.1.1 Key Encapsulation Mechanism (KEM) 14

3.2 Foundational Elements 15

3.2.1 Lattice-based Scheme 15

3.2.2 The Learning With Errors (LWE) Problem 17

3.2.3 The Ring-Learning with Errors (R-LWE) Problem 18

3.2.4 The Module-Learning with Error (M-LWE) 19
3.2.5 CRYSTALS-Kyber 19
3.3 Design Structure of Kyber 20
3.3.1 Keygeneration 20
3.3.2 Encapsulation 20
3.3.3 Decapsulation 21
3.3.4 Kyber.v3 (NIST PQC Round 3) 21
3.3.4.1 KYBER.CPAPKE 21
3.3.4.2 KYBER.CCAKEM 22
3.3.5 Kyber.v4 (NIST PQC Final Round) 24
3.4 Keccak/SHA3 25
3.4.1 Keccak and Sponge Construction 25
3.4.2 Internal Structure and Permutation Function 26
3.4.3 Padding and Domain Separation 27
3.4.4 Security Considerations 27
3.4.5 Variants of Keccak/SHA3 27
3.4.6 Advantages and Applications 28
3.5 Ascon 28
3.5.1 Ascon Algorithm 29
3.5.1.1 Ascon Internal Architecture 29
3.5.1.2 Initialization 30
3.5.1.3 Associated data 30
3.5.1.4 Plaintext/Ciphertext Processing 31
3.5.1.5 Finalization 31
3.5.1.6 Permutation 31
3.5.2 Ascon's Hashing Modes 32
3.6 Summary 32

4 Proposed Methodology **34**
4.1 Overview 34
4.2 Preparation and Planning 36
4.2.1 Development Environment 36
4.2.2 Installation of Necessary Tools 36
4.2.3 Understanding the Kyber Codebase 37
4.2.3.1 AVX2 Implementation 37
4.2.3.2 Reference Implementation 37

4.2.3.3 Key Differences 38
4.2.3.4 Benchmarking Program 38
4.2.4 Understanding the Ascon Codebase 39
4.2.4.1 Ascon Hash Functions 39
4.2.4.2 Core Components of the Ascon Codebase 40
4.3 Identify Keccak/SHA3 Usage in Kyber 41
4.3.1 Role of SHA3-256/512 and SHAKE128/256 in Kyber 41
4.3.2 Locating Keccak/SHA3 Instances in the Code 43
4.4 Ascon Integration Plan 44
4.4.1 Compatibility Consideration 44
4.4.2 Choosing Appropriate Security Level 45
4.4.3 Determining the Equivalent Ascon Hash Function 46
4.5 Replacing Keccak/SHA3 with Ascon 47
4.5.1 Replacing the Functions 47
4.6 Testing and Benchmarking 48
4.6.1 Unit Testing 49
4.6.2 Integration Testing 50
4.6.3 Benchmarking 50
4.7 Summary 51

5 Results and Analysis **52**
5.1 Introduction 52
5.1.1 Test Environment 52
5.2 Benchmarking Paramemters 53
5.3 Hashing Function Efficiency 53
5.3.1 Ascon vs. Keccak: Speed Comparison 53
5.4 Performance Evaluation 54
5.4.1 Results 54
5.4.1.1 Key Generation 54
5.4.1.2 Encapsulation 55
5.4.1.3 Decapsulation 56
5.4.2 Overall ML-KEM Performance 57
5.4.3 Overall ML-KEM with Ascon Performance 59
5.4.4 Performance Comparison Between ML-KEM and ML-KEM with Ascon 60
5.5 Analysis of Matrix Generation Cycles in ML-KEM Based on Ascon . . 60

5.5.1 Hash Rate Mismatch with Ascon-Hash 60
5.5.2 Impact of Squeeze Length on Efficiency 62
5.6 Comparison Analysis 62
5.7 Security Analysis of the Ascon-based Kyber Implementation 64
5.7.1 128-bit Security Against Collision Attacks 64
5.7.2 Quantum Resistance and the LWE Problem 65
5.7.3 Impact on Lattice-based Design 65

6 CONCLUSION AND FUTURE RECOMMENDATION 67
6.1 Conclusion 67
6.2 Future Recommendation 68

List of Tables

3.1 Parameters of Kyber 19
3.2 Variants of Keccak/SHA3 and Their Parameters 28
3.3 Design Specifications for Ascon-128 and Ascon-128a. 29
3.4 Design Parameter Length for Ascon-Hash and Ascon-XOF. 33

4.1 Kyber's Security Levels 45
4.2 Comparison between Ascon's Hash and Keccak/SHA3 Security 46
4.3 Ascon Hash Functions Replacing Keccak in Kyber 47

5.1 Ascon vs Keccak - Speed Comparison 53
5.2 Performance Comparison of Key Functions in ML-KEM and ML-KEM with Ascon, Measured in CPU Cycles (cycles/ticks) 54

List of Figures

3.1 A KEM [1] 15
3.2 2-dimensional lattice basis [1] 16
3.3 Good and bad basis [1] 17
3.4 The flowchart of KYBER.CCAKEM encryption [2] 23
3.5 The flowchart of KYBER.CCAKEM decryption [2] 24
3.6 The sponge construction [3] 26
3.7 Ascon AE encryption's internal architecture. (The decryption procedure is the same; however, *PT* is obtained by processing *CT*). 30
3.8 Ascon hashing's sponge mode [4] 32

4.1 Flow of proposed methodology 35
4.2 Generating random seeds [1] 42
4.3 Generating enteries of matrix [1] 42

5.1 Performance comparison for key_generation 55
5.2 Performance comparison for encapsulation 56
5.3 Performance comparison for decapsulation 57
5.4 Performance of ML-KEM 58
5.5 Performance of ML-KEM with Ascon 59
5.6 Performance comparison between ML-KEM and ML-KEM with Ascon 61

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

AEAD: Authenticated Encryption with Associated Data

AES: Advanced Encryption Standard

D-H: Diffie-Hellman

DES: Data Encryption Standard

ECDSA: Elliptic Curve Digital Signature Algorithm

FO: Fujisaki-Okamoto

ISE: Instruction Set Extensions

IoT: Internet of Things

KEM: Key Encapsulation Mechanism

LWE: Learning with Error

M-LWE: Module-Learning with Error

ML-KEM: Module-Lattice-Based Key-Encapsulation Mechanism Standard

NIST: National Institute of Technology

OQS: Open Quantum Safe

PKE: Public Key Encryption

PQC: Post-Quantum Cryptography

PRNG: Pseudorandom Number Generator

R-LWE: Ring-Learning with Error

RFID: Radio Frequency Identification

RISC: Reduced Instruction Set Computer

RSA: Rivest Shamir Adleman

SIS: Short Integer Solution

TLS: Transport Layer Security

XOF: Extendable Output Function

Abstract

As quantum computing continues to evolve, it poses significant risks to current cryptographic methods, making the transition to post-quantum cryptography essential. CRYSTALS-Kyber, officially standardized by NIST in August 2024 as ML-KEM, is a leading solution designed to resist quantum attacks. This thesis investigates an enhancement to ML-KEM by replacing its existing Keccak-based hashing function with Ascon, recently selected by NIST for its lightweight cryptographic properties known for its efficiency, particularly in environments with limited resources like embedded systems. The core objective of this research is to evaluate the performance impact of this change. Testing was carried out on a personal laptop, using Kyber's original test cases to measure CPU cycles consumed by key cryptographic operations both before and after replacing Keccak with Ascon. The results demonstrate that the integration of Ascon significantly improves computational efficiency while maintaining the cryptographic integrity and security of ML-KEM. Though this work does not aim to enhance ML-KEM's security—which is already ensured by its design—it offers a justification that the substitution of the hashing function does not negatively impact its cryptographic integrity. The key contribution of this research lies in making ML-KEM more suitable for resource-constrained environments, particularly embedded systems, by improving its efficiency and reducing computational overhead. By exploring the practical benefits of Ascon's integration into a post-quantum cryptographic standard, this thesis contributes to the optimization of secure, quantum-resistant lightweight algorithm for real-world applications, paving the way for its effective use in embedded systems and similar platforms.

Keywords: Post-Quantum Cryptography, Lightweight Cryptography, CRYSTALS-Kyber, ML-KEM, Ascon Hash Functions, Keccak/SHA3 Replacement.

Chapter 1

Introduction

1.1 Background

Currently the main cryptographic primitives being used in over digital world to strengthen our communication channels, digital currencies and authentication mechanisms are symmetric and asymmetric primitives. These are constructed to provide authentication, confidentiality, integrity and non-repudiation. Symmetric cryptography also named as secret-key cryptography is used for encryption and integrity purposes. Nowadays most widely used symmetric cryptographic algorithms are Advanced Encryption Standard (AES), Data Encryption Standard (DES). Similarly, asymmetric cryptography also known as public-key cryptography is used for authentication and key distribution. Elliptic Curve Digital Signature Algorithm (ECDSA), Rivest Shamir Adleman (RSA), Digital Signature Algorithm (DSA), Deffi-Hellman (D-H) and Elliptic Curve Diffie-Hellman (ECDH) are the commonly used asymmetric cryptographic primitives.

The security of communication has emerged as a critical issue in our increasingly digitalized environment. The swift progress of quantum computing technology poses an unparalleled danger to established cryptography systems, which have long been the foundation of safe and secure communication. These quantum computers would easily solve the mathematical problems that are currently being used in our cryptography by exploiting the quantum mechanical phenomenon as these devices are exponentially faster than conventional computer. If large-scale quantum computers are ever built, they will compromise the security of many commonly used public-key cryptosystems such as ECDSA, RSA, DSA, D-H, rely on the assumed difficulty of mathematical problem like integer factorization and discrete logarithm. Peter Shor created Shor's algorithm in 1994, a quantum computing method capable of solving integer factorization problems efficiently with enough qubits [5]. This breakthrough poses a significant threat to the security of the above mentioned cryptographic algorithms.

To address this challenge, the field of post-quantum cryptography (PQC) has emerged. PQC aims to develop cryptographic algorithms that remain secure even in the face of quantum attacks. These algorithms are based on mathematical problems believed to be resistant to quantum computing capabilities. Among these, lattice-based cryptography has gained particular attention due to its strong security properties and efficiency. A standout candidate in this field is CRYSTALS-Kyber, a lattice-based key encapsulation mechanism (KEM) that has become a frontrunner in the quest for quantum-resistant cryptography.

1.1.1 The NIST PQC Standardization Project

NIST, the U.S. National Institute of Standards and Technology, in 2016, initiated a competition to standardize PQC primitives that offer security not only for classical computers but also for quantum computers [6]. The NIST competition was structured in multiple rounds, with each round involving rigorous evaluation of the submitted algorithms based on criteria such as security, performance, and ease of implementation.Over the course of 2016 to 2022, three rounds of evaluations took place to assess the proposed PQC primitives.

In the summer of 2022, NIST revealed its intention to standardize two lattice-based cryptographic primitives: CRYSTALS-Kyber, which is a KEM, and CRYSTALS-Dilithium, which is a digital signature algorithm. By standardizing these lattice-based primitives, NIST aims to promote the adoption of post-quantum secure cryptographic algorithms in various applications and systems. The standardization process continued with ongoing evaluations of additional algorithms to ensure a comprehensive set of quantum-resistant tools.

In August 13, 2024, NIST has finalized the selection of three Federal Information Processing Standards (FIPS) for PQC. These are Module-Lattice-Based Key-Encapsulation Mechanism Standard(FIPS 203) or ML-KEM, which is derived from the Kyber submission [7], Module-Lattice-Based Digital Signature Standard (FIPS 204)[8], Stateless Hash-Based Digital Signature Standard (FIPS 205)[9].

1.1.2 NIST's Lightweight Cryptography Competition

NIST conducted two workshops (in July 2015 and October 2016) to solicit and engage with community to understand the need and application of lightweight algorithms. After that NIST announced the need of an open call for proposals to standardise algorithms for lightweight cryptography [10]. Each submission was meant either to implement an authenticated encryption with associated data (AEAD) functionality and optionally also implement the hashing functionality [10]. This competition ran for more than 5 years in multiple rounds.

In March 2019, NIST received 57 submissions to be considered for standardisation. The first round of the NIST lightweight cryptography standardisation process began with the announcement of 56 Round 1 in April 2019 and ended in August 2019, with 32 candidate algorithms advancing to the second round of the evaluation process. Some of the algorithms were eliminated from consideration early in the first evaluation phase in order to focus analysis on the more promising submissions, primarily based on the weaknesses brought forward by the third-party analysis. The second round of the NIST's lightweight cryptography standardisation concluded when 10 finalists were announced in March 2021. Out of these finalists, 6 had AEAD functionality while 4 additionally had hashing functionality as well. On February 7, 2023, NIST announced the decision to standardise the Ascon family for lightweight cryptography applications.The evaluation criteria and selection process, which was based on public feedback. and internal review of the finalists is described in a report [11].

There are several reasons for Ascon family selection given by NIST [11]. Ascon includes both AEAD and hash functions, as well as additional Extendable Output Functions (XOFs), allowing it to satisfy a wide range of application needs with low additional for additional functionalities. Ascon is the most mature of the finalists in terms of security. The Ascon family had already been presented and analyzed as part of the CAESAR competition [12]. Performance in constrained environments, such as dedicated hardware and embedded systems, was a significant factor in the decision, with Ascon performing very well in both hardware and software [11].

1.1.3 CRYSTALS-Kyber_Key Encapsulation Mechanism

Kyber, known as ML-KEM, is a lattice-based KEM that has quickly risen to prominence within the field of PQC. Built on the Learning With Errors (LWE) problem—a complex mathematical challenge that underlies many lattice-based protocols—Kyber is designed to be both efficient and secure. It comes with various parameter sets, letting users fine-tune the balance between security and performance to fit their needs.

Kyber's selection by NIST as a top choice for post-quantum encryption underscores its importance in strengthening our digital security against emerging quantum threats. It provides robust security guarantees and is efficient enough to be used in a wide range of applications. ML-KEMs, unlike general-purpose public-key encryption (PKE) scheme, are not intended for encrypting application data. Instead, they are specifically created to establish a shared secret between communication partners in cryptographic protocols such as Transport Layer Security (TLS), just like the Diffie-Hellman Key-Exchange method, which is currently one of the best available options. Even with Kyber's strong performance, there is always room for exploration and improvement. One area of interest is finding a replacement for the Keccak hash function that might deliver even better performance or security in certain situation. This thesis delves into one such alternative: the Ascon hash function.

1.1.4 The Cryptograghic Hash Functions

Cryptographic hash functions are the backbone of many cryptographic systems, playing a crucial role in ensuring data integrity, authentication, and secure key generation. A hash function takes an input (or "message") and produces a strings of fixed-sized, typically a digest that appears random. Even a minor change in the input should result in a vastly different output, making hash functions indispensable for maintaining the security of cryptographic protocols. The XOF is the extended form of cryptographic hashes that produces the output of variable length for a given message.

In the context of PQC, and particularly in lattice-based schemes like Kyber, hash functions are used extensively. They contribute to key generation, encapsulation and decapsulation processes, and ensure that the system is secure against both classical and quantum adversaries. The performance and security of the entire cryptographic scheme can depend heavily on the choice of hash function.

Keccak, the current hash function employed in Kyber, is known for its robustness and versatility. It is the foundation of the SHA3 family of hash functions, which were standardized following a NIST competition held between 2007 and 2012 [13]. Keccak's sponge construction allows it to generate outputs of variable length, making it highly adaptable for different cryptographic needs. However, despite its strengths, the cryptographic community continues to explore alternatives that might offer better performance or security, particularly in resource-constrained environments like IoT devices.

One such alternative that has recently gained traction is the Ascon [4] hash function. Ascon was developed with efficiency and simplicity in mind, specifically to excel in environments where computational power and memory are limited. It was introduced as part of the CAESAR competition (Competition for Authenticated Encryption: Security, Applicability, and Robustness), which took place from 2014 to 2019 [14]. The goal of this competition was to identify the most secure and efficient authenticated encryption schemes for a variety of applications. Ascon not only stood out during the competition but was also selected by NIST for standardization in February 2023, marking it as a new standard for lightweight cryptography.

Ascon's design is all about balance—combining simplicity and speed without compromising security. Like Keccak, it uses a sponge construction, but it's optimized to reduce the computational load, which leads to faster execution times and lower energy usage. These qualities make Ascon an attractive option for replacing Keccak in applications like Kyber, where you need both top-notch security and efficiency. With NIST's recent endorsement, Ascon is set to play a significant role in the future of cryptography, especially in areas where every bit of performance and security counts.

1.2 Motivation

The motivation for this study arises from the critical needs of securing the digital word in the quantum era. It first claims to provide a notable improvement in performance of Kyber with the replacement of Keccak with lightweight cryptography (LWC) scheme making it suitable for embedded systems.. The effectiveness of LWC systems is well known, as they provide faster cryptographic operations than conventional hashing algorithms. This improvement is essential to maximize Kyber's overall effectiveness. Furthermore, because LWC schemes use fewer system resources, they are especially well-suited for contexts with limited resources, including embedded systems and the Internet of Things (IoT). LWC schemes are lightweight, yet they offer enough security for a lot of applications, therefore Kyber is able to keep its strong security posture. Adopting a LWC scheme also puts Kyber in line with cutting-edge technologies, making it easier to integrate into new platforms and gadgets. Also, Kyber can ensure its durability and relevance in the cryptographic landscape by proactively adopting LWC schemes, which will help to future-proof it against changing security risks and technical breakthroughs. We get following benefits by merging the Ascon with Kyber:

- Reduce the computational overhead that will lead to performance improvement
- Adjust to specific requirements, suitable for low-power devices
- Increase resource efficiency, specifically in resource constrained devices
- Keep up sufficient security even after reducing the usage of system resources
- Beneficial for limited memory applications due to size decrease of code.

By this replacement Kyber would also be suitable for following applications:

- IoT devices and networks
- Embedded systems and microcontroller-based applications
- Mobile and wearable devices demanding lightweight cryptographic procedures
- Secure messaging and email encryption platforms
- Cloud computing and edge computing environments with resource constraints

1.3 Problem Statement

Kyber is developed through a two-step process: initially, it provides PKE that is IND-CPA secure by utilizing a Module-LWE and then with little transforms it into KEM that is an IND-CCA secure. This transformation is known as Fujisaki-Okamoto (FO) transformation. One can transit from CPA-secure PKE to CCA-secure KEM feasibly through FO transform [6]. In the CCA transform, public key (pk) and ciphertext are hashed. Based on the Keccak permutation, the hashes utilized in Kyber are SHAKE-128, SHA3-256, SHAKE-256, and SHA3-512. Although they increase robustness, they are not required for the security reduction. Additionally, hashing pk strengthens defenses against a specific category of multi-target attacks that aim to exploit protocol flaws [15]. But on contrary they are speed-critical component.

SHA3 has the standing of not being a fastest hash function in software [16]. Also, common embedded processors are simple, low-cost and low power devices, having a simple instruction set. The ARM family of embedded processors is the most widely used RISC (reduced instruction set computer) architecture processors in use today, with over 200 billion ARM chips produced till 2021 [17], used in smartphones, laptops and other embedded systems. Considering that this hashing significantly affects the scheme's overall performance and make it not suitable for embedded devices [15].

1.4 Research Objective

Our focus is to replace the SHA3/Keccak of Kyber by the lightweight cryptography alternate to make it suitable for embedded devices. A lightweight alternative Ascon [4] from the NIST Lightweight cryptography competition called the NIST LWC [14] is better suited for IoT applications. This work will thoroughly analyse the effects of this replacement in terms of code/memory/stack sizes, throughput efficiency, etc. The goal is to create a fully functional lightweight Kyber system that can generate keys, encrypt and decode data using a lightweight hashing method, and analyse the results. The main objectives of thesis are:

- Exploration and understanding of Kyber and Ascon.
- Integration of the Ascon with the Kyber and comparing the merger of these with existing Kyber.

1.5 Thesis Structure

- Chapter 1: This opening section highlight the brief background of cryptographic primitives, the quantum threat imposed on it, the brief introduction of PQC and the NIST PQC Standardization project, Categories of PQC algorithms, moves towards the motivation for this study, and then lists the research objectives after highlighting the problem statement.

- Chapter 2: It gives the literature review for this study.
- Chapter 3: This chapter presents the design paradigm of Kyber, encompasses its foundational elements, its strength in case of security, detailed exploration of LWC scheme, the design paradigm of Ascon and its benchmarks.
- Chapter 4: This chapter offers the proposed methodology for the replacement of Ascon with Kyber, pinpoint the Keccak functions usage in Kyber, shows the comparison of Ascon with Keccak and check their compatibly in context of their parameters and security.
- Chapter 5: It shows the performance results, comparison of existing Kyber with the merger one and analysis and security analysis.
- Chapter 6: It is about conclusion and some future work.

Chapter 2

Literature Review

Different families of PQC have emerged, each based on distinct underlying security principles. These families include multivariate-based, symmetric cipher/hash-based, code-based, lattice-based and isogeny-based schemes. Lattice-based encryption is one of the most studied algorithms for public key cryptography in the context of PQC. A lattice is a structure made up of an infinite network of points, where each point is represented by a vector. The set of vectors that define the lattice points is known as a basis. In lattice-based encryption, messages are represented as vectors, and the public key is a matrix that is used to transform these messages into ciphertexts.

There are several types of lattice-based schemes, including: Encryption schemes such as LWE [18] , Ring-Learning with Error (Ring-LWE) [19], and NTRU [20], Signature schemes such as Falcon, Rainbow, and Dilithium, Key exchange protocols such as Kyber and Frodo. Lattice-based schemes have been extensively researched, and many of them have been proposed and implemented in hardware [21].

Kyber is built upon a lattice-based cryptosystem, which is a prominent approach in the field of PQC. The security of lattice-based schemes like Kyber generally relies on well-established, complex mathematical problems such as LWE [18], Short Integer Solution (SIS) [22], and NTRU [20, 23]. These problems are considered difficult to solve, even with the advent of quantum computing, making them a robust foundation for cryptographic security .

Lattice-based cryptography utilizes these problems to create secure encryption schemes. LWE, for example, involves solving equations with small errors added to linear equations, while SIS focuses on finding short solutions to linear equations with integer coefficients. NTRU, another lattice-based scheme, relies on polynomial rings and their hardness properties to ensure security [24].

In addition to these core problems, researchers have proposed variants and extensions of LWE, SIS, and NTRU. These variants aim to offer more efficient cryptographic constructions while maintaining strong security guarantees. Such advancements often involve algebraic techniques that enhance both performance and security proofs [25].

Among the early post-quantum cryptographic algorithms selected for standardization by the NIST, a significant number are based on lattice-based schemes. Specifically, three of the initial four algorithms chosen for standardization fall within this category, underscoring the importance and promise of lattice-based cryptography in securing future communications against quantum threats [26].

There are two main practical approaches to lattice-based encryption schemes: one is based on the LWE problem or its variations, while the other relies on NTRU.

The LWE assumption is grounded in the difficulty of distinguishing between two distributions: the distribution of the pair $A, A.s + e$ and a uniform distribution. In this context, A represents a matrix whose entries are uniformly random, s is a vector of the same dimension, and e is a vector of small random coefficients drawn from a specific error distribution. The challenge is to differentiate between this noisy version of the vector and a truly uniform distribution. The hardness of this problem remains even when$s$ is chosen from a distribution similar to the error vector e [27].

In addition to LWE, the Ring-LWE problem provides a variant that is specifically adapted for polynomial rings. The Ring-LWE assumption is based on the difficulty of solving a similar problem but within the structure of a ring, which can lead to more efficient implementations in practice. Research has demonstrated that solving instances of Ring-LWE is as hard as distinguishing between the noisy and uniform distributions, further solidifying its role in lattice-based cryptography [19].

Both LWE and Ring-LWE are central to modern lattice-based encryption schemes, offering robust security foundations against potential quantum attacks while enabling practical cryptographic implementations.

In Ring-LWE schemes [19], operations involved expressions like $A.s + e$, where specific ring contained polynomials as all variables. The major reason Ring-LWE encryption is more efficient than LWE encryption is because utilizing a ring Rq of bigger degree n allows one to transmit more bits. The amount of bits that may be communicated is related to the size of the ring. On the other hand, a smaller k suggests a more algebraic structure make the system more attackable and introduces the module-learning with error (M-LWE) scheme.

Kyber, introduced by Bos et al. in 2017, based on M-LWE is a KEM [15] . Its design draws from the Ring-LWE LPR (introduced by Lyubashevsky, Peikert, and Regev for Ring-LWE at Eurocrypt 2010 [19]) encryption scheme. When the value of parameter k is set to 1 and is defined, the system becomes a Ring-LWE. If the ring Rq is defined as a polynomial ring with dimension higher than 1 and k is greater than 1, then the system relies on the difficulty of the M-LWE. However, in the module-LWE-based Kyber, the key difference lies in A as a matrix (often with a small dimension $k = 3$ having a fixed-sized polynomial ring as variables, while s and e are vectors over the same polynomial ring. These resulting schemes are as efficient as ones that are based on Ring-LWE, but have additional flexibility and security advantages at almost no cost but only part where this scheme is less effective than Ring-LWE is when dealing with large random $k * k$ matrix A constructed by using an XOF from some seed leads to slight increase in the running time that is used in matrix expansion [15].

Structured variants of the LWE problem, such as those based on polynomial rings, offer notable benefits in terms of efficiency. These variants can speed up computations and reduce the sizes of keys and ciphertexts, making them more practical for real-world applications. For example, R-LWE and M-LWE use algebraic structures to optimize performance and resource usage. However, this added structure brings its own set of challenges. The additional patterns introduced in these variants can potentially create new security vulnerabilities that are not present in the more general, unstructured LWE problem. Finding the right balance between efficiency and security becomes more complex, as adjustments in one area can significantly impact the other. On the other hand, standard LWE does not rely on any specific algebraic structures. This lack of structure allows for easier scaling and adaptation but often results in lower efficiency compared to its more structured counterparts. While standard LWE offers robust security, it tends to be less efficient in terms of computational and storage requirements. M-LWE tries to strike a balance between these two approaches. By incorporating some structure to improve efficiency while still retaining a level of generality, M-LWE provides a practical compromise. It combines the efficiency gains of structured variants with the strong security guarantees of the traditional LWE problem, making it a versatile choice for many applications [15].

Kyber utilizes a fixed polynomial ring, specifically $\mathbb{Z}_q[X]/(X^{256} + 1)$, as its underlying structure. This choice of ring helps to standardize operations and facilitates efficient computation. To accommodate different levels of security, Kyber adjusts the module rank, with three distinct versions available: Kyber512, Kyber768, and Kyber1024, corresponding to module ranks of $k = 2, 3$, and 4 respectively. One of the key features of Kyber is its use of the Number-Theoretic Transform (NTT) for polynomial multiplication. The NTT allows for faster polynomial arithmetic by transforming polynomial multiplication into simpler element-wise multiplications in the transformed domain. After performing multiplication in the NTT domain, Kyber rounds the resulting ciphertext to further reduce its size and maintain efficiency. In terms of noise distribution, Kyber opts for a simpler and more efficient centered binomial noise distribution rather than the Gaussian noise often used in other schemes. The centered binomial distribution is easier to handle and provides a good balance between performance and security. There are several ways to balance the trade-offs between secret-key size and decapsulation speed. If minimizing the secret-key size is a priority, you can opt not to store H(pk) and avoid including the public key as part of the secret key. Instead, you would recompute the public key during the decapsulation process. Additionally, by not keeping the secret key in the NTT domain, you can compress each coefficient to just 5 bits, which reduces the total size of the three polynomials to only 320 bytes[28]. Overall, Kyber's approach to polynomial arithmetic and noise distribution is designed to optimize both computational efficiency and security, making it a robust choice for post-quantum cryptographic applications [15].

Moreover, since key generation relies on two 32-byte seeds to produce all necessary randomness, you can further reduce the storage requirement by saving only these seeds. During decapsulation, you would then regenerate the keys from these seeds.

Many organizations have begun integrating Kyber into their systems as part of a hybrid cryptographic approach [29]. The Internet Engineering Task Force (IETF) has provided drafts related to PQC, though no finalized implementation references have yet been published. In response to these drafts, several organizations are already adopting these early-stage recommendations to bolster their security and mitigate the "Harvest Now, Decrypt Later" problem [30]. For example, Apple has announced that it will employ its own post-quantum cryptographic protocol for iMessage, named "PQ3." This protocol utilizes a hybrid approach that combines ML-KEM with Kyber and ECDH for key exchange mechanisms [31]. This strategy reflects a growing trend among companies to integrate post-quantum cryptographic methods into their security infrastructure, preparing for a future where quantum computing poses a real threat to traditional cryptographic systems.

Many major technology players and organizations are adopting Kyber in their systems, either currently or as part of future plans, particularly within a hybrid cryptographic framework. For instance, Chrome and Microsoft Edge, both of which are based on the Chromium engine, are exploring or have already started integrating Kyber as part of their hybrid approach [30]. Similarly, Cloudflare is incorporating Kyber into its security protocols [32], and Mozilla is working towards including it in the Firefox Nightly builds [33].

The Signal protocol, which is widely used for secure messaging, is also planning to integrate Kyber in its encryption processes [34, 35, 36]. Amazon has announced that its s2n-tls library and s2n-quic protocol will support Kyber [37, 38], and AWS Key Management Service (KMS) along with other AWS services are expected to adopt Kyber as well [39, 40]. Cisco and Proton Mail are also on board with supporting or planning to use Kyber in their cryptographic solutions [41, 42].

In addition to these implementations, several libraries and toolkits are embracing PQC and hybrid key exchange methods, including Kyber. The Open Quantum Safe (OQS) project is a notable example, offering next-generation hybrid KEM solutions [43]. BoringSSL, a popular SSL library, is incorporating PQC techniques [44]. In the C++ ecosystem, the Botan 3.2.0 cryptographic library supports PQC algorithms [45], while WolfSSL includes them as well [46]. The Bouncy Castle Crypto package for Java also provides support for these modern cryptographic approaches [47].

Moreover, various language-specific libraries are emerging to facilitate the use of PQC algorithms, such as KyberLib, a dedicated library for Kyber [48], and the Python-based library "pqcrypto" [49]. Additionally, there are implementations in Rust and other programming languages, reflecting the growing interest and adoption of post-quantum cryptographic methods across different platforms and environments.

Chrome , Microsoft Edge being Chrome based [30], Cloudflare [32], Mozilla in Firefox Nightly [33],signal protocol [34, 50, 51], amazon to their s2n-tls [37], s2n-quic [38], AWS KMS [39] and also in AWS family [51], Cisco [41], Proton Mail [42] all are now supporting or being plan to used Kyber as a hybrid approach. In addition, some common libraries or toolkits are also supporting PQC include next generation hybrid KEM such as OQS (Open Quantum Safe) project [43], BoringSSL [44], a C++ Cryptographic library the Botan 3.2.0 [45], the Bouncy Castle Crypto [47] package for Java and different implementations for different languages be present (like KyberLib [48], Python based library "pqcrypto" [49]).

Kyber is developed through a two-step process: initially, it provides PKE that is IND-CPA secure by utilizing a M-LWE and then with little transforms it into KEM that is an IND-CCA secure. This transformation is known as FO transformation. One can transit from CPA-secure PKE to CCA-secure KEM feasibly through FO transform [28]. In the CCA transform, public key (pk) and ciphertext are hashed. Based on the Keccak permutation, the hashes utilized in Kyber are SHAKE-128, SHA3-256, SHAKE-256, and SHA3-512. Although they increase robustness, they are not required for the security reduction.

Kyber KEM [7], based on Kyber round 3 [15], standardized by NIST as a final PQC algo-rithm, is now named ML-KEM or FIPS 203. This ML-KEM specified in this standard adheres to the aforementioned structure. Specifically, this standard first explains the K-PKE public-key encryption method (named as KYBER.CPAPKE in Kyber.v3), and then it employs the K-PKE algorithms as subroutines to describe the ML-KEM algorithms. The K-PKE scheme should not be utilized as a stand-alone scheme since it is insufficiently secure. This standard has only some differences from the Kyber round 3 are the setting of fixed 256 bits length of shared secret key, a modified version of the FO transform, see, [52], [53], removal of the hashing step of the initial randomness m in the Kyber encryption method, before its being utilization (as this standard mandates the usage of NIST-approved randomness generating), include the explicit input validation procedures [7].

Even prior to Kyber's final standardization, it had already been widely adopted by major companies such as Google, Apple, Cloudflare, and Mozilla, as well as integrated into various cryptographic libraries. Recently, Google has already implemented the ML-KEM in Google's cryptography library, BoringSSL and TLS from Kyber768+X25519, to 0x11EC for ML-KEM768+X25519, further solidifying its position in the industry [54].

In the design of Kyber, the choice was made to instantiate all hash functions using those derived from Keccak, as standardized in FIPS 202 [3]. Specifically, SHAKE-128 is used to generate the matrix A, SHAKE-256 is employed for generating noise polynomials, while SHA3-256 and SHA3-512 are used to instantiate functions H and G, respectively. This approach ensures that all symmetric primitives in Kyber are based on the same foundational primitive, the Keccak-f1600 permutation. The only deviation is that for key generation, different implementations may select the PRNG (Pseudorandom Number Generator) that best balances performance and security on their respective platforms. Although alternative symmetric primitives could offer improved performance on many platforms—such as using SHA-256 for all hashes (with output extension via MGF1 (Mask Generation Function) and AES in counter mode for seed expansion, which would be faster on hardware supporting AES and SHA-256—there are trade-offs. Additionally, hashing pk strengthens defenses against a specific category of multi-target attacks that aim to exploit protocol flaws [15]. But on contrary they are speed-critical component.

SHA3 has the standing of not being a fastest hash function in software [16]. Also, common embedded processors are simple, low-cost and low power devices, having a simple instruction set. The ARM family of embedded processors is the most widely used RISC architecture processors in use today, with over 200 billion ARM chips produced till 2021 [17] , used in smartphones, laptops and other embedded systems. Considering that this hashing significantly affects the scheme's overall performance and make it not suitable for embedded devices [15].

The need for lightweight cryptography stems from its use in the IoT. The IoT today plays a pivotal role in connecting and exchanging information. However, the IoT revolution is a double-edged sword, driving the promise of digital transformation on one hand and opening a plethora of potential security and privacy vulnerabilities on the other. In 2020, the literature statistics of the SCOPUS database, for a 10th straight year, reported the highest number of Iot attacks and threat related articles [55]. Since the current widely deployed cryptographic algorithms can be computationally intensive, it can be a challenge to fit them on embedded IoT (edge) devices, due to the stricter resource requirements (for example tight area and energy budgets) for these use cases. These systems run on extremely resource constraint platforms, for example mobile tokens without battery or RFID (Radio Frequency Identification) or medical implants that do not allow change of batteries. When deployed in extremely high volume, these IoT devices are very cost-sensitive as we are talking about per-volume pricing model. These constraints led to active research into a new sub-field of cryptography, so called the lightweight cryptography, that aims to provide solutions tailored for resource-constrained devices.

This work focuses to replace the SHA3/Keccak of Kyber by the lightweight cryptography alternate to make it suitable for embedded devices. A lightweight alternative Ascon [4] from the NIST Lightweight cryptography competition called the NIST LWC [14] is better suited for IoT applications. This work thoroughly analyse the effects of this replacement in terms of its efficiency. The goal is to create a fully functional lightweight Kyber system that can generate keys, encrypt and decode data using a lightweight hashing method, and analyse the results.

Chapter 3

Preliminaries

3.1 Overview

PQC has emerged as a critical area of research as the advent of quantum computing threatens the security of classical cryptographic algorithms. In response to these challenges, NIST initiated a competition to identify and standardize quantum-resistant cryptographic schemes. Kyber, a lattice-based KEM, gained prominence by advancing through all stages of the competition. It was selected in Round 3 of the NIST PQC competition, eventually being standardized in 2024 as ML-KEM (FIPS-203). This marked a significant milestone in the development of secure encryption algorithms capable of withstanding quantum threats.

A core component of Kyber's structure is its use of Keccak/SHA3 as the hash function. Keccak's role in ML-KEM is essential for various cryptographic functions, such as key generation and encapsulation. However, as cryptographic schemes continue to evolve, lightweight cryptography has gained traction for applications in resource-constrained environments. One such lightweight cryptographic solution is Ascon, a hashing scheme known for its efficiency and minimal computational overhead, making it an attractive alternative to Keccak in specific contexts.

This chapter delves into the preliminary concepts surrounding Kyber, the importance of its selection in the NIST PQC competition, and its transition to ML-KEM. It also highlights the hash functions used within these algorithms (Keccak and Ascon) and sets the stage for a performance-based analysis of their integration into PQC frameworks.

3.1.1 Key Encapsulation Mechanism (KEM)

A KEM uses asymmetric algorithm to send symmetric keys between client or server or between two parties who want to exchange their data in secure channel unlike

Diffie-Hellman KEM, in which the symmetric key is directly generated by some mutual computations. In asymmetric scheme, party A or server generates the public and private keys (pk, sk) and sends public key pk to party B or client. Client encapsulates the symmetric key ss _a shared key that both parties are going to use to exchange information_ by using that public key sk and send the cipher-text c to server. Upon receiving the cipher-text c, server decapsulates the cipher-text c by using its private key sk and retrieves the symmetric key (ss). This is how KEM works and ensures a secure and authenticated exchange. Clients and server then use this shared symmetric key for communication or data exchange. Kyber works exactly like this, as illustrated in figure 3.1.

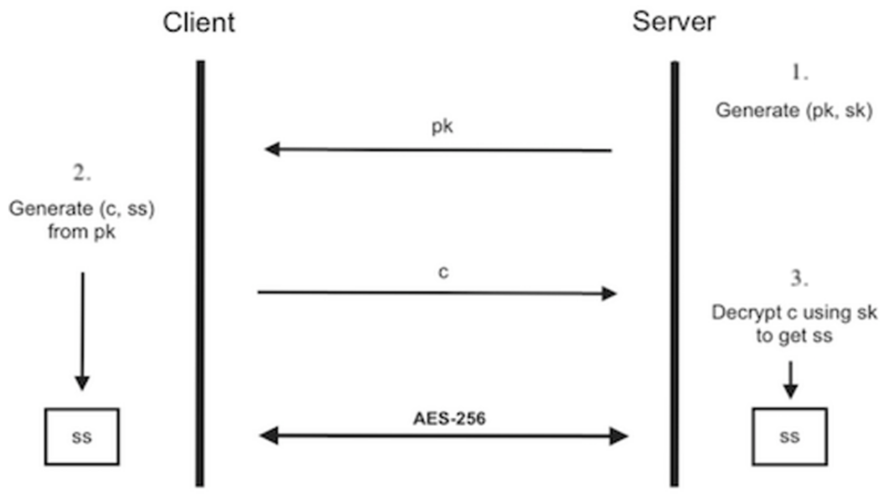

Figure 3.1: A KEM [1]

Kyber generates keys using M-LWE problem. To get grip on M-LWE, there is need to understand its foundational elements that is LWE problem, which further extends into R-LWE problem that then gives the M-LWE problem. Following is the complete elaboration of these problems to learn the hardness they provide in Kyber.

3.2 Foundational Elements

3.2.1 Lattice-based Scheme

Lattice-based cryptography is a promising and widely favored candidate for PQC schemes. In general lattices are regular spaced grid of points. If we define lattice in mathematics, it is a discrete subgroup formed by adding together elements in some inner product space with n-dimensions or it is a set of all integers linearly combination of basis vectors $\mathbf{b_1}, \mathbf{b_2}., ..., \mathbf{b_n} \in \mathbb{R}^n$, as given in 3.1

$$\Lambda = \left\{ \sum_{i=1}^{n} z_i \mathbf{b}_i \mid z_i \in \mathbb{Z}, \mathbf{b}_i \in \mathbb{R}^n \right\} \tag{3.1}$$

Where:

- Λ is the lattice.
- $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ are linearly independent vectors in $\mathbb{R}^n$ (called the basis vectors).
- $z_1, z_2, \ldots, z_n$ are integers from $\mathbb{Z}$.
- n is the dimension of the lattice.

In an finite or infinite space, if there is 2-dimensional lattice, as shown in figure 3.2, then there would only need of two basis vectors to generate a complete lattice.

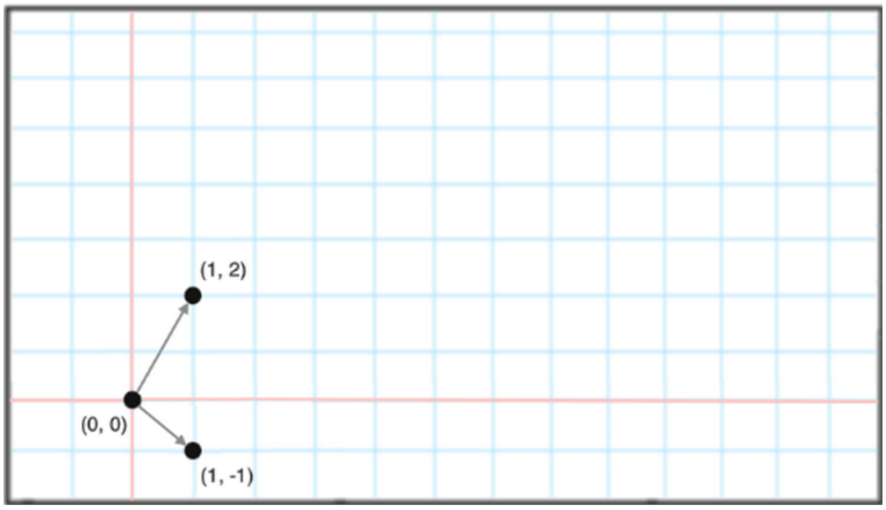

Figure 3.2: 2-dimensional lattice basis [1]

Mainly, we look two types of basis, good basis and bad basis. A good basis is the one where vectors are almost perpendicular to each other (or orthogonal) or if the vectors are short in short distance and bad basis is if the vectors are at long distance or have almost zero angle (too close to each other) as shown in figure 3.3.

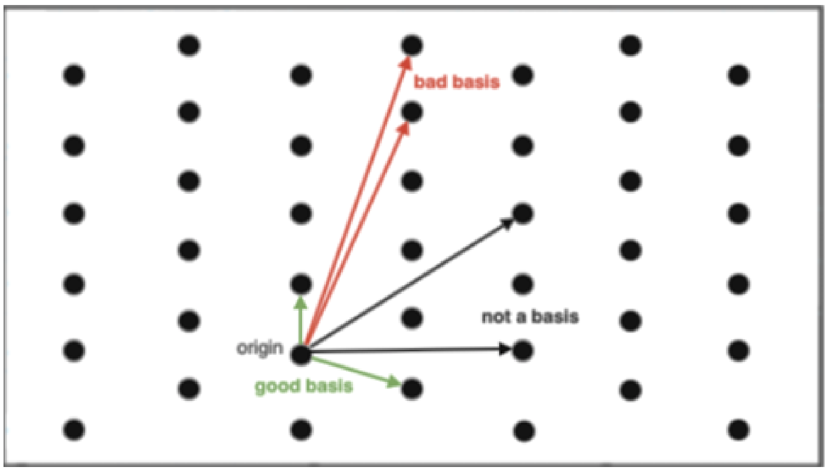

Figure 3.3: Good and bad basis [1]

Despite their simple mathematical description, lattices are associated with many challenging problems of significant interest in both mathematics and theoretical computer science. The most well-known and fundamental among these are the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). SVP is finding the closest point to the origin in the lattice and CVP is finding a point closest to a specific point. To implement these problems for cryptographic function, a lot of work has been done, in which SIS problem, LWE problem and NTRU problems are in focused. As Kyber is based on LWE problem , so next section are the basic elaboration of this problem [56].

3.2.2 The Learning With Errors (LWE) Problem

Suppose there are some systems of linear equations

$$\begin{aligned} 2x^3 + 3x^2 + 4x &= 7 \\ y^3 + 4y^2 + 2y &= 4 \\ 3z^3 + 6z^2 + 5z &= 1 \end{aligned}$$

If A are the coefficients and b are the constants and s are the variable of linear equations then this system can be represented in matrix form as:

$$A * s = b$$

$$\begin{pmatrix} 2 & 3 & 4 \\ 1 & 4 & 2 \\ 3 & 6 & 5 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 1 \end{pmatrix}$$

Imagine A and b as the public keys and s as the secret key vector which is the solution of these linear equations. In that case, these linear equations can easily solve and compute s by using linear algebra methods. Some commonly used method to solve these linear equations are Gaussian Elimination, Cramer's rule and matrix elimination etc. If small whole numbers are added as a noise vector e as shown in following.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 1 \end{pmatrix}$$

Where:

$$\mathbf{A} = \begin{pmatrix} 2 & 3 & 4 \\ 1 & 4 & 2 \\ 3 & 6 & 5 \end{pmatrix}, \quad \mathbf{s} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 7 \\ 4 \\ 1 \end{pmatrix}$$

Thus, the equation becomes:

$$\mathbf{A} * \mathbf{s} + \mathbf{e} = \mathbf{b}$$

In that scenario, computing s becomes more complex. To make equation more difficult to compute, modular arithmetic is added. To find s from this equation is called LWE problem [18].

3.2.3 The Ring-Learning with Errors (R-LWE) Problem

In R-LWE, the mathematical challenge is to hide a secret polynomial in noisy data that is sampled from a structured ring. It involves the multiplication and addition of polynomials. Like

$$\mathbf{a}(\mathbf{x}) * \mathbf{s}(\mathbf{x}) + \mathbf{e}(\mathbf{x}) = \mathbf{b}(\mathbf{x}) modq$$

Here, $a(x), (s(x)$ and $e(x)$ are polynomials from a polynomial ring $Z_p[X]/(X^n + 1)$ with all their coefficients are from Z_p . But the coefficients of $s(x)$, and $e(x)$ are small. The multiplication is done by converting a polynomial into a circular matrix [19]. For example

$$\mathbf{a}(\mathbf{x}) = 2x^3 + 3x^2 + 4x + 7$$

$$\mathbf{s}(\mathbf{x}) = x^3 - x + 1$$

and if the ring is defined over modulo $x^3 + 1$. The circular matrix will be written as:

$$\mathbf{a}(\mathbf{x}) = \begin{pmatrix} 2 & -7 & -4 & -3 \\ 3 & 2 & -7 & -4 \\ 4 & 3 & 2 & -7 \\ 7 & 4 & 3 & 2 \end{pmatrix}, \mathbf{s}(\mathbf{x}) = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 1 \end{pmatrix}$$

In this way multiplication of $a(x) * s(x)$ is performed, then $e(x)$ noise is added like traditional polynomial addition except coefficients are from Z_p . This is called R-LWE. Here, the number of polynomials is 1 and is denoted by k.

3.2.4 The Module-Learning with Error (M-LWE)

If the value of k is increased from 1 then this problem is known as M-LWE problem. For example, if k=2 the matrix would look like this:

$$\mathbf{A} = \begin{pmatrix} a_1(x) & a_2(x) \\ a_3(x) & a_4(x) \end{pmatrix}$$

It would be four polynomials in a matrix A sample from that are from $Z_p[X]/(X^n+1)$ and each polynomial is multiplied by secret s by converting itself into circular matrix. If the matrix A is of 4 by 4 and $k = 4$, it would look as:

$$\mathbf{A} = \begin{pmatrix} a_1(x) & a_2(x) & a_3(x) & a_4(x) \\ a_5(x) & a_6(x) & a_7(x) & a_8(x) \\ a_9(x) & a_10x & a_{11}(x) & a_{12}(x) \\ a_{13}(x) & a_{14}x & a_{15}(x) & a_{16}(x) \end{pmatrix}$$

Here, $a1(x)$ **.......** $a16(x)$ are all polynomials and called entries or elements of matrix A. that are from $Z_p[X]/(X^n+1)$.

3.2.5 CRYSTALS-Kyber

Kyber is a big deal in the world of PQC. At its core, it is a KEM based on M-LWE problem that is lattice-base cryptographic scheme—a branch of mathematics that remains tough to crack even with quantum computers. This toughness comes from the LWE problem, a foundational puzzle in lattice-based cryptography that makes Kyber incredibly secure. Its lattice-based scheme, flexible parameter and efficiency makes it stand out till now.

Table 3.1 represents the parameters of Kyber[15, 7].

Table 3.1. Parameters of Kyber

	n	p	k
Kyber512	256	3329	2
Kyber768	256	3329	3
Kyber1024	256	3329	4

- n denotes the degree of polynomials.
- p denotes the small prime number used for the modular arithmetic of coefficients.
- k is the number of polynomials used in matrix.

3.3 Design Structure of Kyber

3.3.1 Keygeneration

In Kyber, the public key contains two elements a matrix A of kk size and a vector of polynomials t. The private key s and noise e consist of k number of polynomials. Each polynomial have a degree of n. They are generated using small, random coefficients. A matrix is generated using coefficients in modulo p. For example,

$$\mathbf{s(x)} = (x^3 - x^2, -x - 1)$$

$$\mathbf{A} = \begin{pmatrix} 2x^3 + 3x^2 + 4x - 7 & x^3 + 4x^2 + 2x - 4 \\ 3x^3 + 6x^2 + 5x + 1 & 3x^3 + 6x^2 + 5x \end{pmatrix}, \mathbf{e} = (x, -x^2 - 1)$$

and

$$t = (7x^3 - 9x^2 + 12, -11x^3 - 14x^2 + 6x - 9)$$

then by performing calculation

$$A * s + e = t$$

The public (A, t) is generated. So, the private key s is keep save and public is send to the clients or party B. This is how keys are generated in Kyber.

3.3.2 Encapsulation

On getting public key $(A, t\ )$, client encrypts key k that he wants to be shared to server for data exchange or communication. Following equation are used to compute ciphertext c.

$$u = \mathbf{A^T} r + e_1$$

$$v = \mathbf{t^T} r + e_2 + k$$

Here, the r , e_1 and e_2 are polynomials and also have small coefficients sampled from $Z_p[X]/(X^n + 1)$. The ciphertext (u, v) send to the server.

3.3.3 Decapsulation

Upon recieving of ciphertext (u, v), secret key s is used to get the key k back that will be further used for data exchange.

$$k' = v - \mathbf{s}^{\mathbf{T}} u$$

$$k' = \mathbf{e}^{\mathbf{T}} r + e_2 + k + \mathbf{s}^{\mathbf{T}} e_1$$

By removing the noise, shared key k is retried. [15] elaborates Key generation, encapsulation and decapsulation of Kyber in detail.

3.3.4 Kyber.v3 (NIST PQC Round 3)

The first PQC algorithm based on a lattice that NIST selected for standardization is Kyber KEM. By varying the size of the matrix k, one can directly alter the relative balance between security and performance; for security levels 1 (Kyber512), 3 (Kyber768), and 5 (Kyber1024), correspondingly, there are varying options for k. The security level determines how to modify the noise parameter η. Kyber development occurs in two phases. KYBER is an IND-CPA safe public-key encryption technique.KYBER is CPAPKE and IND-CCA2-secure KEM with minimal FO transform.CCAKEM [15].

3.3.4.1 KYBER.CPAPKE

The three parts of Kyber.CPAPKE are encryption (Kyber.CPA.Enc), decryption (Kyber.CPA.Dec), and key generation (Kyber.CPA.KeyGen).

Kyber.CPA.KeyGen(): The Key creation function creates a public key pk and secret key sk before Alice and Bob are able to communicate. Initially, vectors $\mathbf{s}$ and $\mathbf{e}$ are formed via binomial distribution sampling, whereas a matrix $\boldsymbol{A}$ is immediately generated in the NTT domain by uniform sampling. Subsequently, the polynomial $\hat{t} = \hat{A} \circ \mathrm{NTT}(s) + \mathrm{NTT}(e)$ is calculated, and $sk = (\mathrm{NTT}(s))$ and $pk = (\hat{t}, \rho)$, where ρ is a random number, yield the key pair.

Kyber.CPA.Enc(pk, m, r): Initially, in the NTT domain, uniform sampling generates the matrix $\boldsymbol{A}^T$. Subsequently, vectors $\mathbf{r}$, $\mathbf{e}_1$, and polynomial e_2 are formed by seed r. Next, we compute the vectors $\mathbf{u}$ and v as follows: $v = \mathrm{NTT}^{-1}(\ \hat{t^T} \circ \hat{r}) + e_2 + \mathrm{Decompress}_q(\mathrm{Decode}_1(m), 1)$ and $\mathbf{u} = \mathrm{NTT}^{-1}(\ \hat{A^T} \circ \hat{r}) + \mathbf{e}_1$. Afterwards, v and $\mathbf{u}$ are encoded and compressed to create c_1 and c_2. Ultimately, $c = (c_1 || c_2)$, the ciphertext, is given back.

Kyber.CPA.Dec(sk, c): From the input c, the vectors $\mathbf{u}$ and v are compressed. Then, using the formula $m' = \mathrm{Encode}_1(\mathrm{Compress}_q(v - \mathrm{NTT}^{-1}(\hat{s^T} \circ \mathrm{NTT}(\mathbf{u}), 1)))$. The function's output is returned as m'.

3.3.4.2 KYBER.CCAKEM

The three primary steps of the IND-CCA2 secure Kyber KEM, known as Kyber.CCAKEM and submitted to NIST PQC Round 3, are key generation (Kyber.CCA.KeyGen), key encapsulation (Kyber.CCA.Enc), and key decapsulation (Kyber.CCA.Dec). Kyber's prime, p, is modified from $7,681$ to $3,329$, allowing NTT to accelerate the polynomial multiplication in Kyber. Based on the Kyber.CPA, the Kyber.CCA implementation employs the FO transform [57].a functional explanation of each of Kyber's three core functions.CPA are explained as follows, and for additional information about Kyber, the reader is kindly referred to [15] and [7].

Algo. 1, 2, and 3 show the Kyber.CCA.KeyGen, Kyber.CCA.Enc and Kyber.CCA.Dec functions in Kyber.CCA, respectively. To produce the public keys pk and sk', Kyber.CCA.KeyGen() invokes the Kyber.CPA.KeyGen() function. Next, sk=($sk'||pk||H(pk)||z$) is used to construct the secret key sk, where H stands for SHA3-256. The public key pk is received by Kyber.CCA.Enc(), which then uses the Kyber.CPA.Enc(pk, m, r) function to produce the ciphertext c.Next, SHAKE-256 generates the shared secret ss. The shared secret ss and ciphertext c are included in the output. The inputs for Kyber.CCA.Dec() are the secret key sk and the ciphertext c. The message m is recovered using the Kyber.CPA.Enc(sk,c) function first, and the new ciphertext c' is then computed using Kyber.CPA.Dec(pk,m',r'). The encryption is successful if there is a success in the comparison between c and c'.

Algorithm 1 Kyber.CCA.KeyGen()

```
1: **Output**: Public key $pk$, Secret key $sk$
2: $z = B^{32}$
3: $(pk, sk') :=$ Kyber.CPA.KeyGen()
4: $sk := (sk'||pk||H(pk)||z)$
5: **return** $(pk, sk)$
```

Algorithm 2 Kyber.CCA.Enc(pk)

```
1: **Input**: Public key $pk$
2: **Output**: Ciphertext $c$, Shared key $ss$
3: $m = B^{32}$
4: $m =$H$(m)$
5: $(\bar{K}, r) :=$ G$(m||$H$(pk))$
6: $c :=$ Kyber.CPA.Enc$(pk, m, r)$
7: $ss :=$ KDF$(\bar{K}||$H$(c))$
8: **return**$(c, ss)$
```

Algorithm 3 Kyber.CCA.Dec(c, sk)

```
1:  Input: Ciphertext $c$, Secret key $sk$
2:  Output: Shared key $ss$
3:  $m'$ := Kyber.CPA.Dec($sk$,$c$)
4:  ($\bar{K}'$,$r'$) := G($m'||h$)
5:  $c'$ := Kyber.CPA.Enc($pk$,$m'$,$r'$)
6:  if $c = c'$ then
7:      $ss$ := KDF($\bar{K}'||$H($c$))
8:  else
9:      $ss$ := KDF($z||$H($c$))
10: end if
11: return $ss$
```

The figures 3.4 and 3.5 show encryption and decryption flowchart of Kyber KEM and highlight the usage of Kyber.CPA.Enc and Kyber.CPA.Dec role in KYBER.CCAKEM.

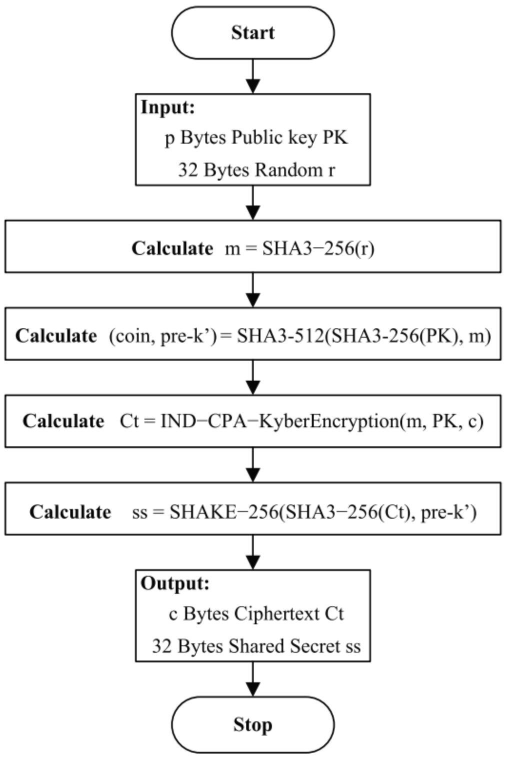
Start
Input:
p Bytes Public key PK
32 Bytes Random r
Calculate m = SHA3−256(r)
Calculate (coin, pre-k') = SHA3-512(SHA3-256(PK), m)
Calculate Ct = IND−CPA−KyberEncryption(m, PK, c)
Calculate ss = SHAKE−256(SHA3−256(Ct), pre-k')
Output:
c Bytes Ciphertext Ct
32 Bytes Shared Secret ss
Stop

Figure 3.4: The flowchart of KYBER.CCAKEM encryption [2]

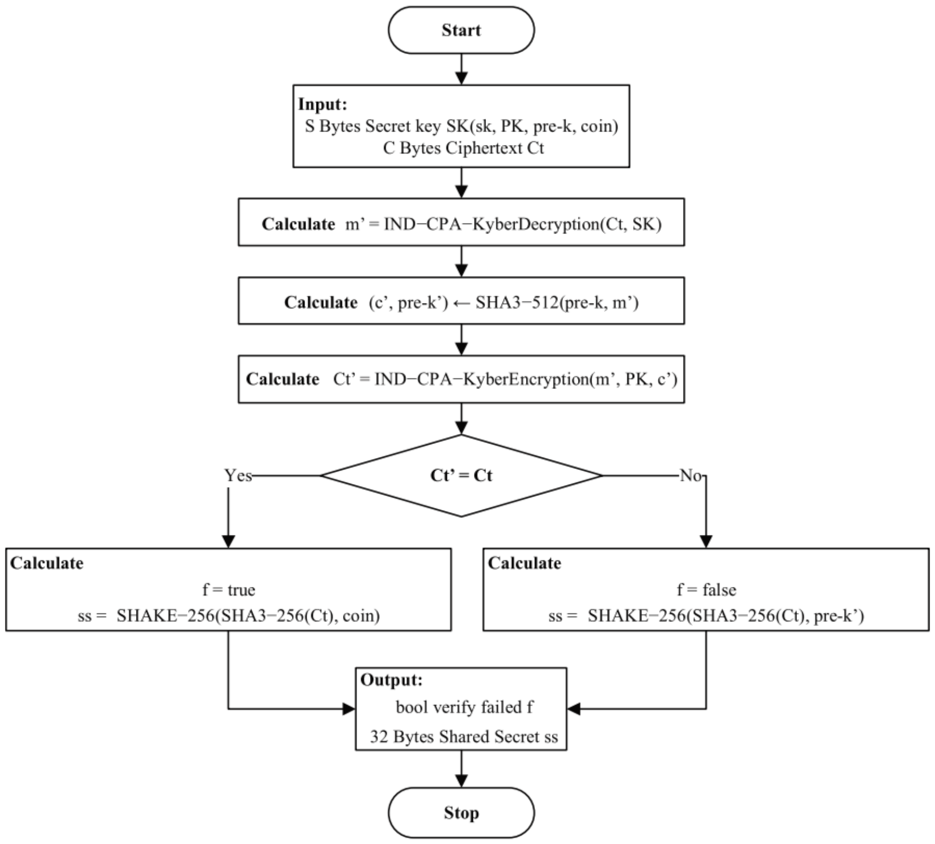
Start
Input:
S Bytes Secret key SK(sk, PK, pre-k, coin)
C Bytes Ciphertext Ct
Calculate m' = IND−CPA−KyberDecryption(Ct, SK)
Calculate (c', pre-k') ← SHA3−512(pre-k, m')
Calculate Ct' = IND−CPA−KyberEncryption(m', PK, c')
Ct' = Ct
Yes
No
Calculate
f = true
ss = SHAKE−256(SHA3−256(Ct), coin)
Calculate
f = false
ss = SHAKE−256(SHA3−256(Ct), pre-k')
Output:
bool verify failed f
32 Bytes Shared Secret ss
Stop

Figure 3.5: The flowchart of KYBER.CCAKEM decryption [2]

3.3.5 Kyber.v4 (NIST PQC Final Round)

Kyber KEM [7], based on Kyber.v3 [15], standardized by NIST as a final PQC algorithm, is now named ML-KEM or FIPS 203. This ML-KEM specified in this standard adheres to the aforementioned structure. Specifically, this standard first explains the K-PKE public-key encryption method (named as KYBER.CPAPKE in Kyber.v3), and then it employs the K-PKE algorithms as subroutines to describe the ML-KEM algorithms. The K-PKE scheme should not be utilized as a stand-alone scheme since it is insufficiently secure. This standard has only some differences from the Kyber.v3 as illustrated below.

- The shared secret key's length is fixed at 256 bits in this specification. This specification allows this key to be used as a symmetric key straight away in applications.
- Compared to the third-round specification, the ML-KEM.Encaps and ML-KEM.Decaps algorithms in this specification employ an altered FO transform

version, see, [52], [53].

- The initial randomness m in the Kyber.CCA.Enc() method was first hashed before being utilized in the third-round specification. In particular, there is an extra step in Algorithm 2 that carried out the operation $m = H(m)$. This step serves as a protection against the application of flawed randomness generation techniques. This step is superfluous and is not carried out in ML-KEM, as this standard mandates the usage of NIST-approved randomness generating.
- Explicit input validation procedures that are absent from the third-round specification are included in ML-KEM specification. As an instance, ML-KEM.The byte array holding the encapsulation key must properly decode to an array of integers modulo q in order for encaps to function, with no modular reductions.

Throughout this thesis, the terms "Kyber" and "ML-KEM" are used interchangeably in various chapters and sections. It is important to clarify that ML-KEM represents the final standardized version of Kyber, which was adopted by NIST in August 2024. While the name Kyber is commonly associated with earlier versions, ML-KEM is the official name for the finalized cryptographic scheme. Therefore, references to Kyber and ML-KEM in this document should be understood as referring to the same algorithm, with ML-KEM being its most current iteration.

3.4 Keccak/SHA3

Keccak, the cryptographic hash function that underpins the SHA3 standard (FIPS202 [3]), represents a significant departure from earlier members of the Secure Hash Algorithm (SHA) family, such as SHA-1 and SHA-2. Its unique design is based on what's called a “sponge construction,” which is a flexible framework applicable to various cryptographic needs, including hashing, authenticated encryption, and random number generation.

3.4.1 Keccak and Sponge Construction

At the heart of Keccak is the sponge construction, which consists of two primary phases: absorbing and squeezing. The process operates on a fixed-size state, usually 1600 bits for SHA3, divided into two sections: the **rate (r)** and the **capacity (c)**. The rate controls how much of the input is processed at a time, while the capacity is tied to the security strength of the algorithm, where $c = 2n$, with n being the desired output size in bits (e.g., 256 bits for SHA3-256). Figure 3.6 illustrate the sponge construction of Keccak/SHA3.

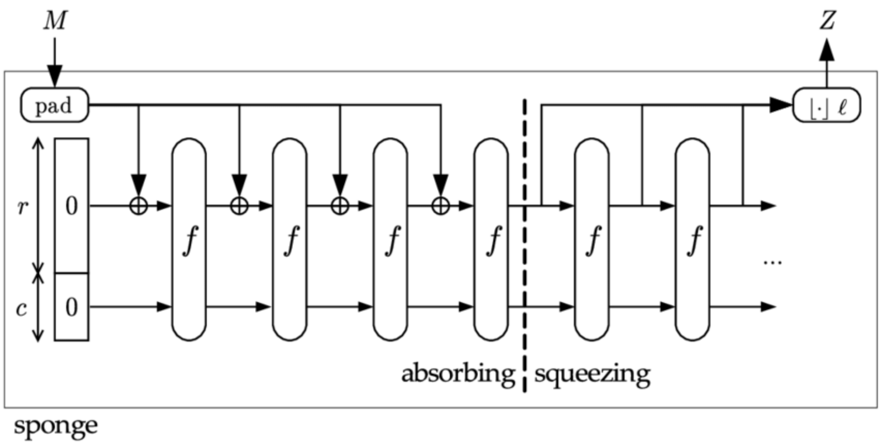

Figure 3.6: The sponge construction [3]

- **Absorbing Phase**: During this phase, the input message is padded and divided into blocks of size r. Each block is XORed with the first r bits of the state, followed by the application of the permutation function f. This process continues until all message blocks have been absorbed into the state.
- **Squeezing Phase**: Once the absorbing phase is complete, the state is squeezed to produce the hash output. The first r bits of the state are extracted as output, and if more output is required, the permutation function f is applied again to generate additional output bits.

3.4.2 Internal Structure and Permutation Function

The internal state of Keccak is represented as a three-dimensional array of bits, organized into a $5 \times 5 \times w$ matrix, where w is the word size (e.g., 64 bits for a 1600-bit state). The permutation function f, which operates on this state, is composed of a sequence of operations applied over several rounds. Each round consists of the following five steps:

1. **Theta (θ) Step**: This step involves bitwise operations across the rows of the matrix. It introduces diffusion by ensuring that every bit of the state depends on every bit of the previous round. The operation involves XORing each bit in the matrix with a value derived from neighboring columns, contributing to the diffusion property.
2. **Rho (ρ) Step**: In the Rho step, each bit in the matrix is rotated by an offset that varies depending on its position within the matrix. This step ensures that the bits are rearranged within the state, contributing to the permutation's complexity.

3. **Pi (π) Step**: The Pi step is a simple permutation of the matrix positions. It rearranges the bits within the matrix according to a fixed pattern, further contributing to the diffusion and ensuring that bits from different positions in the matrix are mixed together.

4. **Chi (χ) Step**: The Chi step is a non-linear operation applied to each row of the matrix. In this step, each bit is XORed with a non-linear function of the two succeeding bits in the same row. This non-linearity is crucial for the security of the hash function, as it prevents simple linear relationships between input and output.

5. **Iota (ι) Step**: The Iota step introduces a round-dependent constant into the state. This step ensures that each round of the permutation is unique, preventing symmetries and simplifying potential attacks. The round constant is XORed with the first bit of the state, breaking any potential linear patterns that might have formed.

3.4.3 Padding and Domain Separation

Keccak uses a specific padding scheme, known as *multi-rate padding (pad10*1)*, which ensures that the message is aligned to the block size required by the sponge construction. The padding appends a single '1' bit followed by the necessary number of '0' bits and ends with another '1' bit. This padding method guarantees that the message length is always a multiple of the rate r.

Additionally, Keccak supports domain separation, a technique that allows the same permutation function to be used for different purposes (e.g., hashing vs. MAC generation) without risk of overlap. This is achieved by appending specific bit strings to the input message before processing.

3.4.4 Security Considerations

Keccak's security is primarily determined by the capacity c of the sponge construction. For a desired output length n, the security level against collision and pre-image attacks is approximately $2^{n/2}$ and 2^n respectively, assuming an ideal permutation function. The design of Keccak ensures resistance to known cryptographic attacks such as differential and linear cryptanalysis, as well as providing adequate protection against side-channel attacks due to its simplicity and regular structure.

3.4.5 Variants of Keccak/SHA3

Keccak/SHA3 has several variants, each tailored to different security levels and application needs. Table 3.2 summarizes the key parameters of these variants:

Table 3.2. Variants of Keccak/SHA3 and Their Parameters

Variant	**Output Length (bits)**	**State Size (bits)**	**Rate (r) (bits)**	**Capacity (c) (bits)**	**Number of Rounds**
SHA3-224	224	1600	1152	448	24
SHA3-256	256	1600	1088	512	24
SHA3-384	384	1600	832	768	24
SHA3-512	512	1600	576	1024	24
SHAKE128	Extendable Output	1600	1344	256	24
SHAKE256	Extendable Output	1600	1088	512	24

3.4.6 Advantages and Applications

Keccak's design offers several advantages, including:

- **Flexibility**: The sponge construction allows Keccak to be easily adapted for different output lengths and cryptographic functions beyond hashing, such as stream encryption and authenticated encryption.
- **Efficiency**: Keccak's permutation-based approach enables efficient implementation in both software and hardware, with high throughput and low area requirements.
- **Security**: The unique structure of Keccak provides strong security guarantees, making it a robust choice for cryptographic applications in the post-quantum era.

Overall, Keccak's adoption as the SHA3 standard marks a significant advancement in cryptographic hash functions, providing a strong foundation for secure digital communications in the face of emerging quantum threats. However, as PQC continues to evolve, the exploration of alternative hash functions, such as Ascon, remains a critical area of research to ensure continued security and efficiency.

3.5 Ascon

Ascon is the new standard for lightweight cryptography, chosen by the NIST in the USA.

3.5.1 Ascon Algorithm

Keyed initialization and finalization routines make up Ascon's duplex mode of operation. The design parameters (state size, rate, capacity, key length) and operating modes (Ascon-128 and Ascon-128a) are carefully chosen to achieve compact implementation.

Ascon is based on a duplex mode of operation, it consists of keyed initialization and finalization functions. To achieve compact implementation, the design parameters (state size, rate, capacity, key length) and operating modes (Ascon-128 and Ascon-128a) are diligently selected. Table 3.3 lists the two Ascon operating modes along with the recommended length of their design parameters. For lightweight use cases in IoT contexts, Ascon-128 (main recommendation) and Ascon-128a (secondary recommendation) are perfect.

Table 3.3. Design Specifications for Ascon-128 and Ascon-128a.

Parameters	**Ascon-128**	**Ascon-128a**
Security Level, Nonce, Key Size, Tag	128-bit	128-bit
Block Size	64-bit	128-bit
Rate / Capacity	64-bit / 256-bit	128-bit / 192-bit
Internal State	320-bit	320-bit
Number of Rounds (a/b)	12 / 6	12 / 8

3.5.1.1 Ascon Internal Architecture

Ascon AE's internal design is displayed in Figure 3.7. Ascon runs in both of its operational modes on a 320-bit state denoted by S. There are four steps involved in updating the state: initialization, AD processing, PT/CT processing, and finalization. While the permutation function f is used six times in processing AD and PT/CT, it is used twelve times in the initialization and finalization stages. The 320-bit state is divided into inner (S_c) and outer (S_r) components, with r and c denoting, respectively, rate and capacity. The parameter sizes of r and c for both Ascon operating modes are given in Table 3.3 and are also mathematically represented in Equation 3.2. When realized in hardware, the 320-bit internal state is modeled by five 64-bit registers each. Access to the internal state begins at byte 0, the most significant byte (or bit) of a_0, and ends at byte 39, the least significant byte (or bit) of a_4.

$$S = Sr\|Sc = a0\|a1\|a2\|a3\|a4. \tag{3.2}$$

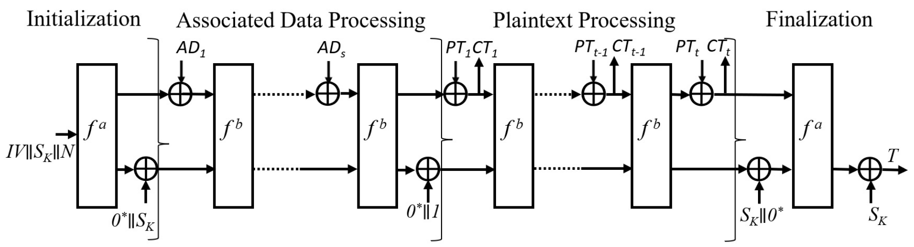

Figure 3.7: Ascon AE encryption's internal architecture. (The decryption procedure is the same; however, PT is obtained by processing CT).

3.5.1.2 Initialization

Concatenating three parameters yields an input that can serve as Ascon's initial state. These parameters consist of the pre-defined initialization vector (IV), nonce N, and secret key S_K. In order to ensure that the confidentiality requirement is met, N must be refreshed for each encryption procedure. After undergoing 12 rounds of permutation f, the original state is XORed with the secret key, or S_K. The entire initialization process is represented by equations 3.3 and 3.4.

$$S \leftarrow IV\|S_K\|N \tag{3.3}$$
$$S \leftarrow f^a(S) \oplus (0^{320-S_k}\|S_K) \tag{3.4}$$

3.5.1.3 Associated data

Only when AD is present is processing of AD necessary; in other cases, it can be omitted. For the AD, Ascon uses a chunk-based processing strategy in which each chunk is made up of r bits. A single 1 and the least number of 0s necessary to make the AD a multiple of r are appended, a process known as padding, to guarantee that the entire length of the AD is a multiple of the r-bit size. After that, the altered AD value is split up into i chunks of r bits. Information in AD might not always be private, but it needs to be protected from any possible intrusion. After six rounds of permutation, each block of the AD is then integrated into the internal state. One bit 1 is XORed to the state's least significant bit once all of the AD has been processed. Equations 3.5 and 3.6 provide the mathematical representations for managing the AD processing, respectively.

$$S \leftarrow f^b((S_r) \oplus AD_i)\|S_c) \tag{3.5}$$
$$S \leftarrow S \oplus (0^{319}\|1) \tag{3.6}$$

3.5.1.4 Plaintext/Ciphertext Processing

Similar to the preceding step, this step also involves padding in order to obtain the r-bit multiple of PT. The padded plaintext is divided into smaller pieces, and each piece is then encrypted. The encryption procedure is demonstrated by equations 3.7 and 3.8. Each chunk of PT_i and the state r-bit of state S_r are subjected to an XOR operation, producing a single block of ciphertext message CT_i. The following chunk is processed after the state is updated six times using the permutation function f. .

$$S_r \leftarrow S_r \oplus PT_i, \quad CT_i \leftarrow S_r \tag{3.7}$$

$$S \leftarrow f^b(S) \tag{3.8}$$

3.5.1.5 Finalization

This step generates a 128-bit Tag T. This T is utilized during the decryption procedure for verification. If the T is confirmed, the decryption process's result is legitimate PT. If not, CT has been altered and is unreliable. Encryption and decryption in Ascon work in the same ways. The main distinction is that CT_i is processed during decryption rather than PT_i. When it comes to hardware, encryption and decryption can be performed on the same setup. The internal state and key S_K are XORed before 12 permutation rounds are carried out in the finalization step. After XORing the least significant 128 bits in the state with the least significant 128-bit of the key S_K, tag T is produced.

In the finalization step, internal state and key S_K are XORed and then 12 permutation rounds are performed. Tag T is generated when the least significant 128-bit of the key S_K is XORed with the least significant 128 bits in the state. The encryption algorithm's outputs are T and C_T. The equations 3.9 and 3.10 display the mathematical representation.

$$S_r \leftarrow f^a(S \oplus (0^r \| S_K \| 0^{320-r-k})) \tag{3.9}$$

$$T \leftarrow \lceil S \rceil^{128} \oplus \lceil S_k \rceil^{128} \tag{3.10}$$

3.5.1.6 Permutation

The permutation function, an SPN-based round transformation f that is applied iteratively to the state, is the crucial operation in Ascon. The three steps that make up the permutation function are shown in Figure **??**. The function begins by adding a constant, and then sums the round constant c_r with the register word x_2 from the internal state for each round of permutation. The following operation is carried out on 64 5-bit S-boxes in the substitution layer. The S-box's cryptographic features have been enhanced through the utilization of affine transformation. Additionally, because the S-box relies on a small number of logical operations when implemented in hardware, it provides a high degree of parallelism. Diffusion operations on each register word x_i of the 64-bit are carried out in the diffusion layer in accordance with the pre-defined constants set by the Ascon team. Every word in the state is subjected to the linear diffusion layer.

3.5.2 Ascon's Hashing Modes

The Ascon family of functions include the extendable output functions Ascon-XOF and Ascon-XOFa with sponge-based modes of operation as ilustrated in figure 3.8, and the hash functions Ascon-Hash and Ascon-Hasha. With a minimum hash size of 256 bits, both offer 128-bit security. The hashing modes and the authorized encryption modes both employ the same low-cost 320-bit permutation.

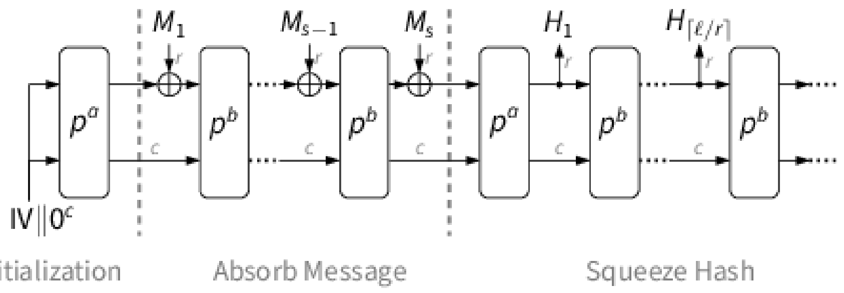

Figure 3.8: Ascon hashing's sponge mode [4]

After absorbing the message M in 64-bit blocks M_i, the hashing modes compress the hash value H in 64-bit blocks. The state is subjected to the b-round permutation p^b following each absorbed or compressed block, with the exception of the last. Following the final message block, the entire a-round permutation p^a is applied at initialization and finalization. Table 3.4 gives the design parameter length for Ascon hashes.

See the Ascon document [4] for more information on things like the values of the IV and round constants, padding rules, and the (almost same) decryption method.

3.6 Summary

Kyber's selection as ML-KEM in the final round of the NIST PQC competition underscores its significance as a robust post-quantum cryptographic solution. As a lattice-based KEM, it meets the stringent security demands necessary for safeguarding against

Table 3.4. Design Parameter Length for Ascon-Hash and Ascon-XOF.

Algorithms	**Ascon-Hash**	**Ascon-Hasha**	**Ascon-XOF**	**Ascon-XOFa**
Security	128-bit	128-bit	128-bit	128-bit
Output	256-bit	256-bit	arbitrary	arbitrary
Rate / Capacity	64-bit / 256-bit	64-bit / 256-bit	64-bit / 256-bit	64-bit / 256-bit
State	320-bit	320-bit	320-bit	320-bit
Rounds (a/b)	12 / 12	12 / 8	12 / 12	12 / 8

quantum computing threats. The use of Keccak as its primary hash function has proven effective in maintaining cryptographic integrity and security within ML-KEM. However, the potential for lightweight cryptographic schemes like Ascon to enhance performance in resource-constrained environments introduces an important area of exploration. Ascon's ability to generate hashes with lower computational overhead makes it a viable candidate for replacing Keccak in specific applications, particularly where efficiency is critical. This chapter has established the foundational knowledge necessary to understand the subsequent performance evaluations and analysis presented later in the thesis. By considering both Keccak and Ascon, the research aims to assess how these hash functions contribute to the performance and suitability of ML-KEM across different hardware environments, with a particular focus on their efficiency in embedded systems.

Chapter 4

Proposed Methodology

4.1 Overview

This chapter presents the methodology developed for evaluating the integration of the Ascon hash function into the Kyber cryptographic algorithm, replacing the originally used Keccak function. The objective of this methodology is to provide a systematic framework that examines the impact of this substitution on both the performance and security of Kyber.

The proposed approach is structured to cover several key aspects: the design and implementation of the Ascon hash function in place of Keccak, performance benchmarking to compare the computational efficiency of the two hash functions, and security analysis to ensure that Kyber's integrity remains uncompromised by the change. Each step in the methodology is critical to understanding how Ascon performs within the Kyber framework, particularly in environments where resource constraints play a significant role, such as embedded systems.

The methodology is divided into distinct layers and modules, each addressing a specific phase of the integration process. This layered architecture enables a thorough and organized evaluation, from the initial design of the replacement to the final testing and validation. A detailed flow diagram is provided to illustrate the process, followed by an in-depth explanation of each component.Each step in the figure 4.1 is connected, illustrating how the methodology moves from preparation through to the final analysis of Ascon's impact on Kyber.

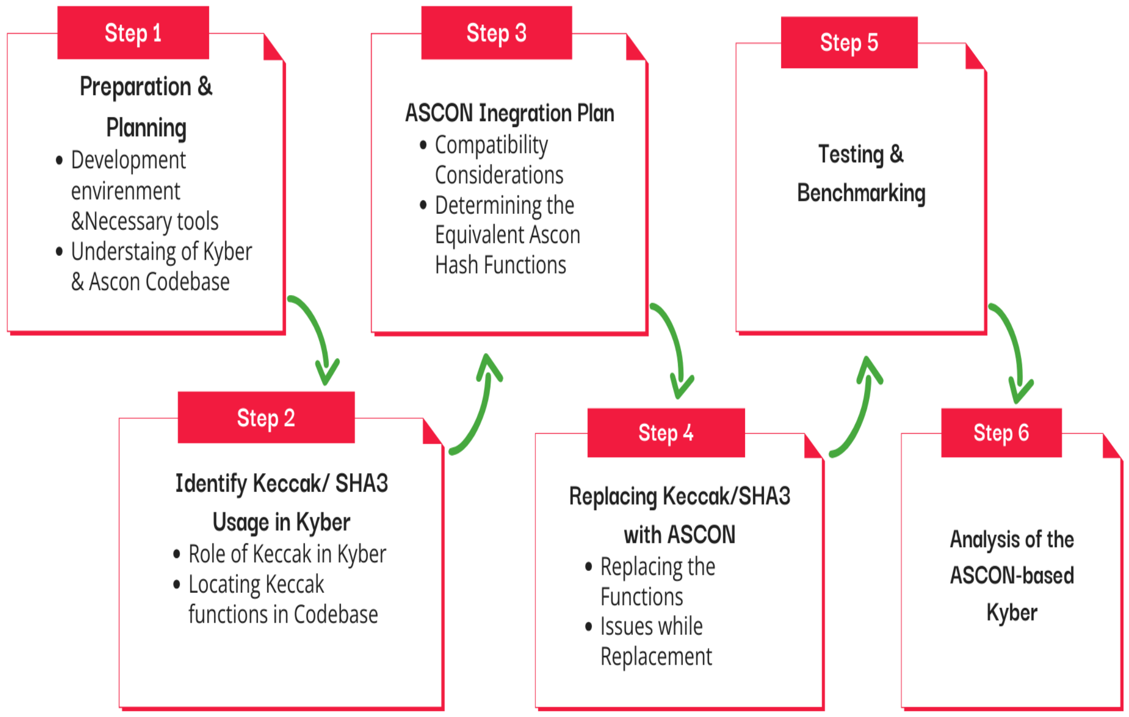
Step 1
Preparation & Planning
• Development envirenment &Necessary tools
• Understaing of Kyber & Ascon Codebase
Step 2
Identify Keccak/ SHA3 Usage in Kyber
• Role of Keccak in Kyber
• Locating Keccak functions in Codebase
Step 3
ASCON Inegration Plan
• Compatibility Considerations
• Determining the Equivalent Ascon Hash Functions
Step 4
Replacing Keccak/SHA3 with ASCON
• Replacing the Functions
• Issues while Replacement
Step 5
Testing & Benchmarking
Step 6
Analysis of the ASCON-based Kyber

Figure 4.1: Flow of proposed methodology

By following this structured approach, the methodology aims to deliver a comprehensive evaluation of how well Ascon integrates into Kyber, offering insights into its performance and potential advantages for post-quantum cryptographic systems.

The first step is Preparation and Planning that involves outlining the project scope and goals, setting up the necessary tools, determining the resources needed for development, testing, and benchmarking, ensuring all preliminary requirements are met for a smooth transition. 2nd step is Identify Keccak/SHA3 Usage in Kyber that provides the detailed examination of where and how Keccak/SHA3 is used within the Kyber cryptographic algorithm. By understanding these specific points, a clear plan can be developed for replacing Keccak with Ascon without disrupting the core functionality of Kyber. Based on the identified points of Keccak/SHA3 usage, 3rd step is Ascon Integration Plan that formulates a concrete plan for integrating Ascon into Kyber. The integration plan addresses technical details, such as modifying cryptographic functions, ensuring compatibility, and maintaining efficiency throughout the replacement process. The next step is Replacing Keccak/SHA3 with Ascon in which the actual replacement of Keccak with Ascon takes place. This involves reworking the hashing operations within Kyber to utilize Ascon while preserving the overall structure and design of the algorithm. Careful attention is given to ensure the replacement is seamless and effective. Once the replacement is complete,there comes the Testing and Benchmarking step to conduct comprehensive testing to verify the correctness of the Ascon integration. This is followed by performance benchmarking, where the updated Kyber (ML-KEM with Ascon) is compared against the original (ML-KEM with Keccak) by measuring CPU cycles, execution time, and other performance metrics. Finally comes the Analysis of Ascon-based Kyber step that involves analyzing the performance and security of the Ascon-based Kyber. This includes reviewing the benchmarking results, assessing the efficiency improvements, and conducting a security analysis to ensure that Kyber's security properties remain intact with the new hash function in place, described in Chapter 5. Following sections present the detailed description of each step.

4.2 Preparation and Planning

To successfully integrate Ascon's hashing scheme with the Kyber algorithm, a well-structured approach is necessary. This section outlines the preparatory steps required to set up the development environment and install the necessary tools to carry out the research effectively.

4.2.1 Development Environment

The development environment plays a crucial role in ensuring that the implementation and testing of the integration are carried out smoothly. For this research,

- **Ubuntu 22.04 LTS**: A widely-used Linux distribution, offers robust support for C++ development,is selected as the operating system for the development environment, which is essential for implementing cryptographic algorithms like Kyber and Ascon. Ubuntu 22.04 provides a stable and up-to-date environment with extensive documentation and community support. Additionally, it offers a vast repository of software packages, making it easier to install the necessary tools and dependencies required for this research.
- **Hardware**: Testing on PC that support Processor "Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz 2.71 GHz" and system type is "64-bit operating system, x64-based processor"

4.2.2 Installation of Necessary Tools

To begin the implementation process, the first step involves setting up the tools needed for C++ development. The following tools and libraries were installed:

Compilers: A C compiler, such as `gcc`, is installed to compile the source code. The compiler ensures that the code adheres to the C standards and can be successfully built into executable programs.

Build Systems: Tools like `make` or `CMake` are utilized to manage the build process. These tools automate the compilation and linking of code, streamlining the development workflow.

4.2.3 Understanding the Kyber Codebase

It involves examining how cryptographic functions are implemented and where hashing functions are utilized. The integration process focuses on replacing existing hash functions, specifically Keccak/SHA3, with Ascon functions while ensuring that all aspects of the algorithm remain functional and secure. The official GitHub repository of Kyber [58], implemented in C, is cloned into the development environment for further integration and testing. In repository, there are two key directories: `ref` and `AVX2`, each serving different purposes.

4.2.3.1 AVX2 Implementation

Advanced Vector Extensions 2 (AVX2) is an extension to the x86 instruction set that provides support for 256-bit vector operations. It is particularly useful for accelerating cryptographic algorithms, including those involved in lattice-based cryptography like Kyber. It can significantly enhance the performance of cryptographic algorithms by enabling parallel processing of data. This is achieved through its wide vector registers and specialized instructions that allow for efficient execution of multiple operations simultaneously. For cryptographic algorithms, such as those used in Kyber, AVX2 can improve the speed of operations like polynomial arithmetic, matrix multiplication, and hashing [28]. AVX2 instructions enable operations on large data vectors, enhancing performance, especially in computation-heavy tasks. This implementation is:

- Tailored for high performance and speed, making it suitable for production environments
- More complex and harder to understand due to its reliance on AVX2 instructions and low-level optimizations

4.2.3.2 Reference Implementation

The `ref` directory includes a reference implementation of Kyber, written in plain C. This version is designed to be simple and easy to understand, making it ideal for those who want to grasp the Kyber algorithm and its components. It's not optimized for speed but is excellent for:

- Gaining insights into how Kyber works
- Testing and verifying the correctness of the algorithm
- Providing a baseline for comparing performance with more optimized versions

4.2.3.3 Key Differences

Following are the key differences between both implementation:

- **Performance:** The `AVX2` implementation runs significantly faster becuse of instructions.
- **Code Complexity:** The `AVX2` code is more complex, requiring a solid grasp of AVX2 and advanced optimization techniques.
- **Portability:** The `ref` implementation is more portable and can be compiled on a variety of platforms since it uses standard C libraries. In contrast, the `AVX2` implementation is specific to x86-64 architectures with AVX2 support.

For this work we are using ref implementation for the replacement of Keccak with ascon.

4.2.3.4 Benchmarking Program

For benchmarking implementations, speed test programs are provided for x86 CPUs that utilize the Time Stamp Counter (TSC) or the actual cycle counter provided by the Performance Measurement Counters (PMC) to assess performance. To compile these programs, the command `make speed` is used. This command generates executables named `test/test_speed$ALG` for all parameter sets denoted as `$ALG`.

The programs report both the median and average cycle counts from 1,000 executions of various internal functions as well as the API functions for key generation, encapsulation, and decapsulation. By default, the Time Stamp Counter is employed for these measurements. If actual cycle counts from the Performance Measurement Counters are desired, the environment variable `CFLAGS` should be set to `-DUSE_RDPMC` before compilation.

It is important to note that the reference implementation located in the `ref/` directory is not optimized for any specific platform. This reference implementation prioritizes code clarity and thus operates at a significantly slower rate compared to a trivially optimized, yet platform-independent, implementation.

4.2.4 Understanding the Ascon Codebase

Before integrating Ascon's hashing scheme into the Kyber algorithm, it is essential to thoroughly understand the Ascon codebase. To delve deeper into the Ascon codebase, including its implementation details and usage examples, the project's GitHub repository [59] is an invaluable resource. It provides access to the source code and documentation, offering insights into how Ascon's cryptographic functions are implemented. 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). In this work, only the hash function and extendable output function are needed as Kyber makes use of the SHA3 HASH and SHAKE XOF functions to perform these operations meaning this is all that needs to be replaced. The repository for Ascon v1.2 has multiple implementations optimized for different system architectures and requirements. These implementations are designed to provide flexibility based on the hardware or application context each within its specific folder structure.

- **Optimized implementations** include various versions tailored for specific hardware or performance needs, such as speed or size optimization for 64-bit and 32-bit systems, bit-interleaved designs, and adaptations for 8-bit microcontrollers and ESP32. These versions are designed to enhance performance or reduce resource usage based on the target environment.

- **Reference implementation** of the Ascon hash algorithm provides a standard, straightforward version that emphasizes clarity and correctness over performance. It serves as the baseline for verifying the algorithm's functionality.

4.2.4.1 Ascon Hash Functions

Ascon offers following main types of cryptographic hash functions:

- **Ascon-Hash** is located in the folder `crypto_hash/asconhashv12`, and it uses 12 rounds in both the initialization and finalization phases to produce a fixed-length output (256 bits).

- **Ascon-Hasha** is found in `crypto_hash/asconhashav12` and, like Ascon-Hash, it generates a 256-bit output with 64-bit blocks. However, it uses 12 rounds for initialization but only 8 rounds for finalization, which can optimize certain performance factors.

- **Ascon-XOF** is in the folder `crypto_hash/asconxofv12`. It allows for an arbitrary output length and uses 64-bit blocks, with 12 rounds for both initialization and finalization, making it suitable for XOF.

- **Ascon-XOFa** can be found in `crypto_hash/asconxofav12`. It also allows for arbitrary-length output and uses 64-bit blocks, but like Ascon-Hasha, it employs 12 rounds for initialization and 8 rounds for finalization to enhance efficiency in specific scenarios.`crypto_hash/asconxofv12`. It allows for an arbitrary output length and uses 64-bit blocks, with 12 rounds for both initialization and finalization, making it suitable for XOF.

4.2.4.2 Core Components of the Ascon Codebase

To effectively integrate Ascon, it is important to understand its core components :

1. **State Representation**: Ascon uses a fixed-size array of bits or bytes to represent its internal state. This state undergoes various transformations during hashing and cryptographic operations. The structure and size of this state are defined in the codebase and are crucial for the efficient implementation of the Ascon algorithm.

2. **Permutation Function**: A key element of Ascon is its permutation function, which is responsible for mixing and transforming the internal state. This function includes:

   - **Substitution**: Introduces non-linearity by replacing bits in the state.
   - **Permutation**: Reorders bits or bytes to spread information across the state.
   - **Mixing**: Combines bits from different areas of the state to improve diffusion and security.

   The design of the permutation function ensures both security and efficiency, making it resistant to various cryptographic attacks.

3. **Padding Scheme**: Ascon includes a padding scheme to handle messages of different lengths. Padding ensures that the input message fits into the algorithm's internal state correctly. This involves appending specific bits to align the message with the required block size.

4. **Absorb and Squeeze Functions**: Ascon-Hash operates in two main phases:

   - **Absorb**: This phase processes the input message by incorporating it into the internal state through a series of transformations.
   - **Squeeze**: The final hash value is extracted from the internal state, producing the output by applying additional transformations and extracting the required number of bits.

These phases are designed to produce secure outputs efficiently, while maintaining resistance to potential attacks.

5. **Initialization and Finalization**: Ascon functions also include initialization and finalization routines:

   - **Initialization**: Prepares the internal state for processing by setting it to a specific value.
   - **Finalization**: Completes the hashing process and produces the final output, ensuring that any remaining data is processed correctly.

   After getting familiar with Ascon codebase, the next step is to locate the Keccak function in Kyber.

4.3 Identify Keccak/SHA3 Usage in Kyber

The integration of Ascon's hashing scheme into the Kyber algorithm necessitates a thorough understanding of how the existing Keccak functions are utilized within Kyber. This step involves an in-depth study of the Kyber codebase to pinpoint all occurrences of Keccak/SHA3 and comprehend their specific roles in the algorithm's overall functioning.

Before replacing Keccak with Ascon, it is important to fully understand how Keccak operates within Kyber. This step involves:

4.3.1 Role of SHA3-256/512 and SHAKE128/256 in Kyber

To understand the role and usage of Keccak in Kyber , it is essential to first grasp the architecture of Kyber. This includes understanding how the key is generated, encapsulated, and decapsulated, and where Keccak is utilized within these processes.

In Kyber, key generation involves creating a master key that will be used across different layers of encryption. Keccak-based hash functions are crucial here, as they generate high-entropy keys by transforming random inputs into secure cryptographic keys. This ensures that the keys are both random and secure, providing a strong foundation for the encryption process. The process of generating a new public and private key pair begins by creating random seeds for the public matrix A, the secret s, and the noise e. This starts with generating an array of 32 random bytes, each ranging from 0 to 255. These random bytes are then hashed using SHA3-512, which produces a 64-byte digest that serves as the foundation for the key pair given in figure 4.2.

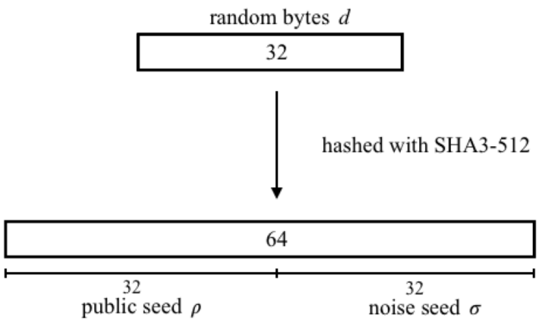

Figure 4.2: Generating random seeds [1]

The public seed is used to create the public matrix A. In Kyber-512, To generate each matrix entry, the public seed is hashed using SHAKE-128, combined with the index of the entry in the matrix (i, j). This array is then processed by a sampling function, which extracts or rejects values based on the byte array to generate a polynomial of length 256, which becomes the entry (i, j) in matrix A as given in figure 4.3.

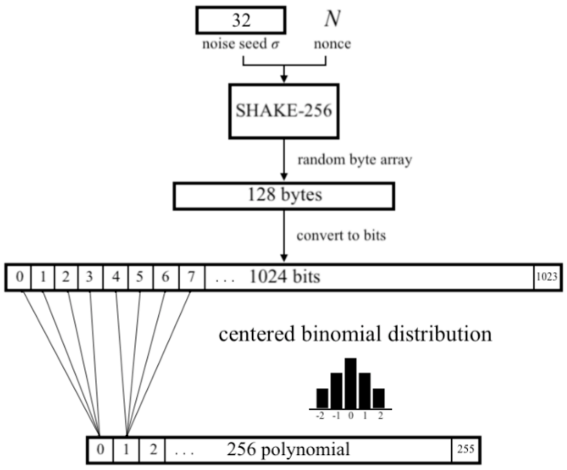
32
noise seed σ
N
nonce
SHAKE-256
random byte array
128 bytes
convert to bits
0 1 2 3 4 5 6 7 ... 1024 bits 1023
centered binomial distribution
-2 -1 0 1 2
0 1 2 ... 256 polynomial 255

Figure 4.3: Generating enteries of matrix [1]

After initializing matrix A, the secret vectors s and noise vectors e are sampled. This involves using **SHAKE-256**, which takes the noise seed and a nonce (an incrementing integer) as inputs. The output is a 128-byte array. This array is then converted into a 1024-bit array. Each 4-bit segment of this array is reduced to a single value, producing a 256-length output array. This method is applied to both the secret vector s and the noise vector e. as shown in the fig. 8c in [1] After this key computation is being perform using $A.s + e = t$. Then there are encapsulation and decapsulation steps elaborated in [1].

FO Transform is for CCA security and to acheive this hashes are being used. Fig 11a, 11b and 11c in [1] illustrate hashes usage in key genearation , encapsulation , decapsulation to make scheme CCA secure.

After understanding the role of Keccak in Kyber, next step is to identify the Keccak function in code and checking the Ascon hash function's compatibility with Keccak.

4.3.2 Locating Keccak/SHA3 Instances in the Code

By first examining the main components of the algorithm, such as key generation (`keypair`), encryption (`enc`), and decryption (`dec`), it becomes clear how Keccak functions contribute to the overall security and efficiency. This understanding can then guide the subsequent steps in identifying specific instances of Keccak within the codebase. With a clear understanding of how Kyber operates, the next step involves systematically identifying all instances of Keccak within the codebase. This process can be approached in the following way:

- **Codebase Exploration**: The Kyber codebase is explored using tools like grep to search for specific functions related to Keccak, such as `sha3_256()`, `sha3_512()`, `shake128()`, and `shake256()`. This work covers both header files and source files to ensure comprehensive identification. This approach allows for mapping out where and how these functions are utilized, offering a clearer picture of their role within Kyber.

- **Identifying Key Functions**: Algorithm utilizes Keccak's `sha3_256`, `sha3_512`, `shake128` and `shake256` functions.Functions such as `shake128_absorb_once()` and `shake128_squeezeblocks()` are of particular interest. Understanding the inputs and outputs of these functions is essential for effectively replacing them with ascon equivalents. The file with the name of `fips202.c` is written for handling the hashing and extendable-output functions that Kyber relies on. It includes the implementation of SHA3-256, SHA3-512, SHAKE128, and SHAKE256, all based on the Keccak family of functions. The file ensures that these operations meet the FIPS 202 standards, providing a solid foundation for the algorithm's security and performance.

- **Analyzing Dependencies**: A dependency analysis reveals how different parts of the Kyber codebase rely on Keccak. This analysis can trace the flow of data and show how the outputs of these hash functions influence other operations. This ensures that any changes made during the integration process will not compromise Kyber's integrity or security.

- **Documenting the Findings**: Each instance of Keccak/SHA3 usage is noted, including details such as function names, their locations within the code, and their specific roles in Kyber. This serves as a reference for the integration of Ascon, ensuring a smooth transition and maintaining the robustness of the algorithm. By carefully studying and documenting the Kyber codebase, the process of integrating Ascon can be carried out with confidence, ensuring that the algorithm remains secure and efficient.

4.4 Ascon Integration Plan

4.4.1 Compatibility Consideration

When integrating Ascon's hashing scheme into the Kyber algorithm, ensuring compatibility is a key step for successful implementation. Compatibility involves several critical aspects such as data formats, function interfaces, and overall system integration. Here's a closer look at these considerations

- **Data Formats**: It's important to ensure that the data formats used by Ascon's hashing functions match those required by Kyber. This means checking that the size and structure of inputs and outputs are consistent. Verify that any encoding or padding techniques employed by Ascon are compatible with Kyber's methods. For example, Ascon uses specific padding schemes to manage messages of varying lengths, and these need to be correctly handled to ensure seamless integration.
- **Function Interfaces**: Function interfaces define how different parts of the system interact. Ascon's functions should align with the API specifications expected by Kyber. This includes ensuring that function names, parameters, and return values are consistent with Kyber's existing code. Make sure that the conventions for calling Ascon's functions match those used in Kyber. This involves how functions are called and how data is passed between them.
- **Code Integration**: Integrating Ascon's code into Kyber involves to integrate Ascon's hashing functions into Kyber's code structure while maintaining organization and readability. This involves modifying existing files or adding new ones.
- **Testing and Validation**

  Conduct unit tests on Ascon's functions to confirm they work correctly on their own. This includes checking for accurate behavior and performance.

4.4.2 Choosing Appropriate Security Level

In the Kyber cryptographic system, the parameter k is crucial as it defines the structure and dimensions of matrix A. This matrix plays a fundamental role in the processes of Kyber's key generation and encryption.

When generating keys (secrect and public keys), a matrix A is constructed using a deterministic process that involves pseudorandom functions and transformations, using a seed that comes from an initial random seed. Matrix A is used during the encryption phase to transform a randomly chosen vector associated with the message under encryption into a format that combines it with the public key.

The lattice problem gets harder and the security is increased as the greater the value of k and the more rows the matrix A contains. Larger key sizes and higher computational costs, however, also result in slower processing and higher memory use. Resource efficiency and security are trade-offs [15]. Kyber accepts three k values. Every value has a corresponding security level,shown in table 4.1, denoted by a bits number such as 512, 786 or 1024.

Table 4.1. Kyber's Security Levels

	k	**Security Levels**
Kyber512	2	1 (AES-128)
Kyber768	3	3 (AES-192)
Kyber1024	4	5 (AES-256)

The "512" in Kyber512, in contrast to the "2048" in RSA-2048, does not imply that the bit length is 512 bits. The security level is represented by an arbitrary number. The true bit length is substantially longer. The public key for Kyber512 [15] is 5888 bits. Because Kyber768 provides a fair trade-off between computing economy and cryptographic strength, the document [15, 7] suggests utilizing it for the majority of applications.

However, the stated equivalent security bits are only valid for the Kyber that is using SHA3 hashed. Ascon Hash could constitute a weak point in the security chain that includes the hashing algorithm and Kyber if it is less safe than SHA3 Hash. It is said that a chain's strength is defined by its weakest link. Comparing the official documents' requirements, it seems that Ascon-Hash is in fact less reliable than the two most often used SHA3 configurations, SHA3-256 and SHA3-512. "Collision resistance" and "second pre-image resistance" are the metrics that is selected for comparison, shown in table 4.2 [4, 3].

- "**collision resistance**" estimates the probability that two different input produce the same hash output, stressing the overall resilience of the hash function against all potential inputs.

- "**second pre-image resistance**" highlights the defense against deliberate data falsification by measuring the likelihood of discovering a different input that generates the same hash as a previously known input.

Table 4.2. Comparison between Ascon's Hash and Keccak/SHA3 Security

	Collision Resistant	**Pre-image Resistant**
SHA3-256	128 bits	256 bits
SHA3-512	256 bits	256 bits
Ascon-Hash	128 bits	128 bits

Ascon-Hash only offers 128 bits of security, as can be seen in the table, making it weaker than SHA3-256 and SHA3-512. As a result, substituting Ascon-Hash for the SHA3 hash would jeopardize the 192 bits of security offered by Kyber768. Kyber768 would become no more secure than Kyber512 with the replacement. Consequently, it appears that Kyber512 would make a better foundation for Kyber light. Ascon-Hash will not slow down Kyber512's 128 bits of security, but it would be much lighter than Kyber768, making it better suited for low-powered devices.

4.4.3 Determining the Equivalent Ascon Hash Function

To effectively replace Keccak/SHA3, it is essential to determine the Ascon hash function that corresponds to the specific roles of Keccak/SHA3 within the Kyber algorithm. This involves matching the functions based on their cryptographic properties, such as output length, security levels, and performance.

The process begins by analyzing the specific uses of Keccak/SHA3 identified in the previous step, including functions like `sha3_256()`, `sha3_512()`, `shake128()`, and `shake256()`. For each of these, the equivalent Ascon function is selected based on its ability to fulfill the same cryptographic requirements while maintaining or improving performance in lightweight environments.

For example, Ascon-XOF can serve as a suitable replacement for SHAKE128 and SHAKE256, providing flexible output lengths and robust security features. Similarly, Ascon-Hash may be chosen to replace SHA3-256 and SHA3-512, ensuring that the overall integrity and security of the Kyber algorithm remain intact.

By carefully selecting and implementing these Ascon's hash functions, the transition from Keccak to Ascon is achieved without compromising the algorithm's effectiveness, particularly in the context of quantum-resistant cryptography.

4.5 Replacing Keccak/SHA3 with Ascon

The final step involves replacing the identified Keccak/SHA3 functions with the selected Ascon hash functions within the Kyber algorithm. This step is critical as it requires careful modification and updating of the codebase to ensure that the integration is seamless and maintains the algorithm's integrity.

4.5.1 Replacing the Functions

Once the appropriate Ascon functions have been chosen, the existing Keccak/SHA3 functions in Kyber are replaced with these Ascon equivalents. This replacement process includes adjusting and updating all supporting function calls to ensure that they are compatible with the new Ascon-based operations. Care is taken to maintain the original flow and structure of the Kyber algorithm, making sure that the new functions integrate smoothly without introducing errors or vulnerabilities.

Modify the Kyber codebase to replace SHA3-256 with Ascon-Hash, and SHAKE256 with Ascon-XOF. SHAKE128 absord and squeeze function are replaced with Ascon's absorb and squeeze functions. This requires adjusting the functions used for key generation and encapsulation/de-capsulation. To Integrate Ascon's absorb and squeeze functions into Kyber, ensuring the same structure used by Keccak is followed to maintain compatibility with Kyber's lattice-based opera- tions. Since Kyber512 is the preferred security level. The definition found in *params.h* sets this. Table 4.3 illustrates the mapping between the original Keccak hash functions used in Kyber and their corresponding replacements with Ascon hash functions.

Table 4.3. Ascon Hash Functions Replacing Keccak in Kyber

Keccak Functions	**Ascon Hash Functions**
sha3_256()	ascon_hash32() (32 bytes output)
sha3_512()	ascon_hash64() (64 bytes output)
shake256()	ascon_xof()
shake128_absorbonce()	ascon_absorb()
shake128_squeezeblocks()	ascon_squeeze()

Five hashing functions found in *fips202.h* are used by Kyber reference implementation that are `sha3_256()`, `sha3_512()`, `shake256()`, `shake128\_absorbonce()`, `shake128\_squeezeblocks()`. There are two SHA3 fixed-length output functions: `sha3_512()` with output length 512 bits and `sha3_256()` with output length 256 bits. `shake256()` offers a security level equal to a 256-bit hash algorithm by utilizing the SHAKE variation of SHA3, which allows variable length output. SHAKE256, SHAKE128 is another XOF from the SHA3 family. The `shake128_absorbonce()` method is invoked to absorb the input into internal states before the standard `shake128()` function is called, following the buffer allocation and internal state initialization. The output from internal states is then squeezed by calling `shake128_squeezeblocks()`. Either one of the subroutines is only ever invoked once. As an alternative, we can call each of these subroutines separately. Once `shake128_absorbonce()` has been called, `shake128_squeezeblocks()` can be called several times. In the event that the initial round of output is rejected, this feature would be helpful. In this scenario, the application might keep refreshing the output by calling `shake128_squeezeblocks()` until the output is approved. If the full `shake128()` function is used instead, it would unnecessarily absorb the input for multiple time, thus reducing efficiency.

There is just one fixed-length output function available in the reference code from the ascon-c library [59], the Ascon-Hash. In order to replace all of the five SHA3 functions utilized by Kyber, the reference Ascon codes are altered to match the five functions. The output length definition is eliminated to build an XOF function. `ascon_xof()` replaces the `shake256()`. `ascon_hash()` replaces `sha3-256()` and `sha3-512()`. Fixed-length outputs of 256 bits and 512 bits are produced by setting output length parameter value to 32 bytes or 64 bytes.

By separating the initialization state, input absorb, and output squeeze phases into three independent functions, the functions `ascon_inithash()`, `ascon_absorb()` and `ascon_squeeze()` are created respectively. These functions uses in place of `shake128_absorbonce()` and `shake128_squeezeblocks()`. These all six functions are defined in *ascon.h* (a file replaces the *fips202.h*). Finally, Ascon-based Kyber is generated by substituting equivalent calls to Ascon's hash functions for any SHA3 or SHAKE function calls.

Additionally, by changing the value of Ascon_Hash_Rounds from 12 to 8, Ascon-Hash is converted into Ascon-Hasha, which is another hashing algorithm from the Ascon family. Ascon-Hasha is an even lighter algorithm than Ascon-Hash. It uses 8 rounds of permutation instead of 12 round, sacrificing some security for faster speed. Ascon-Hasha and Ascon-XOFa can also be used instead of Ascon-Hash and Ascon-XOF respectively.

4.6 Testing and Benchmarking

After successfully replacing, it is essential to conduct a thorough testing and benchmarking process. This phase ensures that the integration of Ascon maintains the expected functionality and performance of Kyber while adhering to security standards. Both unit and integration testing are critical to verifying that the system works as intended, while benchmarking provides insights into the performance impact of this modification

4.6.1 Unit Testing

After the replacement, the new Ascon functions undergo rigorous unit testing to verify that they perform as expected. These tests focus on ensuring that each function produces correct outputs, handles edge cases properly, and meets the required security standards. Unit testing focuses on verifying that individual functions, particularly those affected by the replacement of Keccak with Ascon, work correctly in isolation. The primary aim is to confirm that the Ascon-based implementation produces the intended results for hashing operations, key generation, encryption, and decryption. The following steps outline the test procedure:

1. The first step in unit testing is to ensure that the newly integrated Ascon hash functions are operating as expected. This includes verifying that the outputs from functions like `ascon_hash()`, `ascon_xof()`, `ascon_absorb()` and `ascon_squeeze()` match the expected values for given inputs. Since Ascon differs from Keccak, it's important to validate that the absorbed and squeezed states behave according to the Ascon specification.

2. The key generation process is tested to verify that the public-private key pairs generated are valid and usable. Since the hashing function plays a role in key generation, the new Ascon-based implementation must produce keys that meet Kyber's cryptographic requirements.

3. Unit tests are conducted on the encryption function to confirm that the ciphertext generated from a given plaintext is consistent with the expected output. A sample plaintext message is encrypted using the generated public key. The ciphertext is compared to the plaintext to ensure they differ, validating that the encryption process is functioning as intended. Additionally, multiple encryptions of the same message are performed to confirm that different ciphertexts are produced due to the randomness in encryption.

4. Decryption is tested by ensuring that encrypted messages can be decrypted back to their original form. This verifies that the Ascon-based encryption process is reversible and that no errors have been introduced during the integration of the hash function.

5. Tests are also conducted for edge cases, such as encrypting and decrypting an empty message or a message of maximum allowable size. Any alterations in the ciphertext or the use of incorrect private keys should result in a decryption failure, indicating the robustness of the algorithm against manipulation.

4.6.2 Integration Testing

After individual components are verified through unit testing, integration testing ensures that all parts of the system work seamlessly together with the Ascon hash functions. Integration testing is critical to confirming that the new hash function correctly interacts with Kyber's key generation, encryption, and decryption functions. Any issues that arise during this phase are addressed promptly to ensure a smooth transition. The main focus of integration testing is to validate the complete cryptographic flow. In this scenario, the public and private keys are generated, a message is encrypted using the public key, and then decrypted with the private key. The decrypted message is compared with the original plaintext to ensure the entire process is functioning correctly with Ascon.

Kyber's codebase provides test cases that assess the key generation, encryption, and decryption functions individually. These tests are essential, but for integration testing, they must be executed in combination to ensure the entire cryptographic process works smoothly. For example, generating a key pair, encrypting a message, and then decrypting it should yield the original message, validating that these components interact without errors.

As with unit testing, integration tests also cover edge cases, such as encrypting an empty message or altering the ciphertext before decryption. These tests confirm that the system continues to operate securely and accurately when confronted with extreme or abnormal inputs. Some test cases in the Kyber codebase also evaluate performance, measuring the time and resources consumed by the key generation, encryption, and decryption processes. While primarily used for benchmarking, these tests help confirm that the cryptographic operations perform efficiently and that there are no significant slowdowns or memory issues when the components are integrated.

4.6.3 Benchmarking

Benchmarking is vital to understanding the performance implications of replacing Keccak with Ascon. This involves measuring the time and resource consumption of key operations—key generation, encryption, and decryption—and comparing them to the original Keccak/SHA3-based implementation.

- **Performance Metrics**: Benchmarks are run to measure the time required for key generation, encryption, and decryption. These metrics allow us to compare the computational efficiency of Ascon against Keccak/SHA3. If Ascon performs faster, it could enhance the overall efficiency of Kyber, especially in resource-constrained environments.

- **Memory Usage**: Alongside performance, it is important to measure the memory footprint of the Ascon-based Kyber implementation. This includes assessing how much memory is consumed during the cryptographic processes and whether Ascon requires more or less memory compared to Keccak/SHA3.

- **Security Trade-offs**: While benchmarking, it's important to consider any potential trade-offs between performance and security. Although Ascon is a lightweight hash function, it is still designed to meet strict security requirements. Benchmarks help confirm that the system remains secure without compromising on performance, particularly under high-stress or large-scale use cases.

By systematically replacing, testing, and evaluating the new Ascon functions within Kyber, the effectiveness of the integration can be validated, ensuring that the modified algorithm not only maintains but potentially enhances its performance and security in a post-quantum cryptographic context.

4.7 Summary

The integration of Ascon's hashing scheme into the Kyber algorithm is a carefully structured process designed to enhance the algorithm's security and efficiency, particularly in the context of quantum-resistant cryptography. The methodology follows a logical sequence of steps, each contributing to a smooth and effective transition from the existing Keccak/SHA3 functions to Ascon.

The process begins with establishing a robust development environment, ensuring all necessary tools and resources are in place to support the integration effort. Understanding the inner workings of Kyber and identifying the precise usage of Keccak/SHA3 within the codebase sets the foundation for replacing these functions with Ascon. The selection of Ascon as the lightweight hash function is driven by its proven efficiency, NIST standardization, and suitability for quantum-resistant cryptography.

Once the appropriate Ascon functions are selected, they are seamlessly integrated into the Kyber algorithm, with adjustments made to support function calls and ensure compatibility. Rigorous unit testing and integration testing are conducted to verify the correctness and security of the new implementation. Finally, a performance comparison between the Ascon-based implementation and the original Keccak/SHA3-based version provides valuable insights into the effectiveness of the integration.

By following this comprehensive methodology, the modified Kyber algorithm is expected to maintain, if not enhance, its performance and security, making it a robust choice for post-quantum cryptographic applications. The careful planning and execution of each step ensure that the transition to Ascon is not only successful but also brings measurable improvements in the algorithm's overall functionality.

Chapter 5

Results and Analysis

5.1 Introduction

In this chapter, the findings from the implementation of Ascon as a hashing function within the Kyber framework, replacing Keccak, will be discussed. The focus will be on the comparison of performance metrics, security implications, and overall system behavior. These results will be analyzed in the context of the efficiency and security of post-quantum cryptographic systems, particularly KEM.

5.1.1 Test Environment

The benchmarking tests were conducted using a virtual machine running Ubuntu 22.04 on the user's personal laptop windows 10 as a host OS with the following specification:

- **Processor**: Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz (2.71 GHz)
- **System Type**: 64-bit operating system, x64-based processor

While this system provides a reasonable test environment, it is important to note that it is not specifically optimized for lightweight cryptographic performance testing. Lightweight algorithms like Ascon are designed to perform efficiently on resource-constrained devices such as IoT nodes, embedded systems, or low-power microcontrollers. As a result, the observed performance on this relatively more powerful PC does not fully represent the algorithm's behavior in its intended deployment environment. However, these tests provide a comparative baseline for performance when replacing Keccak with Ascon in Kyber.

5.2 Benchmarking Paramemters

Despite the hardware limitation for lightweight cryptogrpahy, we measured the following performance indicators to evaluate the impact of replacing Keccak with Ascon:

- **Hashing Function Efficiency**: The speed performance of Keccak versus Ascon.
- **Performance Evaluation**: The time taken in key generation, encryption, and decryption by ML-KEM before and after Ascon integration, and then the performance comparison of both of them. Performance is evaluated by generating both the average and median of CPU cycles required for each function, based on 1000 executions.
- **Security Impact**: Ensuring no degradation in security from the hashing function switch.

5.3 Hashing Function Efficiency

5.3.1 Ascon vs. Keccak: Speed Comparison

Table 5.1 demonstrates the time of running 1000 times each hash function of Ascon and Keccak.

Table 5.1. Ascon vs Keccak - Speed Comparison

Output Size (Bytes)	Ascon Time (ms)		Keccak Time (ms)	
32	Ascon_Hash	0.007	SHA3-256	0.003
64	Ascon_Hash	0.018	SHA3-512	0.003
256	Ascon_XOF	0.078	SHAKE256	0.005
64	Ascon_squeeze	0.017	SHAKE128_sqeezeblock	0.003

While both Keccak and Ascon perform well on this system, Ascon's hash function shows to take even more time by running 1000 times each hash function of Ascon and Keccak. As the i5-7300U processor is more powerful than the target platforms for lightweight algorithms, such as embedded systems. Therefore, the speed advantage of Ascon may appear smaller or even none here compared to what would be observed on constrained devices.

5.4 Performance Evaluation

In the evaluation of cryptographic systems, performance is a critical metric, particularly when considering the efficiency of KEMs like ML-KEM. The following section presents a comprehensive analysis of the performance impact of replacing the Keccak hash function with Ascon in ML-KEM. The benchmarking tests were conducted using a virtual machine running Ubuntu on the user's personal laptop. The tests measured the CPU cycles taken by key cryptographic operations in ML-KEM, specifically focusing on key generation, encryption and decryption. Performance is evaluated by generating both the average and median of CPU cycles required for each function, based on 1000 executions for both ML-KEM with Keccak and ML-KEM with Ascon. This approach provides a robust measure of computational efficiency by accounting for variations in speed based on cycles and ensuring a reliable assessment.

5.4.1 Results

The performance tests measured CPU cycles for each function, capturing both average and median values and presents the performance comparison of each function. Table 5.2 displays the performance comparison of ML-KEM and ML-KEM with Ascon representing the number of cycles consumed by key functions such as key generation, encapsulation, and decapsulation.

Table 5.2. Performance Comparison of Key Functions in ML-KEM and ML-KEM with Ascon, Measured in CPU Cycles (cycles/ticks)

Functions	**ML-KEM**		**ML-KEM with Ascon**	
	Median	**Average**	**Median**	**Average**
indcpa_keypair	230674	523311	277915	404683
indcpa_enc	315314	529984	360146	376593
indcpa_dec	113372	151813	89074	97222
kyber_keypair_derand	246952	498159	272416	318649
kyber_keypair	251431	374747	299732	339419
kyber_encaps_derand	333462	450195	361228	376708
kyber_encaps	337340	435473	296764	340409
kyber_decaps	437497	606800	412364	466087

5.4.1.1 Key Generation

There are three main keypair functions used in keygeneraion of ML-KEM:

`indcpa_keypair`: The function responsible for generating public and private key pairs in the Invariant CPA (INCPA) scheme.

`keypair`: The function for generating key pairs for the main key exchange algorithm.

`keypair_derand`: The function used to derandomize key pairs for enhanced security.

Graph 5.1 presents performance comparison for keypair in ML-KEM and ML-KEM with Ascon.

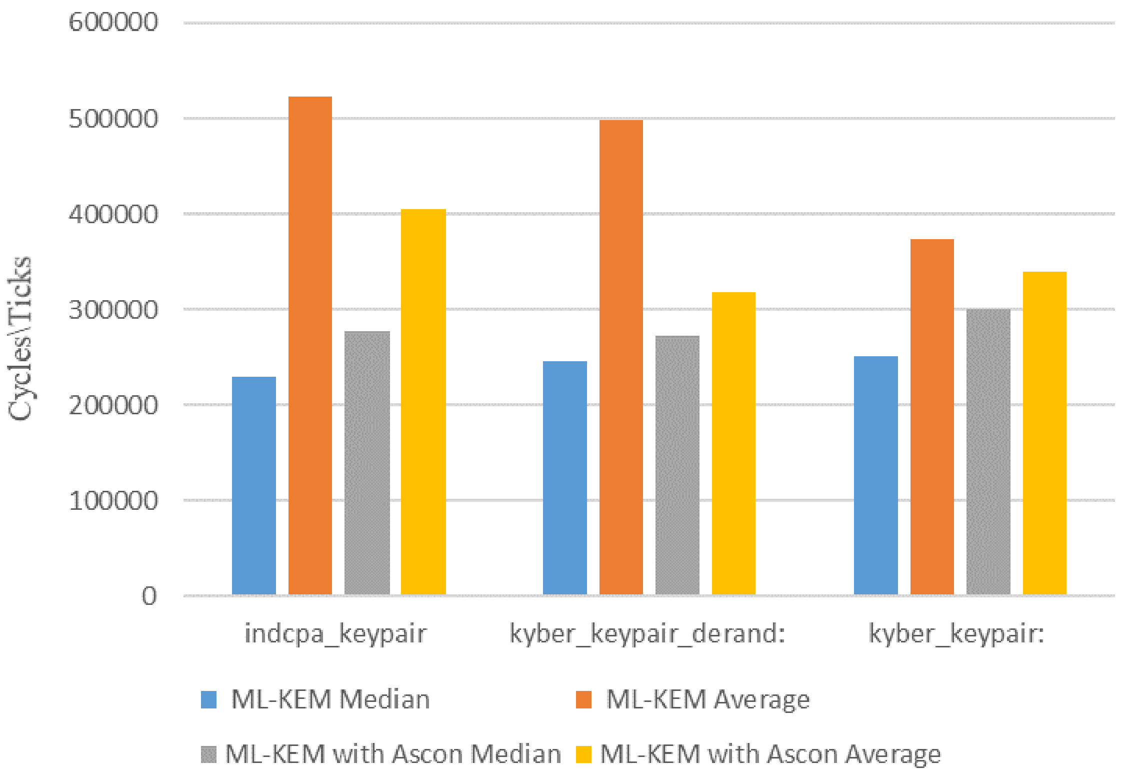

Figure 5.1: Performance comparison for key_generation

The ML-KEM with Ascon implementation showed a slight reduction in both average and median CPU cycles compared to the ML-KEM having Keccak. This efficiency is due to Ascon's simplified internal operations and fewer rounds.

5.4.1.2 Encapsulation

`indcpa_enc`: The function responsible for encryption in the Invariant CPA (INCPA) scheme. This function is critical for securely encrypting messages in the ML-KEM framework.

`encaps`: This function encapsulates a secret key in the main KEM. It is a core operation in the encryption process of ML-KEM.

`encaps_derand`: The function used to derandomize the encapsulation process, ensuring that the encapsulated keys are uniformly distributed and secure. Graph 5.2 illustrates the performance comparison of encapsulation processes in ML-KEM and ML-KEM with Ascon, including standard encapsulation, encapsulation with derandomization, and indcpa-encryption.

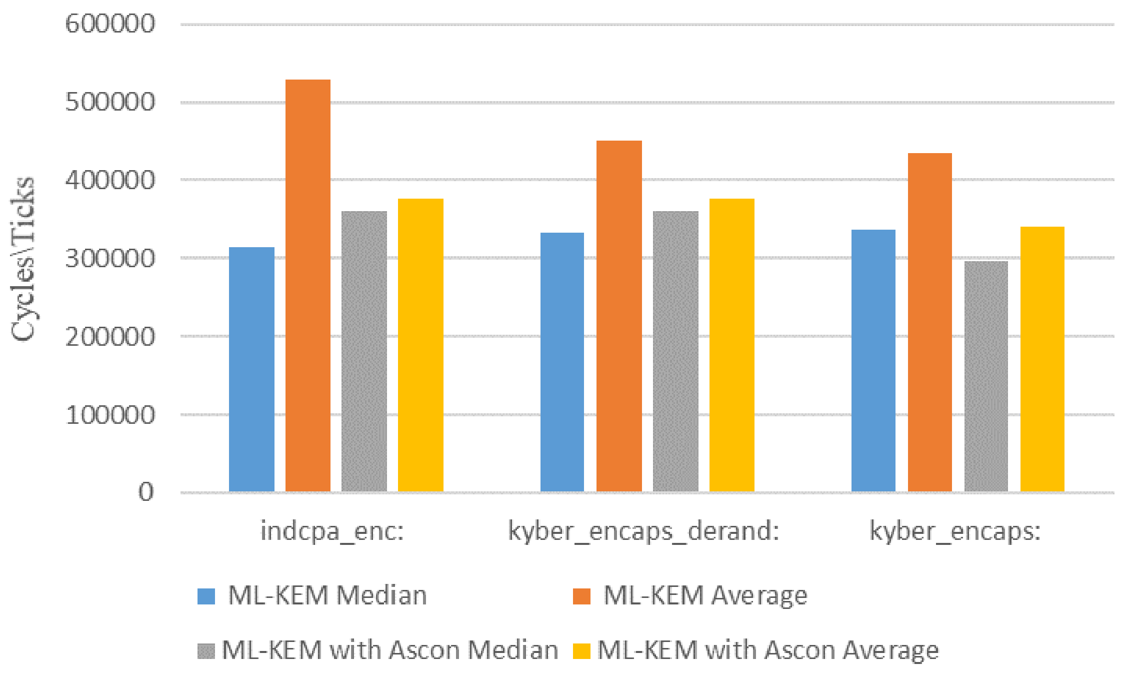

Figure 5.2: Performance comparison for encapsulation

Encryption operations with Ascon were more efficient, with lower average and median CPU cycles observed. This improvement reflects Ascon's lower computational overhead.

5.4.1.3 Decapsulation

`indcpa\_decaps`: This function is responsible for decrypting messages in the INCPA scheme. The comparison between Keccak and Ascon highlights the efficiency gains in decryption operations when using Ascon.

`kyber\_decaps`: The decapsulation function completes the key exchange process by recovering the secret key. The performance comparison illustrates how the use of Ascon affects the speed and efficiency of decapsulation in ML-KEM.

Comparison of separate functions of decapsulations in ML-KEM and ML-KEM with Ascon is illustrated in graphs 5.3.

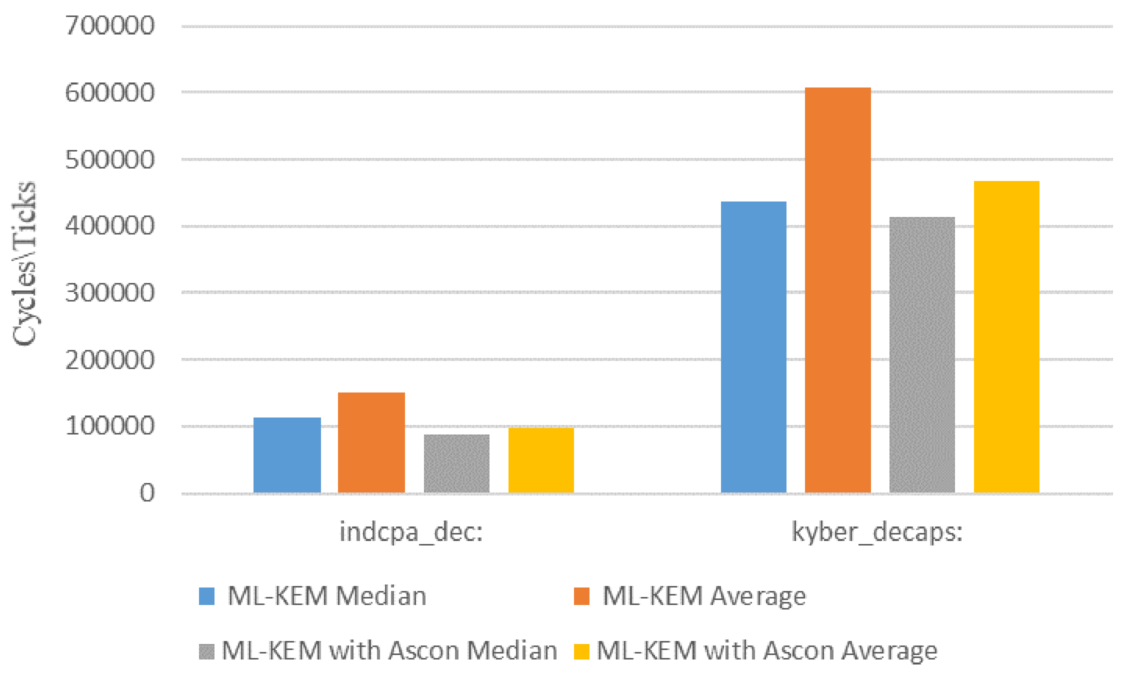

Figure 5.3: Performance comparison for decapsulation

Decryption times also improved with Ascon, demonstrating faster average and median CPU cycles relative to Keccak.

The following section presents an overall analysis of the performance impact of replacing the Keccak hash function with Ascon in ML-KEM. The analysis is based on the number of CPU cycles consumed by various cryptographic functions in both implementations, providing a clear understanding of how the replacement affects overall system efficiency.

5.4.2 Overall ML-KEM Performance

The initial performance evaluation focuses on the standard implementation of ML-KEM, which utilizes the Keccak hash function. A graph 5.4 representing the number of cpu cycles consumed by all functions used in ML-KEM is displayed. The key findings from this analysis reveal the computational overhead introduced by the Keccak-based implementation, particularly in functions that heavily rely on hash operations. The breakdown of cycle consumption allows us to identify potential bottlenecks, with the hash-intensive steps, such as pseudorandom function generation, key encapsulation and key decapsulation, showing the highest cycle counts.

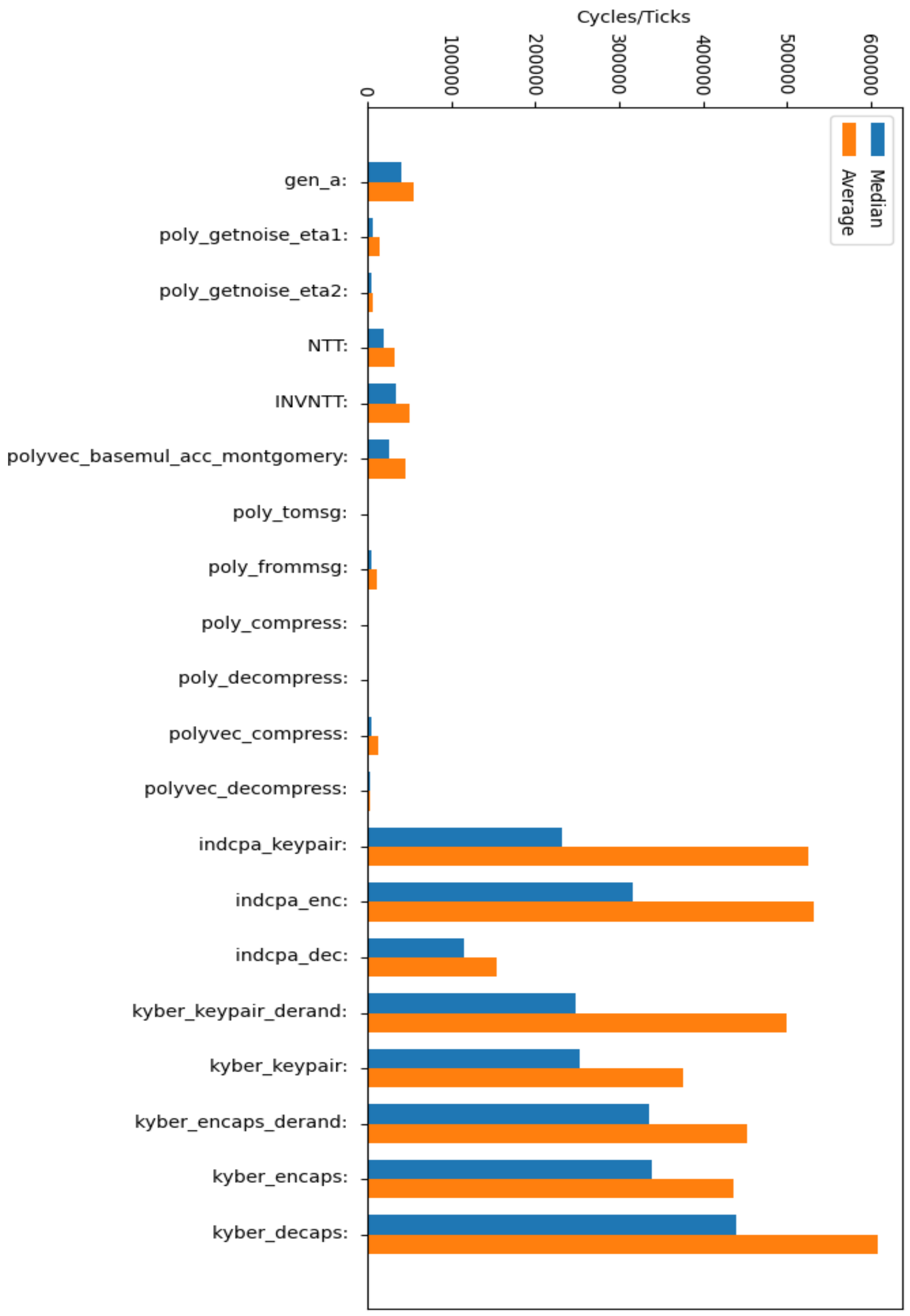

Figure 5.4: Performance of ML-KEM

5.4.3 Overall ML-KEM with Ascon Performance

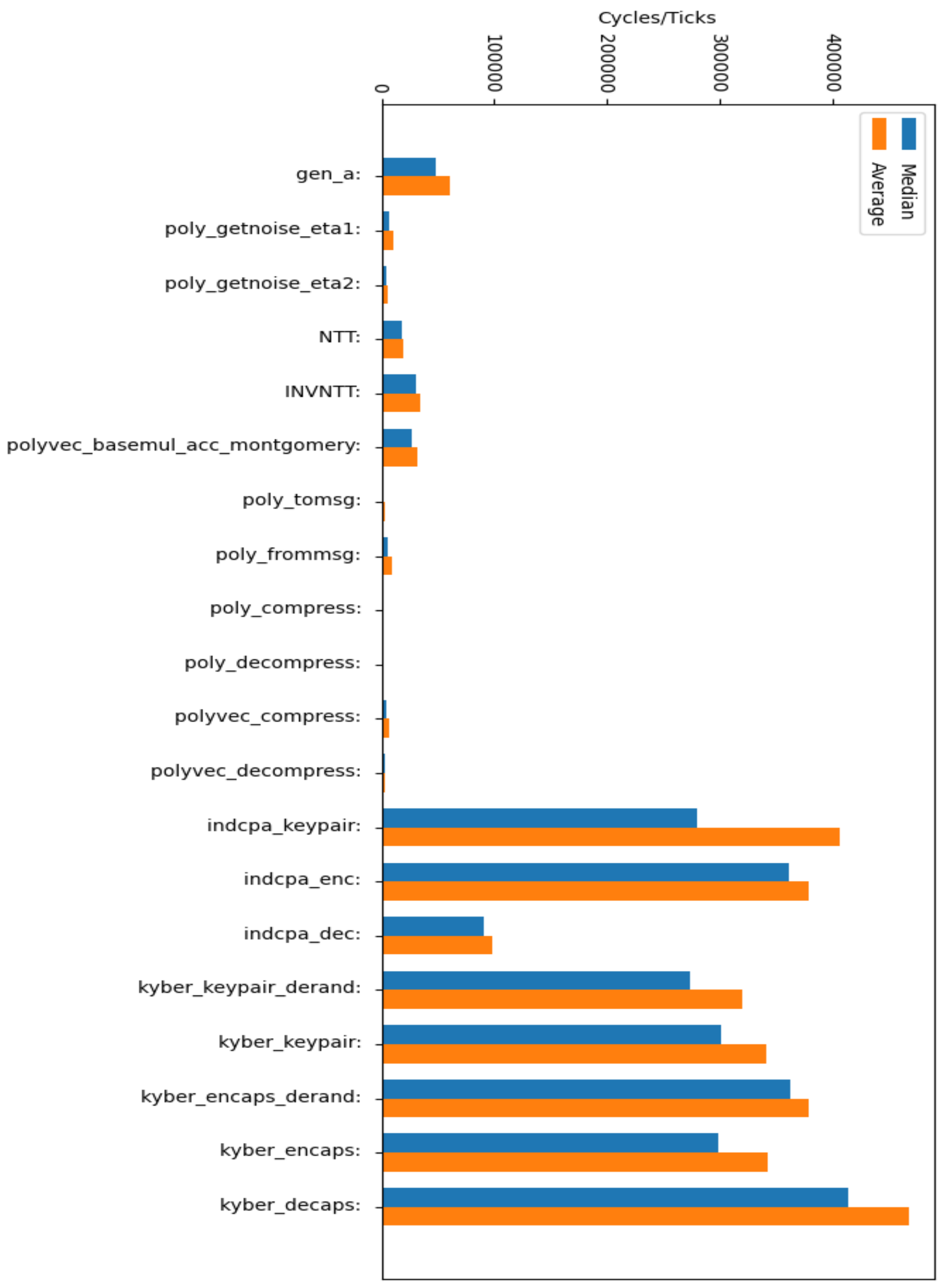

Figure 5.5: Performance of ML-KEM with Ascon

A graph 5.5 representing the number of cpu cycles consumed by all functions used in ML-KEM with ascon is displayed. The key findings from this analysis reveal the reduction in the number of cycles across various functions. particularly in functions that heavily rely on hash operations.

5.4.4 Performance Comparison Between ML-KEM and ML-KEM with Ascon

Nonetheless, Ascon's reduced time, even on this environment, highlights its computational efficiency, which should translate to greater performance gains in low-power environments.

5.5 Analysis of Matrix Generation Cycles in ML-KEM Based on Ascon

In the context of ML-KEM with Ascon, a notable observation is that the matrix generation phase requires more computational cycles compared to the Keccak-based implementation, as can see in graph 5.6. This increase in cycles is largely attributed to a mismatch between the hash rate of SHAKE-128, used in the Keccak-based implementation, and the hash rate of Ascon-Hash. Understanding this discrepancy is essential for optimizing the performance of the Ascon-based ML-KEM system.

IThe matrix generation in ML-KEM is a critical step, as it directly impacts the key generation and encryption processes. Specifically, in the gen_a (used to generate matrix A) method found in the indcpa.c file, a buffer is filled with pseudorandom values. These values are obtained by repeatedly calling the "squeeze" function of the hash algorithm in use. The squeeze function outputs a portion of the hash, which is used to generate the elements of the matrix.

In Keccak-based ML-KEM, SHAKE-128 is the primary hash function used for this purpose. SHAKE-128, being a sponge-based hashing algorithm, has an internal state size known as its "hash rate," which determines how much data can be produced in a single squeeze. SHAKE-128 has a hash rate of 168 bytes, meaning each squeeze operation produces 168 bytes of pseudorandom data. When generating matrix A, the buffer is filled with these values in one operation, making the process relatively efficient.

5.5.1 Hash Rate Mismatch with Ascon-Hash

Ascon-Hash, the lightweight hash function replacing SHAKE-128 in the Ascon-based ML-KEM, is also a sponge-based function. However, the key difference lies in its hash

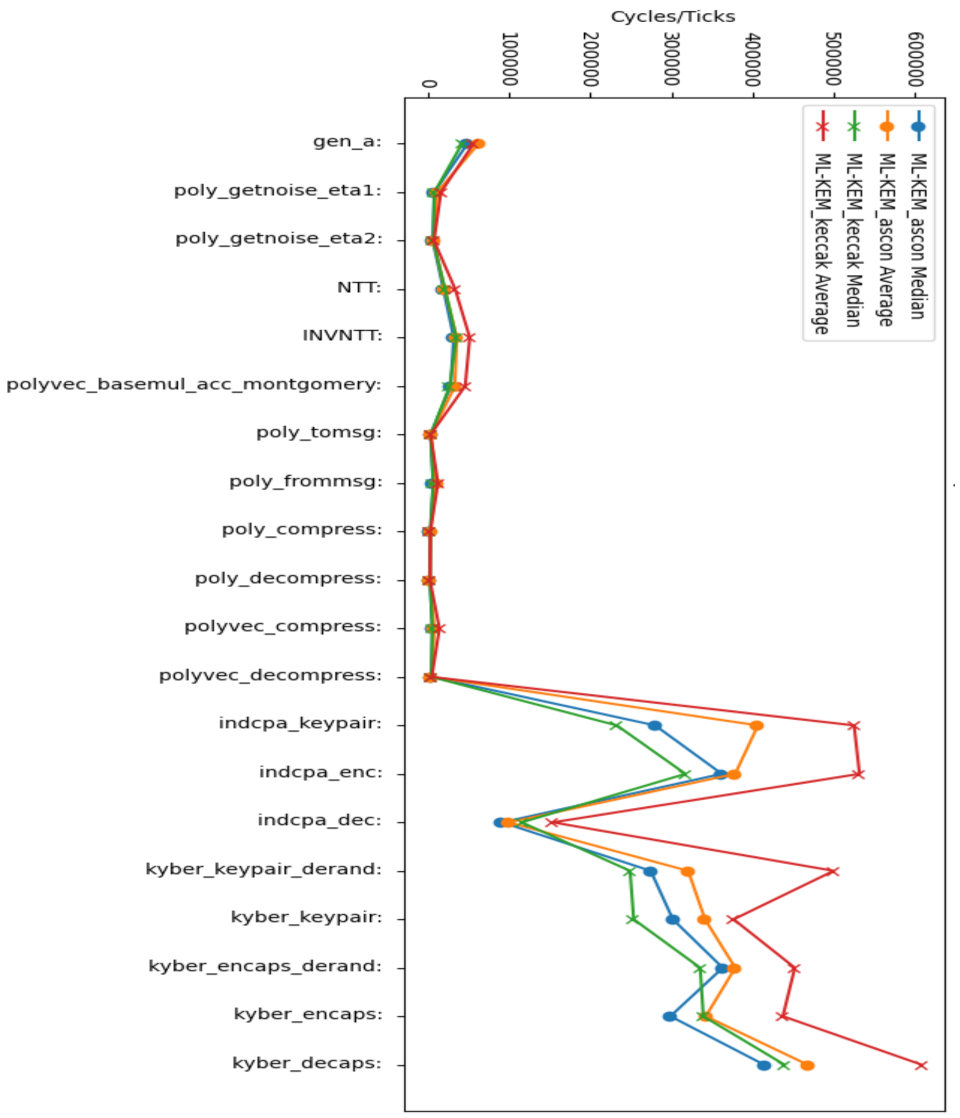

Figure 5.6: Performance comparison between ML-KEM and ML-KEM with Ascon

rate. The hash rate of Ascon-Hash is only 8 bytes, significantly smaller than the 168-byte hash rate of SHAKE-128. This discrepancy has a direct impact on the efficiency of matrix generation.

When the squeeze length remains at 168 bytes (as it is in the SHAKE-128-based implementation), Ascon-Hash requires multiple squeeze operations to fill the buffer. Specifically, it takes 21 squeezes to produce the same amount of data that SHAKE-128 can produce in a single squeeze. This additional overhead leads to a substantial increase in the number of cycles required for matrix generation, thus contributing to the observed performance degradation.

5.5.2 Impact of Squeeze Length on Efficiency

The efficiency of the squeeze operation is closely tied to the match between the squeeze length and the hash rate of the underlying hash function. In SHAKE-128, the squeeze length of 168 bytes is ideal because it perfectly matches the hash rate, resulting in minimal cycle overhead. However, in the case of Ascon-Hash, maintaining a squeeze length of 168 bytes is inefficient due to the much lower 8-byte hash rate. As a result, the squeeze operation must be repeated multiple times to generate the necessary data, leading to increased cycle consumption.

To optimize this process, the squeeze length should be adjusted to better match the 8-byte hash rate of Ascon-Hash. Ideally, setting the squeeze length to 8 bytes would minimize the number of squeezes required and reduce the computational overhead. However, there is a tradeoff involved. A smaller squeeze length, while more efficient in terms of hashing operations, may not provide sufficient entropy for certain cryptographic processes, such as rejection sampling. One way to decide the most ideal squeeze length, is by comparing benchmark results of ascon based ML-KEM with varying squeeze length, from 8 bytes to 168 bytes.

5.6 Comparison Analysis

Lightweight cryptography, like Ascon, is designed to be highly efficient on devices that don't have much processing power or memory—such as embedded systems, small sensors, or IoT devices. However, when we run these same algorithms on powerful machines like your personal laptop, which has a much stronger processor (Intel i5-7300U @ 2.60 GHz), we don't see the same advantages.

Intel processor is optimized for heavy lifting—it can handle tasks that need lots of resources and perform them quickly. In contrast, Ascon is built to minimize the use of resources, making it ideal for smaller, less powerful devices. This is why, when Ascon hashes are tested on laptop against Keccak (a more resource-intensive algorithm), the later performed better in terms of speed. Keccak executed faster because it can fully utilize the high-end processor's capabilities. Meanwhile, Ascon, designed to save resources, has certain optimizations that just don't make a difference on a machine with this much processing power.

Ascon's real strength is exposed when it is deployed on embedded systems, that is, smaller, resource-constrained devices like micro controllers /ARM processors. On those devices, the performance results generally flip. Embedded systems usually have slower processors, with very limited memory, and often run on batteries. In these scenarios, Ascon would be far more efficient than Keccak, as it is designed to work in environments where every byte of memory, every CPU cycle, and every watt of energy matters. On a tiny, low-power device, Ascon would outperform Keccak because Keccak is too resource-heavy for such systems. It uses more memory and more power, which could be a problem for devices that have strict limitations.

When Keccak is replaced with Ascon in Kyber/ML-KEM, there wasn't a significant difference in the number of CPU cycles used for encryption, decryption, and key generation. This is because the bulk of Kyber's work involves other complex operations (like matrix calculations), so the hashing function—whether Keccak or Ascon—doesn't dominate the processing time. On powerful systems, these differences in lightweight vs. regular cryptography just don't show up as clearly. Also On a high-performance machine like laptop running intel processor, the cycle count differences between Ascon and Keccak would be relatively small, as both functions run faster than what's typically measurable at the algorithmic level (especially since Ascon is not particularly optimized for this environment). This is the reason we have not observed a marked difference in cycle counts.

In recent work on the integration of Ascon with Kyber, studies have highlighted the performance advantages of using Ascon as a lightweight alternative to Keccak, particularly in the context of embedded systems. There are the following recent work that demonstrate that

- In [60], a study demonstrated that Kyber-Ascon, when implemented on systems using the RISC-V instruction set, showed significant improvements in memory efficiency. Specifically, Kyber-Ascon achieved a 9% improvement in memory footprint in software and a 10% improvement with Instruction Set Extensions (ISE), which indicates its potential for better memory management in low-complexity embedded environments. The study further explores the balance between performance, energy consumption, and memory usage, showing that Kyber-Ascon is particularly well-suited for applications where memory usage is the primary concern [60]. This research reinforces the notion that Ascon has a lightweight design, while it may not show dramatic performance gains on high-performance systems like laptops or desktops, becomes highly effective in resource-constrained embedded systems. The ability to optimize Kyber for memory efficiency, without significantly compromising performance, makes Ascon a valuable option in scenarios where memory and energy resources are at a premium, such as in IoT devices or low-power microcontrollers.

- In another recent work of integration of Ascon with Kyber undertaken on hardware-based platform, i.e., on Xilinx Virtex-7 FPGA also re-inforces the same notion [61]. In this work, the scheme was designed from scratch and iteratively optimized, with Ascon being used as a lightweight XOF [61]. This represents the first implementation of its kind, further validating Ascon's practicality in lightweight, hardware-based post-quantum cryptographic applications. This design provides a 100 bit of post quantum security and shows a remarkable 3 times improvement in terms of hardware area utilization with respect to the state-of-the-art Kyber KEM standard implementation [61].

Both of these studies highlight the importance of Ascon as a lightweight cryptographic algorithm when used in conjunction with Kyber. The findings illustrate that, although Kyber-Ascon may not provide substantial gains on high-performance machines (as evidenced by the CPU cycle results on a standard laptop), it offers clear advantages in memory efficiency and adaptability when deployed on specialized, resource-constrained hardware platforms. These results not only support the suitability of Ascon for embedded PQC schemes but also provide concrete evidence of its effectiveness in real-world applications, reinforcing its role as a lightweight and efficient solution in PQC.

5.7 Security Analysis of the Ascon-based Kyber Implementation

One of the core aspects of ensuring the robustness of the Ascon-based Kyber implementation is maintaining the security properties, particularly in the context of collision resistance and quantum-resistant features. Kyber is a post-quantum cryptographic scheme, and the integrity of its lattice-based design, especially related to the LWE problem, must remain intact despite the replacement of the hash function. Below are the key considerations and confirmations regarding security:

5.7.1 128-bit Security Against Collision Attacks

Ascon, as a lightweight cryptographic algorithm, has been designed with a focus on efficiency without compromising security. Specifically, the Ascon-XOF and Ascon-Hash functions offer 128-bit security against both collision attacks and second pre-image attacks. This security level is consistent with the requirements of Kyber, which is also designed to provide 128-bit post-quantum security. Thus, the replacement of Keccak/SHA3 with Ascon ensures that Kyber's resistance to collision-based vulnerabilities remains intact.

For the cryptographic operations in Kyber, the use of a 128-bit secure hash function is critical in preventing collisions—where two different inputs produce the same hash output—which could compromise the integrity of the encryption process. With Ascon maintaining this level of security, the system remains safe from such attacks.

5.7.2 Quantum Resistance and the LWE Problem

Kyber's security is based on the hardness of the LWE problem, a problem that is resistant to quantum attacks. The role of the hash function, whether Keccak/SHA3 or Ascon, is to provide secure pseudorandomness during key generation, encryption, and decryption processes. Replacing the hash function with Ascon does not alter the underlying lattice-based structure of Kyber or the LWE problem.

- **Pseudorandomness**: The Ascon hash functions used in Kyber are responsible for generating pseudorandom numbers that are critical for key generation and sampling noise in the encryption and decryption processes. These random numbers are integral to ensuring that the LWE-based encryption scheme is secure against quantum attacks. Ascon's performance in terms of generating secure and unpredictable pseudorandom outputs ensures that the LWE problem's inherent security properties are not compromised.

- **Security Against Quantum Attacks**: The security level of Ascon, combined with the lattice-based LWE problem, ensures that the system maintains quantum resistance. The cryptographic design principles of Kyber remain focused on resisting attacks from quantum computers, and the replacement of the hash function does not affect these properties.

5.7.3 Impact on Lattice-based Design

Kyber's reliance on lattice-based cryptography means that its security comes from the hardness of certain mathematical problems, such as the LWE problem. The hash function in Kyber is primarily used for deriving secure keys and random values but does not alter the mathematical foundation of the scheme. Therefore, switching from Keccak/SHA3 to Ascon does not impact the security derived from Kyber's lattice-based design.

Moreover, the Ascon hash function ensures that key derivation and message encryption remain secure and resistant to both classical and quantum attacks. This replacement preserves the cryptographic strength needed to protect against known vulnerabilities while maintaining Kyber's established resistance to quantum algorithms.

The integration of Ascon hash functions into the Kyber encryption scheme ensures that the system maintains its 128-bit security level against collision and pre-image attacks. Additionally, the quantum resistance provided by Kyber's lattice-based design, particularly with regard to the LWE problem, remains unaffected by the replacement of the hash function. Ascon's efficiency and security make it an appropriate alternative to Keccak/SHA3, preserving Kyber's robustness in both classical and post-quantum environments.

Chapter 6

CONCLUSION AND FUTURE RECOMMENDATION

6.1 Conclusion

This thesis focused on integrating the Ascon hash function into Kyber, now standardized by NIST as MLKEM, as a replacement for the existing Keccak hash function. The primary objective was to evaluate the impact of this change on performance and suitability for embedded systems without compromising Kyber's well-established security. Ascon, a LWC algorithm, is recognized for its efficiency in resource-constrained environments, making it an attractive alternative to Keccak in such contexts.

The testing, conducted on a personal computer, provided valuable insights but also highlighted a key limitation: lightweight algorithms like Ascon are not designed to deliver optimal performance on resource-enriched devices, such as personal computers or servers. In these environments, where computational power and resources are abundant, lightweight algorithms often fail to show significant gains in efficiency or performance. However, the real strength of Ascon lies in its application to embedded systems, where computational and power resources are limited. This research concludes that while the performance improvements on general-purpose hardware were modest, Ascon's integration into MLKEm can lead to substantial efficiency gains when deployed on resource-constrained devices, such as embedded systems or IoT platforms.

Benchmarking was an essential component of this study, with a focus on measuring the performance of Ascon and Keccak hashes and CPU cycles to gauge performance before and after hash functions replacement. The results demonstrated that Ascon reduced computational overhead in key cryptographic functions compared to Keccak, making it a more suitable choice for devices with stringent resource requirements. This suggests that future implementations of MLKEM in embedded systems could benefit from enhanced performance and efficiency, making Ascon a valuable component in these environments.

6.2 Future Recommendation

While this research successfully demonstrated the feasibility of replacing Keccak with Ascon, several avenues for future work remain. One critical area of investigation is to perform additional benchmarking on embedded systems. Since lightweight algorithms are tailored for resource-constrained environments, testing on actual embedded platforms will provide a more accurate assessment of Ascon's performance benefits. This benchmarking should also consider the hash rate, as varying squeeze lengths from 8 bytes to 168 bytes can significantly impact the efficiency of the Ascon-based implementation. Understanding how the hash rate interacts with different squeeze lengths in embedded contexts will help validate the hypothesis that lightweight cryptographic functions, like Ascon, offer superior efficiency and resource optimization in these environments compared to resource-rich devices, where they do not produce the desired performance outcomes.

Ongoing security evaluations are crucial to ensure that ML-KEM with Ascon continues to withstand evolving cryptographic attacks and advances in quantum computing. As the field of PQC matures, continuous benchmarking and security analysis will be necessary to confirm that the integration of lightweight functions like Ascon remains both secure and efficient for future applications. Adjusting the focus on hash rates and their corresponding squeeze lengths during these evaluations will further enhance the robustness of the findings and provide a clearer understanding of the optimal configurations for various operational environments.

Bibliography

[1] A. Tutoveanu, “Active implementation of end-to-end post-quantum encryption,” *Cryptology ePrint Archive*, 2021.

[2] Y. Huang, M. Huang, Z. Lei, and J. Wu, “A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse,” *IEICE Electronics Express*, vol. 17, no. 17, pp. 20200234–20200234, 2020.

[3] M. J. Dworkin, “SHA-3 standard: Permutation-based hash and extendable-output functions,” 2015.

[4] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, “Ascon,” tech. rep., National Institute of Standards and Technology (NIST), May 2021. Accessed: 2024-09-01. [Online]. Available: https://csrc.nist.gov/projects/lightweight-cryptography.

[5] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.

[6] “Announcement and outline of NIST’s call for submissions.” Available at https://csrc.nist.rip/presentations/2016/announcement-and-outline-of-nist-s-call-for-submis, 2016. Accessed: 2024-09-01.

[7] FIPS203, “Module-lattice-based key-encapsulation mechanism standard,” tech. rep., National Institute of Standards and Technology (NIST), August 2024.

[8] FIPS204, “Module-lattice-based digital signature standard,” tech. rep., National Institute of Standards and Technology, August 2024.

[9] FIPS205, “Stateless hash-based digital signature standard,” tech. rep., National Institute of Standards and Technology, August 2024.

[10] National Institute of Standards and Technology, "Submission requirements and evaluation criteria for the lightweight cryptography standardization process," 2018.

[11] M. S. Turan, K. McKay, D. Chang, L. E. Bassham, J. Kang, N. D. Waller, J. M. Kelsey, and D. Hong, "Status report on the final round of the NIST lightweight cryptography standardization process," 2023.

[12] "Caesar: Competition for authenticated encryption: Security, applicability, and robustness." Available at `http://competitions.cr.yp.to/caesar.html`, 2013.

[13] "Cryptographic hash algorithm competition." Available at `https://www.nist.gov/programs-projects/cryptographic-hash-algorithm-competition`. Accessed: 01-09-2024.

[14] "Lightweight cryptography." Available at `https://csrc.nist.gov/projects/lightweight-cryptography`. Accessed: 2024-09-01.

[15] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber algorithm specifications and supporting documentation," *NIST PQC Round 3*, 2020.

[16] A. Langley, "Maybe skip SHA-3." Available at `https://www.imperialviolet.org/2017/05/31/skipsha3.html`, 2017. Blog post on ImperialViolet, accessed September 23, 2024.

[17] "Arm partners have shipped 200 billion chips." Available at `https://newsroom.arm.com/blog/200bn-arm-chips`. Accessed September 23, 2024.

[18] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, no. 6, pp. 84–93, 2005.

[19] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proceedings of the 2010 Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pp. 1–23, 2010.

[20] D. Stehlé and R. Steinfeld, "Making ntru as secure as worst-case problems over ideal lattices," in *Advances in Cryptology - EUROCRYPT 2011*, pp. 27–47, Springer, 2011.

[21] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota, "Post-quantum lattice-based cryptography implementations," *ACM Computing Surveys*, vol. 51, no. 1, pp. 1–41, 2019.

[22] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*, pp. 99–108, 1996.

[23] C. Chuengsatiansup *et al.*, "Modfalcon: Compact signatures based on module-ntru lattices," *Proceedings of the 2020 ACM Conference on Computer and Communications Security (CCS)*, pp. 853–866, 2020.

[24] A. Wang, D. Xiao, and Y. Yu, "Lattice-based cryptosystems in standardisation processes: A survey," *IET Information Security*, vol. 17, no. 2, pp. 227–243, 2023.

[25] D. Micciancio, "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions," *computational complexity*, vol. 16, pp. 365–411, 2007.

[26] "Post-quantum cryptography standardization: Initial round report," technical report, National Institute of Standards and Technology, 2022.

[27] B. Applebaum, D. Cash, C. Peikert, and A. Sahai, "Fast cryptographic primitives and circular-secure encryption based on hard learning problems," in *Proceedings of the 2009 Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pp. 595–618, 2009.

[28] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 353–367, IEEE, 2018.

[29] Netmeister, "Post-quantum cryptography (pqc) and updates." Available at https://www.netmeister.org/blog/pqc-2024-01.html#8. Accessed: 2024-09-04.

[30] Chromium Blog, "Protecting chrome traffic with hybrid cryptography." Available at https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html. Accessed: 2024-09-04.

[31] Apple Security, "imessage pq3: Post-quantum cryptography protocol." Available at https://security.apple.com/blog/imessage-pq3/. Accessed: 2024-09-04.

[32] Cloudflare Blog, "Post-quantum cryptography: General availability." Available at https://blog.cloudflare.com/post-quantum-cryptography-ga/. Accessed: 2024-09-04.

[33] Mozilla, "Firefox desktop channels." Available at https://www.mozilla.org/en-US/firefox/channel/desktop/. Accessed: 2024-09-04.

[34] Signal Blog, "Post-quantum encryption: An overview." Available at https://signal.org/blog/pqxdh/. Accessed: 2024-09-04.

[35] E. Crockett, C. Paquin, and D. Stebila, "Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh," Tech. Rep. Report 2019/858, Cryptology ePrint Archive, 2019. Accessed: 2024-09-04.

[36] J. Brendel, R. Fiedler, F. Günther, C. Janson, and D. Stebila, "Post-quantum asynchronous deniable key exchange and the signal handshake," in *PKC 2022, Part II* (G. Hanaoka, J. Shikata, and Y. Watanabe, eds.), vol. 13178 of *LNCS*, pp. 3–34, Heidelberg: Springer, 2022. Accessed: 2024-09-04.

[37] AWS, "s2n-tls: A tls library." Available at https://github.com/aws/s2n-tls. Accessed: 2024-09-04.

[38] AWS, "s2n-quic: A quic implementation." Available at https://github.com/aws/s2n-quic. Accessed: 2024-09-04.

[39] AWS, "Post-quantum tls now supported in aws kms." Available at https://aws.amazon.com/blogs/security/post-quantum-tls-now-supported-in-aws-kms/. Accessed: 2024-09-04.

[40] AWS, "Post-quantum hybrid sftp file transfers using aws transfer family." Available at: https://aws.amazon.com/blogs/security/post-quantum-hybrid-sftp-file-transfers-using-aws-transfer-family/. Accessed: 2024-09-04.

[41] Cisco, "Quantum encryption in cisco vpns." Available at: https://www.cisco.com/c/en/us/td/docs/routers/ios/config/17-x/sec-vpn/b-security-vpn/m-sec-cfg-quantum-encryption-ppk.pdf. Accessed: 2024-09-04.

[42] Proton Mail Blog, "Post-quantum encryption: Proton mail's approach." Available at https://proton.me/blog/post-quantum-encryption. Accessed: 2024-09-04.

[43] "Open quantum safe." Available at https://openquantumsafe.org/. Accessed September 23, 2024.

[44] BoringSSL Project, "Boringssl." Available at https://boringssl.googlesource.com/boringssl/. Accessed September 23, 2024.

[45] Botan Project, "Discussion on kyber support in botan." Available at https://github.com/randombit/botan/discussions/3747. Accessed September 23, 2024.

[46] "wolfssl embedded ssl/tls library." Available at https://www.wolfssl.com/products/wolfssl/. Accessed September 23, 2024.

[47] Bouncy Castle Project, "Bouncy castle specification and interoperability documentation." Available at https://www.bouncycastle.org/documentation/specification_interoperability/. Accessed September 23, 2024.

[48] KyberLib, "About kyberlib." Available at https://kyberlib.com/about/index.html. Accessed September 23, 2024.

[49] "pqcrypto: Post-quantum cryptography." Available at https://pypi.org/project/pqcrypto/. Accessed September 23, 2024.

[50] E. Crockett, C. Paquin, and D. Stebila, "Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh," *Cryptology ePrint Archive*, 2019.

[51] J. Brendel, R. Fiedler, F. Günther, C. Janson, and D. Stebila, "Post-quantum asynchronous deniable key exchange and the signal handshake," in *IACR International Conference on Public-Key Cryptography*, pp. 3–34, Springer, 2022.

[52] "Discussion about Kyber's tweaked FO transform." Forum post on pqc-forum, 2023. Available at https://groups.google.com/a/list.nist.gov/g/pqcforum/c/WFRDl8DqYQ4.

[53] "Kyber decisions, part 2: FO transform." Forum post on pqc-forum, 2023. Available at https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/C0D3W1KoINY/m/99kIvydoAwAJ.

[54] Google Security Blog, "A new path for Kyber on the web." Available at https://security.googleblog.com/2024/09/a-new-path-for-kyber-on-web.html. Accessed September 23, 2024.

[55] R. R. Krishna, A. Priyadarshini, A. V. Jha, B. Appasani, A. Srinivasulu, and N. Bizon, "State-of-the-art review on iot threats and attacks: Taxonomy, challenges and solutions," *Sustainability*, vol. 13, no. 16, p. 9463, 2021.

[56] A. Wang, D. Xiao, and Y. Yu, "Lattice-based cryptosystems in standardisation processes: A survey," *IET Information Security*, vol. 17, no. 2, pp. 227–243, 2023.

[57] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Annual international cryptology conference*, pp. 537–554, Springer, 1999.

[58] PQCRYPTO, "Kyber: A post-quantum public-key encryption scheme." Available at https://github.com/pq-crystals/kyber. Accessed: 2024-09-04.

[59] Ascon Team, "Ascon C implementation." Available at https://github.com/ascon/ascon-c/tree/main/crypto_hash, 2023. Accessed: 2024-09-22.

[60] C. Gewehr, L. Luza, and F. G. Moraes, "Hardware Acceleration of Crystals-Kyber in Low-Complexity Embedded Systems with RISC-V Instruction Set Extensions," *IEEE Access*, 2024.

[61] S. Kundu, A. Ghosh, A. Karmakar, S. Sen, and I. Verbauwhede, "Rudraksh: A compact and lightweight post-quantum key-encapsulation mechanism," *Cryptology ePrint Archive*, 2024.