Crypto Agility

Transition to post-quantum safe algorithms for secure key exchange and certificate generation.

Bachelor Thesis
Spring Semester 2023

Authors
Christopher Hilfing
Lara Gubler
Petra Heeb

Advisor
Prof. Dr. Purandare Mitra

External Advisors
Dr. Basil Hess
Dr. Martin Schmatz

Eastern Switzerland University of Applied Sciences

Abstract

The quantum era is arriving, which poses a significant threat to traditional encryption and public-key cryptography standards. Quantum computing breaks many cryptographic algorithms as the underlying mathematical problems could be solved by quantum computers within a short amount of time. With the appearance of quantum computers, cryptographic algorithms have also evolved. New quantum-safe algorithms have been standardized in the past year, but only a few applications already use them. To ensure a secure environment this will need to change.

Faced with these challenges and the rapid improvements in the area of quantum computing, the global cybersecurity landscape plunges into a highly precarious state. It is therefore important to test and deploy the new cryptographic algorithms today.

This bachelor thesis aims to demonstrate how two recently standardized post-quantum secure algorithms can be used by testing their compatibility with a Hardware Security Module (HSM) in a controlled environment. To demonstrate how these could be implemented in a quantum-safe manner at a later stage, two different use cases will be realized. The used quantum-resistant algorithms CRYSTALS-Kyber and CRYSTALS-Dilithium are based on the module lattices problem.

First, two Proof of Concepts (PoCs) were implemented, that demonstrate the compatibility between the HSM architecture and the two CRYSTALS algorithms. Afterward work on the two use cases began in parallel. The use case: Bring Your Own Key (BYOK) demonstrates how locally generated keys can be imported into the HSM in a quantum-safe manner. In this demonstration, the Key Encapsulation Mechanism (KEM) CRYSTALS-Kyber is used to generate a shared secret so that the client application can communicate to the HSM using AES.

The second use case focuses on a Public Key Infrastructure (PKI) based on a post-quantum secure infrastructure. The HSM is used as a key store to secure the identity of the Root Certificate Authority (CA), which acts as the root of trust. This ensures that keys are never exposed in clear text in memory. Furthermore, the quantum-safe signature scheme CRYSTALS-Dilithium is used to sign certificates which further increases security.

The outcomes of this research provide valuable insights into the implementation of quantum-safe algorithms in practical high security scenarios. In addition, the implementations facilitate the replication of a similar use case for an enterprise architecture and the transition from today's legacy algorithms to the new secure post-quantum algorithms with increased efficiency.

Both the BYOK and PKI implementations could also still be extended to provide more functionality, and higher security standards based on the algorithm versions or protocol used. The PKI implementation could also be further improved by using a quantum-safe variant of Transfer Layer Security (TLS).

Management Summary

Quantum computing is coming, and it poses a significant threat to the security of many cryptographic algorithms currently in use. This is not only a problem for the future, but also has an impact on software running today, as an attacker could simply store important traffic and then try to decrypt it later.

This is why research into quantum-safe algorithms has been going on for years. In 2022, NIST announced that the CRYSTALS-Dilithium and CRYSTALS-Kyber algorithms will be part of the new post-quantum cryptography standards. CRYSTALS-Dilithium will be the new standard for digital signature algorithms, whereas CRYSTALS-Kyber will be the new standard for public key encryption and key establishment.

This thesis shows the implementation of two use cases to demonstrate how these two post-quantum algorithms can be used.

Procedures and Technologies

Several phases were required to test whether these algorithms could run on a Hardware Security Module (HSM) provided by Marvell, and then to implement the use cases. Starting with the familiarization phase to look at each of the algorithms and become familiar with the HSM, followed by an implementation phase to successfully test the compatibility of the algorithms with the hardware, and then the implementation of the use cases: Bring Your Own Key (BYOK) and Public Key Infrastructure (PKI). The BYOK implementation allows the user to generate a Cryptographic Recovery Key (CRK) locally and then securely import it into the HSM. This is done by first using CRYSTALS-Kyber to generate a shared secret and then using AES to transport the CRK to the HSM instead of using RSA, which is not quantum-safe. The PKI is a certificate organization that ensures the identity and origin of keys. In this implementation of a PKI, the root Certificate Authority (CA) is signed by the HSM and thus keys are never exposed in plaintext in memory. In addition, the keys for the certificates are generated using CRYSTALS-Dilithium, a new quantum-safe signature scheme, rather than the non-quantum-safe algorithms used today.

Results

The following results were achieved during this work:

compatibility

Two proofs of concept were written to show that the HSM provided by Marvell could run the two algorithms CRYSTALS-Dilithium and CRYSTALS-Kyber on the Secure Machine, as it was not clear at the start of the project whether this was possible.

PoC: Bring Your Own Key (BYOK)

The implementation of the BYOK PoC demonstrates how CRYSTALS-Kyber can be used to transport a CRK into the HSM in a quantum-safe manner.

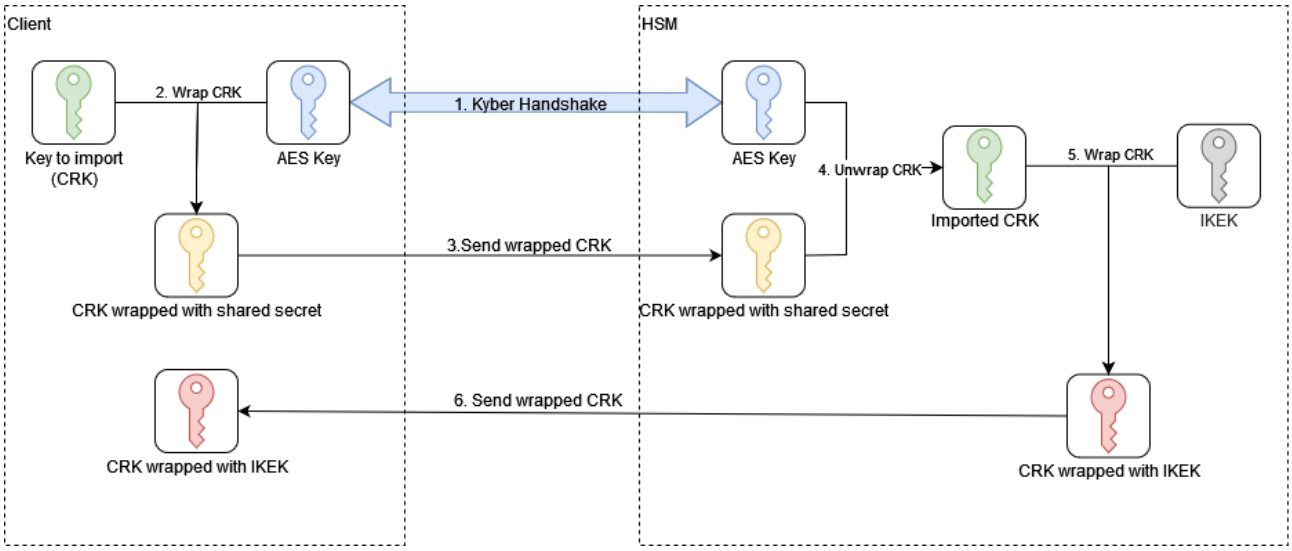

Figure 1: PoC: Bring Your Own Key (BYOK)

Public Key Infrastructure (PKI)

The implementation shows how a PKI can be implemented using a quantum-safe algorithm and never exposing secret keys in clear text in memory. This PKI implementation ensures that keys are secured in an HSM and can thus be stored in a protected manner. In addition, the keys are generated using quantum-safe algorithms and cannot be broken by a quantum computer.

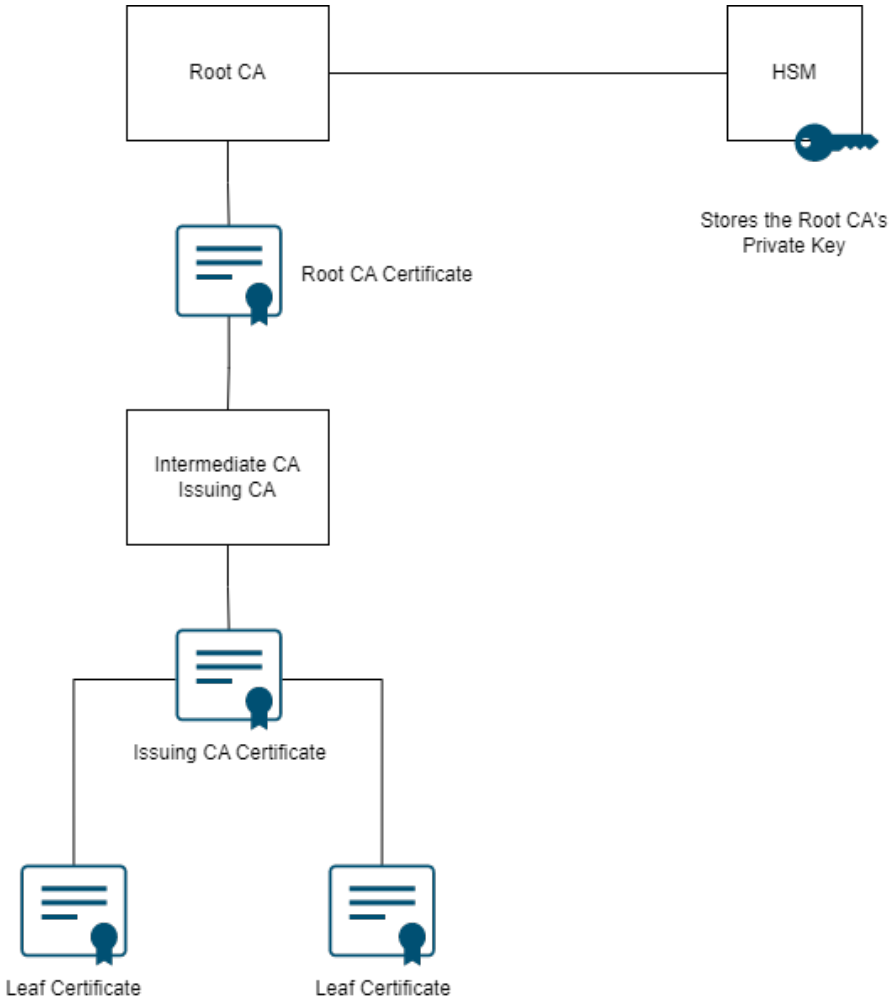

Figure 2: Public Key Infrastructure (PKI)

Outlook

This thesis demonstrates a first step towards a quantum-safe future. The implementation described can be extended and improved to make it suitable for production environments and even more secure than it is now.

The BYOK implementation is currently only a minimum viable product. There are still ways to increase the security, for example by using an authenticated version of the Kyber handshake.

The PKI implementation can also be extended to support quantum-safe TLS, for example. In addition, this implementation is currently based on OpenSSL version 1.1.1, which is about to expire. Migration to version 3 should therefore also be considered.

Acknowledgements

We want to say a big thank you to our supervisors, without you such work would not have been possible.

First of all, we want to thank our professors Prof. Dr. Nathalie Weiler and Prof. Dr. Mitra Purandare for their supervision. The feedback and support was always a great help and made our work easier.

We also want to thank our industrial partners Dr. Martin Schmatz and Dr. Basil Hess from the IBM Research Lab in Rüschlikon for their support and guidance in all technical matters. We have always been able to count on you and get your opinions on our decisions. The provision of the infrastructure, as well as your supervision of the work, was excellent.

The collaboration with IBM Research and the University on this project was very exemplary, and we were able to experience and learn a good mixture of practice and theory without any effort.

Thank you very much for this experience.

Contents

List of Figures

List of Tables

Glossary

AES-GCM AES-GCM (Advanced Encryption Standard-Galois/Counter Mode) is a symmetric encryption algorithm that combines the AES block cipher with the Galois/Counter Mode (GCM) for authenticated encryption and data integrity. . 78

AES-KWP AES Key Wrap Algorithm with Padding. This algorithm provides secure key wrapping by encrypting the key with another encryption key.. 78

AIV The Alternative Initial Value (AIV) is a 32-bit constant concatenated to a 32-bit message length indicator (MLI).. 76

IND-CCA2-secure indistinguishability under adaptive chosen ciphertext attack.. 33, 77

Jira Is a tool for project management and agile planning processes.. 35, 65, 79

Kyber Kyber is an IND-CCA2-secure key encapsulation mechanism (KEM) made by IBM. . 79

Linux Is an open-source operating system.. 76

LiquidSecurity LiquidSecurity is a product family from the company Marvell (HSM vendor).. 26

Makefile target A target is usually the name of a file that is generated by a program; examples of targets are executable or object files.. 46

MIPS LiquidSecurity is a product family from the company Marvell (HSM vendor).. 26, 43, 48

MVP Minimum Viable Product (MVP) is the minimal version of a product for using it.. 65, 79

NIST National Institute of Standards and Technology (NIST) is an agency of the United States that work on innovation in technologies and standards.. 19, 21, 37

OpenSSL Toolkit for general-purpose cryptography and secure communication.. 34

PKCS#11 Public Key Cryptography Standard is a platform-independent API for cryptographic tokens.. 32, 62, 76, 105

static public key A cryptographic key that has a long period of use and often also is meant to be utilized in multiple instances of a cryptographic key establishment scheme, is referred to as a static key.. 11, 78

Acronyms

ADC Application Delivery Controller. 33, 34

AES Advanced Encryption Standard. 63, 75, 77–79, 95

API Application Programming Interface. 28, 32, 34, 60, 62, 64, 74, 94

AU Appliance User. 24

BYOK Bring Your Own Key. 11, 19–21, 75, 95

CA Certificate Authority. 54–57, 60, 61, 66, 71, 74, 95

CO Crypto Officer. 24

CoT Chain of Trust. 53, 54, 65, 66

CPU Central Processing Unit. 26

CRK Customer Root Key. 62, 63, 75–80, 86, 91, 95, 105

CRL Certificate Revocation List. 57, 61, 71

CSP Cloud Service Provider. 75

CSR Certificate Sign Request. 57, 60

CU Crypto User. 24

DNSSEC Domain Name System Security Extensions. 30

EOL End Of Life. 64

GDPR General Data Protection Regulation. 30

HSM Hardware Security Module. 19–21, 24–34, 43, 47, 48, 51, 54, 59–66, 71, 72, 74–77, 79, 83, 86, 92, 94, 95, 102, 103

IKEK Instance Root Key. 63, 76, 91, 95

KAT Known Answer Test. 66, 81

KCV Key Check Value. 76, 77, 83, 105

KEK Key Encryption Key. 63

KEM Key Encapsulation Mechanism. 33, 77, 95

LWE Learning With Errors. 33, 77

MCO Master Crypto Officer. 24

MEK Master Encryption Key. 63

MKEK Master Key Encryption Key. 75–77, 79, 80, 91

MVP Minimum Viable Product. 99, 103

NCSC National Cyber Security Center. 58

NFR Non-Functional Requirements. 66, 67, 81

OCSP Online Certificate Status Protocol. 57, 61

PCI Peripheral Component Interconnect. 30

PCO Partition Crypto Officer. 24

PEK Public Encryption Key. 90, 119, 120

PEM Private Enhanced Mail. 57

PKI Public Key Infrastructure. 19–21, 33, 36, 52, 54, 55, 57–59, 61–64, 71, 95

PoC Proof of Concept. 11, 19–21, 27, 43, 47, 48, 50, 51, 75–77, 82, 83, 86, 91, 94, 95, 99, 105

PQC Post-Quantum Cryptography. 32

QSC Quantum Safe Cryptography. 21

RoT Root of Trust. 53, 54, 95

RSA Rivest–Shamir–Adleman. 19, 75, 95

RTE Run-Time Engine. 26

SDK Software Development Kit. 26

SM Secure Machine. 24, 26, 29, 43–45, 48, 50, 51, 62, 68, 71, 78, 82, 86, 89, 95

SMF Secure Machine Framework. 26

SMW Secure Machine World. 26

TLS Transport Layer Security. 19, 33, 34, 71, 95

VM Virtual Machine. 25, 28, 59, 76

Part I

Technical Report

Chapter 1 Introduction

The quantum era is arriving, posing a significant threat to traditional encryption and public key cryptography standards.

Cryptographic systems are based on a mathematical problem. Rivest–Shamir–Adleman (RSA), for example, is based on the so-called factorial problem, which was previously thought to be secure. However, with the advent of quantum computers, some of these mathematical problems are now considered insecure, such as the Factorial Problem. The reason for this is that the Factorial Problem assumes that there is no factorization procedure with a polynomial running time. However, the Shor algorithm, a new factorization method for quantum computers, runs in polynomial time. The result of this new method is that legacy algorithms such as RSA are broken. A similar problem exists with other cryptographic systems or protocols, such as Transport Layer Security (TLS).

Due to the rise of quantum computing, organizations need to adopt quantum-safe algorithms to ensure the integrity, authentication and non-repudiation of applications, infrastructure and data.

In response to the predicted threats, NIST initiated a process to solicit, evaluate, and standardize quantum-resistant public key algorithms. Three out of the four algorithms selected (CRYSTALS-Kyber, CRYSTALS-Dilithium and Falcon), which are suitable for the new standard defined by NIST, were developed with the help of IBM researchers. The fourth algorithm (SPHINCS+) is a stateless hash-based signature scheme developed by another team.

CRYSTALS-Kyber is the new standard for public key encryption, while CRYSTALS-Dilithium is used as the new digital signature scheme. The Falcon digital signature algorithm has been selected as the standard to be used in situations where Dilithium would be prohibited in space. [1]–[4]

In this work, the two algorithms CRYSTALS-Kyber and CRYSTALS-Dilithium are tested on a Hardware Security Module (HSM). These new algorithms can be used to create a post-quantum safe environment for future use. The first step is to test whether these algorithms are compatible with the corresponding HSM architecture, and thus whether it is possible to build a post-quantum safe application. Subsequently, the two use cases, Bring Your Own Key (BYOK) and Public Key Infrastructure (PKI), will be implemented post-quantum secure to create a sensible approach for use.

The technical documentation comprises the theoretical and practical parts of our work, with a technical focus. First, the scope of this work is described in more detail. Then, in addition to the users of the application, the basic architecture and the individual components are described. This is followed by a small theoretical section on signatures, which forms the basis for the rest of the report. The Proof of Concept (PoC) of the algorithms is followed by a theoretical and practical part on the two use cases BYOK and PKI. The technical documentation ends with a conclusion. The entire project management part of this work is included in the project documentation.

Chapter 2 Problem Definition

With the standardization of post-quantum algorithms such as CRYSTALS-Dilithium, the use of these algorithms in combination with a Hardware Security Module (HSM) has become a relevant topic of discussion. The CRYSTALS-Dilithium implementation has not yet been tested on the Marvell LiquidSecurity HSM used at IBM. In this thesis, we address how obtain such a post-quantum HSM:

The first step is to provide a Proof of Concept (PoC) for the CRYSTALS-Dilithium and the CRYSTALS-Kyber algorithms, showing how these algorithms can run on the Marvell HSM.

Then, two different use cases (Bring Your Own Key (BYOK) and Public Key Infrastructure (PKI)) will be implemented using the Kyber and Dilithium algorithms. These implementations will serve as demos and show how these post-quantum algorithms can be used for real-world use cases.

Chapter 3 Task Definition

NIST has announced the first post-quantum secure cryptographic protocol standards for the quantum computing era. The CRYSTALS-Dilithium and CRYSTALS-Kyber algorithms are an option for the use cases of a Public Key Infrastructure (PKI) and Bring Your Own Key (BYOK) for Quantum Safe Cryptography (QSC).

3.1 Background

With this new standard, new implementations based on the standardized cryptographic algorithms CRYSTALS-Dilithium and CRYSTALS-Kyber have to be tested. For this purpose, proof of concepts have to be carried out on so-called Hardware Security Module (HSM)s. To test and implement the use cases, IBM is providing an HSM from Marvell. This Proof of Concept (PoC) will test whether the Marvell HSM is compatible with the new cryptographic algorithms.

3.2 Tasks

1. Familiarize yourself with the new infrastructure: server, HSM, VM etc.
2. Create a Proof of Concept (PoC) for the CRYSTALS-Dilithium algorithm on the Marvell HSM.
3. Create a Proof of Concept (PoC) for the CRYSTALS-Kyber algorithm on the Marvell HSM.
4. Implement use cases (or Proof of Concept (PoC) for use cases) for the post-quantum safe algorithms (Dilithium and Kyber).
5. Write a thesis report documenting the implementation, analysis and results.

3.3 Expected Results

- Documentation of the project work
- Proof of Concepts for Dilithium and Kyber algorithms
- Demonstration of the results
- Presentation of the results to the industrial partner

3.4 Partner

This project work will be carried out with the external partner the Zurich Lab of IBM Research in Rüschlikon.

Dr. Martin Schmatz and Dr. Basil Hess are the local advisors at IBM Research.

Access to the workplace and any special equipment, will be agreed between the students and the industrial partner.

Chapter 4 Personas

To support the agile development of the application, role-based personas are defined.

This helps to prioritize features and take user needs into account when functional decisions have to be made. Furthermore, personas support the verification and revision of the defined requirements.

4.1 Host-App Users

The main goal of the personas defined below is to use and create Host-Apps with post-quantum algorithms.

4.1.1 Security Engineer

Name:
Selina

Role:
Security Engineer

Background:
Selina is a Security Engineer with 7 years of experience in this field of work. She is responsible for conducting thorough risk assessments, identifying vulnerabilities within a network and fixing them.

Goals:
The primary goal of Selina is to protect the confidentiality, integrity and availability of the organization's assets from cyber attacks. She is responsible for identifying vulnerabilities and implementing security controls to mitigate risks.

Tasks:
Selina's tasks include analyzing security logs, monitoring network traffic, responding to security incidents and conducting vulnerability assessments.

Challenges:
In her role as a security engineer, Selina faces several challenges. These include keeping up with the latest security threats and technologies and managing security risks across multiple systems and networks.

4.1.2 Software Developer

Name:
Eric

Role:
Software Developer

Background:
Eric is a Software Developer with 6 years of experience in this field of work. He is responsible for developing and testing secure software that meets the organization's and the user's needs.

Goals:
The primary goal of Eric is to develop high-security applications that meet the stakeholder's requirements. Moreover, he is responsible for maintaining code quality and ensuring the protection of sensitive data from cyber threats.

Tasks:
Eric's responsibilities include implementing secure software features, testing the application, as well as identifying and remediation of security vulnerabilities.

Challenges:
In his role as Software Developer, Eric faces several challenges. These include keeping up with the latest security technologies, meeting strict security requirements, and managing the complexity of secure software design.

4.1.3 System and Cloud Engineer

Name:
Alex

Role:
System and Cloud Engineer

Background:
Alex is a System and Cloud Engineer with 12 years of experience in this field of work. He is responsible for building and maintaining the organization's IT infrastructure including network and cloud infrastructure.

Goals:
The primary goal of Alex is to ensure the reliability, security, and scalability of the system and cloud infrastructure.

Tasks:
Alex's tasks include designing and deploying system and cloud infrastructure, configuring and managing virtual machines, and ensuring compliance with security regulations and requirements.

Challenges:
In his role as a System and Cloud Engineer, Alex faces several challenges. These include managing the complexity of the infrastructure, staying up to date with the latest system and cloud technologies and ensuring high availability.

4.1.4 Manager

Name:
Alina

Role:
Manager

Background:
Alina is a Manager with 17 years of experience in this role. She is responsible for ensuring that her team delivers high-quality products and services that meet the organization's and the customer's requirements.

Goals:
Alina's primary goal is to lead her team to success by delivering high-quality products on time and providing customers with IT solutions that offer high security with the latest security standards and technologies. Another goal of her role as manager is to ensure that her team members are motivated and engaged and that collaboration and knowledge exchange within the team is always practiced.

Tasks:
Alina's tasks include communicating with stakeholders, ensuring compliance with organizational policies and security standards, and coordinating project activities such as setting project goals and timelines.

Challenges:
In her role as a Manager, Alina faces several challenges. These include managing the team and conflicts and dealing with uncertainties and risks. Additionally, Alina has to present the customer with tailor-made solutions using the latest security standards that are not yet widely used and convince customers of the benefits and importance of these.

4.2 HSM and Secure Machine Users

The following personas define users who maintain and configure the Hardware Security Module (HSM) and Secure Machine (SM). These personas correspond to the available roles on the HSM. Each role has specific permissions for the HSM that determines which operations a user is allowed to perform. [5]

4.2.1 Master Crypto Officer (MCO)

Description:
The Master Crypto Officer (MCO) performs administrative tasks on the HSM including creating and backing up partitions, and setting system policies. There is only one MCO for the HSM.

4.2.2 Crypto Officer (CO) / Partition Crypto Officer (PCO)

Tasks:
The Crypto Officer (CO) or Partition Crypto Officer (PCO) is responsible for setting policies and performing user management operations for the partitions of the HSM.

4.2.3 Crypto User (CU)

Tasks:
The Crypto User (CU) can perform key management and cryptographic operations. Key management operations include creating, deleting, sharing, importing, exporting, and deriving cryptographic keys. Cryptographic operations include encryption, decryption, signing, and verifying by using cryptographic keys.

4.2.4 Appliance User (AU)

Tasks:
The Appliance User (AU) performs two primary tasks. Retrieving the audit logs and performing cloning and synchronization operations. The AU can synchronize keys across all HSMs in a cluster.

Chapter 5 Architecture

This chapter provides an overview of the overall architecture and aims to provide a comprehensive understanding of its structure. Various layers of the architecture and their functionality are described here, including components and the interaction between them.

The Architecture Overview diagram provides a comprehensive view of the entire architecture.

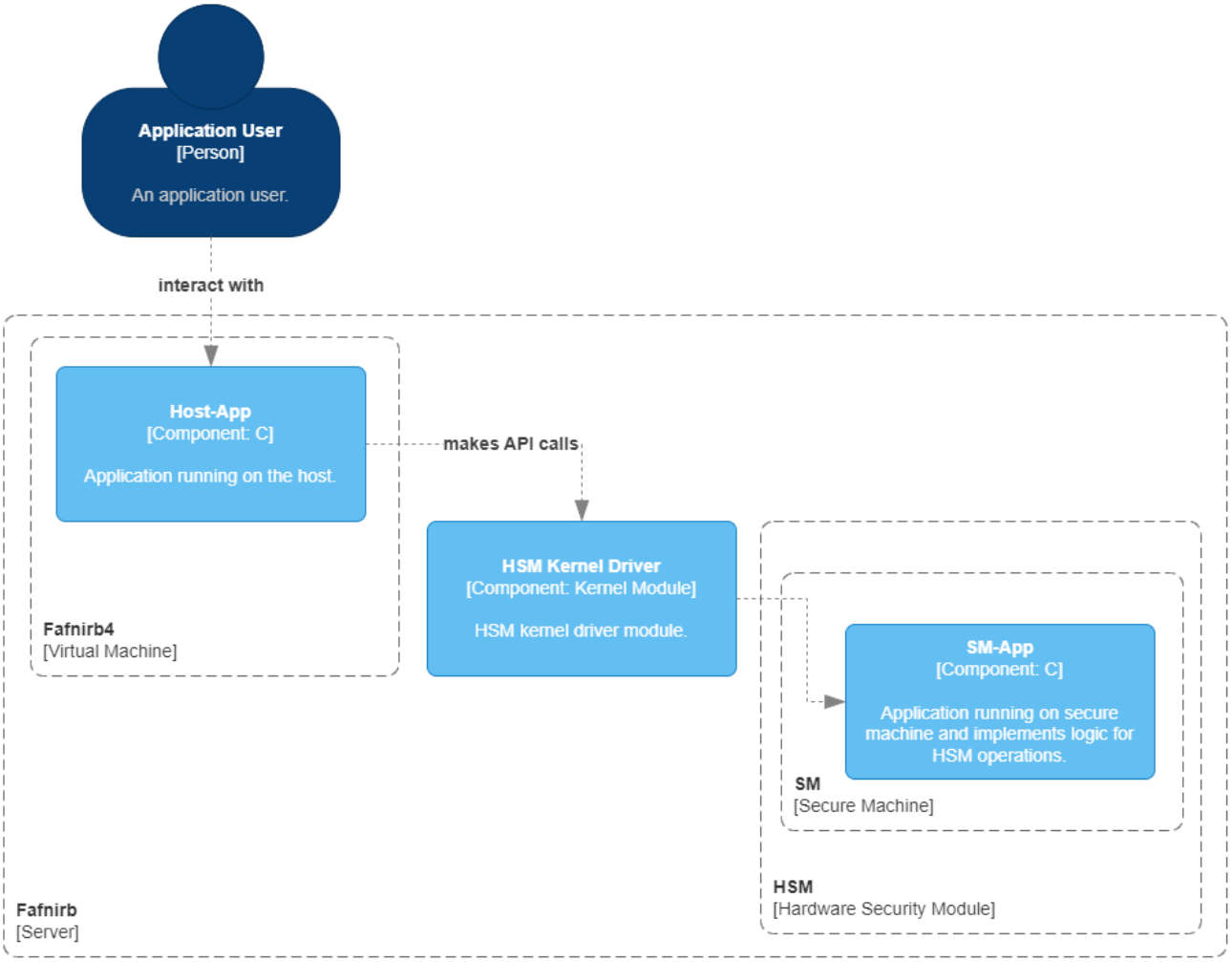

Figure 5.1: Architecture Overview

5.1 Architectural Components

This section provides a detailed description of the architectural components shown in the Architecture Overview diagram.

5.1.1 Fafnirb4

The Virtual Machine (VM) named `Fafnirb4` is located on the `Fafnirb` server. This setup simplifies the connection between the host (`Fafnirb4`) and the server (`Fafnirb`), as there is no interconnecting network to consider between the device where the application is installed and the server where the driver and Hardware Security Module (HSM) are located.

5.1.2 Fafnirb

`Fafnirb` is a server, located at IBM Research Zurich. This server is used to access the HSM.

5.1.3 HSM Kernel Driver

The kernel driver is provided by the HSM vendor and installed on the server. This driver communicates with the server (`Fafnirb`) and the HSM.

5.1.4 Hardware Security Module (HSM)

The HSM consists of multiple MIPS Central Processing Unit (CPU)s of the OCTEON family. For each cryptographic operation on the HSM, three HSM CPUs are used.

5.1.5 Secure Machine (SM)

The Secure Machine (SM) is a partition within the HSM that can be customized. In this thesis, all custom functions that need to be executed in a secure environment will be executed in the SM. An example of this would be signing a message with Dilithium. This is not yet possible on the provided HSM, as it does not support Dilithium.

5.1.6 Host Application (Host-App)

The custom C application on the host (`Fafnirb4`), which also communicates with the HSM or SM, is called the Host-App. It is written in C.

5.1.7 Secure Machine Application (SM-App)

The SM-App is an HSM user application that runs inside the LiquidSecurity HSM. This application implements new or custom algorithms and protocols. In this case, for example, the SM-App implements the CRYSTALS-Dilithium algorithm to protect any key material from being exposed in clear text in memory and to provide the Dilithium functionality.

Secure Machine World

OCTEON core 3 provides the Secure Machine World (SMW). This is a flexible, secure Run-Time Engine (RTE) for SM-Apps. The SMW contains an SM Manager for communication channels with pre-defined filters. The SMW is included in the LiquidSecurity HSM Firmware.

Secure Machine Framework

The Secure Machine Framework (SMF) provides the Software Development Kit (SDK) for developing SM-Apps and defines security policies that set the rules under which SM-Apps operate. The SMF is included in the LiquidSecurity HSM Firmware.

5.2 Build and Deploy Process

The Application Build and Deploy Process diagram describes the build and deploy process and shows at what point the two applications (Host-App and SM-App) communicate with each other and then how they are terminated. (The Host-App acts as a client, sending requests to the SM-App)

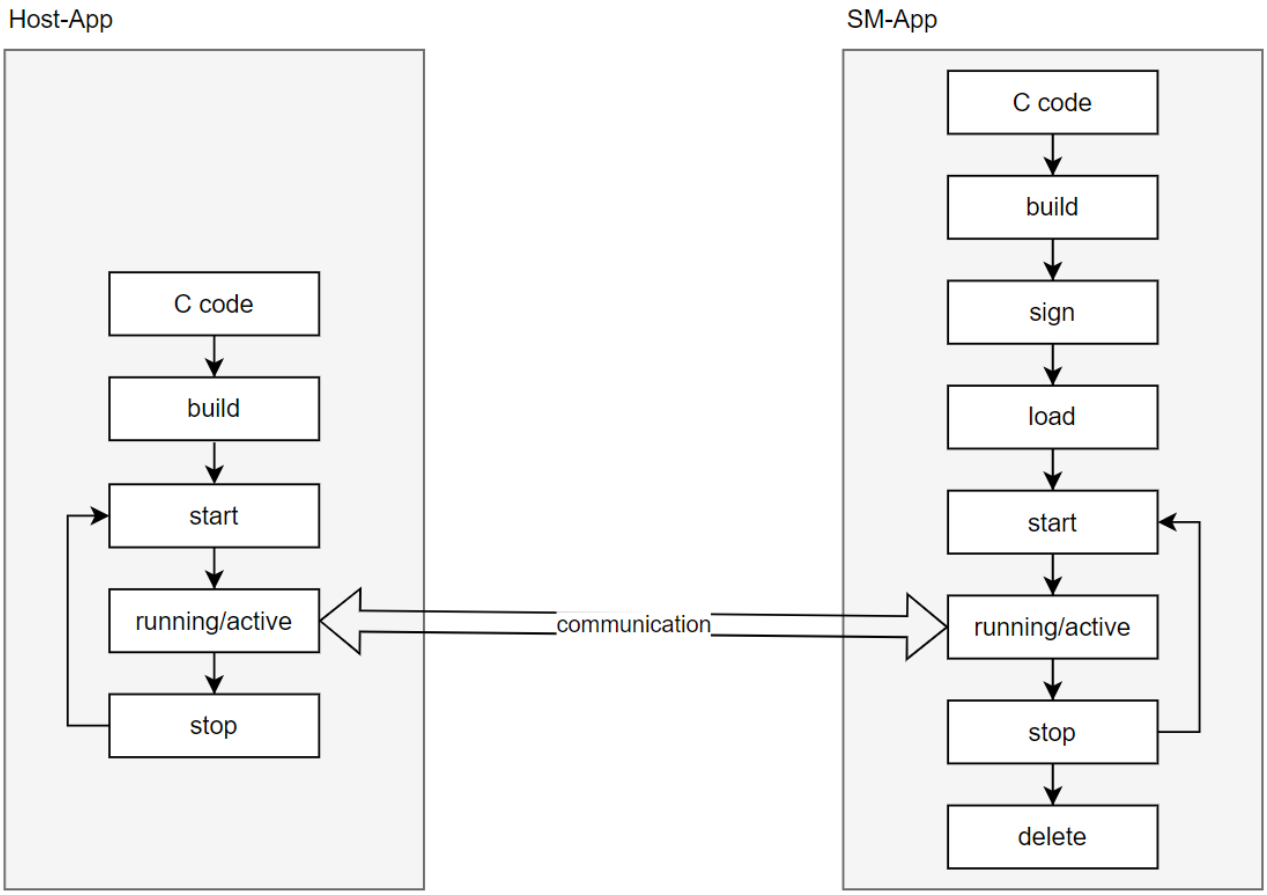

Figure 5.2: Application Build and Deploy Process

The process for building and deploying the SM-App is described on the right. This is the same for each Proof of Concept (PoC) and use case that has been implemented. Once the SM-App has been built and deployed, additional steps are required to run it, as the HSM will only run a signed software and the software must be loaded onto the HSM. When terminated, the SM-App is also deleted to keep the HSM "clean".

On the left is the simple build and deploy process for the Host-App is described. This is the same for each Proof of Concept (PoC) and use case that has been implemented.

5.2.1 Host-App to SM-App

The diagram Host-App to SM-App describes the communication between the Host-App and the SM-App.

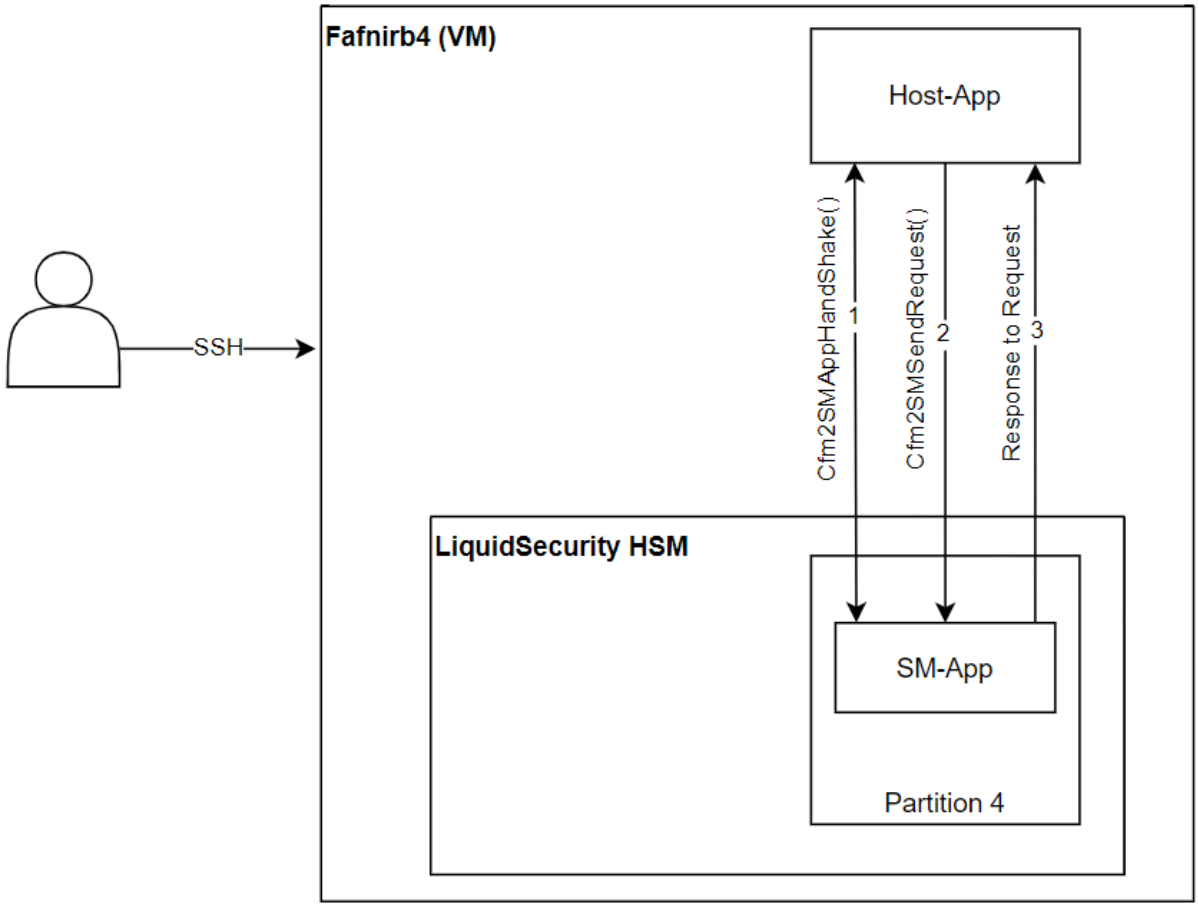

Figure 5.3: Host-App to SM-App

After opening a session with the HSM partition, the Host-App can communicate with the SM-App by performing a handshake. In step one, the Host-App uses the *Cfm2AMAppHandshake* function to initiate a handshake with the SM-App. In step two, after a successful handshake, the Host-App can send requests to the SM-App using the *Cfm2AMSendRequest* function. In step three, the SM-App can send data in response to *Cfm2AMSendRequest*s. These functions are provided by the `Cfm2Util` installed on the VM, which is an API published by Marvell.

To end the communication, the Host-App sends a *cleanup request* and calls the *Cfm2SMAppCleanup* function.

The Request-Response Flow diagram illustrates the Request-Response flow between the Host-App and the SM-App.

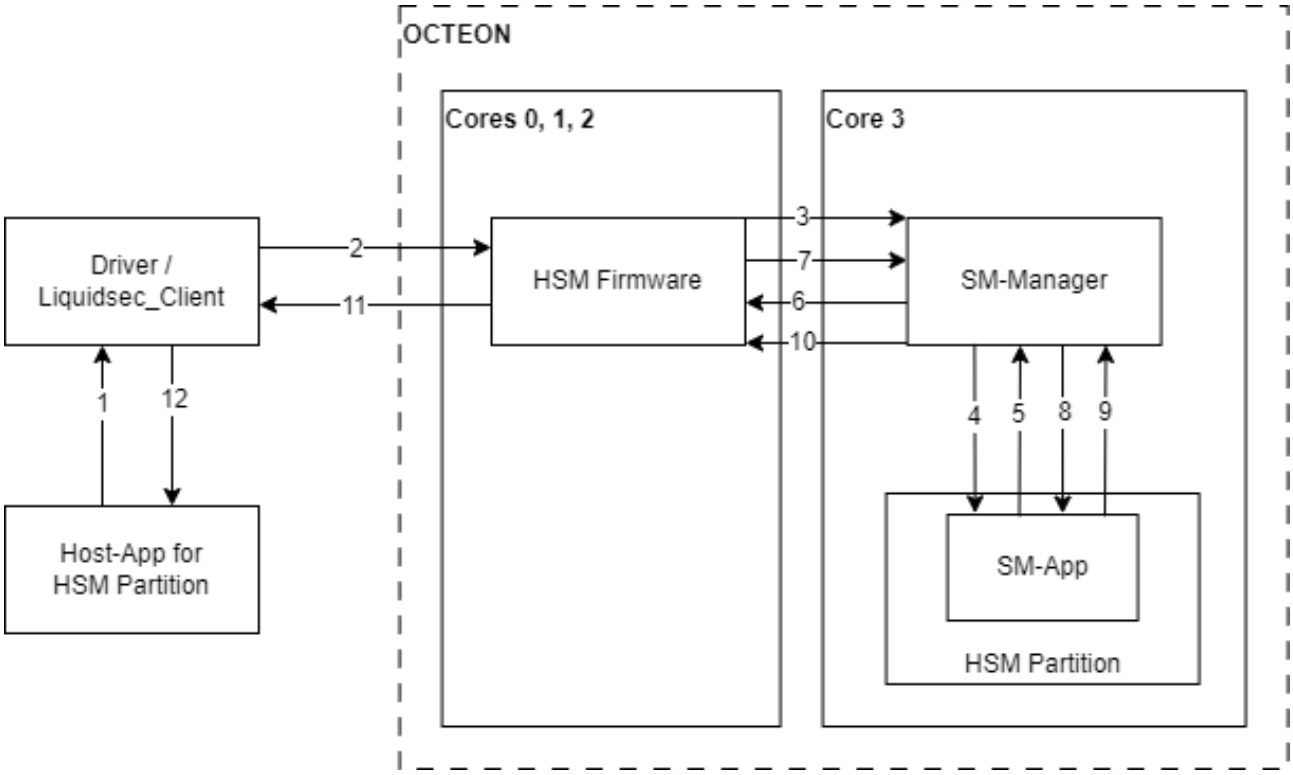

Figure 5.4: Request-Response Flow

1. Host-App sends a request to Driver/Liquidsec_client
2. Driver/Liquidsec_client sends a request to HSM firmware
3. HSM firmware sends a request to SM Manager
4. SM Manager sends a request to SM-App
5. SM-App sends firmware request to SM Manager
6. SM Manager sends firmware request to HSM firmware
7. HSM firmware sends a response to SM Manager
8. SM Manager sends firmware response to SM-App
9. SM-App sends a response to SM Manager
10. SM Manager sends a response to HSM firmware
11. HSM firmware sends a response to Driver/Liquidsec_client
12. Driver/Liquidsec_client sends a response to the Host-App

Chapter 6 Components

All the components required are explained in this chapter.

6.1 Hardware Security Module (HSM)

Hardware Security Module (HSM)s are hardened, tamper-resistant hardware devices used to perform cryptographic operations, such as generating, protecting and managing keys used to encrypt and decrypt data and create digital signatures and certificates.

They provide extra security for sensitive data in transit, in use, and at rest by validating the integrity and authenticity of components in a system and process, and storing cryptographic keys in a trusted environment.

The HSM is an external device that can be attached to a computer, server or network server via the Peripheral Component Interconnect (PCI).

6.1.1 Trusted Environment

As stated in NIST IR 8320: "The threat landscape has evolved to encompass more advanced attacks. Attackers are pushing lower in the platform stack threatening the platform firmware and hardware". [6]

Because the HSM is used to provide cryptographic keys for critical functions, it must meet the highest security standards for data security and privacy compliance.

- European Union's General Data Protection Regulation
- PCI Data Security Standard
- Domain Name System Security Extensions (DNSSEC)
- FIPS 140-3
- Common Criteria

To comply with regulations such as the General Data Protection Regulation (GDPR), an HSM can be used to provide the necessary security controls to ensure platform integrity.

In addition, an HSM must be free of malware and viruses, and protected from exploits and unauthorized access by restricting access via network interfaces. A key feature of an HSM is tamper-resistance. This is important to achieve the required level of security. The entire design of an HSM protects and hides cryptographic information, providing a trusted environment.

6.1.2 HSM vs Software Cryptographic Providers

HSMs offer several advantages over software cryptographic providers, including:

- Secured key management process
- Increased system throughput
- Strong key generation
- Easy to integrate, configure and use

[7]

6.1.3 Use-Cases

The Hardware Security Module has many use cases, namely:

- Cloud and Containers
- Public Key Infrastructures
- Privilege Access and Secret management
- Encryption and Tokenization
- Key management
- Digital Signing
- TLS/SSL Applications
- Identity and Authentication
- Payment

Critical Applications

Another purpose of using an HSM is to separate the cryptographic operations from the code. Not separating cryptographic operations from ordinary operations could allow attackers to interfere with cryptographic operations.

For applications such as online banking, this is particularly important.

6.2 PKCS

PKCS#11 is a standard API used for cryptographic operations on the HSM.

The figure below visualizes the communication flow at a high level.

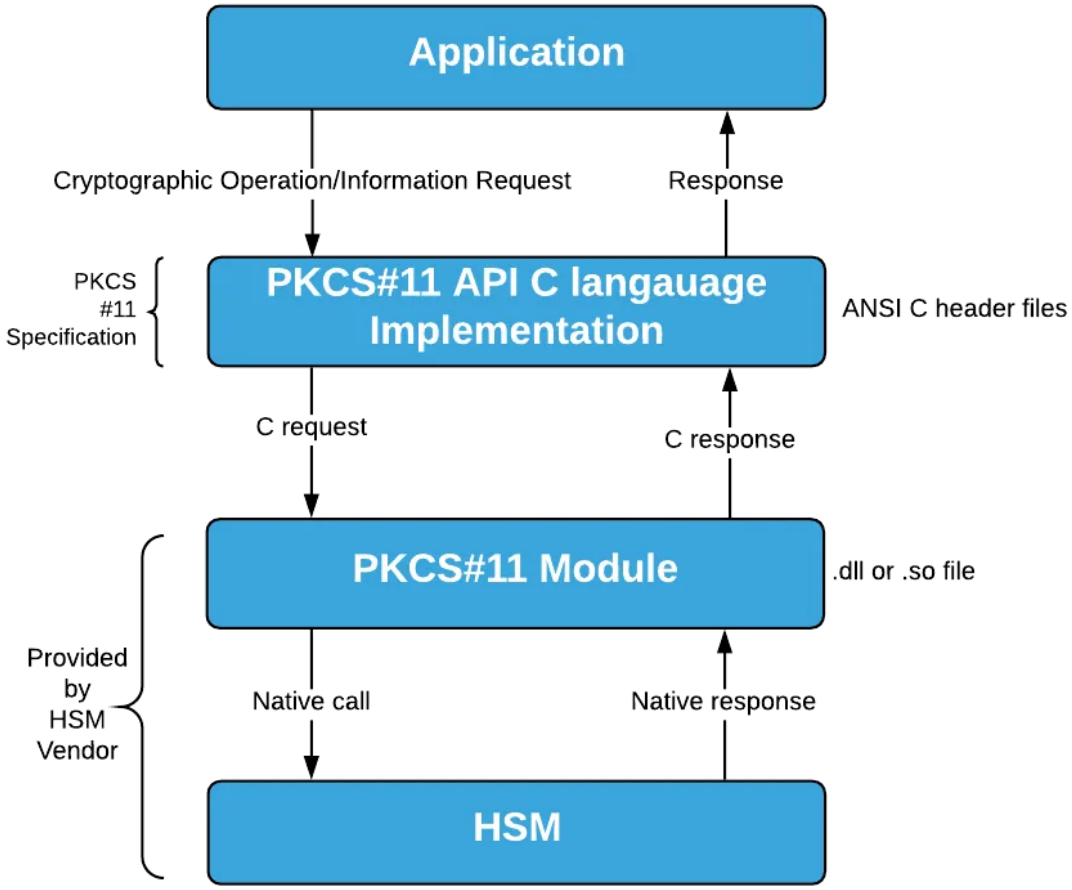

Figure 6.1: PKCS Communication Flow

Applications include the PKCS#11 API and interact with the HSM via API calls. [8]

6.3 Algorithms

The following describes the post-quantum safe algorithms used to provide some basic information about the new algorithms.

6.3.1 CRYSTALS-Dithilium

Dilithium is a new digital signature scheme proposed by an international consortium led by IBM researchers. It is strongly secure based on the hardness of lattice problems over module lattices and has been selected as the signature standard for Post-Quantum Cryptography (PQC) defined in the NIST post-quantum cryptography project. This algorithm is recommended by the CRYSTALS team to be used in the following ways [9]:

- Use Dilithium in a so-called hybrid mode in combination with an established "pre-quantum" signature scheme.
- Use the Dilithium3 parameter set, which — according to a very conservative analysis — achieves more than 128 bits of security against all known classical and quantum attacks.

and like this from CNSA 2.0 [10]:

- Use the Dilithium level five parameters for all classification levels.

6.3.2 CRYSTALS-Kyber

Kyber is a new IND-CCA2-secure Key Encapsulation Mechanism (KEM), whose security is based on the hardness of solving the Learning With Errors (LWE) problem over module lattices proposed by IBM. It is one of the candidate algorithms for the NIST post-quantum cryptography project. This algorithm is recommended by the CRYSTALS team to be used in the following ways [11]:

- Use Kyber in a so-called hybrid mode in combination with established "pre-quantum" security; for example in combination with elliptic-curve Diffie-Hellman.
- Use the Kyber-768 parameter set, which — according to a very conservative analysis — achieves more than 128 bits of security against all known classical and quantum attacks.

and like this from CNSA 2.0 [10]:

- Use the Kyber level five parameters for all classification levels.

6.4 Application Delivery Controller (ADC)

An Application Delivery Controller (ADC) is deployed to support applications with network protocols. The ADC handles a large number of algorithms to deliver the unencrypted data received over the network to the target application. [12]

6.5 TLS/SSL Offload

TLS Offload describes the process of removing Transport Layer Security (TLS) encryption from the incoming traffic and forwarding the decrypted traffic to the internal server. The device that handles the TLS encryption and the handshake is called ADC and is described in the Application Delivery Controller (ADC) section. This frees up processing power for the intended application or website.

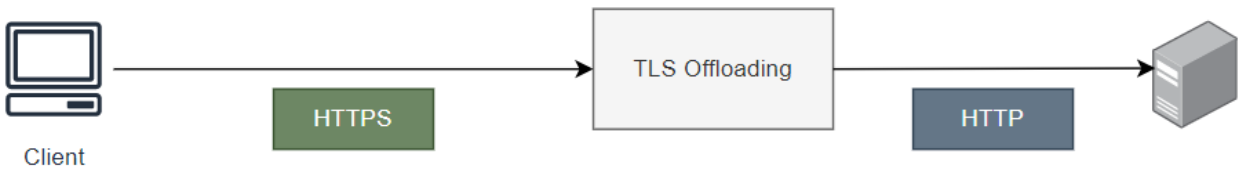

Figure 6.2: TLS/SSL Offload

6.5.1 TLS/SSL Offload to HSM

To reduce the risk of key compromise, an HSM can be used to perform the TLS encryption and decryption. This is more secure, as the keys are located in a secure environment. [13]

In the PKI use case this was also done. The creation of the dilithium keys is not done on the host running openssl. Rather, the request to create the key is passed from openssl to the HSM. The key is then generated on the hsm. This approach significantly increases security.

6.6 TLS/SSL Bridging

TLS Bridging is similar to TLS Offloading. The main difference is that the message is re-encrypted as it leaves the ADC. [14]

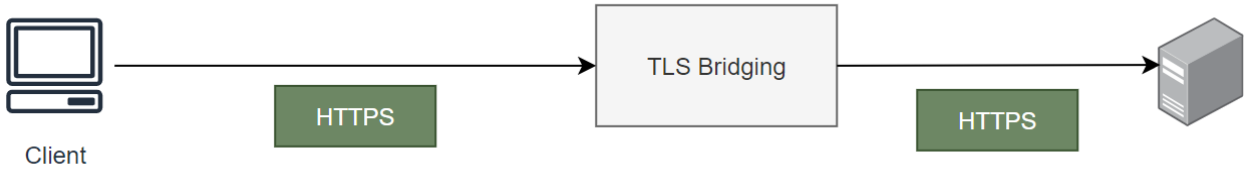

Figure 6.3: TLS/SSL Bridging

6.6.1 Benefits

Traffic Inspection: The bridge server can run malware detection and antivirus software to inspect the packets before they enter the end server. [15]

Access Control: In online banking systems, TLS Bridging can provide access control based on user identity. [15]

6.7 OpenSSL

OpenSSL is a widely used open-source cryptography and secure communication toolkit. It is commonly used to implement secure communications for web applications, email, and other types of networked communication. It is also used to secure and manage digital certificates, which are used to verify the identity of websites and other digital entities. [16]

6.7.1 OpenSSL Engine

The OpenSSL Engine API is a low-level interface used for adding alternative implementations of cryptographic primitives. However, the Engine interface is deprecated in OpenSSL version 3.0. The Provider API is recommended instead. [17]

In this project, the Engine is used to implement custom OpenSSL functions. The reason for this is that the Providers are much more complex and our advisors have been using Engine until now. One example is a custom signing function that is used to send the signing request directly to the HSM, rather than executing it in an insecure environment like on the server.

6.7.2 OpenSSL Provider

OpenSSL Providers can be used to provide custom algorithm implementations and operations, such as encryption, decryption, and signing [18]

When using cryptographic algorithms via the high-level APIs, a provider is selected. This provider then provides the custom algorithm implementation. [19]

6.8 Liboqs

Liboqs is part of the Open Quantum Safe (OQS) project. Liboqs is an open-source C library for quantum-safe cryptographic algorithms. It includes the two cryptographic algorithms used in this thesis: Dilithium and Kyber.[20]

Chapter 7 Quality Measures

The quality measures focused on in this project are defined in this chapter.

7.1 Definition of Done (DoD)

We define a task as completed when the following requirements are met and the reviewer closes the task in Jira.

Documentation:

- Grammar has been checked
- The content has been proofread and checked for accuracy
- The use of glossary entries and acronyms has been judicious

Code:

- Style guidelines were followed
- All tests have been successful
- Code has been reviewed or developed using pair programming
- There are no known critical bugs

Code and Documentation:

- The branch has been merged with the main branch

7.2 Quality Control and Assurance

This section describes the quality control and assurance process used for this project.

Table 7.1: With the help of Jira (Kanban) we defined the following states:

State (Jira Kanban)	Description
To Do	A task has been defined, but no one has started working on it.
In Progress	A task has been assigned to a team member. The team member has already started working on the task.
Needs Review	The assigned team member has completed the task. Now the task needs to be assigned to a reviewer. Comments are used to provide the reviewer with additional task-specific details. For example, the name of the branch where the task was completed should be included in the comments section.
Done	The reviewer has reviewed the task and checked the items listed in the DoD section. If everything is OK, the task can be closed.

Chapter 8 Signatures

Part of this thesis is the implementation of the quantum-safe signature scheme CRYSTALS-Dilithium to create digital certificates and signatures. In this chapter, the relevance of this topic is explained to give a better overview of the implications of the results.

Signatures are used in different use cases and at different levels of a system using cryptographic operations. It is relevant to the implementation of the integrity, authenticity and non-repudiation of the security standard.

8.1 What is a Signature?

A signature can be created by using a key pair. The sender calculates the hash of a message and signs it with their private key, the message is now signed. The private key is known only to the sender, while the public key is known to everyone. Anyone who receives this signed message can now calculate the hash of the original message and use the signed hash to verify it with the sender's public key. The hash obtained from the calculation and the signature should now match. If not, the message has been tampered with during transmission. The figure Signature Workflow visualizes this process.

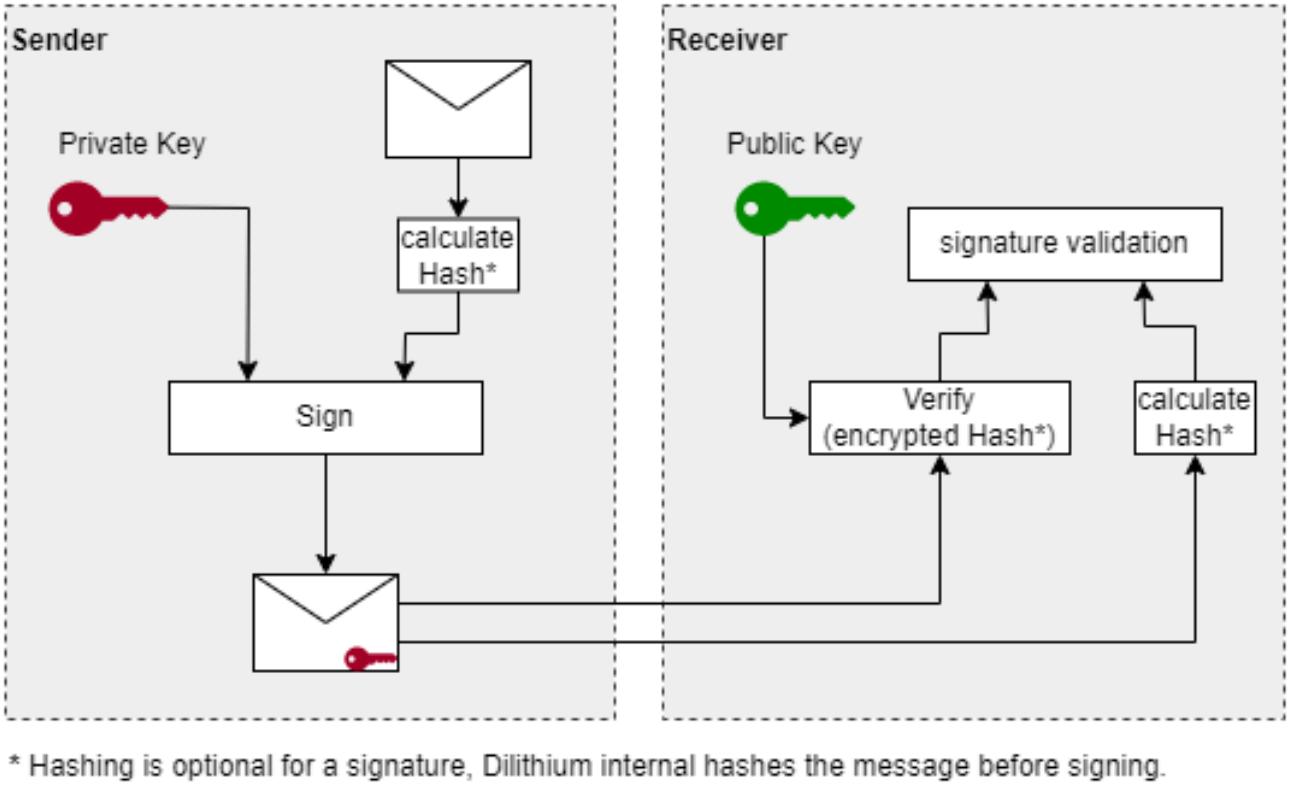

Figure 8.1: Signature Workflow

8.2 Relevance of Signatures

Signatures provide integrity, authentication and non-repudiation.

- Integrity: A recipient can be confident that the message has not been altered. [21]
- Authentication: A recipient can be confident that the message originates from the sender. [22]
- Non-repudiation: The sender of the message receives proof of the delivery. The recipient can be confident that the message originates from the sender. [23]

This thesis demonstrates the transition from a non-quantum safe signature scheme to CRYSTALS-Dilithium, a post-quantum safe signature scheme. To demonstrate this, a Public Key Infrastructure (PKI) is implemented using this new algorithm.

8.2.1 Recommendation for Key Management

Especially for critical services such as online banking, the process of periodically rotating keys and renewing the certificates is of high importance. According to NIST, short crypto periods enhance security. However, short crypto periods also have drawbacks. NIST states that more frequent key changes also increase the risk of key exposure. Another reason for long-lived private keys is that Internet-facing services and applications need to maintain a continuous and secure connection. Frequent renewal of private keys can disrupt services.

Nevertheless, the security of the keys should be checked on a regular basis and the keys should be renewed or replaced if necessary. [24]

Chapter 9 Project Structure

This chapter details the project structure of the SM-App and the Host-App to give a better overview, of where to find the files.

9.1 SM-App

An SM-App project is divided into these four main folders:

- **libs:** Contains all libraries and a Makefile to build them.
- **liquidsec_api:** Contains header files and shared functionality provided by Marvell. This is a submodule. The corresponding GitHub repository is: liquidsec_api.
- **src:** Contains the code to run the application.
- **tests:** Contains all the automated tests for the application.

libs contains the following files and directories:

- **[library]:** A folder for each library with the corresponding name. For example, the folder called "kyber".
- **Makefile:** This Makefile will build all the libraries in this folder.

src contains the following files and directories:

- **extra:** This folder contains the **app.cfg** file needed to properly configure the application.
- **api.h:** This file contains all the defined requests and responses needed to communicate with the SM-App.
- **main.c:** This file contains the "main" and the logic to parse the arguments used to start the program, if necessary.
- **Makefile** This Makefile builds the SM-App. Dependencies (libraries) must be built before this Makefile is executed.

There are other files in this folder that contain application logic, but these are application specific.

tests contains the following files and directories:

- **project.yml:** This file configures the test framework(ceedling).
- **test_[filename]:** This folder contains tests for the specified file. Each file in this folder tests a different function and is named accordingly.
- **mockable_headers:** This folder contains custom mock header files. Using one of these files can simplify the mocking of a dependency, as a file in this folder contains only the bare minimum to define a function, as opposed to the real counterparts that define multiple functions and require other header files to define all the types used.

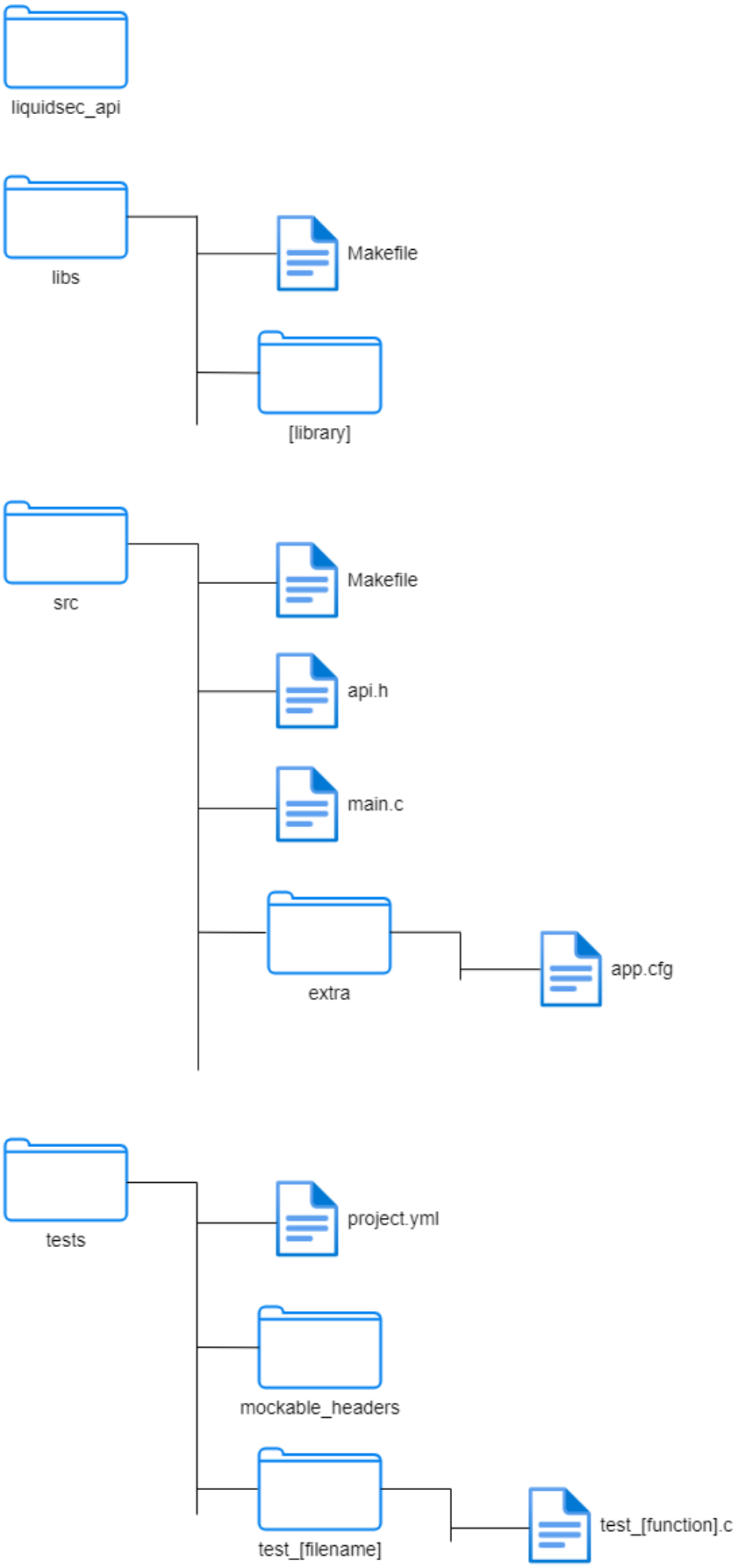

Figure 9.1: SM-App structure

9.2 Host-App

A Host-App project is divided into these four main folders:

- **libs:** Contains all libraries and a Makefile to build them.
- **liquidsec_api:** CContains header files and shared functionality provided by Marvell. This is a submodule. The corresponding GitHub repository is liquidsec_api.
- **src:** Contains the code to run the application.
- **tests:** Contains all automated tests for the application.

libs contains the following files and directories:

- **[library]:** A folder for each library with the corresponding name. For example, the folder called "kyber".
- **Makefile:** This Makefile will build all the libraries in this folder.

src contains the following files and directories:

- **api.h:** This file contains all defined requests and responses needed to communicate with the SM-App.
- **main.c:** This file contains the "main" and the logic to parse the arguments used to start the program, if necessary.
- **Makefile** This Makefile builds the Host-App. Dependencies (libraries) must be built before this Makefile is executed.

Other files in this folder contain application logic, but these are application specific.

tests contains the following files and directories:

- **project.yml:** This file configures the test framework(ceedling).
- **test_[filename]:** This folder contains tests for the specified file. Each file in this folder tests a different function and is named accordingly.
- **mockable_headers:** This folder contains custom mock header files. Using one of these files can simplify the mocking of a dependency, as a file in this folder contains only the bare minimum to define a function, as opposed to the real counterparts that define multiple functions and require other header files to define all the types used.

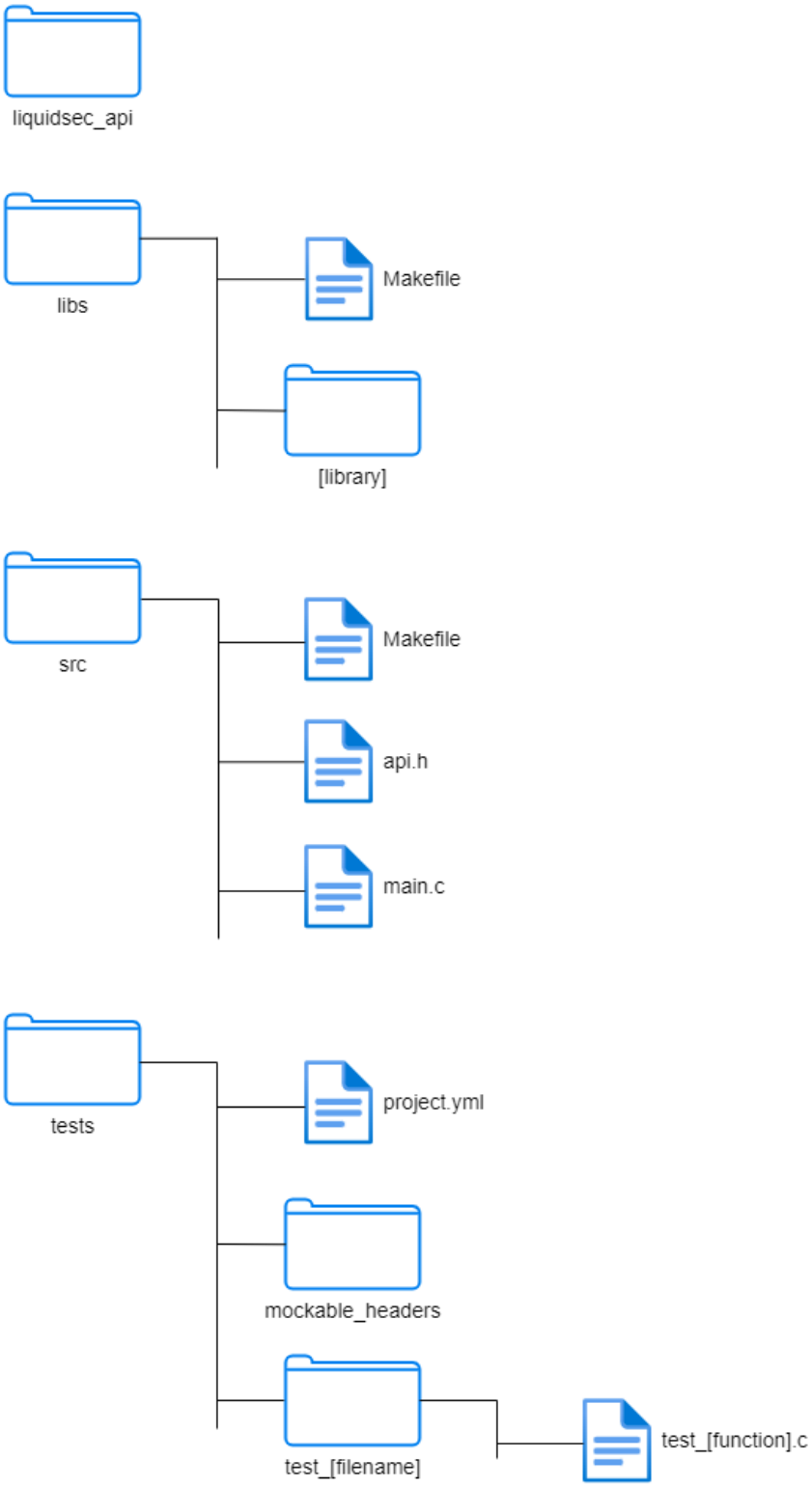

Figure 9.2: Host-App structure

9.3 liquidsec_api

liquidsec_api is a repository of shared functionality and header files provided by Marvell that is included as a submodule in the Host-Apps and SM-Apps for the implemented use cases.

It is split into these four main folders:

- **hostapp_common:** This folder contains the unmodified source code provided by Marvell that is required by each Host-App. These files define functions, for example, to perform a handshake with the SM-App or to initialize a Host-App.
- **smapp_common:** This folder contains the source code provided by Marvell with only a few changes required by each SM-App. These files define functions, for example, to perform a handshake with the Host-App or to initialize an SM-App. The only changes made to these files are minor refactoring.
- **include:** This folder contains headers provided by Marvell that are required to build a Host-App or SM-App.
- **wrappers:** This folder contains wrappers for functions that are commonly used by the Host-App and the SM-App. This includes functions for allocating memory, for example. Wrappers have been written to make otherwise not testable functions testable when using a testing framework like Ceedling.

Where these Marvell-provided files were copied from, and more information about using this repository for a Host-App or SM-App, can be found in the GitHub repository.

Chapter 10 Proof of Concept: Dilithium on Hardware Security Module

At the beginning of this thesis, it was uncertain whether the CRYSTALS-Dilithium algorithm could run on the MIPS architecture of the Hardware Security Module (HSM). This was identified as a major risk, as this algorithm was to be used for parts of the project later. Therefore, it had to be proven to work first so that there would still be time to find other solutions in case of incompatibility. All the code, a guide to setting up and running this Proof of Concept (PoC) and run it can be found in the PoC Dilithium GitHub repository.

10.1 PoC

This application tests if and how Dilithium can be used on the Secure Machine (SM) by processing a request to sign a message and sending the signed message as a response. To test if Dilithium works the signature is validated before the SM-App sends the response to the Host-App.

10.2 Setup

The PoC is based on the setup described in the figure PoC Dilithium on HSM Setup. The implementation of the CRYSTALS-Dilithium algorithm is placed in the HSM. On the other hand, there is the Host-App, which is only responsible for requesting cryptographic operations. In this case, it requests the algorithm to sign a test message.

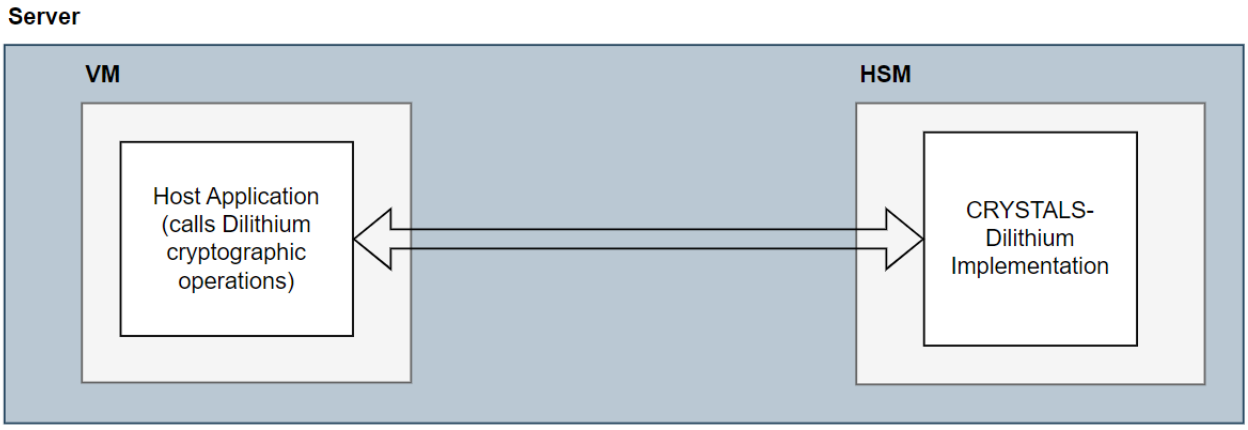

Figure 10.1: PoC Dilithium on HSM Setup

10.2.1 CRYSTALS-Dilithium Implementation

We decided to use the CRYSTALS-Dilithium implementation from the official GitHub repository. The Dilithium3 implementation is used as a base for further implementations and customizations in this project, as requested by IBM. The difference between the Dilithium versions is the security level, which is given by the key length used.

10.3 SM-App Project Structure

The SM-App is divided into these three main folders:

- **include:** Includes all header files for easier local development
- **sm-dilithium:** Contains the code to run the application on the Secure Machine (SM)

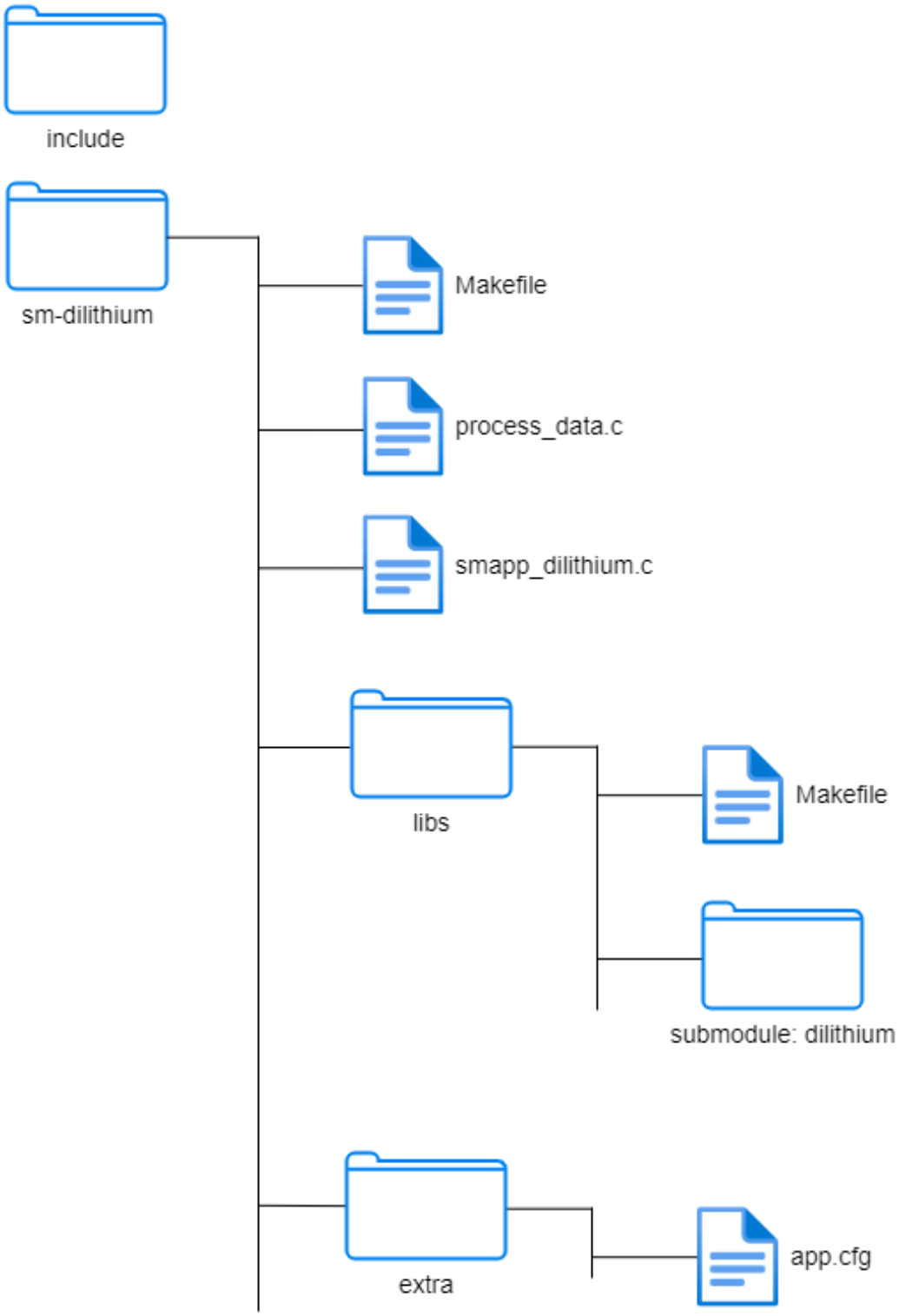

Figure 10.2: Folder Structure and most important files (SM-App)

10.3.1 Application

The application code is located in the **sm-dilithium** folder, which consists of the following files/folders:

- **Makefile:** This is the Makefile that builds the application. The other Makefiles e.g. for building the Dilithium library, are called from this one.
- **process_data.c:** This file contains the logic to sign a message and verify a signature with Dilithium.
- **smapp_dilithium.c:** This file contains the logic to communicate with the Host-App.
- **libs:** This folder contains the Dilithium submodule and the Makefile to build the library
- **extra:** This folder only contains the **app.cfg** file to configure the application

The Makefile in libs is needed to build a static library from the Dilithium submodule, even though the Dilithium submodule contains a Makefile to build a shared library. The reason for this is that it needs to be cross-compiled to be compatible with the Secure Machine (SM). Another aspect is that an SM-App using a static library is much easier to load on the SM than one using a shared library, so a static library is preferred.

10.4 Host-App Project Structure

The Host-App consists of only a few files. The most important ones are:

- **Makefile:** This is the Makefile that builds the application. It is the same as the one used in the bit-flip example except for a change in the name of the Makefile target.
- **SmTestDilithium.c:** This file contains all logic. It is mostly the same code as the Host-App from the bit-flip example. The only changes are to the data sent to the SM-App, which is then signed.
- **include:** This folder contains several header files for easier local development.

The folder structure is as follows:

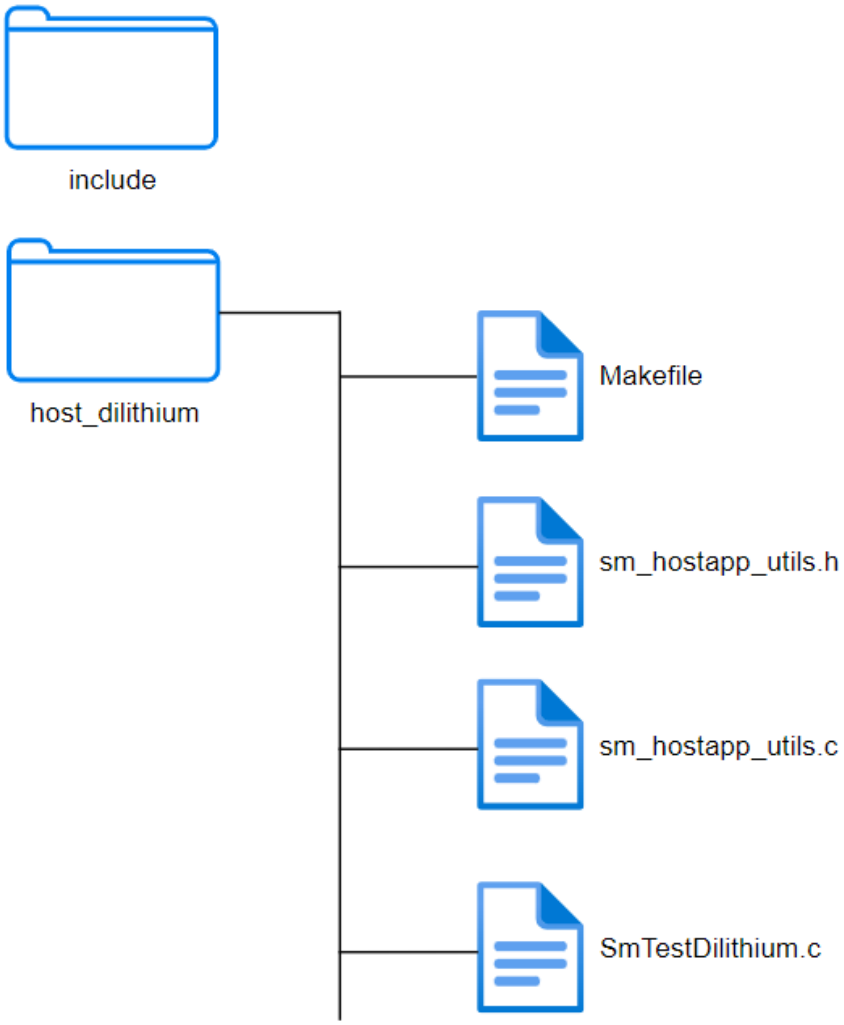

Figure 10.3: Folder Structure and most important files (Host-App)

10.5 Process

To test whether the PoC is successful or not, the following steps are performed by the Host-App and the SM-App. The Dilithium algorithm is implemented on the SM-App to generate a key pair, sign a message and then verify the signature. If the validation is correct at the end, the PoC works as expected.

Table 10.1: Process PoC Dilithium

Host-App	**SM-App**
1. Setup connection to SM-App	1. Setup connection to Host-App
2. Send a message to be signed to the SM-App	3. Create Dilithium3 key pair
	4. Sign message using the generated Dilithium3 keys
	5. Validate the signature
	6. Send the signed message back to the Host-App
7. Output the signed message to the console	

10.6 Success Specification

To validate that the PoC is working correctly, the following requirements were checked:

1. The Dilithium code should be able to be built successfully.
2. The Host-App should be able to send a request to the HSM.
3. The Host-App should receive the signed response for the request.
4. After signing the request on the HSM the signature must be valid. (This was checked on the HSM before the response was sent to the Host-App)

10.7 Result

The PoC for Dilithium on the HSM has been implemented and tested as described. The described success specification defines the individual points that must be fulfilled in order to test the compatibility between the algorithm and the HSM. With the implementation described above, all specifications were met. The communication between the Host-App and the SM-App was also tested and defined as successful. This concludes the PoC.

Chapter 11 Proof of Concept: Kyber on Hardware Security Module

At the beginning of this work, it was uncertain whether the CRYSTALS-Kyber algorithm could be implemented on the MIPS architecture of the Hardware Security Module (HSM). This was identified as a major risk, as this algorithm was to be used for parts of the project later. Therefore, it had to be proven to work first so that there would still be enough time to find other solutions in case of incompatibility. The full code, a guide to setting up and running this Proof of Concept (PoC) can be found in the PoC Kyber GitHub repository.

11.1 PoC

This application tests if and how Kyber can be used on the SM by handling a request to flip bits in the same way as bit-flip-sm does and performing a test to see if Kyber works before sending a response. To test if Kyber works, a simple handshake is simulated inside the Secure Machine (SM) and the resulting keys are compared to see if they are the same.

The PoC is based on the sample application bit-flip-sm. Bit-flip is the simplest example program provided by Marvell. The code for bit-flip has already been rewritten to be more modular, making it a good choice for quickly adding Kyber to create this PoC. For this PoC, only an SM-App was written, since the only uncertainty was whether Kyber would run on the SM. To call the SM-App the regular bit-flip-host app can be used as the response of the SM-App is the same as the one generated by bit-flip-sm.

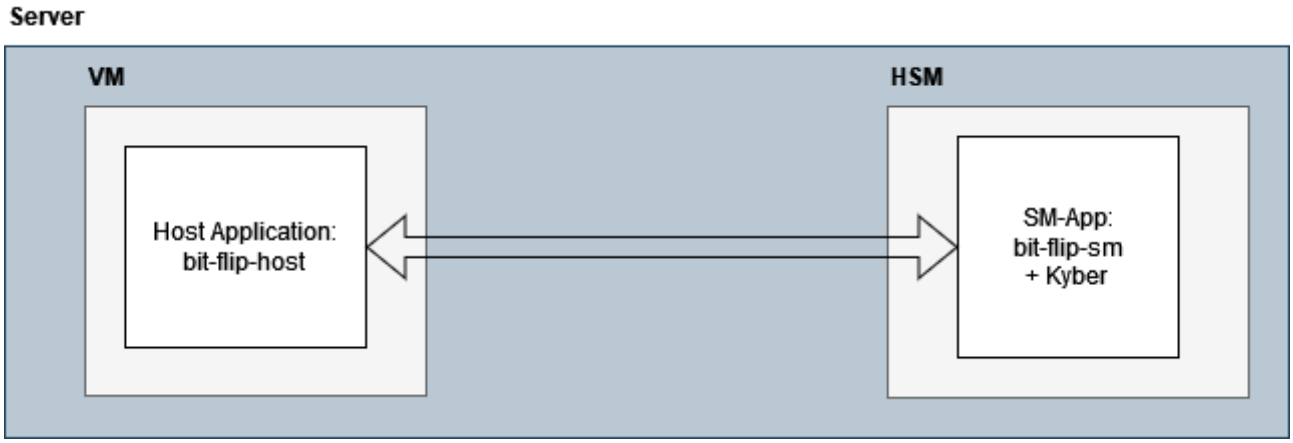

Figure 11.1: PoC Kyber Setup

11.2 Setup

11.2.1 CRYSTALS-Kyber Implementation

We decided to use the CRYSTALS-Kyber implementation from the official GitHub repository. This Kyber implementation is used as the base for further implementations and customizations in this project, as this was requested by IBM.

11.3 Project Structure

This project is divided into these three main folders:

- **include:** Contains all header files for easier local development
- **sm-bit-flip:** Contains the code to run the application

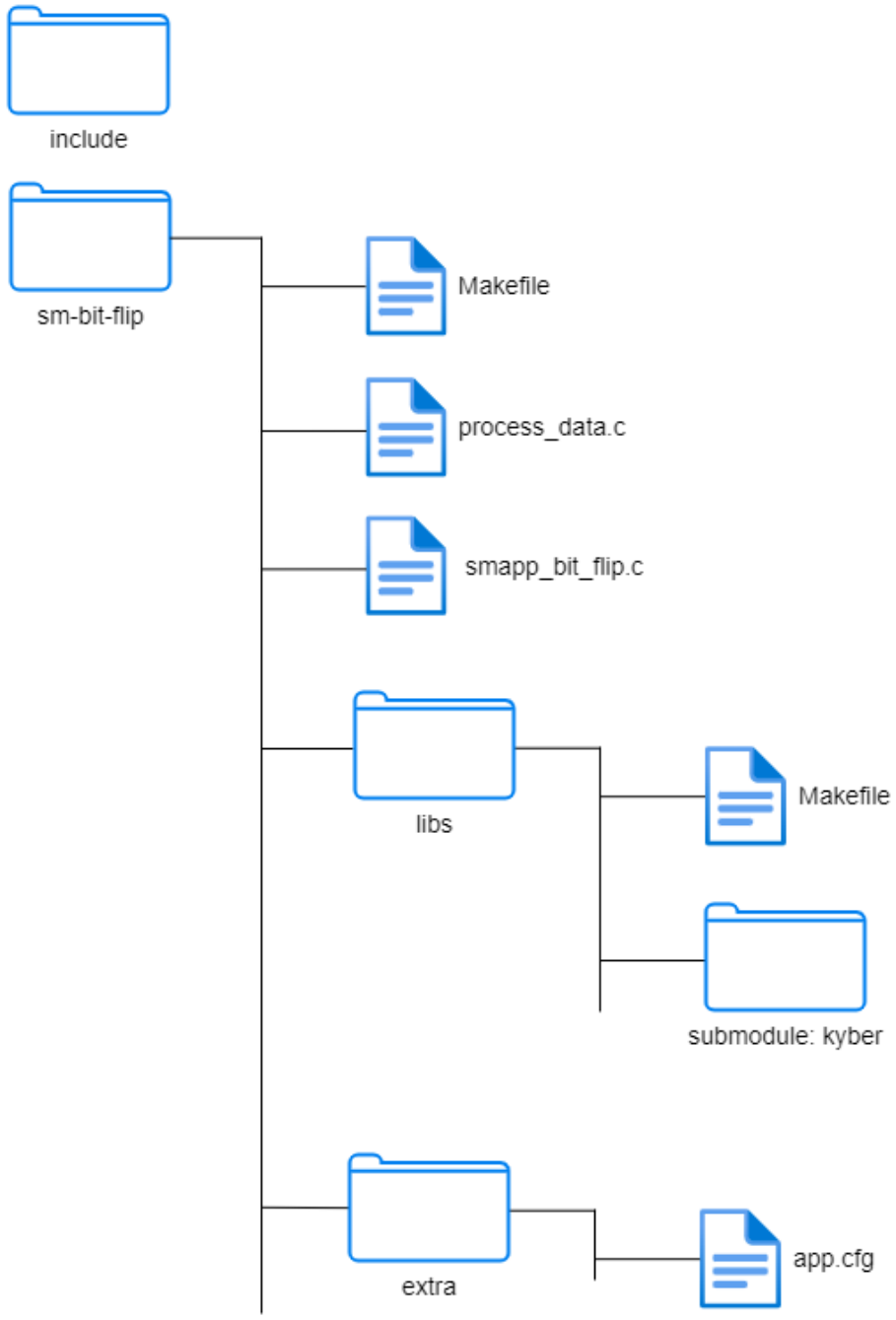

Figure 11.2: Folder Structure and most important files

Makefiles:

Makefiles are used to build the program. The reason for this is that Makefiles were used in Marvell's sample SM-Apps and could also be used to build this PoC with just a few changes.

11.3.1 Development

For easier development, there is an **include** folder that contains all the header files needed for the SM-App.

11.3.2 Application

The application code is located in the **sm-bit-flip** folder and consists of the following files/folders:

- **Makefile:** This is the Makefile that builds the application. The other Makefiles e.g. for building the Kyber library are called this one.
- **process_data:** This file contains the logic to test a Kyber handshake and to flip bits.
- **smapp_bit_flip:** This file contains the logic to communicate with the Host-App.
- **libs:** The libs folder contains the Kyber submodule and the Makefile to build the library.
- **extra:** This folder only contains the **app.cfg** file to configure the application.

The Makefile in libs is needed to build a static library from the Kyber submodule, although the Kyber submodule already contains a Makefile to build a shared library. The reason for this is that it needs to be cross-compiled to be compatible with the SM. Another aspect is that an SM-App using a static library is much easier to compile on the SM than one with a shared library, so a static library is preferred.

11.4 Process

To test whether the PoC is successful or not, the following steps are performed by the Host-App and the SM-App. The Kyber algorithm is implemented in the SM-App. Each time the SM-App receives a request, it performs a Kyber handshake with itself. After generating the shared secrets, it compares them to ensure that both parties have the same key, which indicates that the handshake has been successfully completed. If successful, the request is treated as a bit flip request and each bit of the requested message is flipped, and a log message is written by the SM-App indicating the successful processing of the request.

Table 11.1: Process PoC Kyber

Host-App	**SM-App**
1. Setup connection to SM-App	1. Setup connection to Host-App
2. Send a bit flip request to the SM-App	3. Do a Kyber handshake with itself
	4. Flip all bits in the request
	5. Send the resulting message back to the Host-App
6. Output the message with the flipped bits to the console	

11.5 Success Specification

To validate that the PoC is working correctly the following requirements were checked:

1. The Host-App should receive the same response as if bit-flip-sm was running on the SM.
2. The handshake is successful.
3. The SM logs should show the message "correct keys".

11.6 Result

The PoC for Kyber on the HSM was implemented and tested as described. The described success specification defines the individual points that must be fulfilled in order to test the compatibility between the algorithm and the HSM. With the implementation described above, all specifications were met. The communication between the Host-App and the SM-App was also tested and defined as successful. This concludes the PoC.

Chapter 12 Public Key Infrastructure (PKI) - Theory

This chapter deals with the theory regarding a Public Key Infrastructure (PKI). It serves as a basis for understanding and comprehending the following implementation.

The secure transfer of information is essential when exchanging data. One of the biggest challenges, however, is verifying that a public key belongs to the entity it is assumed to belong to. To verify the public keys origin a so-called PKI is used. A PKI can help to ensure the secure exchange of sensitive data, providing core elements of cyber security such as data confidentiality, data integrity and user authentication. With a PKI managing digital certificates in a public key cryptography scheme and ensuring that the certificate can be trusted can be achieved. [25]

12.1 Introduction to Public Key Infrastructures

This chapter serves as the basis for understanding this use case, first explaining the terminology and concepts used, and then describing how a PKI is constructed and the components required.

12.2 Public and Private Keys

Asymmetric cryptography is used to encrypt data within a PKI. The public key is used to encrypt data and the private key to decrypt data.

Asymmetric cryptography is also used to sign data within a PKI. The private key is utilized to sign the data and the public key to verify the signature.

To implement a post-quantum secure PKI, Dilithium3 was chosen to generate the keys. Dilithium version 3 was chosen because of the key length and at the request of IBM. This is for research purposes only, the standard is Dilithium version 5 (Dilithium5).

Dilithium uses a slightly different naming convention. Instead of the terms private key and public key, Dilithium uses the terms public key and secret key. [26]

12.3 Certificates

A certificate is a digitally signed document used to prove the validity of a public key. The digital certificate contains information about the key, the identity of its owner (subject), and the signature of an entity that has verified the contents of the certificate (issuer). [27]

The certificate is used to associate an identity with a public key. [28]

12.4 Chain of Trust (CoT)

The Chain of Trust (CoT), also known as Certificate Chain, describes a validation process of components from a trusted point. A CoT is often used by certificates and their validation. The trusted point is also called Root of Trust (RoT) and is also called the Root Certificate. This trusted certificate is used to sign other certificates. Any certificate that has been signed by a root certificate, or by multiple certificates that have been signed by the root certificate, is validated and can also sign other certificates. This creates a CoT.

In the following Chain of Trust diagram the keys and their relationships are depicted.

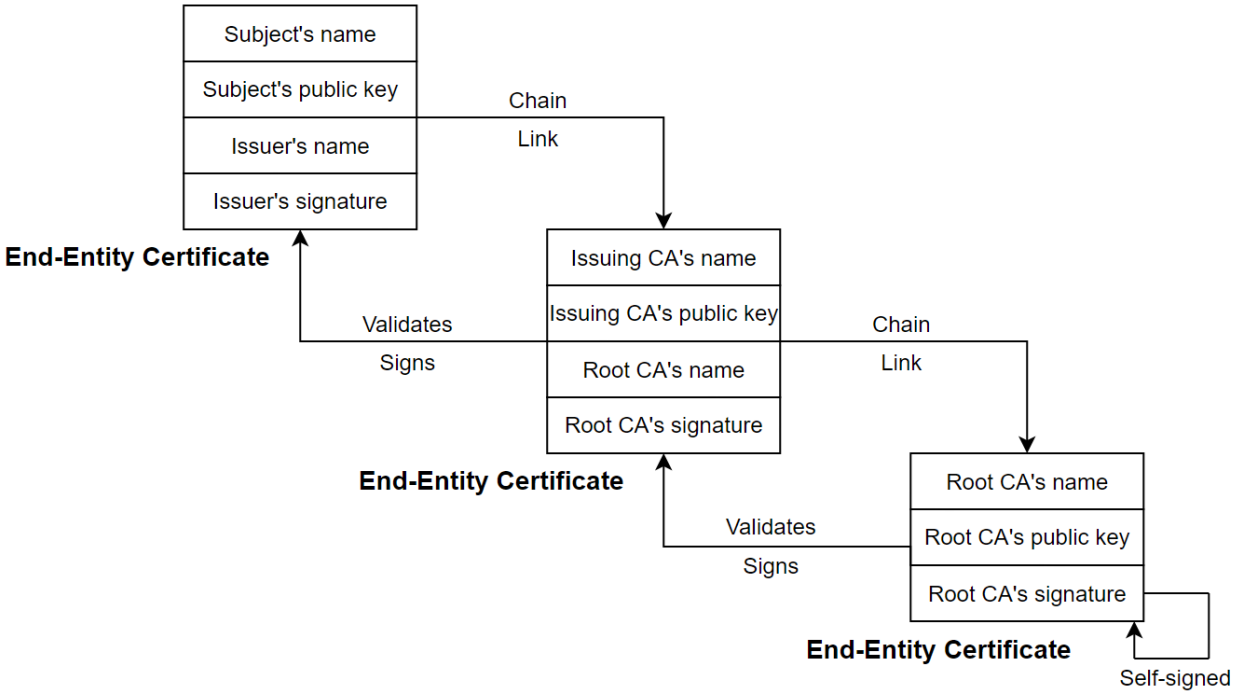

Figure 12.1: Chain of Trust (Source: [29])

12.5 Root of Trust (RoT)

The Root of Trust (RoT) is the root point in a security CoT and can validate other components. These components can then validate further components. This (CoT) can then be used to validate a component, with the validation process going backwards until a RoT is reached. This is also known as transitive trust, as a system is trusted that another system trusts that the user trusts.

12.5.1 Hardware Root of Trust (RoT)

When using cryptographic keys for authentication, encryption and decryption, it is important to protect these cryptographic keys. The Hardware Security Module (HSM) is a hardware component specified for such critical operations. The placement of the root certificate inside an HSM is called a Hardware RoT.

12.6 PKI Entities

The following components are part of the PKI.

12.6.1 Certificate Authority

The Certificate Authority (CA) is a trusted component of a PKI and can issue certificates. A CA can have different roles depending on its position in the hierarchy. [30]

Roles:

- **Root CA:** Issues Intermediate CA certificates.
- **Intermediate CA:** Issues issuing CA certificates.
- **Issuing CA:** Issues certificates to end-entities.

The next section provides an overview of a PKI hierarchy and explains the roles of a CA according to their position in the hierarchy.

12.7 PKI Hierarchy

Three different types of hierarchies can be used to set up a PKI. These are: one-tier, two-tier and three-tier hierarchies.

Due to the design of one-tier hierarchies, where a single CA acts as both Root CA and Issuing CA, this hierarchy is not recommended for production scenarios. Not separating the roles of the CAs poses a security risk, as a compromise of this single CA leads to a compromise of the entire PKI. This is the reason, why the one-tier hierarchy is not described in more detail in this thesis. [31]

The two terms CRL and OCSP that appear in the figures below are described in detail in the linked sections.

12.7.1 Two-Tier Hierarchy

The two-tier hierarchy includes an `Offline Root CA` and `Issuing CAs`. The `Issuing CAs` are authorized by the `Offline Root CA`. [32]

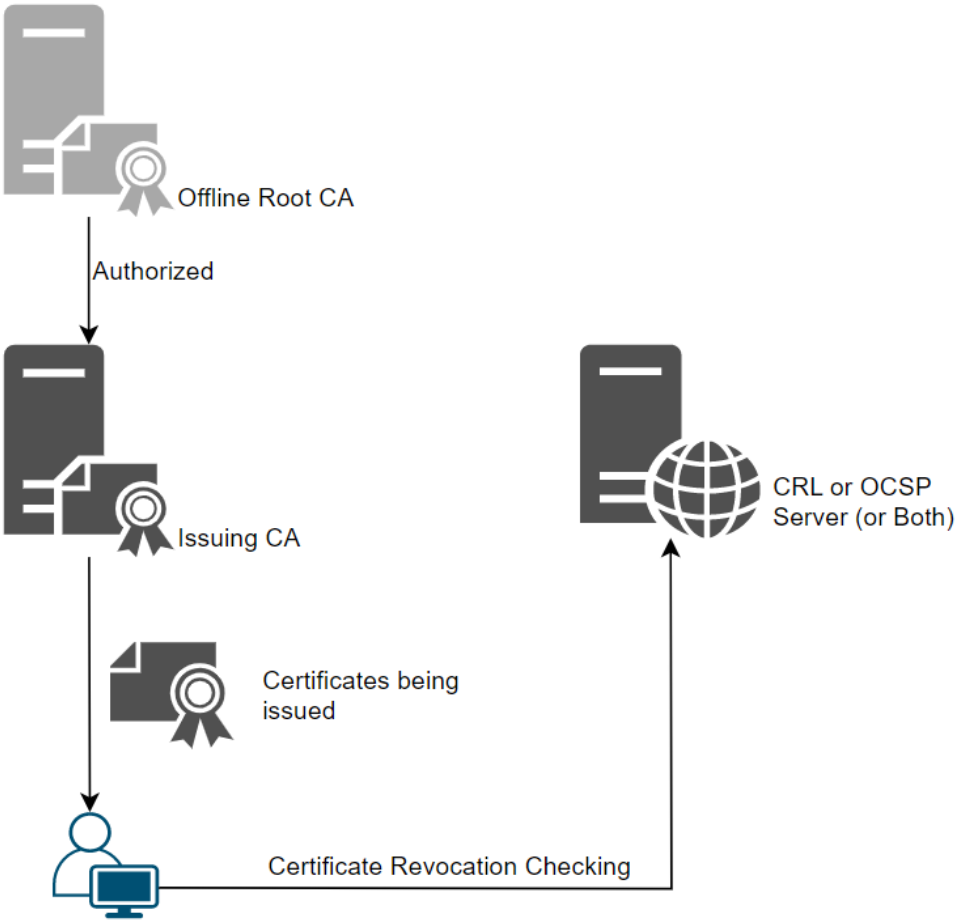

Figure 12.2: PKI Two-Tier Hierarchy

12.7.2 Three-Tier Hierarchy

The three-tier hierarchy includes the `Offline Root CA`, the `Intermediate CA` and the `Issuing CA`. In this case, the `Intermediate CA` is authorized by the `Offline Root CA` and the issuing CA is authorized by the `Intermediate CA`. [32]

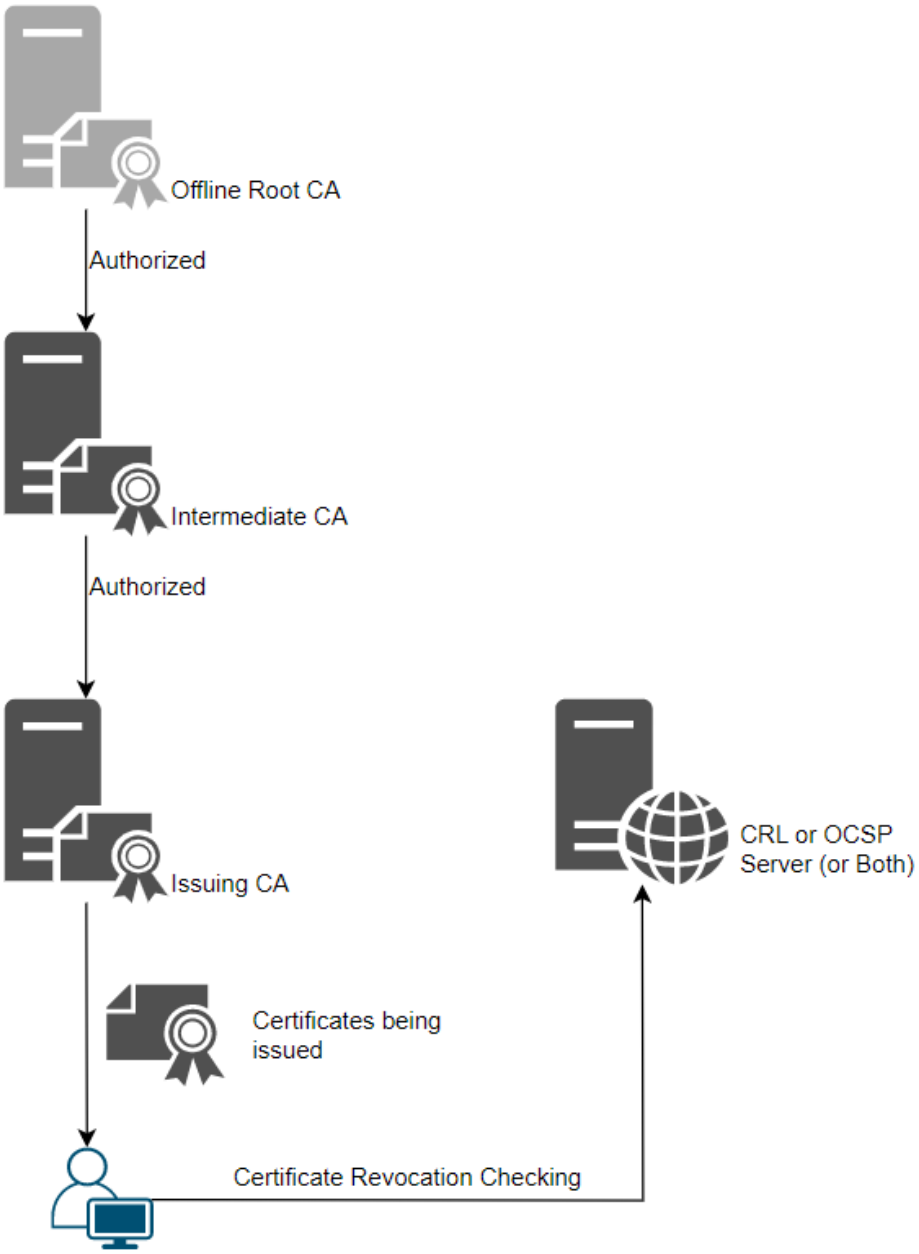

Figure 12.3: PKI Three-Tier Hierarchy

12.8 Online Certificate Status Protocol (OCSP)

Online Certificate Status Protocol (OCSP) is a protocol for checking whether a certificate is still valid or has been revoked. The reason for revoking a certificate is if the private key is stolen or accidentally exposed, because the signature on a corresponding certificate is only secure if the private key is known to only one instance.

The certificate is checked against the issuer's Certificate Revocation List (CRL). As there are many certificates in use around the world, these lists are long. Instead of checking the Certificate Revocation List (CRL) itself, the OCSP is used to check the status of a certificate by sending a request to the vendor. [33]

Online Certificate Status Protocol (OCSP) will not be used in the Public Key Infrastructure (PKI) use case because of the requirement of the online availability of the Certificate Authority (CA) which is not the case in this implementation.

12.9 Certificate Revocation List (CRL)

The Certificate Revocation List (CRL) is a list derived from the CA of a certificate. If the private key of a certificate is published, the corresponding signature is no longer secure. To spread this information and allow users to check if a certificate is invalid, a so-called CRL is available online. If a certificate is listed there, the certificate and its signature are no longer secure. [34]

Certificate Revocation List (CRL) will not be used in the Public Key Infrastructure (PKI) use case because of the timelimitation. In a further development process the CRL will be used.

12.10 Certificate Signing Request (CSR)

When an entity wants to obtain a certificate, it must first create what is called a Certificate Sign Request (CSR) to request the certificate. This CSR is created on the server where the certificate will be installed. [35]

12.10.1 Information of CSR

The CSR contains several pieces of information used to create the certificate, such as the following. [36]

- Information about the company and the website
- Public key (which will be placed in the certificate later)
- Information about the key type and key size (e.g. RSA 2048)

The CSR is usually based on a base64 Private Enhanced Mail (PEM)-format. This format is described in detail in the Private Enhanced Mail (PEM) section below.

12.11 Private Enhanced Mail (PEM)

The Private Enhanced Mail (PEM) format contains a single public certificate or an entire certificate chain. This file format allows an entire certificate chain to be installed directly from a single file. It is the most common format for X.509 certificates, CSRs and cryptographic keys. All elements are base64 ASCII encoded with plain-text headers and footers such as `---BEGIN CERTIFICATE---` and `---END CERTIFICATE---`. For PEM files there are various file extensions such as `.crt`, `.pem`, `.cer`, `.key` and many more. [37]

Chapter 13 Use Case: Public Key Infrastructure (PKI)

According to the National Cyber Security Center (NCSC), there will be a period where conventional algorithms and quantum-resistant algorithms will run in parallel to sign data in Public Key Infrastructure (PKI) systems. However, the goal is to transition completely to quantum-resistant algorithms. [38]

In this chapter, a demonstration of an implementation of a post-quantum secure PKI by using the CRYSTALS-Dilithium3 Signature Scheme is described. The PKI use case consists of several components. The figure Overview PKI and the corresponding description provide an overview of the components of this use case.

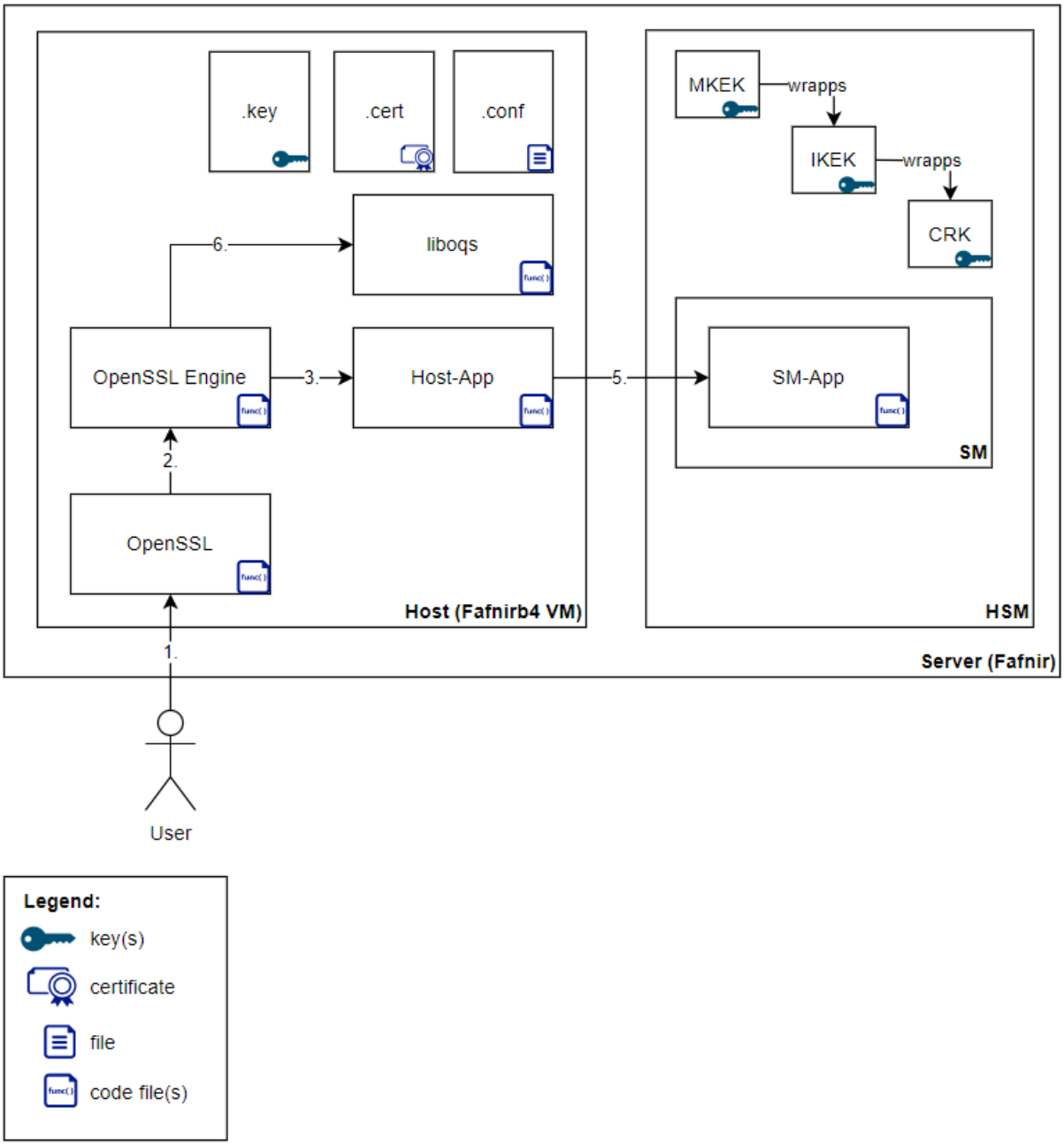

Figure 13.1: Overview PKI

1. A User uses the command line tool provided by OpenSSL to execute the required functions. In this use case, the focus is on the `key generation`, the `signature` and the `signature validation` functions. When running OpenSSL commands, a Engine must be specified to define the source of the function implementation. All of these functions are defined in OpenSSL itself, in liboqs and some of them also in the Host-App.
2. OpenSSL takes this input and forwards the functions to the specified Engine. Functions entered without an Engine are handled by OpenSSL itself.
3. In the Engine, the functions passed by OpenSSL are defined and linked to custom functions. These custom functions are defined in liboqs or the Host-App because these libraries are linked with the Engine.
4. If the function is forwarded from the Engine to the Host-App, the implemented functions (`key generation` or `signature`) are called.
5. The Host-App forwards these functions to the HSM, because anything related to secure keys should never be done outside the secure environment of the HSM. Specifically, the Host-App calls the SM-App where the cryptographic operations are performed.
6. If the function is forwarded from the Engine to the liboqs, the implemented function (`signature validation`) is called from liboqs.

The reason for using functions defined in liboqs or the Host-App instead of OpenSSL directly is that this PKI uses the newly standardized signature algorithm CRYSTALS-Dilithium for key generation and signatures. OpenSSL currently does not support this type of algorithm. Liboqs instead supports all functions of CRYSTALS-Dilithium, but in this use case the keys should only be available on the HSM and therefore the key generation and the signature, where the secured private keys are involved, have to be done directly on the SM-App, which has access to the keys on the HSM.

13.1 User

The user is anyone who uses OpenSSL with the liboqs and the Engine. In this case, a user is someone, who understands the technical documentation to use and extends the setup if needed.

13.2 Host

In this project, the host is a Virtual Machine (VM) located on the server to which the HSM is connected. The advantage of this is that there is no network in between the host and the server. In an active PKI, the hosts are distributed and connected to the server via the network.

13.2.1 OpenSSL

OpenSSL is a widely-used open-source toolkit for cryptography and secure communication. It is commonly used to implement secure communications for web applications, email, and other types of networked communications. It is also used to secure and manage digital certificates that are used to verify the identity of websites and other digital entities. In this project, OpenSSL version 1.1.1 is used to implement a Public Key Infrastructure (PKI) because of the used Engine where the OpenSSL version 1.1.1 is required.

OpenSSL can be used from the command line to perform a large number of different cryptographic operations. The extension built in this thesis uses the OpenSSL command line as user interaction.

13.2.2 Engine

The OpenSSL Engine is a low-level interface used to add alternative implementations of cryptographic primitives. In this project, the Engine will be used to implement custom OpenSSL functions. The Engine is an interface that is placed on the command line of OpenSSL. When a command is entered and the Engine implements a customized version of the OpenSSL function, OpenSSL will pass the function call to the Engine to use the customized function instead of the original OpenSSL function. In this case, OpenSSL is also extended with `liboqs`, which is an extension for post-quantum safe algorithms. This provides a subset of several newly standardized algorithms. The Engine can also replace these functions with a custom implementation, which is the case in this project.

The implemented Engine passes the following required functions to the HSM:

Key Generation Key generation for Dilithium3 is redirected to the Host-App, as this operation should be done in the secure environment of the HSM.

Create Signature The function to create Dilithium3 signatures is also redirected to the Host-App, as these operations should also be performed on the HSM.

Signature Validation To validate a Dilithium3 signature, only the public key is used. Therefore, the validation can be done directly with the liboqs implementation of Dilithium3.

13.2.3 Liboqs

Liboqs is part of the Open Quantum Safe (OQS) project. Liboqs is an open-source C library for quantum-safe cryptographic algorithms. Among other algorithms, it includes the two cryptographic algorithms: Dilithium and Kyber.[20]

In this project, the liboqs library is used for some quantum-safe cryptographic operations. The functions provided by this library regarding Dilithium are not used because Dilithium keys are stored in the HSM in this use case. Therefore, these functions are handled by the Host-App as described below.

13.2.4 Host-App

The Host-App only forwards cryptographic requests to the SM-App. The SM-App can then handle these requests. To communicate with the SM-App on the HSM, the `Cfm2Util` API is used.

13.2.5 Root CA Configuration File

The configuration file for the root Certificate Authority (CA) is located on the host because this is where we want to create the CA. To create a CA using OpenSSL, we have to create a configuration for OpenSSL to do so.

Root CA Configuration File Format

Following is a description of the different sections defined in our root CA configuration file.

[ca] This section is mandatory. It is used to tell OpenSSL to use the options from the `[CA_default]` section.

[CA_default] This section defines global constants such as the various directory and file locations for our CA generation. We also define the location of the root key pair, defaults for the certificate revocation list, expiration dates and the default message digest to use.

[policy_strict] The strict policy is applied to all root CA signatures. This ensures that the root CA is only used to sign intermediate certificates that match the defined parameters. This policy is linked in the previous section; `[CA_default]`.

[policy_loose] This policy applies to all intermediate CA signatures and allows the intermediate CA to sign a wider range of certificates, which may come from a variety of third parties.

[req] In this section the options for the OpenSSL request tool are defined. These options are applied when creating certificates or Certificate Sign Request (CSR)s.

[req_distinguished_name] The configuration in this section defines the information required in a Certificate Sign Request (CSR).

[v3_ca] This configuration is an extension for a typical CA that can be applied when signing certificates. We use it for X509 certificates with version 3.

[v3_intermediate_ca] This configuration is an extension for a typical intermediate CA for X509 version 3 certificates.

[usr_cert] We use this section to sign client certificates.

[server_cert] This section is an extension for signing server certificates.

[crl_ext] In this section the definitions for Certificate Revocation List (CRL)s can be made.

[ocsp] In this section the definitions for Online Certificate Status Protocol (OCSP) signing certificates can be made.

```
[ ca ]
...

[ CA_default ]
...

[ policy_strict ]
...

[ policy_loose ]
...

[ req ]
...

[ req_distinguished_name ]
...

[ v3_ca ]
...

[ v3_intermediate_ca ]
...

[ usr_cert ]
...

[ server_cert ]
...

[ crl_ext ]
...

[ ocsp ]
...
```

13.2.6 Root CA

Note:
For this project, IBM has agreed that only the root certificate on the HSM is required to prove the functionality of the PKI implementation. In a production environment, at least a two-tier PKI hierarchy consisting of an offline root CA and at least one intermediate, issuing CA is recommended

13.3 HSM

The Hardware Security Module (HSM) on the other side is connected to the server via an adapter. The HSM is used in the PKI system as a source of trust. The reason for this is that the HSM is usually used for high-security operations and cryptographic key storage. Therefore, the HSM can be used as a source of trust for the Dilithium keys. The keys section describes the mechanism for providing the required security standard in more detail.

13.3.1 SM-App

The SM-App resides in the SM, which is a customizable partition on the HSM used to implement functions, which are not supported by e.g. the PKCS#11#11 API.

The detailed implementation and the usage of the interface are described in the PKI Code SM-App section in the appendix.

13.3.2 Keys

The HSM stores the private keys used to encrypt the keys stored in the Keystore. These keys are needed to achieve the level of security that a PKI should provide.

HSMs manage the entire lifecycle of cryptographic keys. The LiquidSecurity HSM only supports a selection of key types and key lengths. Therefore, a cryptographic key created with Dilithium cannot be imported into the LiquidSecurity HSM. For a storage of the Dilithium keys, a CRK is used to encrypt the Dilithium key before saving it on the host.

13.3.3 Keys Prerequisites

Preparing the keys is a prerequisite before they can be used for signing. Below is a description of the different types of keys used in the environment and the relationships between them. The key generation commands are described in the Use Case Certificate GitHub repository.

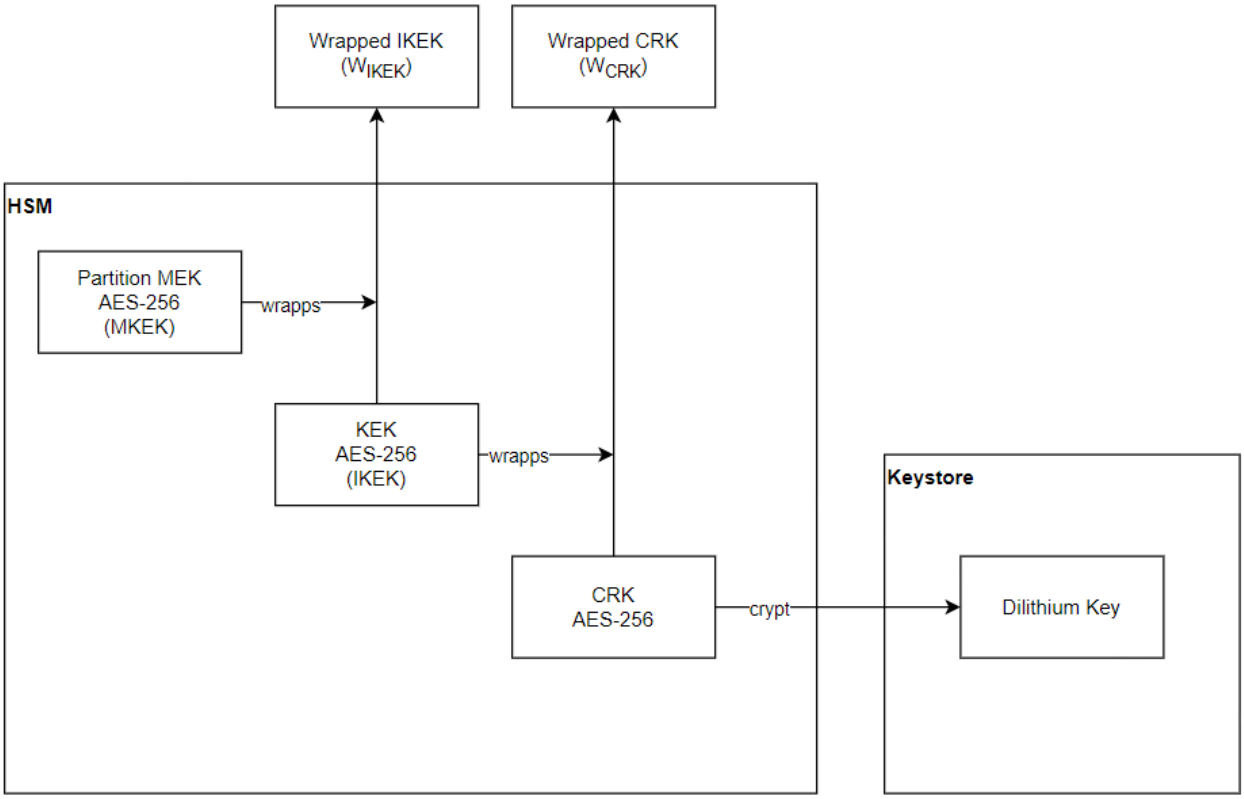

Figure 13.2: PKI Keys Prerequisites

Partition Master Encryption Key (MEK)

The Partition Master Encryption Key (MEK) is used to authenticate and sign digital certificates. In a PKI hierarchy, this is the topmost key and is protected in the secure environment of the HSM.

In the LiquidSecurity HSM, this is an AES key used to encrypt the contents of a partition. The HSM has one partition MEK for each partition.

Key Encryption Key (KEK)

The Key Encryption Key (KEK) is also called Instance Root Key (IKEK). This key is wrapped by the partition MEK. The wrapped KEK is available outside of the HSM because only the HSM itself can unwrap the key for usage.

Cryptographic Recovery Key (CRK)

This key is used for disaster recovery and is wrapped with the IKEK. Like the KEK the CRK is available outside of the HSM in wrapped form.

13.3.4 Key Usages

After the prerequisites are in place, the keys are ready to use for the PKI. This section describes how to use the keys and PKI.

The wrapped CRK key (W_{CRK}) is stored in a database outside the HSM to which the user has access. The W_{CRK} is encrypted with the KEK and cannot be used directly. To use this key, the HSM must first unwrap the wrapped CRK. This mechanism increases security because only the HSM can access the CRK (unwrapped).

The commands for this procedure are found in the Use Case Certificate GitHub repository.

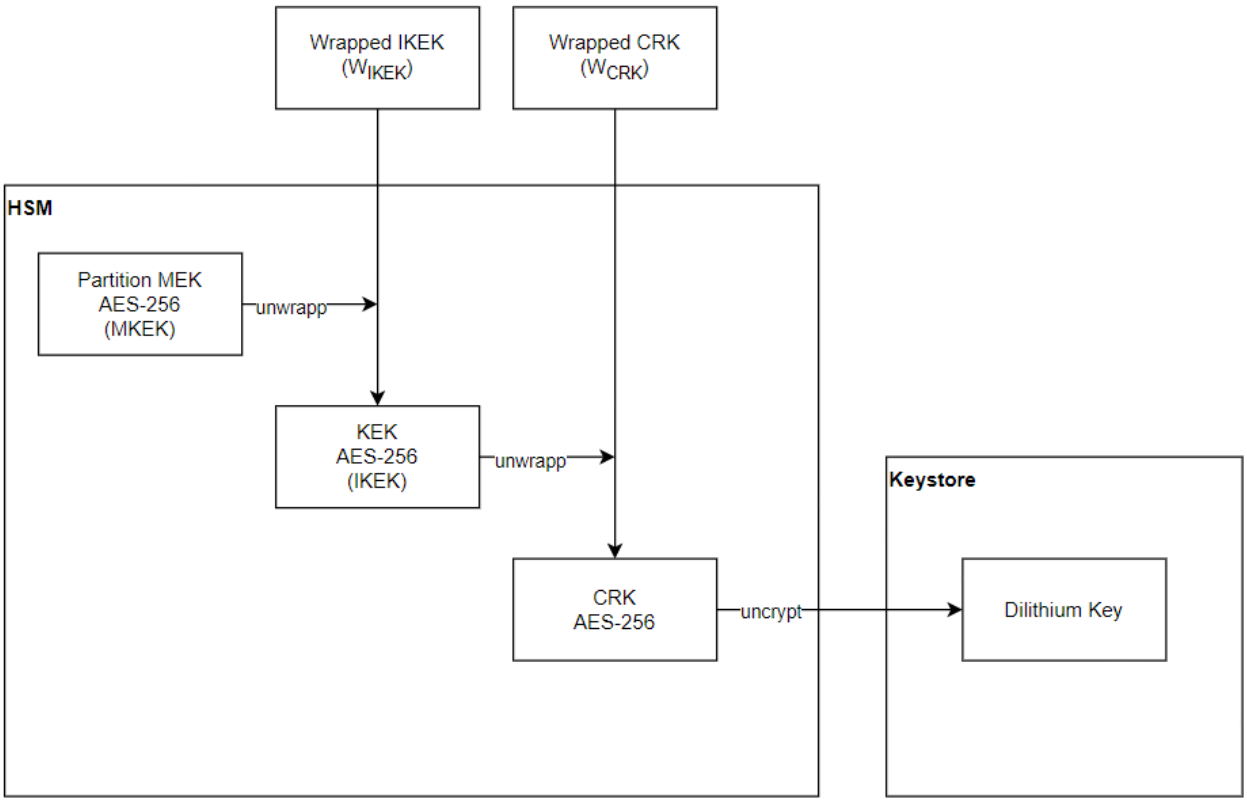

Figure 13.3: PKI Keys Usages

13.4 Challenges

The implementation of the PKI involved several challenges and decisions. The main challenges are discussed below.

13.4.1 Provider and Engine

Deciding whether to implement the OpenSSL Provider or the Engine has proved difficult. In the original specifications of our project, the implementation of the Engine was defined. OpenSSL version 1.1.1 allows the implementation of custom functionality using an Engine.

As OpenSSL version 1.1.1 is considered to be End Of Life (EOL) on the 11th of September 2023, we had to decide whether to continue with the Engine or move to OpenSSL version 3.0.0 and implement a Provider. Since OpenSSL version 3.0.0, the Engine has been replaced by the Provider. The functionality is almost identical.

The challenge here was to decide which version of OpenSSL to use. IBM already had a lot of experience with Engines and was willing to provide us with a sample code from a similar use case. In addition, examples of an engine and provider are publicly available on the Open Quantum Safe Project GitHub.

Despite being warned that it would be more complicated to implement a Provider, the decision was made. The announcement of OpenSSL version 1.1.1 being EOL was reason enough for us to take the risk.

Such an implementation requires a great deal of knowledge in the area, not only of C development but also of cryptography and the application architecture of OpenSSL. After three weeks of trying to understand available provider implementations, reading API documentation and troubleshooting, we were no closer to a working implementation.

Realizing that it would take too long to build up the knowledge to implement a working provider in such a short time, we decided to abandon this idea and implement the Engine.

An engine with a basic implementation of Dilithium2 was developed several years ago by one of our supervisors at IBM. With his knowledge and sample code, we started to implement an engine using OpenSSL version 1.1.1 to provide custom functionality for Dilithium3.

13.5 OpenSSL Changes

Another challenge to overcome was the change to the OpenSSL library. The library is still under development and changes are released regularly.

We started by implementing a suitable engine for our use case, which is based on an already existing Engine for the Dilithium2 algorithm.

Our Engine uses an OpenSSL function that redirects OpenSSL calls to our custom dilithium3 function (`keyprotect_engine_digestsign_dilithium3`). Due to the regular changes to the OpenSSL Library, we didn't realize that the function we were using to forward calls had been deprecated and replaced with a new function called `signctx`.

This change proved to be challenging as `signctx` requires different parameters. To overcome this challenge, we have implemented the new function `my_pkey_oqs_signctx`. This function takes the parameters from `signctx` and passes them one by one to the `keyprotect_engine_digestsign_dilithium3` function. The `keyprotect_engine_digestsign_dilithium3` then establishes communication with the SM-App located on the HSM and forwards the parameters to create the digital signature in the secure environment of the HSM.

13.6 Functional Requirements (FR)

The following requirements were defined with IBM.

Epics

Nr.	Title	Priority
1	Build custom App (Host & Secure Machine)	High
2	Write OpenSSL Engine	High
3	Generate keys on HSM (MKEK, IKEK, CRK)	Medium
4	Generate root CA	Low
5	CoT security measurements	Low

User Stories

In the documentation, the user stories are quite broad to increase readability. They were refined to be easier to work with, but only in Jira since that is where all the time tracking and estimation took place.

To visualize the MVP the corresponding user stories are colored yellow.

Build custom App

Prio.	Story	Done
1	As a security engineer/developer, I want to be able to use the post-quantum safe algorithm implementation on the HSM so that I can use safe algorithms in the future.	NO
2	As a security engineer/developer, I want to be able to use a post-quantum algorithm without exposing any keys at any point so that I can build secure applications.	NO
3	As a security engineer, I want to have the option of using a hybrid implementation (Kyber and RSA or Elliptic Curve) so that I have secure communication.	NO

Write OpenSSL Engine

Prio.	Story	Done
1	As a security engineer, I want to be able to forward customized functions from a post-quantum safe library to an HSM so that I can use my secured environment for development and engineering.	YES

Generate keys on HSM (MKEK, IKEK, CRK)

Prio.	Story	Done
1	As a security engineer, I want to be able to import my keys to the HSM without a length limitation so that I can use my security keys for HSM usage.	YES
2	As a security engineer, I want to be able to secure even keys without a supported key length on the HSM so that I can use different and modern algorithms.	YES

Generate Root CA

Prio.	Story	Done
1	As a security engineer, I want to be able to build a CA based on a configuration file so that I can use it multiple times.	YES
2	As a security engineer, I want to be able to secure my CA keys on the HSM so that I can be sure the keys belong to the CA owner.	NO

CoT security measurements

Prio.	Story	Done
1	As a security engineer, I want the keys to never leave the secure environment	YES

13.7 Non-Functional Requirements (NFR)

In this chapter, all NFRs are described which are valid to this use case.

NFR-1	
Requirement	The written code should be testable.
Measure(s)	Write automated unit tests, integration tests and additional Known Answer Test (KAT) (incl. code coverage checker).
Validation	The code coverage should reach either the same percentage as it was before the extension or 90%
Priority	High

NFR-2	
Requirement	The applications work on a Linux system.
Measure(s)	Test it regularly on Ubuntu 22.04 (servers at IBM).
Validation	System Tests passed expected result
Priority	High

NFR-3	
Requirement	The applications are available in English.
Measure(s)	All displayed texts are in English.
Validation	Corresponding System Test passed
Priority	High

NFR-4	
Requirement	Keys are never exposed in clear text in the memory of the Host-App.
Measure(s)	Manual code reviews
Validation	Manual code review
Priority	High

NFR-5	
Requirement	A security overview for possible threats is made.
Measure(s)	Create a threat model.
Validation	An up-to-date threat model is described in the documentation.
Priority	Medium

NFR-6	
Requirement	The system should print an error but not crash unexpectedly.
Measure(s)	Test the system multiple times with different inputs (system test).
Validation	System test run successfully with no unexpected crashes.
Priority	Medium

NFR-7	
Requirement	Users are notified of wrong or inconsistent inputs.
Measure(s)	Test invalid inputs.
Validation	Corresponding System Test passed
Priority	Medium

NFR-8	
Requirement	When an exception is thrown the application displays a meaningful error message.
Measure(s)	Exceptions are looked at during code reviews to ensure sufficient logging is done.
Validation	Exceptions are captured on the application level and printed. It is also tested if the correct error message is shown using automated tests and manual system tests.
Priority	Medium

13.8 Testing

Unit and Integration tests were written for the SM-App and the Host-App wherever sensible. To do this mocking was used as many functions require the environment of the SM or the server `Fafnirb4`. All these tests can be found in the host-app/tests and sm-app/tests folder of the Use-Case-PKI repository.

To verify that everything works together as expected these additional System tests were specified:

NOTE: Follow the steps of the following README to successfully perform the tests.

https://github.com/BA-HSM/Use-Case-PKI/tree/main/host-app

Table 13.1: System Tests PKI

Nr.	Test Name	Test Steps	Expected Results

1	Build the Engine	`cd /home/fafnirb4/ba_ost/pki/Use-Case-PKI/host-app/src`
		`make clean`
		`cmake .`
		`make`
			[100%] Built target keyprotect_engine

2	Generate a dilithium3 key	`/qsc_deployment/bin/openssl genpkey \`
		`-algorithm dilithium3 \`
		`-out dilithium3.key`
			dilithium3.key was successfully created in the current directory

3	Generate a self-signed dilithium3 certificate	`/qsc_deployment/bin/openssl req \`
		`-config ../config_files/certgen.RootCA.conf \`
		`-key dilithium3.key \`
		`-subj "/C=CH/ST=Switzerland/L=Zurich/O=IBM Research ZRL/OU=IBM ZRL Root/CN=qscsandbox.root.com" \`
		`-new \`
		`-x509 \`
		`-days 7300 \`
		`-extensions v3_ca \`
		`-out dilithium3.cert`
			dilithium3.cert was successfully created in the current directory.

4	Generate a self-signed dilithium3 certificate, providing an unavailable RootCa.conf file	`/qsc_deployment/bin/openssl req \ -config ../config_files/certgen.unavailableRootCA.conf \ -key dilithium3.key \ -subj "/C=CH/ST=Switzerland/L=Zurich/O=IBM Research ZRL/OU=IBM ZRL Root/CN=qscsandbox.root.com" \ -new \ -x509 \ -days 7300 \ -extensions v3_ca \ -out dilithium3.cert`	Error is displayed: "Can't open certgen.unavailableRootCA.conf for reading, No such file or directory".
5	Generate a self-signed dilithium3 certificate with an unavailable extension	`/qsc_deployment/bin/openssl req \ -config ../config_files/certgen.RootCA.conf \ -key dilithium3.key \ -subj "/C=CH/ST=Switzerland/L=Zurich/O=IBM Research ZRL/OU=IBM ZRL Root/CN=qscsandbox.root.com" \ -new \ -x509 \ -days 7300 \ -extensions v4_ca \ -out dilithium3.cert`	Error is displayed: "Error Loading extension section v4_ca".
6	Extract the public key from the certificate	`/qsc_deployment/bin/openssl x509 -pubkey -noout -in dilithium3.cert > dilithium3.pub`	The public key is extracted and written to dilithium3.pub.
7	Sign	`sudo /qsc_deployment/bin/openssl dgst \ -sha256 \ -binary \ -sign dilithium3.key \ -keyform ENGINE \ -engine /home/fafnirb4/ba_ost/pki/Use-Case-PKI/host-app/src/libkeyprotect_engine.so \ -out hugobinary.sig \ keyprotect_engine.c`	First 64 bytes of secret key, message, and signature are displayed. Connection to HSM is terminated.

8	Sign without working connection to the SM-App (e.g wrong password)	`sudo /qsc_deployment/bin/openssl dgst \ -sha256 \ -binary \ -sign dilithium3.key \ -keyform ENGINE \ -engine /home/fafnirb4/ba_ost/pki/Use-Case-PKI/host-app/src/libkeyprotect_engine.so \ -out hugobinary.sig \ keyprotect_engine.c`	Cfm2LoginHSM2 failed with error 163:HSM Error: RET_USER_LOGIN_FAILURE. [error] Failed to initialize Host-App
9	Verify	`/qsc_deployment/bin/openssl dgst \ -sha256 \ -verify dilithium3.pub \ -signature hugobinary.sig \ keyprotect_engine.c`	Message "Verified OK" is displayed.

13.9 Results

The complete PKI functionality is described in the reference figure Overview PKI and the descriptions of the steps below. The current state of this work is as follows. The first step with the OpenSSL command line works. The second step with the forwarder to the engine has also been successfully tested. The command line can be used to perform the operations. For the signing function with Dilithium3, the engine is currently interposed and the custom function is called from the engine. The third step is to link to the Host-App. This also works without any problems. The interface is the `pki.h` file, which provides the required functionality. In the fourth step, the two Dilithium operations for signature and key generation are passed to the Host-App. Currently, only the signature is overwritten on the engine, so the key generation is not implemented. For the signature, the custom function is called, and in our case the signature function is passed to the liboqs implementation. The connection to the Host-App is still established via initialization, handshake and login for the SM-App. The fifth step is the actual forwarding from the Host-App to the SM-App. This works as already described with the initialization, handshake and login to the SM. The Dilithium functionality is implemented on the Host-Apps and SM-Apps, but had to be omitted from the engine due to time constraints. The sixth step with signature validation used by liboqs was also tested successfully.

In a separate step, the creation of the CA using the configuration file was also tested and all required folders and the CRL were successfully created. Linking to the engine turned out to be more complex than expected. Therefore, CRL creation was not linked to the engine. There is a dockerfile for CRL creation that implements the functionality completely.

13.10 Outlook

This section presents further additions and adaptations to the Public Key Infrastructure (PKI) that will allow the PKI infrastructure to be more secure. In the current state, only the certificate keys are post-quantum safe, as well as the signatures with the corresponding keys.

- One possible extension of upgrading the entire infrastructure would be the introduction of quantum-secure Transport Layer Security (TLS). TLS is currently secured with legacy keys that can be broken by quantum computers in a short time. As TLS is today's standard for secure communication connections, post-quantum developments are also relevant for TLS. Therefore, TLS certificates would need to be modified to use quantum-safe key pairs in the future, such as Dilithium.
- To achieve an even higher standard of security, the Dilithium version could be upgraded from Dilithium3 to Dilithium5. The key length of Dilithium5 is much longer and therefore it takes much longer to crack the key. This would mainly require changing the key length in the functions that are called.
- As mentioned in the PKI Hierarchy section, a one-tier hierarchy as described and PKI implemented in our use case is not recommended. A one-tier hierarchy, where the HSM acts as both, the root CA and the issuing (intermediate) CA has been implemented as it is sufficient to demonstrate the functionality of this use case. However, this could be improved in further work to demonstrate a more suitable implementation for production environments.

Chapter 14 Threat Model: PKI

The following threat model is based on the STRIDE methodology, which classifies each threat into a STRIDE category. Each letter of STRIDE is a security threat category (left) responsible for one of the desired properties (right):

Spoofing: Authenticity

Tampering: Integrity

Repudiation: Non-Repudiation

Information disclosure: Confidentiality

Denial of service: Availability

Elevation of privilege: Authorization

14.1 Identify security objectives

The following security objects were identified:

Object	Explanation
SM-App	The SM-App is the only instance in this architecture that can access the keys located on the HSM.
Communication	Communication between the Host-App and the SM-App must also be secure, because if an attacker can access this connection without any security walls, the SM-App can be attacked directly to try to expose sensitive information.

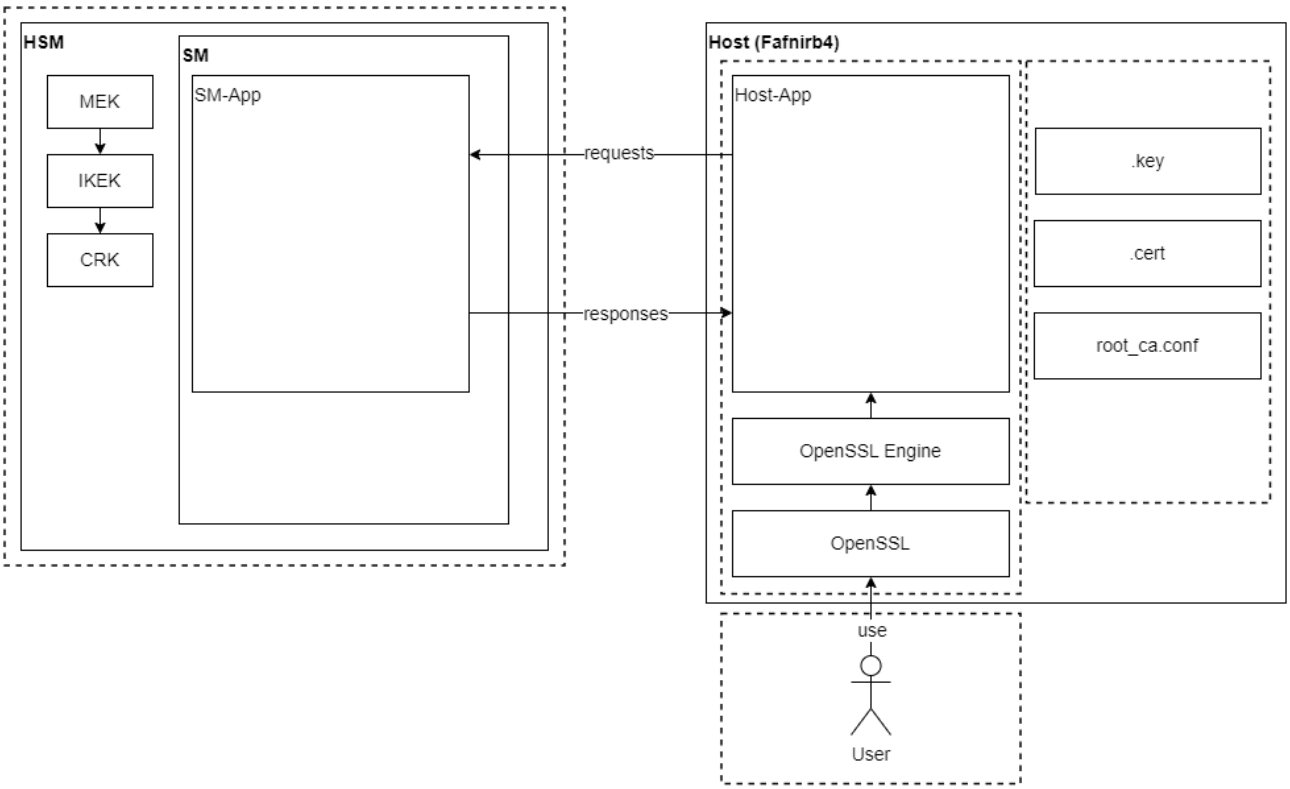

Figure 14.1: Threat Model PKI

Legend:
- - - Trusted Boundary

14.2 Assumptions

- No connection is over networks.
- No tokens are used, only sessions.
- No network between the user and the host, direct input of commands or functions.

14.3 Threat Identification

Table 14.1: Threat Model PKI

Component	Category	Threats	Risk	Mitigation
User	I	Social Engineering	Medium	Informing users about HSM security handling
	I	Blackmail	Low	Good working culture
Host-App	I	Verbose Exception	Medium	Correct exception handling, no critical information disclosure
	T	An attacker could change the behavior of the Host-App or replace it completely	High	Root rights are required to access the folder with the application
Host $\leftrightarrow$ SM-App	T	Change packet content	High	Digital signature
	I	Eavesdropping	High	Encryption
	I	Verbose Exception	Medium	Correct exception handling, no critical information disclosure
	D	Consumes HSM Resources	Low	Set resource limitation for HSM usage
SM-App	I	Verbose Exception	Medium	Correct exception handling, no critical information disclosure
	D	Consumes HSM Resources	Low	Set resource limitation for HSM usage
Root CA	T	Modify configuration file	Medium	Require root right to edit the file
	I	Take keys from CA	High	Only store wrapped secret keys on the host.

14.3.1 User

The users in this particular system are not a big problem. This is because he would need to have more rights than just the crypto user rights e.g. to change the SM-App that uses the HSM. However, it is still advisable to train employees against social engineering attacks.

14.3.2 Host

The host includes the OpenSSL installation, the OpenSSL Engine, the `liboqs` library, and the Host-App. The Host-App is important for the connection to the SM-App and potentially poses a risk by having indirect access to the keys. Running this program requires root privileges on `Fafnirb4` and knowledge of a crypto user's username/password to log in to the HSM.

14.3.3 Root CA

The root Certificate Authority (CA) is signed by the HSM but is not later secured by our application. Because the secret key is encrypted by the HSM.

14.3.4 Connection Host to SM-App

The connection between the Host-App and the SM-App uses only functions provided by Marvell, since there is no network between the Host-App and the SM-App. In the context of this thesis, the Marvell-provided functionality is assumed to be secure.

14.3.5 SM-App

The SM-App is a program that responds to the requests sent by the Host-App. It interacts with the HSM to generate a key, for example. To run this program, a user must first load it into the HSM and start it, which requires the credentials of a crypto officer.

14.3.6 HSM

The HSM and the API through which an SM-App can communicate with the HSM are secured and tested by Marvell. Testing of the HSM is not part of this thesis. In the context of this thesis, the features provided by Marvell and the HSM are considered secure.

Chapter 15 PoC: Bring Your Own Key

For example, Bring Your Own Key (BYOK) allows public cloud users to generate their master key locally and securely transmit it to their Cloud Service Provider (CSP) to protect their data across multiple cloud deployments. This Proof of Concept (PoC) demonstrates how BYOK could be implemented in a post-quantum world. At the time of writing, BYOK typically uses Rivest–Shamir–Adleman (RSA). The Customer Root Key (CRK) is encrypted with the public key and sent to be imported into the Hardware Security Module (HSM). There it is wrapped with the Master Key Encryption Key (MKEK), and the wrapped key is then returned to the client. To show what a secure solution for this use case could look like in a post-quantum world, this PoC was written

15.1 Vision

This Bring Your Own Key (BYOK) PoC was envisioned as a custom Host-App that communicates with a custom SM-App to import a CRK into the HSM in a quantum-safe manner by first performing a Kyber handshake to generate a shared secret and then transporting the CRK to the SM-App using Advanced Encryption Standard (AES). The SM-App then imports the CRK into the HSM using functionality provided by Marvell.

15.2 Description

The following diagram shows at a high level what this PoC does:

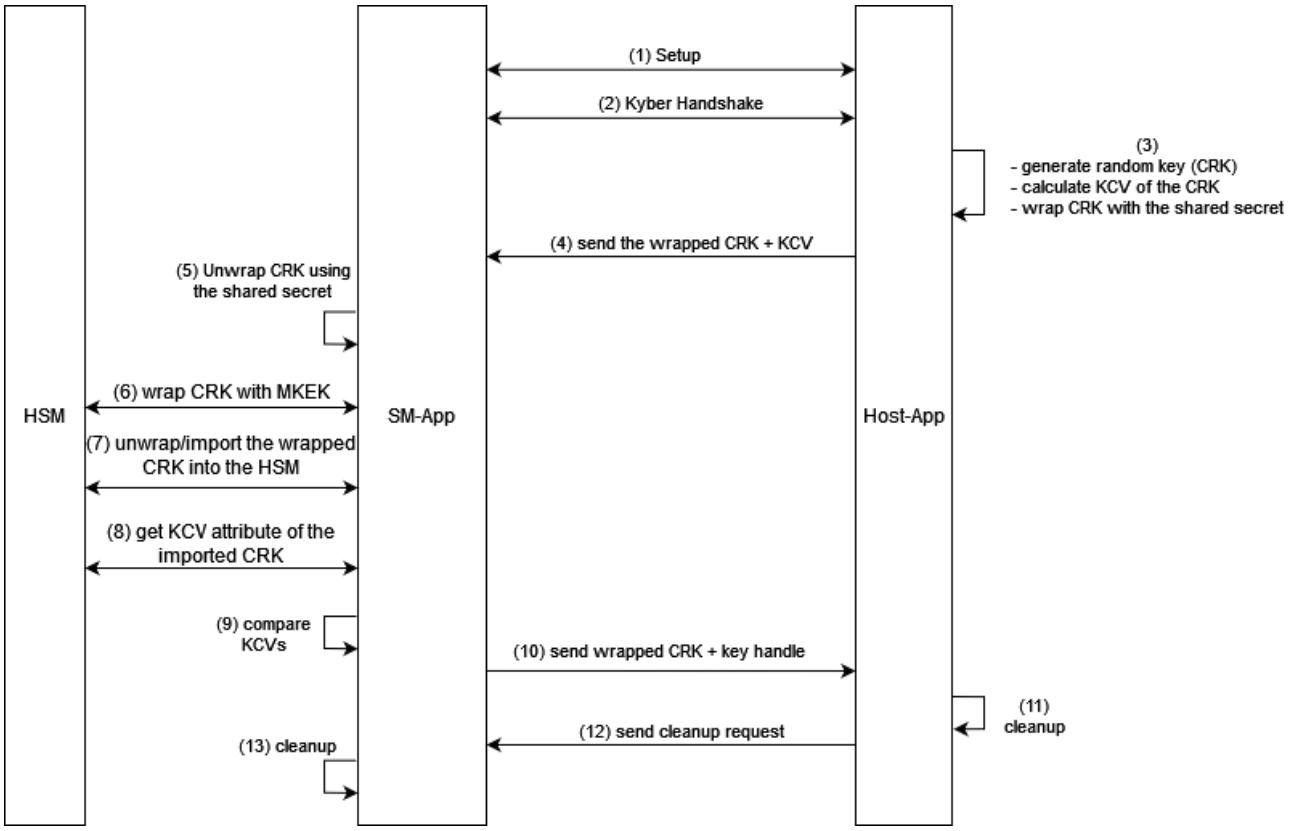

Figure 15.1: Bring Your Own Key (BYOK) using Kyber - PoC

- 1. A handshake is performed between the Host-App and the SM-App so that they can communicate, and a login is performed so that the CRK can later be imported into the HSM.
- 2. A Kyber handshake is performed between the Host-App and the SM-App to obtain a shared secret.
- 3. The Host-App generates a CRK, calculates the Key Check Value (KCV) and wraps the CRK with the shared secret from step 2.
- 4. The Host-App sends the CRK and the KCV to the SM-App.
- 5. Unwrapp the cached CRK using the shared secret.
- 6. Wrap the plaintext CRK with the MKEK on the HSM.
- 7. The MKEK-wrapped CRK is unwrapped in the HSM.
- 8. The SM-App gets the KCV attribute of the imported key from the HSM.
- 9. The SM-App compares the KCV it received from the HSM and the one it received from the Host-App to ensure that the key in the HSM is the same as the one generated by the Host-App.
- 10. If the KCVs are identical, the Host-App receives the key handle of the imported CRK and wraps the CRK with the MKEK generated in step 6. This wrapped CRK is 40 bytes long. The first 32 bytes contain the wrapped key, while the next 8 bytes are the AIV.
- 11., 12. and 13. The Host-App starts the cleanup process, sends the SM-App a cleanup request and then stops. The SM-App starts the cleanup process after receiving the request and then also stops.

Regarding steps 5, 6 and 7: It is not necessary to wrap the key first and then immediately unpack it in the HSM. A plain key can be imported directly without wrapping it first. However, this is done because of a requirement that it should be proven that the wrapped CRK is correctly wrapped and can be unwrapped in the HSM at a later time. Since the same wrapped key is returned to the Host-App that was used to import the key and then checked to see if it was correct, it is certain that the wrapped key is correctly wrapped.

Typically, the CRK is wrapped with the IKEK, which is wrapped with the MKEK. In this PoC, the IKEK is not used to reduce complexity. Instead, the MKEK is used to wrap the CRK directly.

A first attempt was made to use PKCS#11#11 to receive the Key Check Value (KCV) and the wrapped CRK from the HSM. This did not work. This is described in more detail in the Risks encountered section.

15.3 Boundary

The Host-App will only support Linux systems and will only be tested on Ubuntu as this is what runs on IBM's server.

15.4 Components

The various components shown in the Bring Your Own Key with Kyber PoC figure are described in this section.

15.4.1 Server

The server (`Fafnirb`) is an Ubuntu system connected to the HSM. The Virtual Machine (VM) named `Fafnirb4` runs on this server. This is where the Host-App runs. The Host-App must also be compiled on this VM, as all dependencies regarding the SM-App are only present on this VM.

15.4.2 Host-App

The Host-App is a program written in C that communicates with the SM-App. This is what a user would interact with to start the process of importing a key.

15.4.3 HSM

The Hardware Security Module (HSM) is the platform on which the SM-App runs. It is shown as a separate component to represent the communication between the HSM and the SM-App.

15.4.4 SM-App

The SM-App runs custom code that communicates with the HSM and the Host-App. It does one part of the Kyber handshake, wraps the CRK with the MKEK and then unwraps the keys in the HSM. It also verifies that the key in the HSM is the correct one by comparing the KCV of the key generated by the Host-App and the one in the HSM.

15.4.5 Kyber

Kyber is an IND-CCA2-secure Key Encapsulation Mechanism (KEM) whose security is based on the hardness of solving the Learning With Errors (LWE) problem over module lattices. [11]

It is used to exchange the shared secret to later use AES to transfer the CRK. It provides three key exchange protocols.

The simplest one does not authenticate the other side. This is also the Kyber handshake that was implemented in this PoC because of its simplicity. It looks like this:

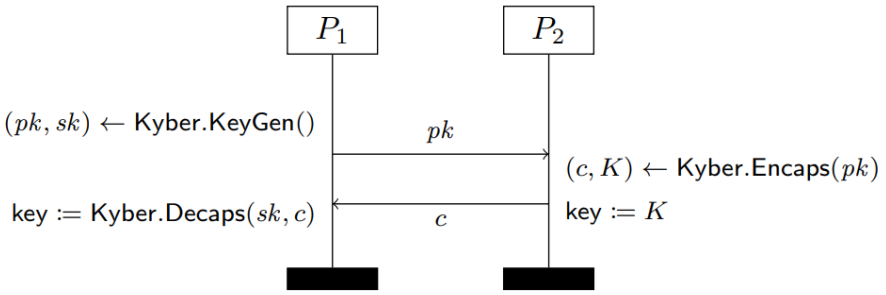

Figure 15.2: Kyber.KE – Key Exchange protocol using the Kyber (Source: [39])

To authenticate both parties (both know each other's static public key) the protocol looks like this:

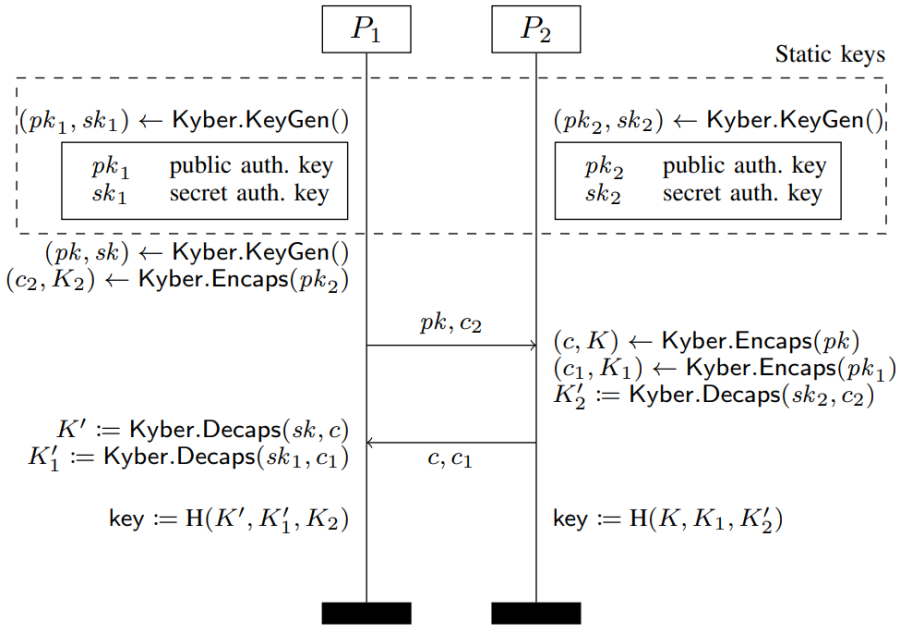

Figure 15.3: Kyber.AKE – Authenticated key exchange protocol using Kyber, where both parties know each other's static public key. (Source: [39])

Finally, to authenticate only one side, the following protocol exists:

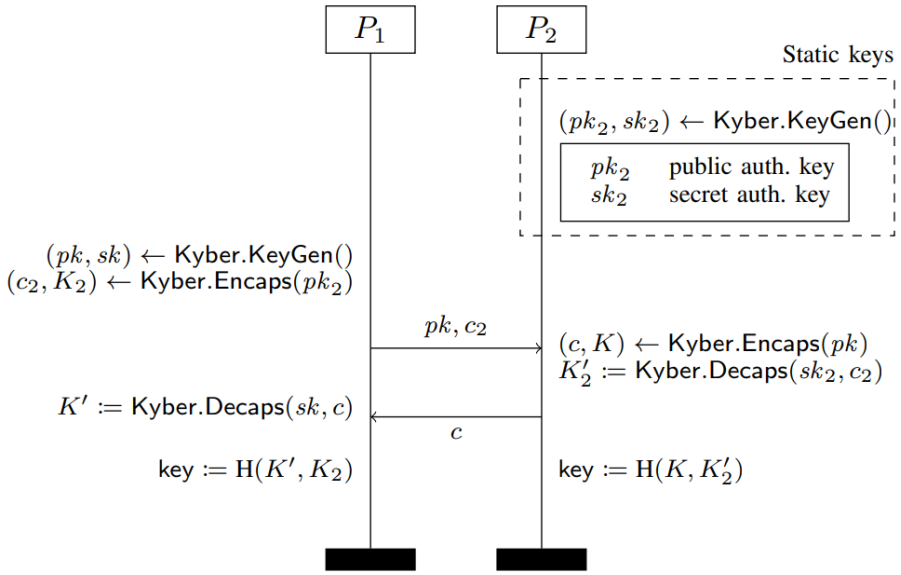

Figure 15.4: Kyber.UAKE – One-sided authenticated key exchange protocol using Kyber, where P1 knows the static public key of P2 (Source: [39])

15.4.6 Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is used to securely transport the CRK from the Host-App to the SM-App. AES-GCM provides both confidentiality (encryption) and integrity (authentication) of the data, ensuring that the ciphertext remains confidential and has not been tampered with. AES-GCM was chosen because of the security requirements and its efficiency. AES also provides a special mode for key wrapping (AES-KWP). This was considered but not used as the Secure Machine (SM) runs OpenSSL version 1.1.1, which does not support unwrapping.

15.5 Functional Requirements (FR)

The following requirements were defined with IBM.

15.5.1 Epics

For this use case, four main tasks are necessary. To secure the connection, firstly Kyber is used to exchange keys. Afterwards, the CRK key must be encrypted using AES and sent to the SM-App. The SM-App must wrap the CRK with the MKEK and then unwrap it into the HSM. Then, it needs to be proven that the key in the HSM is the same as the one generated by the Host-App by checking a nonce, after which the CRK is wrapped with the MKEK and send back to the Host-App.

Nr.	Title
1	Exchange of symmetric keys
2	Transmission of the CRK key
3	Test CRK
4	Export CRK

15.5.2 User Stories

In the documentation, the user stories are quite broad to increase readability. They were refined to be easier to work with, but only in Jira since that is where all the time tracking and estimation took place.

To visualize the MVP the corresponding user stories are colored yellow.

Exchange of symmetric keys

Prio.	Story	Done
1	As a security engineer, I want the key exchange to use Kyber so that it is secure against attacks from quantum computers.	YES
2	As a security engineer, I want the application to support the one-sided authenticated key exchange protocol using Kyber to ensure the handshake takes place with the expected party.	NO

Transmission of the CRK

Prio.	Story	Done
1	As a security engineer I want the CRK to be transmitted using AES so that it is secure against attacks from quantum computers.	YES

Test CRK

Prio.	Story	Done
1	As a user, I want the application to prove that the imported key is the same as the one generated by the Host-App so that I can be sure the correct key is being used.	YES
1	As a user, I want the application to prove that the exported key is correctly encrypted with the MKEK so that I can import the CRK later on again.	YES

Export CRK

Prio.	Story	Done
1	As a user, I want to receive the generated key wrapped with the MKEK as output, so that I can store it somewhere safe.	YES

15.6 Non-Functional Requirements (NFR)

In this chapter, all NFRs are described for this use case.

NFR-1	
Requirement	All communication between the SM App and the Host-App, except for setup and cleanup requests, should be considered as going over a network. For these requests, the confidential contents need to be encrypted using AES-256 GCM or AES-256 CCM.
Measure(s)	Manual code reviews
Validation	Manual code review
Priority	High

NFR-2	
Requirement	A Kyber handshake should be used so that both parties can gain a shared secret. Only Kyber-1024 or Kyber-1024-90s should be used.
Measure(s)	Manual code reviews
Validation	Manual code review
Priority	High

NFR-3	
Requirement	The written code should be testable.
Measure(s)	Write automated unit tests, integration tests and additional Known Answer Test (KAT) (incl. code coverage checker).
Validation	The code coverage should reach the same percentage as it was before the extension or 90%.
Priority	High

NFR-4	
Requirement	The applications work on a Linux system.
Measure(s)	Test it regularly on Ubuntu 22.04 (servers at IBM).
Validation	System Tests passed expected result
Priority	High

NFR-5	
Requirement	The applications are available in English.
Measure(s)	All displayed texts are in English.
Validation	Corresponding System Test passed
Priority	High

NFR-6	
Requirement	Users are notified of wrong or inconsistent inputs.
Measure(s)	Test invalid inputs.
Validation	Corresponding System Test passed
Priority	Medium

NFR-7	
Requirement	When an exception is thrown the application displays a meaningful error message.
Measure(s)	Exceptions are looked at during code reviews to ensure sufficient logging is done.
Validation	Exceptions are captured on the application level and printed. It is also tested if the correct error message is shown using automated tests and manual system tests.
Priority	Medium

15.7 Technologies, Libraries and frameworks

The SM-App and the Host-App were written in C because at the time of writing the SM provided mainly supported C programs and the Kyber library was also only available in C.

15.7.1 Libraries

The following libraries were used:

- kyber1024_ref was used as using Kyber was a requirement by IBM. It was in this PoC so that the SM-App and the Host-App can gain a shared secret. Kyber1024 was chosen as it is the most secure and recommended by the CNSA.

15.7.2 Frameworks

For testing purposes the framework ceedling was chosen. This is due to the following reasons:

- Mocking capabilities
- Ease of use
- Does **not** introduce cmake files to the project

All of the code examples provided by Marvell use Makefiles to build the SM/Host-App. Using cmake would therefore mean rewriting these shared Makefiles to build a Host/SM-App. To avoid having to rewrite these files various testing frameworks were considered.

However, this library also has some disadvantages:

- Header files can either be mocked or not. It is not possible to mock only one function from a header file. Because of this limitation, many files only implement one or two functions, otherwise it is not possible to easily test functions that call other functions in the same file.
- Some functions are very hard to mock. This is especially true of functions in libraries such as OpenSSL since all types must be defined. For this reason, some functions have not been tested with automated tests, as it would be would be too much effort for the added value. Another approach to work around this problem was to write a custom header file that contained only the function of interest and defined all the required types. During testing, this custom header would be used instead of the real one. It was decided on a case-by-case basis whether it was worth to write a custom header. These headers can be found in the test folder under "mockable_headers".

15.8 Host-App

This section describes the Host-App in detail.

15.8.1 Project Structure

A Host-App for this PoC is structured as described here: Project Structure

libs contains the following libraries:

- **kyber:** Kyber is needed to perform the handshake. The official GitHub repository is therefore present here as a submodule so that this library can be used and easily updated. The GitHub repository already contains a Makefile to create a shared library. However, this Makefile is not used because the source of the entropy is intentionally missing, as can be seen in this issue. Instead, a static library is created that contains all required files. A static library was chosen instead of a shared one as there would be little benefit to this PoC by using a shared library and a static library was already needed for the SM-App which meant the code to build it was already present.

host_poc_byok contains the files and folders shown in the diagram below, but also some additional ones:

- **api.h:** This file contains all defined requests and responses needed for communication with the SM-App.
- **main.c:** This file contains "main" and some logic to parse the arguments used to start the program.
- **key_import:** This folder contains the following files that are used to create the request to the SM-App to import a key into the HSM: **aes.c/.h**, **key_import.c/.h:**, **key_import_requests.c/.h:**. **key_import.c** contains the logic for the programme flow, while **key_import_requests.c** contains the code for creating and sending requests and receiving their responses. **aes.c/.h** is needed to encrypt the key with the shared secret so that it can be securely sent to the SM-App. This file also contains the code to compute the KCV.
- **kyber_handshake:** This folder contains the following files, which are used to complete a Kyber handshake between the Host-App and the SM-App: **kyber_handshake.c/.h:**, **kyber_handshake_requests.c/.h:**. **kyber_handshake.c** contains the logic for the program flow, while **kyber_handshake_requests.c** contains the code for creating and sending requests and receiving their responses.
- **request_helper.c/.h:** This file contains a function to create a generic request that can be customized and then sent to the SM-App.
- **setup_communication.c/.h:** This file contains the functions to set up the communication between the SM-App and the Host-App. This includes a handshake and a login.
- **Makefile** This Makefile builds the Host-App. Dependencies (libraries) must be built before this Makefile is executed.

tests contains automated tests for the following files:

- **key_import.c**
- **key_import_requests.c**
- **kyber_handshake.c**
- **kyber_handshake_requests.c**
- **request_helper.c**
- **setup_communications.c**

This folder also contains the files: **helper_functions.c/.h**. These files contain stubs for functions used in multiple tests.

The following graphic also depicts the described project structure for the Host-App, including only the significant files:

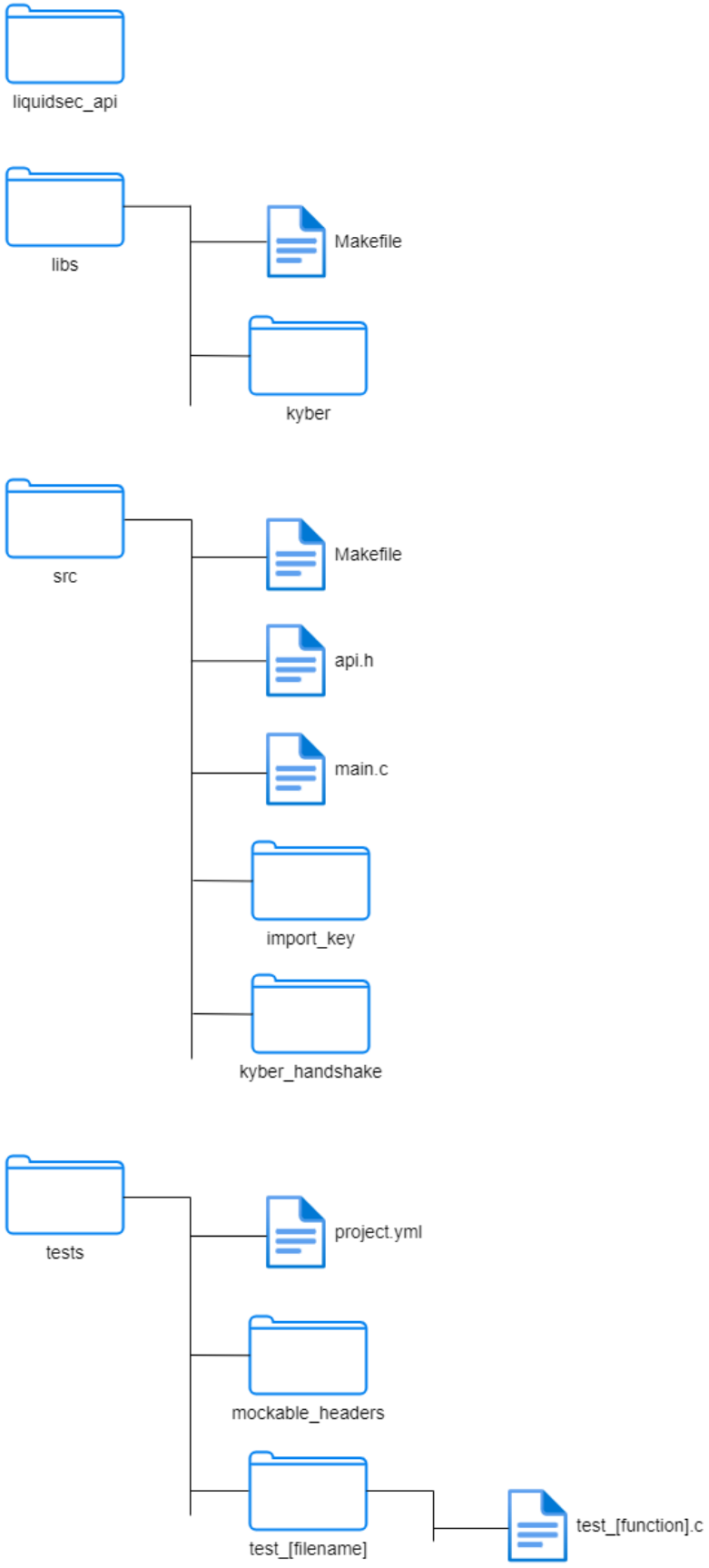

Figure 15.5: Host-App structure

15.8.2 Call graph

The following graphic shows a static call graph of the Host-App, including only the most important calls, to give an how and which files interact with each other:

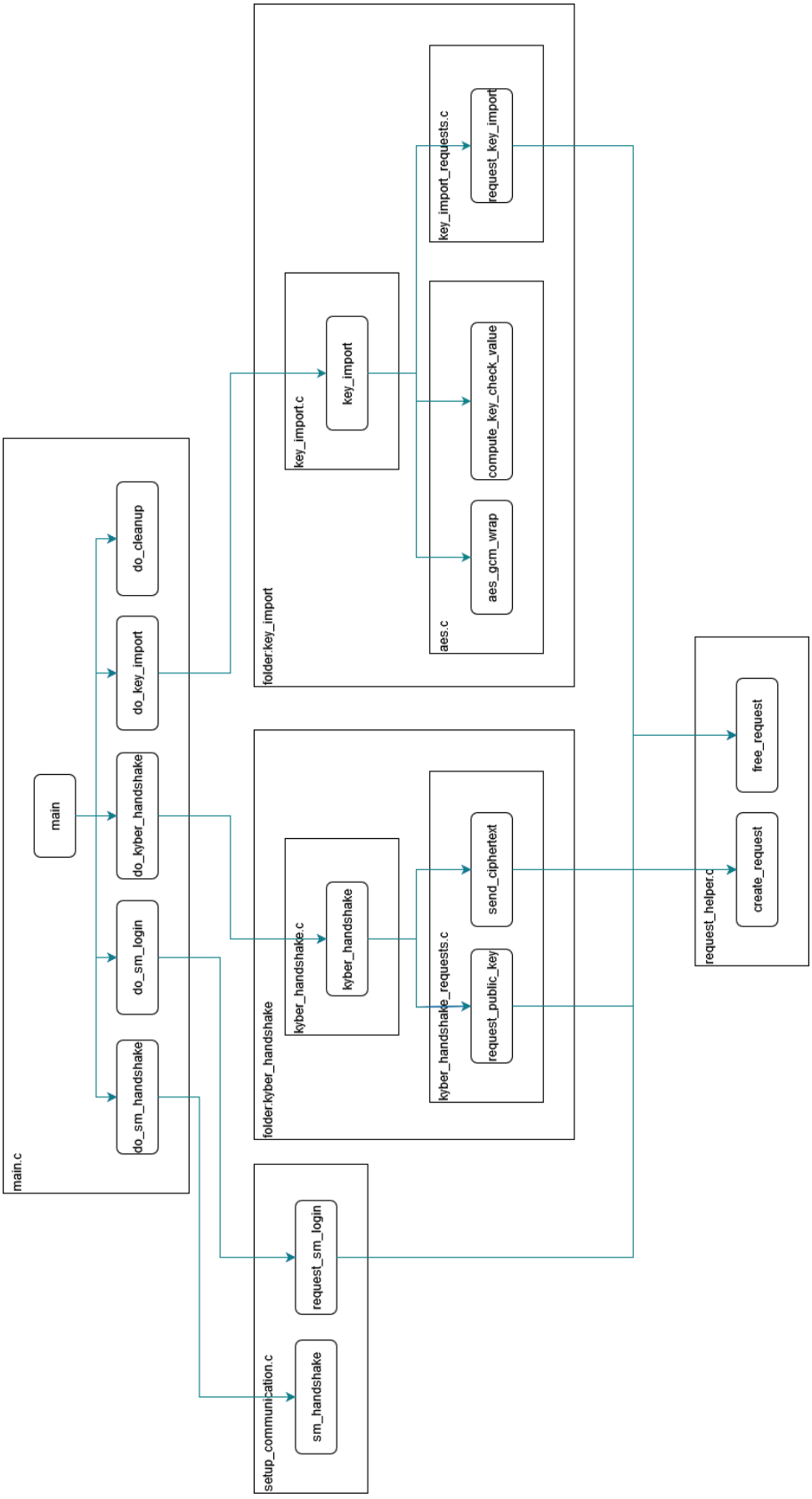

Figure 15.6: Host-App call graph (Overview)

15.9 SM-App

This section describes the SM-App in detail.

15.9.1 Project Structure

An SM-App for this PoC is structured as described here: Project Structure

libs contains the following libraries:

- **kyber:** Kyber is needed to perform the handshake. The official GitHub repository is therefore present here as a submodule so that this library can be used and easily updated. The github repository already contains a Makefile to create a shared library. However, this Makefile is not used because the source of the entropy is intentionally missing, as can be seen in this issue. Instead, a static library is created that contains all the required files. Since it is quite easy to send a program to the SM, but little was found about how to send/set up a shared library to the SM, therfore it seemed easier to use a static library instead of a shared one.

sm_poc_byok contains the files and folders shown in the diagram below, but also some additional ones:

- **api.h:** This file contains all defined requests and responses needed for communication with the SM-App.
- **main.c:** This file contains "main" and some logic to handle requests.
- **key_import:** This folder contains the following files that are used to handle a request to import a key into the HSM: **aes_gcm_key_unwrap.c.c/.h**, **key_import.c/.h:**, **key_import_helper.c/.h:** and **key_import_request_handler.c/.h:**. **key_import_request_handler.c/.h:** contains the logic to handle the request and build a response. This file calls the function to import the key in **key_import.c/.h:**. **key_import.c/.h:** contains code for the control flow of importing a key into the HSM. It the function provided by **aes_gcm_key_unwrap.c/.h** to unwrap the received CRK and then uses functions from **key_import_helper.c/.h:** to interact with the HSM and import the key.
- **kyber_handshake.c/h:** This file handles all Kyber handshake-related requests.
- **request_helper.c/.h:** This file contains a function to respond to the Host-App with a given response.
- **setup_communication.c/.h:** This file contains the login function. A login is necessary to later import a key into the HSM.
- **Makefile** This Makefile builds the Host-App. Dependencies (libraries) must be built before this Makefile is executed.

tests contains automated tests for the following files:

- **setup_communications.c**
- **key_import_helper.c**
- **import_key_request_handler.c**
- **key_import.c**
- **kyber_handshake.c**
- **setup_communication.c**

The following graphic also depicts the described project structure for the SM-App, including only the more significant files:

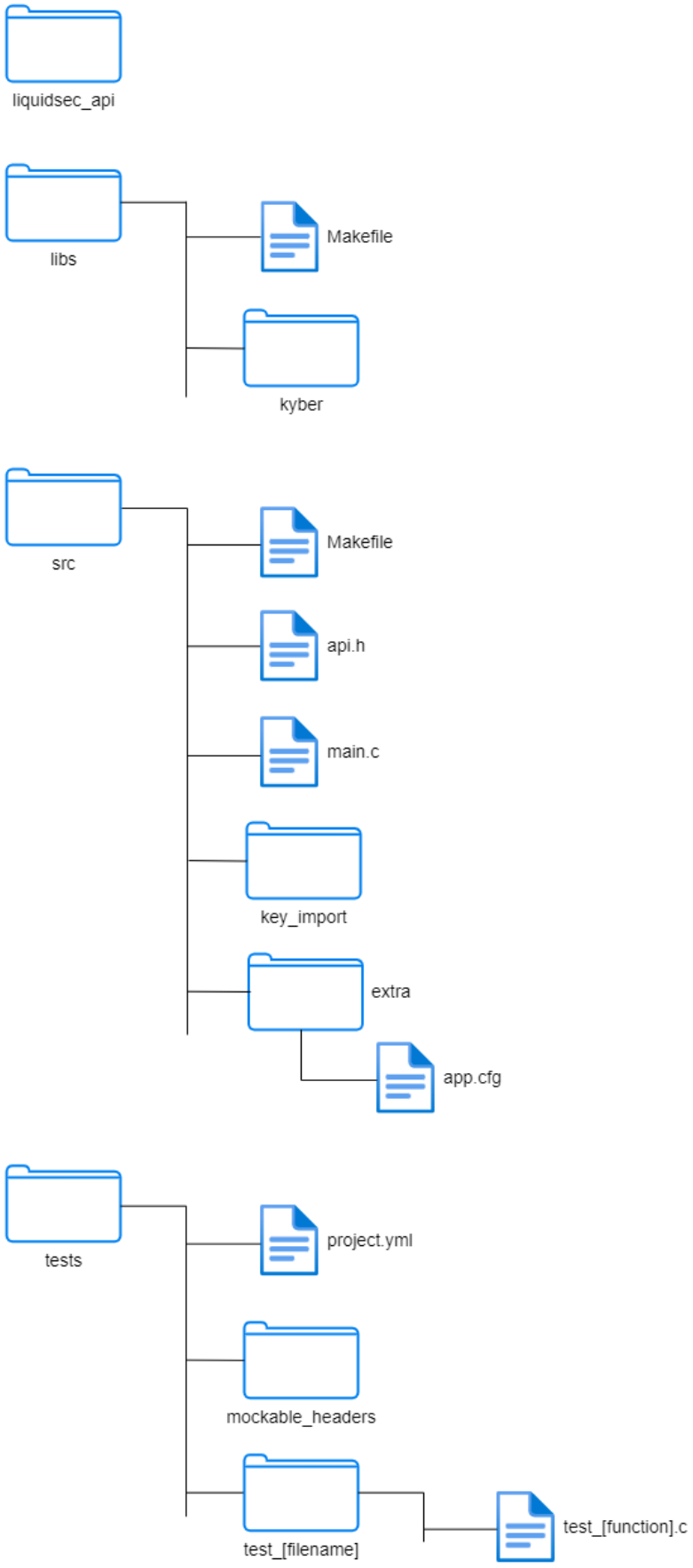

Figure 15.7: SM-App structure

15.9.2 Call graph

The following graphic shows a static call graph of the SM-App, including only the most important calls, to give an how and which files interact with each other:

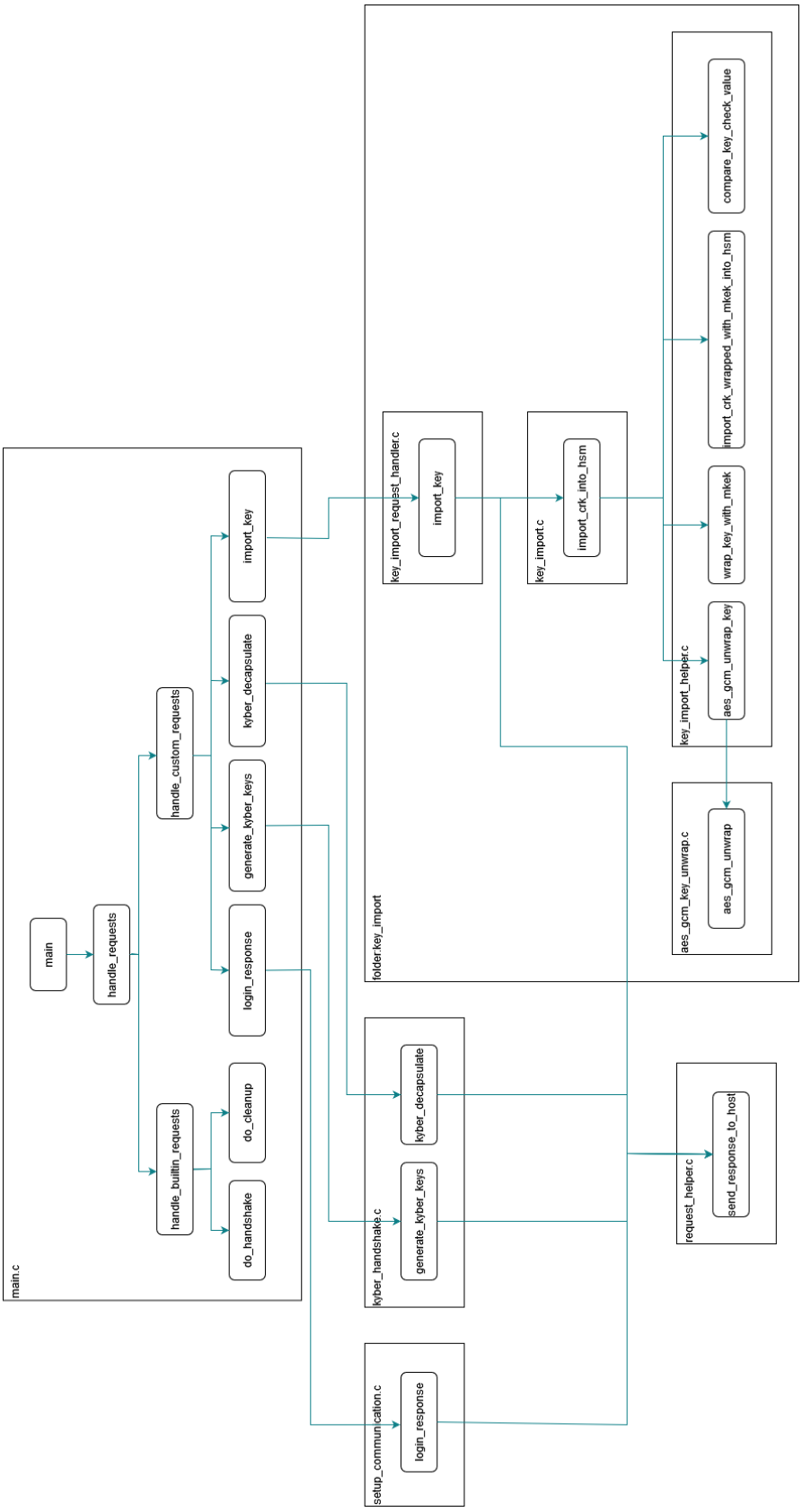

Figure 15.8: SM-App call graph (Overview)

15.10 Development

In this section, the development process including the pipeline and the testing are described.

15.10.1 Pipeline

The pipelines for both the SM-App and the Host-App consist of the following steps:

- **Install SSH key:** As some submodules are in a private repository ssh is used to clone these repositories, therefore an ssh key needs to be added to the ssh-agent in the pipeline.
- **Checkout:** The repository is cloned into the pipeline.
- **Checkout submodules:** All submodules used in the repository are cloned (some using ssh).
- **Run tests:** ceedling is installed and used to run all tests present in the repository.
- **Archive code coverage results:** Finally, the report generated by running the tests, which shows the code coverage, is stored as an artifact of the workflow.

15.10.2 Testing

Unit and Integration tests were written for the SM-App and the Host-App wherever sensible. To do this mocking was used as many functions require the environment of the SM or the server `Fafnirb4`. All these tests can be found in the tests folder in the repository of the Host-App or the tests folder of the SM-App.

To verify that everything works together as expected these additional System tests were specified:

Table 15.1: System Tests BYOK

Nr.	Test Name	Test Steps	Expected Results
1	Run Host-App without arguments	Run the Host-App with the following command: `sudo ./hostapp_poc_byok`	An error is displayed informing the user of the arguments needed to run the application.
2	Run Host-App without some arguments	Run the Host-App with the following command: `sudo ./hostapp_poc_byok -n part -u user -p pass`	An error is displayed informing the user of the arguments needed to run the application.
3	Run Host-App without too long username	Run the Host-App with the following command: `sudo ./hostapp_poc_byok -n part -u AAAAAAAAAAAAAAAAAAAA\AAAAAAAAAAAAAAAAAAAAAAAAA\AAAAAAAAAAAAAAAAA -p pass`	An error is displayed informing the user that the username exceeds the max length.
4	Run Host-App without too long password	Run the Host-App with the following command: `sudo ./hostapp_poc_byok -n part -u user -p AAAAAAAAAAAAAAAAA\AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\AAAAAAAAAAAAAAAAA`	An error is displayed informing the user that the password exceeds the max length.

5	Run Host-App without too long partition name	Run the Host-App with the following command: `sudo ./hostapp_poc_byok -n \AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA \AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA \AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA \AAAAAAAAAAAAAAAAAAAA -u user -p pass`	An error is displayed informing the user that the partition name exceeds the max length.

6	Run Host-App with a too-short pek file	1. Run:
		`echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" > test_pek`
		2. Run the Host-App with the following command:
		`sudo ./hostapp_poc_byok -n part -u user -p pass -f test_pek`
			An error is displayed informing the user that the chosen file has the wrong size.

7	Run Host-App with a too-long PEK file	1. Run:
		`echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" > test_pek`
		2. Run the Host-App with the following command:
		`sudo ./hostapp_poc_byok -n part -u user -p pass -f test_pek`
			An error is displayed informing the user that the chosen file has the wrong size.

8	Run Host-App with incorrect values	1. Run:
		`echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" > test_pek`
		2. Run the Host-App with the following command:
		`sudo ./hostapp_poc_byok -n part -u user -p pass -f test_pek`
			An error is displayed informing the user that the initialization of the application failed.

9	Run Host-App without running the SM-App	Run the Host-App as described in this README	An error is displayed informing the user that the SM-App is not running.
10	Run Host-App and SM-App	1. Run the SM-App as described in this README. 2. Run the Host-App as described in this README.	The output generated looks like the expected result described in the README of the Host-App.
11	Check the Pipelines	Go to the main branch of the Host-App and the main branch of the SM-App.	A green check mark is displayed next to the last commit.

15.11 Outlook

To further improve this PoC, the following changes could be made:

- This implementation uses the basic Kyber handshake (Kyber.KE). This could be improved by using the one-way authenticated key exchange protocol provided by Kyber(Kyber.UAKE) so that the Host-App can verify that it is communicating with the intended party.
- The Host-Application could be extended to be able to receive an IKEK as input, which it would then pass on to the SM-App so that the CRK can be wrapped with this specific IKEK instead of the MKEK to make this PoC more realistic.

Chapter 16 Threat Model: BYOK

The following threat model is based on the STRIDE methodology, which classifies each threat into a STRIDE category. Each letter of STRIDE is a security threat category (left) responsible for one of the desired properties (right):

Spoofing Authenticity

Tampering Integrity

Repudiation Non-Repudiation

Information disclosure Confidentiality

Denial of service Availability

Elevation of privilege Authorization

16.1 Identify security objectives

The following security objects were identified:

Object	Explanation
SM-App	The SM-App is the only instance in this architecture that can access the keys located on the HSM.
Keys	Keys generated by the Host-App must not be exposed.

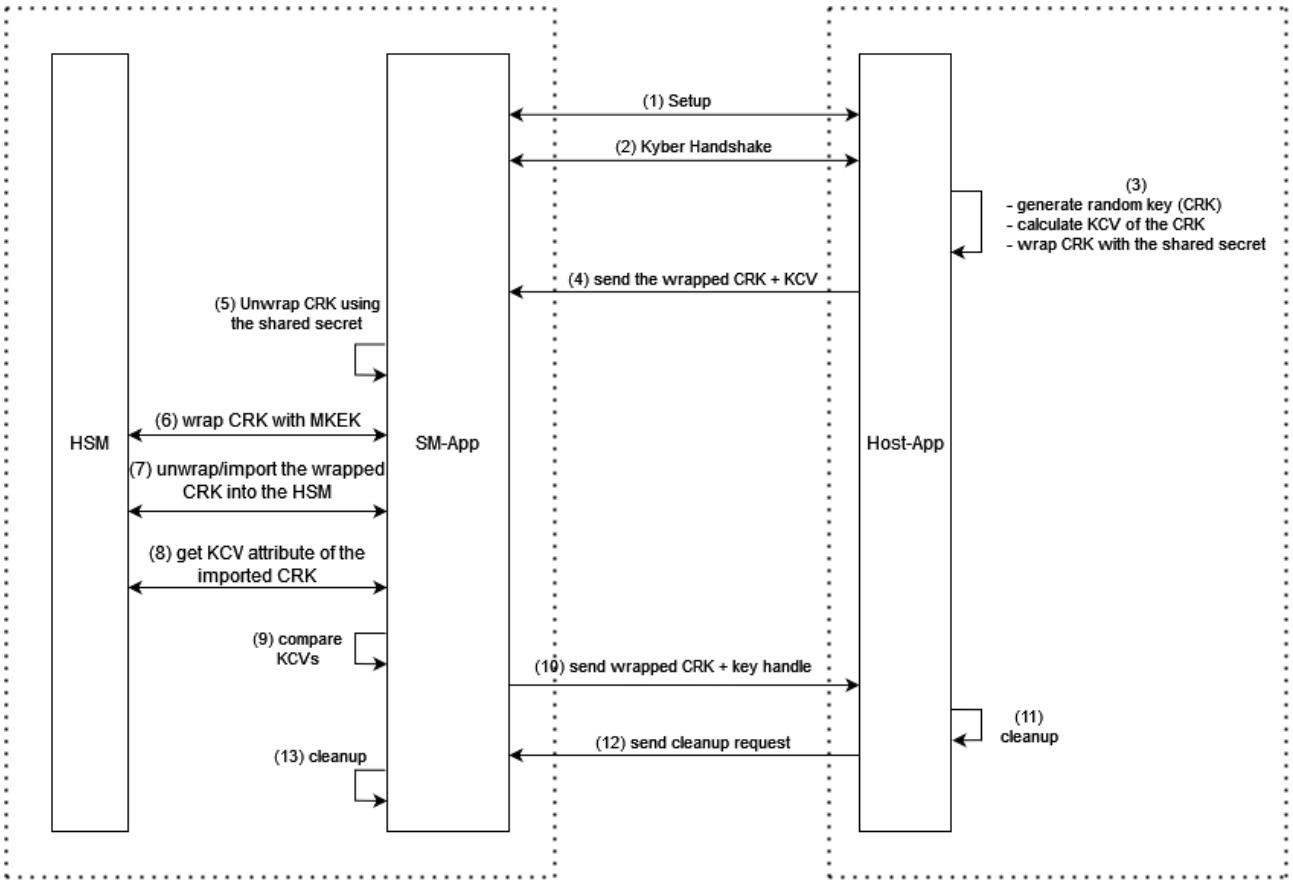

Figure 16.1: Threat Model BYOK

Legend:
- - - Trusted Boundary

16.2 Assumptions

- An attacker does not have root access to the `Fafnirb4` and valid crypto officer credentials. At this point, the SM-App could be replaced.

16.3 Threat Identification

Table 16.1: Threat Model BYOK

Component	Category	Threats	Risk	Mitigation
Host-App	I	Verbose Exception	Medium	Correct exception handling, no critical information disclosure
	T	An attacker could change the behavior of the Host-App or replace it completely	High	root rights are required to access the folder with the application
	I	An attacker with root rights and valid crypto user credentials could try to access the CRK of an earlier session using PKCS#11	Medium	Keys are stored as session objects in the HSM
Host-App ↔ SM-App	T	Change packet content	High	digital signature
	I	Eavesdropping	High	Encryption
SM-App	I	Verbose Exception	Medium	Correct exception handling, no critical information disclosure
	D	Consumes HSM Resources	Low	set resource limitation for HSM usage
Host-App ↔ HSM	-	Trusted	-	-
HSM	-	Trusted	-	-

16.3.1 Host-App

The Host-App is what a user of this program would interact with to start the process of importing a key. Running this program requires root privileges on the `Fafnirb4` and knowledge of a crypto user's username/password of a crypto user to log in to the HSM.

This application receives the username and password of the crypto user as arguments. Usually, this is regarded as a problem, as the username and password can be seen in the bash history. It was discussed during the meeting on 31.05.2023 and approved by IBM to handle it in this manner anyway, since this is only a PoC and other similar example programs provided by Marvell also get the username and password as arguments.

This application receives the crypto user's username and password as arguments. Usually, this is seen as a problem because the username and password can be seen in the bash history. It was discussed at the 31.05.2023 meeting and IBM agreed to handle it this way anyway, since it is only a PoC and other similar examples provided by Marvell also get the username and password as arguments.

16.3.2 SM-App

The SM-App is a program that responds to the requests sent by the Host-App. It interacts with the HSM to import a key, for example. To run this program a user must first load it into the HSM and start it. This requires the credentials of a crypto officer.

16.3.3 Connection Host-App to SM-App

The connection between the Host-App and the SM-App uses only functions provided by Marvell because there is no network between the Host-App and the SM-App. For the purposes of this PoC, however, it is necessary to pretend that there is a network over which all requests other than setup and cleanup requests are sent.

16.3.4 HSM

The HSM and the API through which an SM-App can communicate with the HSM, are secured and tested by Marvell. Testing of the HSM is not part of this thesis. For the purposes of this thesis, the features provided by Marvell are considered secure.

Chapter 17 Conclusion

This thesis deals with the use of newly standardized post-quantum secure algorithms. The two algorithms covered in this thesis are CRYSTALS-Dilithium and CRYSTALS-Kyber. These two algorithms were used in Proof of Concept (PoC)s to demonstrate how different use cases could be implemented. These projects used the Secure Machine (SM) on the Hardware Security Module (HSM) from the manufacturer Marvell, because the provided HSM does not support these algorithms yet and by using the SM for cryptographic operations it could be guaranteed that these keys are never exposed in clear text in memory.

The first step of this thesis was to test the compatibility of these two algorithms of the CRYSTALS family with the architecture of the Marvell HSM. Two PoCs were used to verify functionality and compatibility. After successful completion of the PoCs, use cases for these algorithms were discussed and implemented.

The Bring Your Own Key (BYOK) use case demonstrates the use of the CRYSTALS Kyber Key Encapsulation Mechanism (KEM) algorithm in a practical environment. This use case demonstrates how to securely import a Customer Root Key (CRK) into the HSM. The Kyber algorithm is responsible for establishing a secure connection between the host and the HSM by generating a shared secret, which is then used to transfer the CRK using Advanced Encryption Standard (AES). This is an alternative to the current method of wrapping the CRK using Rivest–Shamir–Adleman (RSA), which is not quantum-safe.

In the second use case, a Public Key Infrastructure (PKI) was set up using the quantum-safe signature algorithm CRYSTALS-Dilithium. In this use case, the HSM was used as the Root of Trust (RoT) for key identity assurance. The keys for the Certificate Authority (CA) and certificates are generated using the Dilithium algorithm. Compared to today's keys, the Dilithium algorithm ensures that the keys will be post-quantum secure.

These implementations are a step toward a quantum-secure future. With these demonstrations, a new, more secure infrastructure can be built.

To further enhance security and improve implementations, additional features and security standards can be added to both use cases. For the BYOK use case, this would include using an authenticated version of the Kyber handshake so that a client can verify that it is communicating with the correct party, or the applications could be extended so that the CRK is correctly wrapped with a supplied IKEK. The PKI use case is also not yet complete. Longer keys, which are even harder to crack, or an extension of the Transport Layer Security (TLS) protocol, which is not yet post-quantum secure, would further increase security.

Part II

Project Documentation

Chapter 18 Project Plan

The following figures show the project plan during the bachelor thesis. The fields with the grey background show the corresponding weeks for the task described on the left.

The Legend describes the color scheme and the milestones.

Project Plan - Crypto Agility

Start Date 20.02.2023
End Date 16.06.2023
Submit Abstract and Poster 12.06.2023

		Priority	W01 20.02 - 26.02	W02 27.02 - 05.03	W03 06.03 - 12.03	W04 13.03 - 18.03	W05 20.03 - 26.03	W06 27.03 - 02.04	W07 3.04 - 09.04 (Holiday)	W08 10.04 - 16.04 (Holiday)	W09 17.04 - 23.04
				M1		M2		M3			M4
1	Business Modelling										
1.1	Rough Project Plan (RUD)	low									
1.2	Detailed Project Plan (Scrum)	low									
2	Requirements										
2.1	Non Functional Requirements	medium									
2.2	Functional Requirements	medium									
3	Architecture										
3.1	Define Architecture	high									
4	Preparation										
4.1	Familiarize with HSMs	low									
4.2	Familiarize with PKCS#11	low									
4.3	Familiarize with Dilithium-Algorithm	low									
4.4	Familiarize with IBM specific Setup	low									
4.5	Create Concept for Use-Case PKI	medium									
4.6	Create Concept for Use-Case Kyber	medium									
5	Implementation										
5.1	PoC Dilithium	high									
5.2	Implement Use-Case: PKI	medium									
5.3	Implement PoC Use-Case: BYOK	medium									
5.4											
6	Test										
6.1	Test Dilithium on MIPS CPU	high									
6.2	Test Use-Case: PKI	medium									
6.3	Test PoC Use-Case: BYOK	medium									
6.4											
7	Configuration & Change Management										
7.1	Update Documentation	low									
7.2	Change requirements	low									
8	Project Management										
8.1	Time Tracking	low									
8.2	Current Task State	low									
8.3	Meeting Notes	low									
8.4	Risk Management	high									
9	Environment										
9.1	Tooling setup	low									
9.2	Repository setup	low									
9.3	CI/CD setup	low									
9.4	Test environment setup	low									

Figure 18.1: Project Plan (Part 1)

Project Plan - Crypto Agility

Start Date 20.02.2023
End Date 16.06.2023
Submit Abstract and Poster 12.06.2023

		Priority	W10 24.04 - 30.04	W11 1.05 - 07.05	W12 8.05 - 14.05	W13 15.05 - 21.05	W14 22.05 - 28.05	W15 29.05 - 04.06 29.05 Holiday	W16 05.06 - 11.06	W17 12.06 - 16.06
						M5		M6		M7
1	Business Modelling									
1.1	Rough Project Plan (RUD)	low								
1.2	Detailed Project Plan (Scrum)	low								
2	Requirements									
2.1	Non Functional Requirements	medium								
2.2	Functional Requirements	medium								
3	Architecture									
3.1	Define Architecture	high								
4	Preparation									
4.1	Familiarize with HSMs	low								
4.2	Familiarize with PKCS#11	low								
4.3	Familiarize with Dilithium-Algorithm	low								
4.4	Familiarize with IBM specific Setup	low								
4.5	Create Concept for Use-Case PKI	medium								
4.6	Create Concept for Use-Case Kyber	medium								
5	Implementation									
5.1	PoC Dilithium	high								
5.2	Implement Use-Case: PKI	medium								
5.3	Implement PoC Use-Case: BYOK	medium								
5.4										
6	Test									
6.1	Test Dilithium on MIPS CPU	high								
6.2	Test Use-Case: PKI	medium								
6.3	Test PoC Use-Case: BYOK	medium								
6.4										
7	Configuration & Change Management									
7.1	Update Documentation	low								
7.2	Change requirements	low								
8	Project Management									
8.1	Time Tracking	low								
8.2	Current Task State	low								
8.3	Meeting Notes	low								
8.4	Risk Management	high								
9	Environment									
9.1	Tooling setup	low								
9.2	Repository setup	low								
9.3	CI/CD setup	low								
9.4	Test environment setup	low								

Figure 18.2: Project Plan (Part 2)

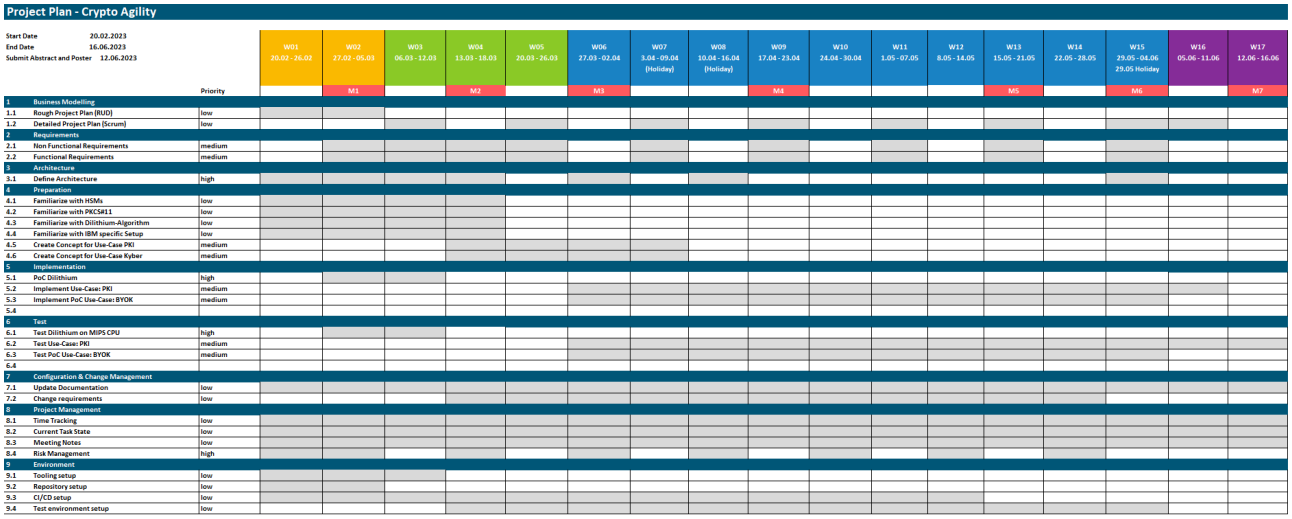

Figure 18.3: Project Plan (Part 3)

Milestones:

M1	Project Setup
M2	PoC
M3	End of Elaboration / Architecture
M4	Interim Presentation
M5	Quality
M6	End of Implementation
M7	Submit Final Documentation

Legend:

- Inception
- Elaboration
- Construction
- Transition

Figure 18.4: Project Plan (Legend)

18.1 RUP - Rational Unified Process

For the long-term plan, it was decided to use RUP. The RUP model divides the project time into four phases. Each phase focuses on a different part of the project.

- Interception
- Elaboration
- Construction
- Transition

18.1.1 Interception Phase

The first phase is the interception phase, which is the start of the project. This is where the vision of the project is written, the scope is defined, and a rough timeline is set.

18.1.2 Elaboration Phase

The main focus of the elaboration phase is to identify the requirements, implement the architecture (draft) and try to identify the risks. This phase is really important to identify problems as early as possible and to provide a structure for the project to work on them later.

It is often used to achieve the Proof of Concept (PoC) or Minimum Viable Product (MVP), which means testing every interface and the communication between the components. With this test, it is easier to find out what would not work properly and which components can be used. By achieving the PoC or MVP, most of the risks can be eliminated or at least reduced.

18.1.3 Construction Phase

During the construction phase, the objective is to put the product's functionality into practice. Additionally, attempt to identify solutions for any lingering risks. Upon completion of this phase, the product is ready for deployment.

18.1.4 Transition Phase

In the final phase, the transition phase, the beta version needs to be tested, and the deployment process can begin. This phase also includes the final steps to complete the project.

18.2 Scrum

The Scrum methodology was used for the short-term plan. The duration of the sprint is two weeks because of previous good experience with this sprint duration and the project's limitations of fourteen weeks. The decision to combine Scrum and RUP was made because it had already been used successfully in a project leading up to the bachelor thesis. This method was also recommended by our professors.

18.2.1 Scrum Meetings

The Scrum methodology is based on a series of meetings, which are described below.

Daily Scrum

This meeting is not held every day, as each team member can only work on this project for up to three days a week. Instead, this meeting is only held on Mondays and Fridays. This allows everyone to be able to work in between meetings and still keep up to date. After each meeting with our advisor from the OST and the advisors from IBM, there was another meeting that lasted about 15 minutes to an hour to discuss any changes that needed to be made or how to proceed.

Sprint Planning

After each sprint, the week starts with a retrospective meeting on Monday, followed by planning for the next sprint. The main topics to focus on are defined, and the product backlog is discussed for to update tasks or priorities.

Sprint Review

The results of the sprint are discussed at the end of each sprint during the Friday meeting.

Sprint Retrospective

This meeting is of great importance for the improvement of the quality of our Scrum Sprints. Following each sprint, the initial segment of the subsequent Monday's meeting is dedicated to retrospection. The focus is on discussing successes and areas for improvement so that we can refine our processes for the upcoming sprint.

18.2.2 Scrum Roles

To manage the project, Scrum roles were used and additional competencies were assigned to each team member.

The assigned competencies should not indicate that this team member is solely responsible for this area of work. It means that this person has the main responsibility. However, all team members can work in any area of the project.

Table 18.1: scrum-roles

Team Member	**Competences**
Christopher Hilfing	Code Quality Assurance
Petra Heeb	Product Owner, Documentation
Lara Gubler	Scrum Master, Architecture

In addition, each team member acts as a developer and contributes code to the project.

18.3 Milestones

- M1 - Project Setup
- M2 - Proof of Concept
- M3 - End of Elaboration / Architecture
- M4 - Interim Presentation
- M5 - Quality
- M6 - End of Implementation
- M7 - Submit Final Documentation

The appendix contains the Milestones Checklist.

18.4 Industrial Partner

This bachelor thesis was carried out with an external partner, the IBM Research Lab in Rüschlikon. The advisors at IBM Research were Dr. Martin Schmatz and Dr. Basil Hess. Access to the workplace was agreed between the students and the industrial partner, as was the use of any special equipment.

18.5 Mentoring

The first weeks of this bachelor thesis were supervised by Prof. Dr Nathalie Weiler. Prof Dr Mitra Purandare then took over as the new advisor.

Dr. Martin Schmatz and Dr. Basil Hess supervised the bachelor thesis at the IBM Research Lab.

At the beginning of the semester, a regular weekly meeting was arranged between the students and the supervisors involved.

Chapter 19 Risk Management

In the following table, the identified risks for this project are summarized. Additionally, they are categorized in terms of severity and probability and visualized these in a risk matrix. The numbers of figure Risk Matrix match the risk number in the table Risks with a detailed description.

Table 19.1: Risks

Nr.	Description	Countermeasure	Severity	Probability
1	Absence of a team member	Communicate through Signal/Teams to re-distribute tasks if necessary	Low	Moderate
2	Scope creep	▪ Regular meetings ▪ Define exactly what is necessary and what is optional ▪ Pay close attention to progress with the usage of project management software	Moderate	Low
3	More time than planned is needed for individual components	A buffer is always included in the schedule	Moderate	Moderate
4	The requirements are incomplete	Check the requirements regularly in the meeting with the advisors	Moderate	Low
5	There are misunderstandings about requirements	Regular presentation of the current state of the project and time for questions during meetings	Low	Low
6	A Meeting with IBM can not take place (e.g. schedule collision)	Questions can also be asked by email	Very Low	Moderate
7	Too little communication	▪ Team members meet regularly ▪ There is a Teams chat for regular communication and a signal chat to quickly message someone	Moderate	Very Low
8	Unknown infrastructure. (Servers, HSM and data are provided by IBM.)	A time buffer is included in the project plan to get acquainted with the new environment.	Low	Very Low
9	No prior experience with similar projects	The first weeks were used to gain some basic knowledge about HSMs and read through the documentation of how to write an app for it.	Low	Very Low
10	Limited experience with the programming language C	The first weeks were used to get a better understanding of the language.	Moderate	Low
11	New architecture	A draft is created, which is discussed with our supervisors.	Low	Low
12	Bugs are noticed only late during the development	Automated unit tests, known answers, integration and sensible system tests will be written.	Moderate	Low

13	Dilithium does not work on a MIPS architecture	It is be tested if it works in the first weeks and if not a different algorithm will be used for this project.	Eliminated	Eliminated - PoC (13.03.2023)
14	HSM limitations will be reached	Potential risks and limitations are discussed during the weekly meetings.	Low	Low
15	HSM documentation: Information about the HSM and how SM-Apps are written can only be found in confidential documentation. This can be a risk if the documentation is not complete or is difficult to work with	Contact people that have read the documentation already	Moderate	Moderate
16	Running out of time, due to the problems with the HSM documentation	It was decided to reduce the scope of the MVP where possible	High	High

Very Low	Low	Moderate	High	Very High	Probability / Severity
					Very High
			16		**High**
7	2, 4, 10, 12	3, 15			**Moderate**
8, 9, 16	5, 11, 14	1			**Low**
		6			**Very Low**

Figure 19.1: Risk Matrix

Chapter 20 Risks Incurred

20.1 Advisor Change

During the first weeks of our bachelor thesis, our advisor unexpectedly left the project. As a result, it was necessary to find a new advisor and do an onboarding to bring her up-to-date.

20.2 Nr. 1, Absence of a team member

During the second week, two of our team members were unable to work for several days due to illness. While this was unfortunate, it did not have much of an impact on the progress of the project.

Table 20.1: Absence of a team member

Date	Time	Comment
Monday 27.02.2023	16h	Two team members were sick
Wednesday 08.03.2023	8h	One team member was sick

20.3 Nr. 3, More time than planned is needed for individual components

The development of an OpenSSL Provider/Engine is taking longer than expected, and due to dependencies, some parts of the project have not been able to continue for two weeks.

Due to the complexity and the knowledge of our advisors, we have had to switch the development from the Provider to the Engine. This involves a different version of OpenSSL and rebuilding the entire OpenSSL setup. The time spent developing the Provider was lost in the switch to the Engine. Neither the provider nor the engine are documented in an easy to understand way, so a lot of time is spent understanding the setup and preparing our environment for development.

20.4 Nr. 6, A Meeting with IBM could not take place

Due to holidays or scheduling conflicts, some meetings could not be held as planned. All meetings were either rescheduled or, at the most there were no meetings with IBM for a week. The dates of the individual meetings can be found in the Meeting Minutes.

20.5 Nr. 15, HSM documentation

During development, many problems were encountered with the Marvell documentation. They are all listed in the appendix under Risk: HSM documentation.

20.6 Nr. 16, Running out of time

At the beginning of May, we realized that we would not be able to meet the optimal time budget of 360 hours per person. This was due to a number of risks that delayed us or took up more time than planned.

20.7 PKCS#11 not running on Fafnirb4

The Bring Your Own Key PoC was originally intended to use PKCS#11 to retrieve the wrapped CRK and its KCV, as requested by IBM. Some research was done at the beginning of this project into how difficult it was to use PKCS#11, but as many resources were found with code examples, and as it was known that people at IBM had previously written code using this API, this particular part of the PoC was not considered to be a major uncertainty as to whether it would work or not, and was therefore not considered to be a risk.

However, during the development of this PoC it became apparent that using the provided library to use PKCS#11 did not work. None of the examples provided were able to run on the Fafnirb4 server, including the examples provided by Marvell which were already built.

This issue was discussed in the meeting on 17.05.2023. It was found that this was probably an error in IBM's custom setup. Due to these circumstances it was decided that this PoC will be implemented without the use of PKCS#11.

Chapter 21 Time Tracking Report

For the bachelor thesis, a fixed amount of hours is given. For the whole project, each team member got 360 hours (+/- 20%). Therefore, time tracking was important to manage the time resource.

The table spent time gives an overview of the time spent per team member.

Table 21.1: Spent time per team member

Name	Spent hours
Christopher	399h 10m
Lara	370h 2m
Petra	407h 45m

21.1 Time spent per Category

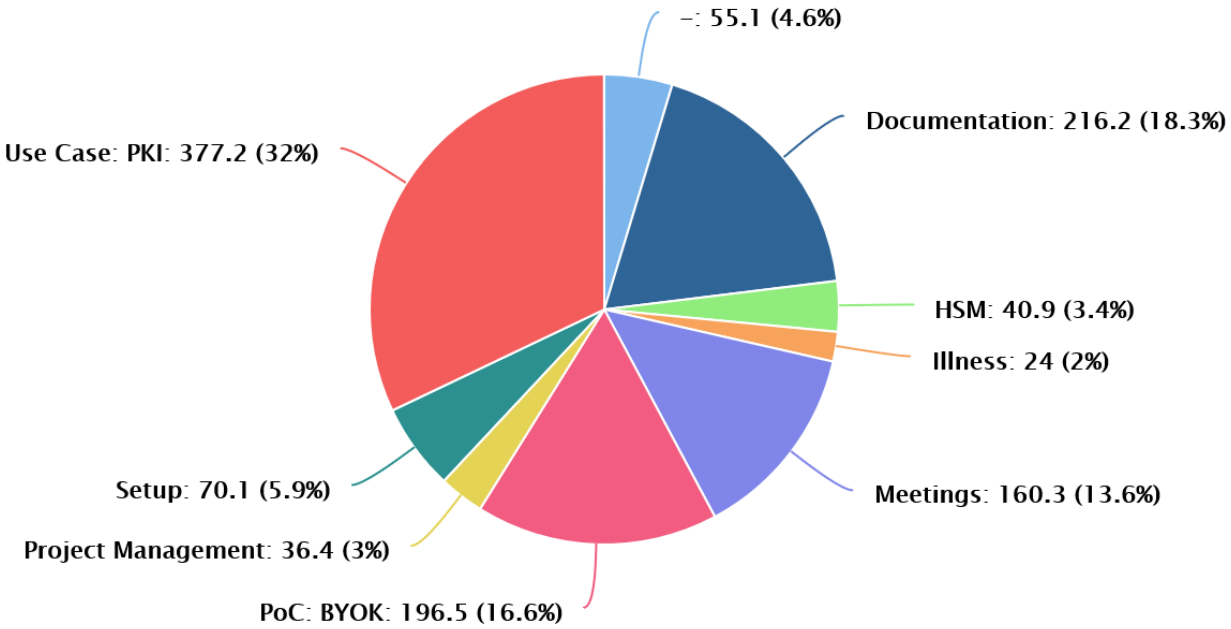

Figure 21.1: Time per Category

The pie chart is divided into the different categories that were used in Jira. The numbers near the category name are the number of hours spent for this category and in percentage how much this is in comparison to the other categories.

Category Description:

Setup : The setup category is for the initialization part of the project, to set up Jira, the GitHub repository and some installations.

HSM : The HSM category includes the Proof of Concepts for the Dilithium and Kyber algorithm done at the beginning of the project.

Documentation : The documentation tasks, which are not directly linked to an implementation task, are in the documentation category to track the time used for project documentation.

Meetings : All meetings are tracked in this category to show the meeting time for internal meetings and advisor synchronizations.

Project Management : Project management includes risk management, time tracking, project planning, and all the other organizational matters that needed to be done.

PoC: BYOK : Implementing the Proof of Concept for Bring Your Own Key belongs to this category, including documentation regarding each task.

Use Case: PKI : Implementing the use case Public Key Infrastructure belongs in this category, including documentation regarding each task.

Illness : If a team member was sick the time was tracked in this category because of a risk based on the time in this category.

- : This category is for tasks not connected to one of the above categories. This could be tasks for reworking all repositories and so on.

21.1.1 Christopher

The main focus of Christopher was the implementation of the Proof of Concept for Bring Your Own Key. Therefore, he spent most of his time in this category.

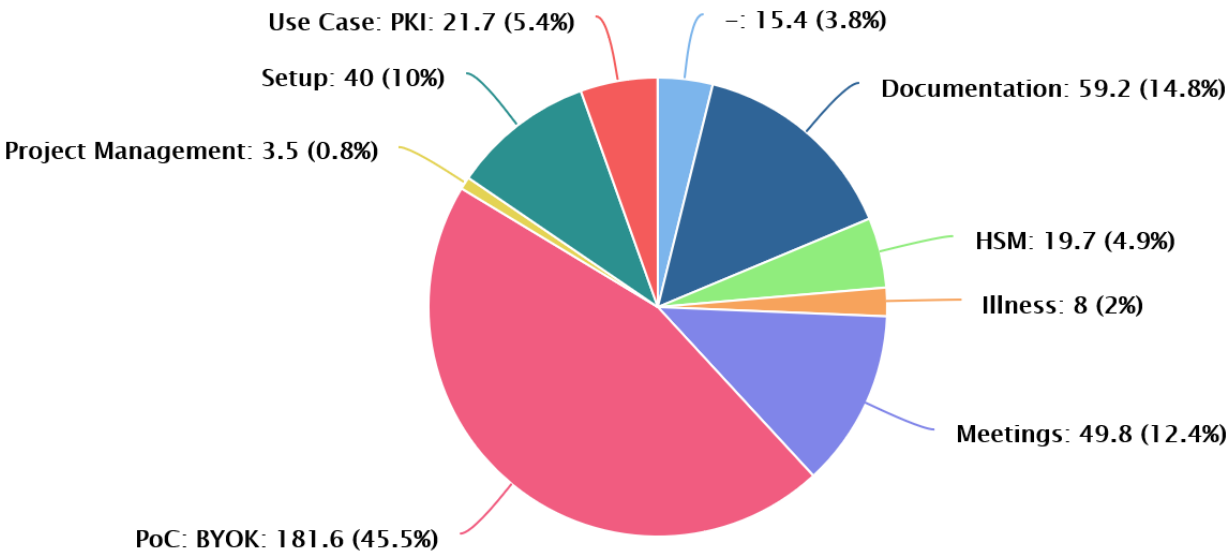

Figure 21.2: Time per Category - Christopher

21.1.2 Lara

Lara was mainly responsible for the OpenSSL Engine implementation in the use case of Public Key Infrastructure. Because of the challenges in this part of the project, the time amount was higher than planned.

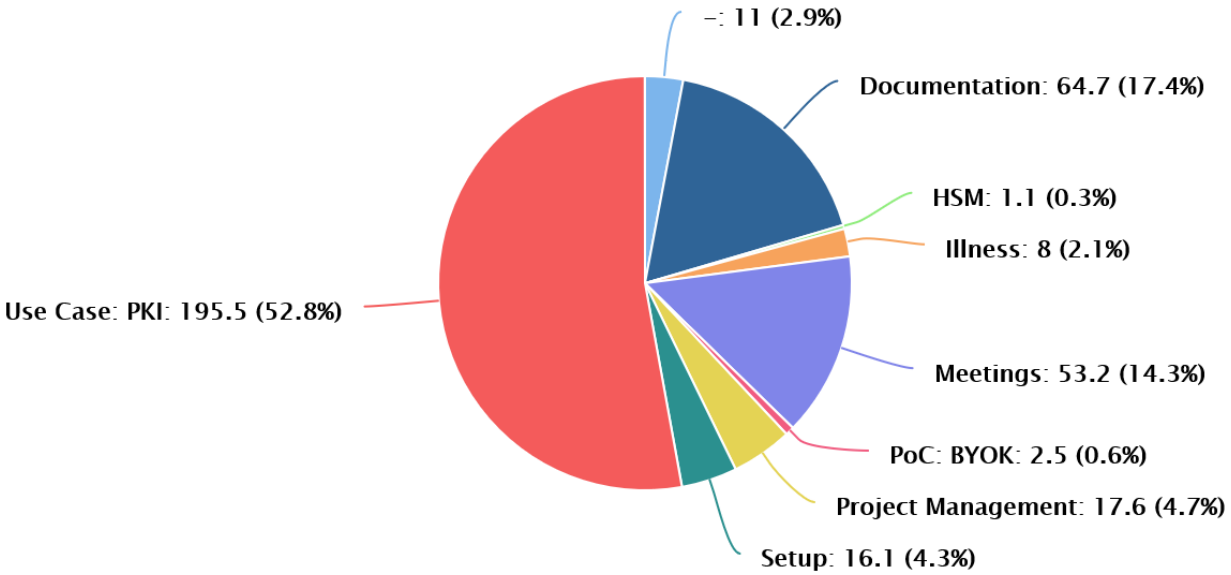

Figure 21.3: Time per Category - Lara

21.1.3 Petra

The main task of Petra was the SM-App and Host side implementation of the Dilithium algorithm in the Public Key Infrastructure use case.

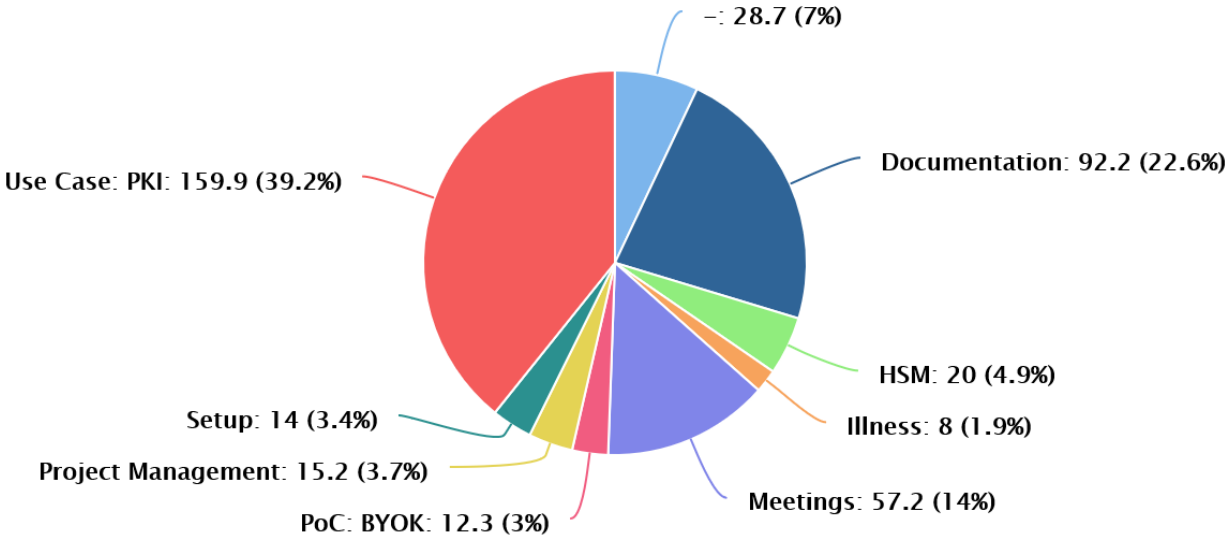

Figure 21.4: Time per Category - Petra

Bibliography

[1] IBM. "Ibm scientists help develop nist's quantum-safe standards." (), [Online]. Available: https://research.ibm.com/blog/nist-quantum-safe-protocols. (accessed: 08.03.2023).

[2] IBM. "Ibm: Quantum computing poses an 'existential threat' to data encryption." (), [Online]. Available: https://venturebeat.com/security/ibm-quantum-computing/. (accessed: 08.03.2023).

[3] NIST. "Post-quantum cryptography." (), [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography. (accessed: 08.03.2023).

[4] IBM. "How we quantum-proofed ibm z16." (), [Online]. Available: https://research.ibm.com/blog/z16-quantum-safe-migration. (accessed: 08.03.2023).

[5] Marvell, "Ls1 sdk documentation release 2.08-01 rc08," Chapter Managing Users.

[6] M. B. et al. "Nist ir 8320, hardware-enabled security: Enabling a layered approach to platform security for cloud and edge computing use cases." (), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8320.pdf. (accessed: 03.03.2023).

[7] M. Karu. "Secure cryptographic operations with hardware security modules." (), [Online]. Available: https://medium.com/@mevan.karu/secure-cryptographic-operations-with-hardware-security-modules-d54734834d7e. (accessed: 23.02.2023).

[8] M. Karu. "Standard api for connecting hsms with client applications." (), [Online]. Available: https://medium.com/@mevan.karu/standard-api-for-connecting-hsms-with-client-applications-6296eb187d89. (accessed: 23.02.2023).

[9] IBM. "Dilithium." (), [Online]. Available: https://pq-crystals.org/dilithium/. (accessed: 24.02.2023).

[10] NSA. "Announcing the commercial national security algorithm suite 2.0." (), [Online]. Available: https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF. (accessed: 24.02.2023).

[11] IBM. "Kyber." (), [Online]. Available: https://www.pq-crystals.org/kyber/. (accessed: 24.02.2023).

[12] Citrix. "What is an application delivery controller (adc)?" (), [Online]. Available: https://www.citrix.com/solutions/app-delivery-and-security/what-is-application-delivery-controller.html. (accessed: 22.03.2023).

[13] H. W. Sandeep Batta Luis Carlos Silva. "Use ibm cloud hyper protect crypto services to offload nginx tls." (), [Online]. Available: https://developer.ibm.com/tutorials/use-hyper-protect-crypto-services-to-offload-nginx-tls/. (accessed: 13.06.2023).

[14] J. Hallberg, *Using secure sockets layer bridging and content filtering mechanisms to provide defense in-depth when publishing ssl encrypted web hosts.* 2005. [Online]. Available: https://www.sans.org/white-papers/1573/ (visited on 04/20/2023).

[15] ClickSSL. "What is ssl offloading? features and benefits of ssl offloading." (), [Online]. Available: https://www.clickssl.net/blog/what-is-ssl-offloading-features-benefits-of-ssl-offloading. (accessed: 20.04.2023).

[16] OpenSSL. "Welcome to openssl." (), [Online]. Available: https://www.openssl.org/. (accessed: 12.06.2023).

[17] OpenSSL. "Engines." (), [Online]. Available: https://github.com/openssl/openssl/blob/master/README-ENGINES.md. (accessed: 30.04.2023).

[18] OpenSSL. "Provider." (), [Online]. Available: https://www.openssl.org/docs/man3.0/man7/provider.html. (accessed: 30.04.2023).

[19] OpenSSL. "Providers." (), [Online]. Available: https://github.com/openssl/openssl/blob/master/README-PROVIDERS.md. (accessed: 30.04.2023).

[20] open quantum safe project. "Liboqs." (), [Online]. Available: https://github.com/open-quantum-safe/liboqs. (accessed: 26.04.2023).

[21] NIST. "Integrity glossary." (), [Online]. Available: https://csrc.nist.gov/glossary/term/Integrity. (accessed: 26.05.2023).

[22] NIST. "Authentication glossary." (), [Online]. Available: https://csrc.nist.gov/glossary/term/authentication. (accessed: 26.05.2023).

[23] NIST. "Non-repudiation glossary." (), [Online]. Available: https://csrc.nist.gov/glossary/term/non_repudiation. (accessed: 26.05.2023).

[24] E. Barker. "Recommendation for key management: Part 1 – general." (), [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final. (accessed: 19.04.2023).

[25] ensia. "Public key infrastructure (pki)." (), [Online]. Available: https://www.enisa.europa.eu/topics/incident-response/glossary/public-key-infrastructure-pki. (accessed: 26.04.2023).

[26] C. Team. "Crystals-dilithium: A lattice-based digital signature scheme." (), [Online]. Available: https://eprint.iacr.org/2017/633.pdf. (accessed: 13.06.2023).

[27] IBM. "Public key certificate." (), [Online]. Available: https://www.ibm.com/docs/en/sia?topic=osdc-private-keys-public-keys-digital-certificates-27. (accessed: 13.06.2023).

[28] Anuradha. "Build the basic entities in the chain of trust in your organization." (), [Online]. Available: https://www.encryptionconsulting.com/what-is-the-certificate-chain-of-trust/. (accessed: 26.04.2023).

[29] Keyfactor. "What is the certificate chain of trust?" (), [Online]. Available: https://www.keyfactor.com/blog/certificate-chain-of-trust/. (accessed: 14.03.2023).

[30] N. C. S. Center. "Design and build a privately hosted public key infrastructure." (), [Online]. Available: https://www.ncsc.gov.uk/collection/in-house-public-key-infrastructure. (accessed: 27.04.2023).

[31] Microsoft. "Securing pki: Planning a ca hierarchy." (), [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786436(v=ws.11). (accessed: 17.05.2023).

[32] P. Long. "Microsoft pki planning and deploying certificate services." (), [Online]. Available: https://www.petenetlive.com/KB/Article/0001309. (accessed: 12.05.2023).

[33] E. Kompedium. "Ocsp - online certificate status protocol." (), [Online]. Available: https://www.elektronik-kompendium.de/sites/net/1907091.htm. (accessed: 12.05.2023).

[34] Globaltrust. "Verwendung der certificate revocation list (crl)." (), [Online]. Available: http://www.globaltrust.eu/php/cms_monitor.php?q=PUB&s=11241qpp. (accessed: 12.05.2023).

[35] E. Consulting. "What is the certificate signing request (csr)?" (), [Online]. Available: https://www.encryptionconsulting.com/education-center/csr/. (accessed: 12.05.2023).

[36] R. Publico. "Ssl-grundlagen: Was ist ein certificate signing request (csr)?" (), [Online]. Available: https://www.globalsign.com/de-de/blog/was-ist-ein-certificate-signing-request. (accessed: 12.05.2023).

[37] S. S. Team. "Pem, der, crt, and cer: X.509 encodings and conversions." (), [Online]. Available: https://www.ssl.com/guide/pem-der-crt-and-cer-x-509-encodings-and-conversions/. (accessed: 12.05.2023).

[38] NCSC. "Design and build a privately hosted public key infrastructure." (), [Online]. Available: https://www.ncsc.gov.uk/collection/in-house-public-key-infrastructure. (accessed: 20.04.2023).

[39] C. Team. "Kyber." (), [Online]. Available: https://eprint.iacr.org/2017/634.pdf. (accessed: 24.02.2023).

Part III

Appendix

Appendix A Milestones Checklist

A.1 M1 - Project Setup

Table A.1: M1 - Project Setup

#	Topic	Status
1	Progress corresponds to project plan	ok
2	Time tracking is up-to-date	ok
3	Documentation is concise and up-to-date	ok
4	Collaboration as a team is defined	ok
5	Roles are assigned	ok
6	Meetings are planned	ok
7	A long-term plan for the project exist	ok
8	A short-term plan for next iteration exists	ok
9	Risks are identified and assessed	ok
10	Risks do have an impact on the long-term plan of the project	ok
11	Documentation is under version control	ok
12	Documentation can be built automatically	ok
13	Issue management is set up	ok
14	Time tracking is set up	ok
15	FR are tracked on multiple levels	ok
16	FR are documented in an appropriate manner	ok
17	FR are prioritized	ok
18	FR are estimated	ok
19	FR with the highest priority are refined	ok
20	NFR are tracked	ok
21	NFR are measurable	ok
22	Process for validating NFR is defined	ok
23	Context diagrams	ok
24	Personas	ok

A.2 M2 - PoC

Table A.2: M2 - PoC

#	Topic	Status
1	Progress corresponds to project plan	ok
2	Time tracking is up to date	ok
3	Documentation is concise and up to date	ok
4	Dilithium prototype is running and can be demonstrated	ok
5	Dilithium prototype covers all layers and tiers (end-to-end)	ok
6	Dilithium prototype focuses on technical risks and important non-functional requirements	ok

A.3 M3 - End of Elaboration and Prototype

Table A.3: M3 - End of Elaboration and Prototype

#	Topic	Status
1	Progress corresponds to project plan	ok
2	Time tracking is up to date	ok
3	Documentation is concise and up to date	ok
4	Project plan was refined	ok
5	High-priority risks were mitigated and risk management was updated	ok
6	Requirements were refined according to priority	ok
7	Domain Model covers all domain concepts relevant for the project	ok
8	DEV-environment is fully set up	ok
9	A test concept is created	ok
10	The role of an architect is assigned	ok
11	Non-functional requirements with the highest priorities are addressed	ok
12	Interfaces to outside systems are clarified	ok
13	Adequate technologies and patterns are evaluated	ok
14	Architecture is documented	ok
15	The architecture is an evolution of the initial draft	ok
16	The architecture is adequate for the problem	ok
17	The chosen technologies are adequate for the problem	ok
18	The functional requirements with the highest priorities are addressed	ok
19	Use of Design Patterns when appropriate	-
20	Kyber prototype is running and can be demonstrated	ok
21	Kyber prototype covers all layers and tiers (end-to-end)	ok
22	Kyber prototype focuses on technical risks and important non-functional requirements	ok

A.4 M4 - Interim Presentation

Table A.4: M4 - Interim Presentation

#	Topic	Status
1	Progress corresponds to project plan	ok
2	Time tracking is up-to-date	ok
3	Documentation is concise and up-to-date	ok
4	Presentation date is set	ok
5	Proofreader, IBM supervisor, a supervisor from the OST, and the external expert were invited	ok
6	A PowerPoint presentation is prepared	ok

A.5 M5 - Quality

Table A.5: M5 - Quality

#	Topic	Status
1	Progress corresponds to project plan	ok
2	Time tracking is up-to-date	ok
3	Documentation is concise and up-to-date	ok
4	Reviews are organized	ok
5	Quality is verified as part of the DEV-process	ok
6	Quality verification is supported by tools	ok
7	Quality measures are documented properly	ok
8	The test concept describes the strategy for testing the product	ok
9	The product is tested on different levels	ok
10	Tests are automated when appropriate	ok
11	Test are performed at an appropriate time	ok
12	Tests cover an appropriate amount of the product	ok
13	Tests with end-users are planned	-
14	Code corresponds to design and documentation	ok
15	Code is "clean"	ok
16	Code quality is verified by tools	ok

A.6 M6 - End of Implementation

Table A.6: M6 - End of Implementation

#	Topic	Status
1	Progress corresponds to project plan	ok
2	Time tracking is up-to-date	ok
3	Documentation is concise and up-to-date	ok
4	The product is tested	ok
6	System tests are successful	ok
7	Code corresponds to design and documentation	ok
8	Code is "clean"	ok

A.7 M7 - Submit Final Documentation

Table A.7: M7 - Submit Final Documentation

#	Topic	Status
1	Progress corresponds to project plan	ok
2	Time tracking is up-to-date	ok
2	Time tracking is added to the documentation	ok
3	Documentation is concise and up to date	ok
4	Signed "Eigenständigkeitserklärung"	ok
5	Signed agreements*, copyrights and rights of use (if applicable)	ok
6	Meeting minutes are in the appendix	ok
7	The task description is attached	ok
8	Abstract, acknowledgment and management summary is written	ok
9	Conclusions and outlook is written	ok
10	The report does not contain signatures, e-mail addresses or telephone numbers	ok
11	All relevant files are submitted	ok

Appendix B Test Protocols

Table B.1: System Tests BYOK: 8.05.2023

Nr.	Test Name	Result
1	Run Host-App without arguments	SUCCESS
2	Run Host-App without some arguments	SUCCESS
3	Run Host-App without too long username	SUCCESS
4	Run Host-App without too long password	SUCCESS
5	Run Host-App without too long partition name	SUCCESS
6	Run Host-App with a too-short PEK file	SUCCESS
7	Run Host-App with a too-long PEK file	SUCCESS
8	Run Host-App with incorrect values	SUCCESS
9	Run Host-App without running the SM-App	SUCCESS
10	Run Host-App and SM-App	SUCCESS
11	Check the Pipelines	NA

Table B.2: System Tests BYOK: 12.05.2023

Nr.	Test Name	Result
1	Run Host-App without arguments	SUCCESS
2	Run Host-App without some arguments	SUCCESS
3	Run Host-App without too long username	SUCCESS
4	Run Host-App without too long password	SUCCESS
5	Run Host-App without too long partition name	SUCCESS
6	Run Host-App with a too-short PEK file	SUCCESS
7	Run Host-App with a too-long PEK file	SUCCESS
8	Run Host-App with incorrect values	SUCCESS
9	Run Host-App without running the SM-App	SUCCESS
10	Run Host-App and SM-App	SUCCESS
11	Check the Pipelines	NA

Table B.3: System Tests BYOK: 26.05.2023

Nr.	Test Name	Result
1	Run Host-App without arguments	SUCCESS
2	Run Host-App without some arguments	SUCCESS
3	Run Host-App without too long username	SUCCESS
4	Run Host-App without too long password	SUCCESS
5	Run Host-App without too long partition name	SUCCESS
6	Run Host-App with a too-short PEK file	SUCCESS
7	Run Host-App with a too-long PEK file	SUCCESS
8	Run Host-App with incorrect values	SUCCESS
9	Run Host-App without running the SM-App	SUCCESS
10	Run Host-App and SM-App	SUCCESS
11	Check the Pipelines	SUCCESS

Table B.4: System Tests BYOK: 09.06.2023

Nr.	Test Name	Result
1	Run Host-App without arguments	SUCCESS
2	Run Host-App without some arguments	SUCCESS
3	Run Host-App without too long username	SUCCESS
4	Run Host-App without too long password	SUCCESS
5	Run Host-App without too long partition name	SUCCESS
6	Run Host-App with a too-short PEK file	SUCCESS
7	Run Host-App with a too-long PEK file	SUCCESS
8	Run Host-App with incorrect values	SUCCESS
9	Run Host-App without running the SM-App	SUCCESS
10	Run Host-App and SM-App	SUCCESS
11	Check the Pipelines	SUCCESS

Nr.	Test Name	Result
1	Run Host-App without arguments	SUCCESS
2	Run Host-App without some arguments	SUCCESS
3	Run Host-App without too long username	SUCCESS
4	Run Host-App without too long password	SUCCESS
5	Run Host-App without too long partition name	SUCCESS
6	Run Host-App with a too-short PEK file	SUCCESS
7	Run Host-App with a too-long PEK file	SUCCESS
8	Run Host-App with incorrect values	SUCCESS
9	Run Host-App without running the SM-App	SUCCESS
10	Run Host-App and SM-App	SUCCESS
11	Check the Pipelines	SUCCESS

Table B.5: System Tests BYOK: 14.06.2023

Table B.6: System Tests PKI: 12.06.2023

Nr.	Test Name	Result
1	Build the Engine	SUCCESS
2	Generate a dilithium3 key	SUCCESS
3	Generate a self-signed dilithium3 certificate	SUCCESS
4	Generate a self-signed dilithium3 certificate, providing an unavailable RootCa.conf file	SUCCESS
5	Generate a self-signed dilithium3 certificate with an unavailable extension	SUCCESS
6	Extract the public key from the certificate	SUCCESS
7	Sign	FAIL
8	Verify	FAIL

Table B.7: System Tests PKI: 14.06.2023

Nr.	**Test Name**	**Result**
1	Build the Engine	SUCCESS
2	Generate a dilithium3 key	SUCCESS
3	Generate a self-signed dilithium3 certificate	SUCCESS
4	Generate a self-signed dilithium3 certificate, providing an unavailable RootCa.conf file	SUCCESS
5	Generate a self-signed dilithium3 certificate with an unavailable extension	SUCCESS
6	Extract the public key from the certificate	SUCCESS
7	Sign	SUCCESS
8	Sign without working connection to the SM-App (e.g wrong password)	SUCCESS
9	Verify	SUCCESS

Appendix C Meeting Minutes

This chapter includes the meeting notes for the weekly synchronization meeting with our advisors from the university and IBM Research.

C.1 Kick-off Meeting 2023-02-22

C.1.1 Deadlines

Table C.1: Kick-off Meeting Deadlines

Date	Comment
April/May	Interim presentation
2023-06-12	Hand-in Abstract, Poster
2023-06-16	Hand-in Documentation
until 2023-08-31	Final presentation and defense

C.1.2 Preparation Steps

- Read Crystals Website
- Read about MIPS Processors
- Read Dilithium NIST standard
- Orient on PKCS#11 (version 204) for HSM programming
- Test "Hello World" on Software HSM (SoftHSM)

C.1.3 First Step

It needs to be tested if the CRYSTALS-Dilithium and CRYSTALS-Kyber work on the HSM (MIPS Processor) The reason for this is that the private key should only exist in the HSM (also the signatures are handled by the HSM) and nowhere else, so the key pairs cannot be generated externally.

C.1.4 Tasks

Table C.2: Kick-off Meeting Tasks

Due Date	Description	Responsible
2023-02-22	Setup GitHub repository	Chris
2023-02-22	Setup Jira project	Petra
2023-02-22	Setup documentation	Petra
2023-02-27	Project plan draft	together

C.2 Weekly Meeting 2023-02-27

C.2.1 Notes

Project Plan : no spring holidays, only for lectures not for thesis

Architecture : change naming from driver to kernel driver

Dilithium implementaiton :

- unoptimized code as starting point
- Basil put some links for the correct implementation code into the Box folder

C.3 Weekly Meeting 2023-03-06

C.3.1 Tasks

- go through HSM documentation for personas
- update what hybrid means (Kyber + RSA elliptic curve) on Functional Requirements (FR)
- update risk matrix graphic
- go through Marco Zanetti's semester thesis (chapter HSM)
- add development architecture - how the application is cross-compiled etc.
- document that known answer tests should be used - next to the unit tests
- diagram architecture - Dilithium should be on the HSM side not on the Host-App side

C.3.2 Notes

Interim presentation :

- 2023-04-17
- same day as Sechseläuten - holiday in Zurich (is still okay)

HSM doucmentation

- go through HSM documentation for personas: crypto officer, user, ...
- write more clearly in each case whether it is a user on the HSM or the Host-App that uses the HSM

Hybrid is Kyber + RSA elliptic curve :

- is not backward compatible
- see Marco Zanetti's work - post-quantum cryptography to run parallel what is needed

Architecture :

- add development architecture
- how is the application cross-compiled etc.

Emulators : Using emulators the Dilithium code ran (Basil tested this)

Documentation Documentation is wrong for building the app (outdated - will be looked at on Wednesday)

C.3.3 Questions

Table C.3: Meeting 2023-03-06 Questions

Question	Answer
Should the Dilithium algorithm be implemented in the secure machine?	yes
Was the custom app implementation tested by IBM?	only one example from the manufacturer was tried

C.4 Weekly Meeting 2023-03-13

C.4.1 Tasks

- Contact Marvell - What can we use in our documentation from their confidential documentation (Martin)
- Contact Martin for further information on how to proceed
- Fafnir is misspelled as "Favnir" on one graphic displaying the architecture

C.4.2 Notes

Use Cases :

- Kyber - one person
- Dilithium (PKI) - two persons

C.4.3 Questions

Table C.4: Meeting 2023-03-06 Questions

Question	Answer
How should the LS1 guides be cited?	We need to contact Marvell and check if we can use their material in our bachelor thesis documentation

C.5 Weekly Meeting 2023-03-20

C.5.1 Notes

Use Case Kyber :

- Need to create a PoC
- Using pkcs11 is a possible way to transfer the key
- How and why it was solved like this (using pkcs11/SM functionality) needs to be documented :)
- Session objects should be used instead of token objects!
- Vision needs to include the communication between the HSM and the SM-App

Use Case PKI :

- Having the intermediate certificate in HSM is great, but having just the root certificate in the HSM would be enough for IBM
- The images(8.2/8.1) need a better description (which key is which? Belonging to where?)

C.6 Weekly Meeting 2023-04-03

C.6.1 Tasks

- update PoC vision for BYOK
- update Key chain image for certificates

C.6.2 Notes

Use Case Kyber : The randomly generated AES256 key will be used as a CRK. Feedback to the image:

- 1 - IKEK needs to be imported via MKEK
- 4 - key needs to be wrapped with IKEK - unwrapped and imported only test afterward
- 10 - needs to be added: CRK key gets wrapped with IKEK (in the HSM) and sent to Host-App
- 5,6,7 - use HMAC on nonce instead of encryption (easier)
- 5,7. - Host-App should talk direktly to HSM via PKCS#11

Use Case PKI : The first image is correct.

- change last "wrap" to "crypt"
- make two images, the first for the wrapping process (prerequisites) and the second one for the unwrapping process (usage)
- CRK and IKEK are in the wrapped format available in the Keystore on the outside of the HSM
- use Martins "notation" to display which key unwraps which

C.7 Weekly Meeting 2023-04-13

C.7.1 Questions

Table C.5: Meeting 2023-04-13 Questions

Question	**Answer**
Any news regarding Marvell - are we allowed to use their images in our documentation?	Not yet
Is a documentation review until next Tuesday possible?	Yes, we will receive an email with their comments

C.8 Weekly Meeting 2023-04-24

C.8.1 Tasks

- Send Martin the documentation regarding the Marvell documentation
- Send Martin the Slides of the interim Presentation (watch out for more meetings)

C.8.2 Questions

Table C.6: Meeting 2023-04-24 Questions

Question	Answer
why does the Dockerfile use an Nginx server?	• The main idea was the Dockerfile should show how the oqs-openssl was built • the nginx is not that interesting for us. If there is enough time - maybe use nginx with our solution for basically free TLS offload • IBM wants super shortlived certificates - OCSP will not be checked anymore - would take longer than a lifetime of the certificate • OCSP is often skipped because of e.g. privacy violations
Regarding the question from the Expert during the interim meeting: Why are requirements the way they are? Should one store the intermediate certificate in the HSM as well?	• in our case the lowest in the chain must be such that it can sign leaf certificates at very high rates • a root CA key is in HSM - it sign once a day a number of the intermediate certificate for a bunch of clusters • the private key of the intermediate inside the cluster would regenerate every couple minutes the certificates inside the cluster • one could also have root and intermediate key in HSM - this would also have its drawbacks: qsc certificates are quite huge which creates a lot of traffic • This is just a PoC/Demo in the real world e.g. the root CA would be offline

C.9 Weekly Meeting 2023-05-08

C.9.1 Questions

Table C.7: Meeting 2023-05-08 Questions

Question	Answer
Was PKCS#11 already used/tested by IBM on the Fafnirb4?	Someone was already working on it 2 years ago. Martin will write an email to ask for helpful code snippets and to introduce us to him so that we can ask him questions directly.

C.10 Weekly Meeting 2023-05-17

C.10.1 Tasks

- Send Martin an invite to discuss the problems with PKCS#11 on the Fafnirb4

C.10.2 Questions

Table C.8: Meeting 2023-05-15 Questions

Question	Answer
Should the OpenSSL provider be modified to send the request to the HSM or the OQS library?	The changes should be made in the provider as multiple of them could be loaded

C.11 Weekly Meeting 2023-05-31

C.11.1 Notes

Diagram BYOK :

- IKEK is MKEK
- IKEK should be brought by oneself (document outlook)
- documentation and diagrams must be adapted - it must always be indicated which key was used exactly
- Diagram has an error - CRK is wrapped with the help of the HSM
- KCV access should be clearly shown as access to an attribute

Demo BYOK :

- errors during cleanup are not particularly important for IBM

C.11.2 Questions

Table C.9: Meeting 2023-05-31 Questions

Question	Answer
BYOK: should the wrapped key end up in a file, or is the output on the console sufficient (since it is only a demo)?	Doesn't have to be in a file - but it should be documented why it is more than 32 bytes.
Is it ok for the SM app to terminate after it has run once?	yes, is okay
May the username and password for the crypto user be given as arguments to the program?	Yes, for the demo it is ok. But it should be documented why it is done this way and why it should not be done this way usually.

C.12 Weekly Meeting 2023-06-05

C.12.1 Notes

PKI : IBM and our advisor were informed that due to slower progress than expected, this use case will not be finished in time.

C.12.2 Questions

Table C.10: Meeting 2023-05-31 Questions

Question	**Answer**
14.06.2023 will be our last time at IBM is Martin and Basil available?	Martin and Basil are available
SA publication (crypto Discovery): someone contacted us and is interested in our SA, what can we share?	Basil has already contacted the person in question and they will handle it.

Appendix D Risk: HSM documentation

In this chapter, the incurred risks of the HSM documentation are shown in more detail.

Deleted because of confidentiality.