POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Design and in-orbit Demonstration of a Post-Quantum Cryptographic Solution Based on KEMTLS-PDK to Enhance Satellite Communication Security

Supervisors:

Prof. Cataldo Basile

Dr. Nicoló Maunero

Eng. Dominik Marszk

Candidate:

Noemi Terzo

Academic Year 2022/2023
Torino

Ai miei genitori,
ai miei fratelli
e ai milioni di sonagli.

“Ho gli occhi
pieni di nuvole e comete.”
-Goliarda Sapienza

Abstract

In recent years, cyber-attacks leveraging quantum algorithms like Shor's and Grover's, executed on cryptanalytically-relevant quantum computers, have demonstrated their potential to compromise and break the cryptographic systems currently safeguarding information systems. Recognizing the urgent necessity to bolster data and communication security in the quantum age, the National Institute of Standards and Technology took a significant step in July 2022 by announcing the four post-quantum cryptographic (PQC) algorithms to standardize by 2024. Notably, Kyber was chosen as key establishment algorithm, marking a pivotal transition towards quantum-safe cryptographic solutions.

In the realm of space technology, current space systems predominantly rely on symmetric cryptographic primitives. However, there is a growing imperative to incorporate asymmetric cryptography into future systems. This strategic shift is driven by the overarching objectives, including enhancing flexibility for federated operations, adopting digital signature-based authentication, improving scalability – essential in new space and for projects with large constellations – and fostering interoperability with terrestrial systems.

Security disruption in space systems, particularly in terms of data integrity and authentication, can have profound consequences during data transmission to and from spacecraft. Data may be corrupted, manipulated, or sent by malicious actors, and unauthorized operations may be performed, representing significant risks. In extreme cases, these risks could result in mission failure, loss of human lives in the case of crewed missions and causing harm to people and property. Furthermore, the lack of data confidentiality in communications between satellites and between ground stations and satellites could expose private and sensitive information to untrusted parties. Therefore, given the quantum threat mentioned earlier, it is necessary to begin adopting quantum-resistant asymmetric cryptographic primitives in space systems.

The current Master's Thesis project was conducted during the internship at the European Space Operations Centre of the European Space Agency (ESA). The Thesis objective is to design and implement a specific solution that integrates Post-Quantum Cryptography into space missions to secure sessions between ground-based Mission Operations (MO) applications and those on spacecraft. To achieve this, a new MO service called Security Service is introduced. It implements KEMTLS-PDK protocol with Kyber512 as key establishment algorithm, and it is used to exchange encrypted messages after the successful creation of a secure session. Software security modules are used for key material storage and a Public Key Infrastructure on ground is employed for generating, revoking, and verifying the X.509 public key certificates of the nodes. The entire implementation has been

built on top of the CCSDS (Consultative Committee for Space Data Systems) MO Message Abstraction Layer (MAL), operationally used by ESA OPS-SAT spacecraft, to ensure ability to demonstrate the project in orbit.

The implementation was tested in a real-life scenario, firstly with OPS-SAT satellite's Engineering Model to provide a representative example of execution, and subsequently during a live ground-station pass with the flying satellite.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nowadays, cyber-attacks based on quantum algorithms, such as Shor's and Grover's ones, pose a significant threat to a large portion of the classical cryptography currently utilized in information systems. Cryptography disruption results in compromised confidentiality, leaving data accessible to unauthorized parties, potentially compromising sensitive information. The implications of such disruptions are particularly grave within Space domain, where lack of confidentiality may lead to improper functioning of space systems and may threat human life in scenarios where those systems involve human presence and information concerning individuals. Given these circumstances, it is necessary to start considering solutions that include Post-Quantum Cryptography to enhance security of space systems against evolving quantum threats.

This master's thesis work is the result of a six-months internship project conducted at the European Space Operations Centre of the European Space Agency, in Darmstadt. The primary objective was to develop a robust and functional architecture able to address the quantum threat by implementing the Post-Quantum Cryptographic protocol KEMTLS-PDK. The KEMTLS-PDK protocol is an Authenticated Key Exchange (AKE) based on a Key Encapsulation Mechanism (KEM) and represents a Post-Quantum Cryptographic evolution of TLS 1.3 protocol. This protocol is utilised on top of the Consultative Committee for Space Data Systems (CCSDS) Message Abstraction Layer, for secure session establishment between parties. These parties are *ground nodes*, i.e., Mission Operations (MO) applications that run on ground, and space nodes, i.e., MO applications onboard of a spacecraft. The proposed architecture includes a Public Key Infrastructure managed on ground, which is used to generate, revoke and verify X.509 public key certificates. Additionally, Software Security Modules are employed on end nodes to store key material, including session keys established through the KEMTLS-PDK-based handshake, and perform symmetric encryption and decryption of messages exchanged within a secure session.

The remainder of this document is structured as follows: Chapter 2 introduces the concepts of KEM and AKE, providing a classical example of a key encapsulation mechanism such as RSA-KEM, and an overview of TLS 1.3 and of TLS 1.3 handshake. Moreover, the concepts of Public Key Infrastructure, Certification Authority, OCSP Responder, OCSP stapling and Hardware Security Modules are explained, since all of them are key elements in the understanding of the architecture to implement.

Chapter 3 delves into Post-Quantum Cryptography (PQC). In order to understand

the importance of transitioning systems to a cryptography that is quantum-resistant, the chapter starts discussing Quantum Computing, Shor's algorithm and Grover's algorithm, that constitute threats against classical cryptography. The time needed to perform this transition is analysed, according to "Quantum Threat Timeline Report 2022" [15]. Then, the chapter explores CRYSTALS-Kyber algorithm, which is NIST's choice for key establishment, and the KEMTLS-PDK protocol.

In Chapter 4, the state-of-the-art in cryptography for space systems is discussed. The discussion starts by analysing a space system architecture. Possible cyber-attacks against space systems are presented to understand the importance of cybersecurity in Space domain. Some available countermeasures against these attacks are described, focusing later to how and where cryptography is implemented in a space system and which are the objectives of using it. At the end of the chapter, ESA, Skudo and CGI project that serves as the basis for the thesis is described.

Chapter 5 is dedicated to the design and subsequent implementation of the thesis project. Starting from the analysis of the designed architecture and of involved entities, the implementation is described in terms of software behaviour. The KEMTLS-PDK-based handshake, developed on top of Message Abstraction Layer, is described step-by-step, to explain what happens during the establishment of a secure session. At the end of the chapter, challenges faced during the implementation phase are also discussed.

Chapter 6 explores the integration of the implemented architecture into a real space system, using OPS-SAT satellite for experimentation. OPS-SAT experimental nature allows people from all around Europe to test their software onboard of the spacecraft. The steps to conduct the experiment, along with results and analysis, are provided.

Chapter 7 concludes the thesis, summarizing the entire procedure that led to the successful implementation of the project. The chapter proposes insights into potential future evolution of the implemented work and of the used tools, to conduct the work to a solution that increases the utilisation of Post-Quantum Cryptography.

Chapter 2

Background

In this chapter, the foundation for a comprehensive understanding of the thesis work is established by presenting key concepts essential to its context. The concepts discussed here, including Key Encapsulation Mechanism, Authenticated Key Exchange, Public Key Infrastructure, TLS 1.3 protocol and Hardware Security Module, have been studied during the Master's Degree courses and are crucial in shaping the landscape of the thesis.

2.1 Key Encapsulation Mechanism

A Key Encapsulation Mechanism (KEM) is an asymmetric cryptographic technique for securing symmetric keys employed to encrypt data without having to use the padding. Given two parties A and B, a KEM allows them to establish a shared secret key ss in the key space K together with its encryption C. The encryption C is used for sharing ss, while ss is used for long data encryption. A formal definition of a KEM is provided in [4]: given a key space K, a KEM is a triple of algorithms $(KG,\ E,\ D)$, where

- KG is the key generation algorithm that outputs key pairs $(pk,\ sk)$, where pk is the public key and sk is the private key.
- E is the probabilistic encapsulation algorithm, that encrypts pk and outputs a pair $(k,\ c)$, where k is a key that belongs to the key space K and c is the corresponding ciphertext.
- D is the decapsulation algorithm used to decrypt c with the private key sk and outputs the shared secret key k' that belongs to K and is equal to k.

This mechanism eliminates the complexity of the padding scheme and the proofs needed to show that the padding is secure, because the symmetric key ss is the result of encapsulation, so it is not necessary to map bit strings into algebraic message space as in traditional public-key encryption. A KEM is implicitly used in the latest version of TLS handshake, which will be described in section 2.4.1 of the current chapter.

An example of a classic KEM is presented in the following section, reserving the description of a post-quantum KEM for the forthcoming Post-Quantum Cryptography chapter.

2.1.1 RSA-KEM

A classic KEM algorithm is RSA-KEM, which is described in [7]. As shown in Figure 2.1, consider two peers, Alice and Bob, who wish to establish a shared secret key for securing communication. The mechanism unfolds as follows:

1. Alice initiates the process by sharing her public key (n, e) with Bob, where n is the modulus and e is the public exponent.

2. Bob generates a random integer r such that $1 < r < n$, which serves as the basis for the shared secret key. To derive the secret key ss, Bob employs a Key Derivation Function (KDF) that takes r as input. The KDF is a one-way function, ensuring that it is computationally infeasible to compute r from the secret key ss. This property is essential because even if an attacker gains access to ss, they cannot deduce the original random integer r.

3. Bob computes the encryption of r as $c = r^e \ mod \ n$ and transmits this ciphertext to Alice.

4. Upon receiving the ciphertext c, Alice uses her private key exponent d to perform the decryption as $r = c^d \ mod \ n$.

5. Alice applies the same KDF used by Bob to r for computing the shared secret key ss. This ensures that both parties possess the identical symmetric key for securing communications.

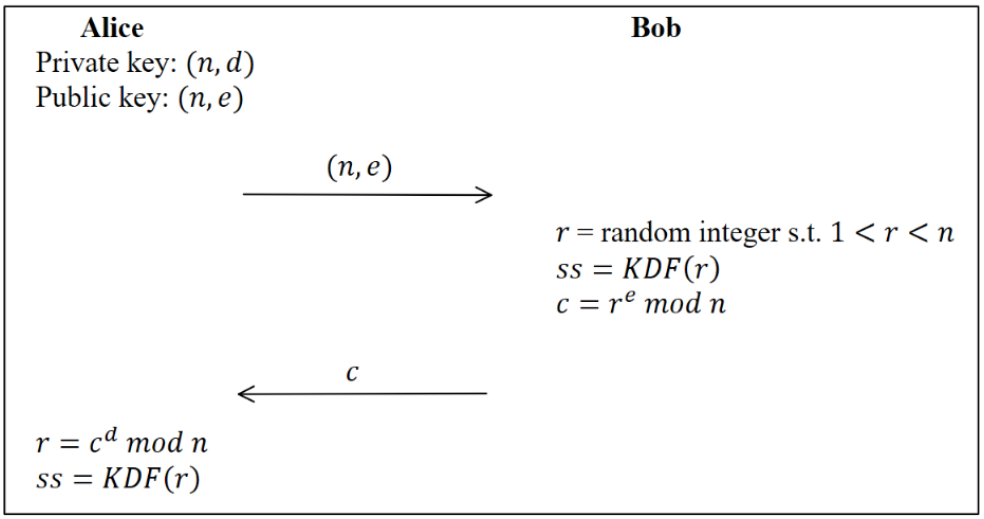

Figure 2.1. RSA-KEM

2.2 Authenticated Key Exchange

The *Authenticated Key Exchange* (AKE) is an important cryptographic primitive utilised in information and network security to establish secure communications over untrusted channels avoiding man-in-the-middle-based attacks. It relies on asymmetric cryptography - each participating entity possesses a keypair composed by a pubic key, which is freely exchanged, and a private key, which must be kept secret - and consists of:

- Negotiation of the cryptographic shared secret key between two parties, to be used for encrypting and decrypting exchanged data.
- Authentication of the parties involved in the key exchange. Authentication can be one-way or mutual, implicit or explicit. It is *explicit* if achieved during the execution of the protocol explicitly using other primitives like digital signature schemes. The authentication is *implicit* when it relies on the ability that only the parties have to compute the session key, without using other primitives [27].

An Internet protocol that uses AKE is TLS, and it is described in section 2.4. An explicitly authenticated key exchange is Diffie-Hellman AKE protocol where the two peers use long-term signature keys to perform authentication. An implicitly authenticated key exchange is Double Diffie-Hellman AKE protocol.

2.3 Public Key Infrastructure

A *Public Key Infrastructure* (PKI) is a technical and administrative infrastructure put in place for the creation, distribution and revocation of Public Key Certificates (PKCs). These certificates are data structures that securely bind a public key, used for cryptographic operations, to specific attributes used to identify the corresponding private key holder. This secure linkage is typically achieved through the digital signature of a trusted authority named Certification Authority, but there are also other techniques as placing certificates on a blockchain or establishing direct trust relationships with personal signatures. PKCs play a pivotal role in achieving non-repudiation of digital signatures, serving as concrete evidence admissible in a court of law, preventing the authors of digital signatures from denying their responsibility. Without a PKC, digital signatures can only provide authentication and integrity because the binding between the identity and the private key misses. The X.509 standard is widely adopted for associating public keys with identities.

2.3.1 Certification Authority

The *Certification Authority* (CA) is the authority responsible for issuing PKCs, digitally signing them with its private key. CAs can be organized hierarchically, where the CA of an organization issues PKCs to subordinate organizations' CAs. A CA can serve as an issuing CA with respect to the PKCs it issues and as a subject CA in relation to the PKC issued to it. CAs also possess the authority to revoke the certificates they have issued and maintain records of PKC statuses in Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP) servers.

2.3.2 Online Certificate Status Protocol

The *Online Certificate Status Protocol* (OCSP) is a client-server protocol designed to ascertain the real-time status of a PKC. The OCSP server responds to specific queries, providing information about the current validity of a particular PKC and digitally signing its responses. When the OCSP server is directly operated by the CA, it has direct access to the CA database.

2.4 Transport Layer Security 1.3

Transport Layer Security 1.3 is the latest version of the Transport Layer Security (TLS) protocol, which establishes a secure transport channel on top of transport layer (Layer 4) of the OSI model. As described in [23], TLS 1.3 provides server authentication and, optionally, client authentication. These authentication mechanisms employ asymmetric cryptography such as RSA and Elliptic-Curve Digital Signature Algorithm (ECDSA), or symmetric pre-shared key (PSK), or Edwards-Curve Digital Signature algorithm (EdDSA). TLS 1.3 ensures both data confidentiality and data integrity, permitting only the intended endpoints to understand the transmitted data and to detect any unauthorized change. It reduces the impact on performances and networking by reducing the setup time handshake latency. The handshake is not in clear because this version of the protocol enhances the encryption both for security and privacy. Privacy is guaranteed because it is not possible to find the client certificate that identifies the person who is using the browser.

Two objectives of TLS 1.3 are the improvement of resiliency against cross-protocol attacks, and the removal of legacy features no longer useful for TLS. The protocol is designed to facilitate a smooth and secure migration of the cryptography it uses. Specifically, TLS 1.3 no longer permits the use of RSA key exchange due to its inability to provide forward secrecy. Consequently, RSA keys are exclusively reserved for server authentication. For key exchange, Ephemeral Diffie-Hellman (DHE) or Elliptic-Curve DHE (ECDHE) are employed, with predefined groups to thwart potential attacks exploiting small DH parameters on the server side.

TLS 1.3 enhances message protection through the use of Authenticated Encryption with Associated Data (AEAD) algorithms and completely eliminates data compression. Digital signature for ephemeral keys is performed using RSA with the modern secure RSA-PSS (Probabilistic Signature Schema), and the entire handshake is digitally signed. Notably, TLS 1.3 does not specify entire ciphersuites but focuses on orthogonal elements, i.e., the encryption cipher, the encryption mode and the HKDF (i.e. Hashed Key Derivation Function). The allowed certificate types are RSA, ECDSA, EdDSA, while the allowed key exchange mechanisms are DHE and ECDHE, so in TLS 1.3 there are only five ciphersuites:

TLS_AES_128_GCM_SHA256

TLS_AES_256_GCM_SHA384

TLS_CHACHA20_POLY1305_SHA256

TLS_AES_128_CCM_SHA256

TLS_AES_128_CCM_8_SHA256

2.4.1 TLS 1.3 handshake

TLS comprises two fundamental protocols: the handshake protocol and the record protocol. The record protocol uses parameters established during the handshake protocol to protect communications, dividing data traffic into a series of records independently protected by means of traffic keys.

The TLS handshake protocol, as an Authenticated Key Exchange (refer to section 2.2 for more information), establishes a secure encrypted channel between a client ad a server. Its objectives encompass:

- Agreement on the algorithms governing data confidentiality and integrity for the specific session.
- Exchange of two random values generated independently by the client and the server. These values are essential for subsequent key generation.

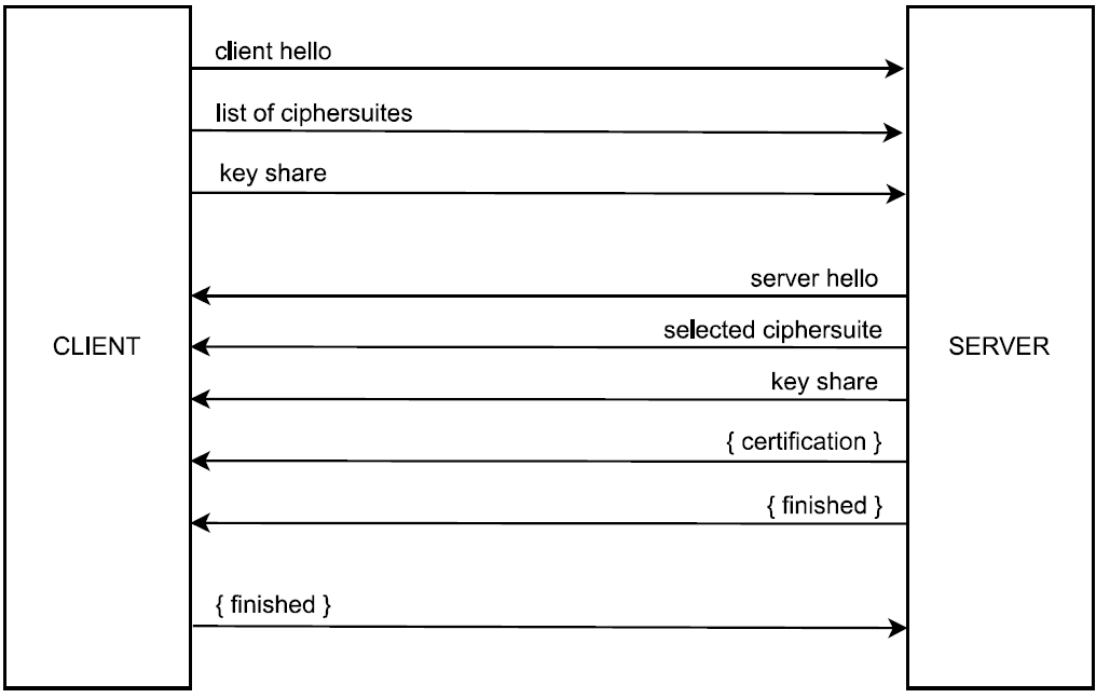

Figure 2.2. TLS 1.3 handshake.

- Establishment of a symmetric key through the employment of DH or ECDH key exchange.
- Negotiation of a session ID (i.e. identifier).
- Exchange of certificates needed for authentication.

The TLS 1.3 handshake is depicted in Figure 2.2: the client initiates the handshake by transmitting a *Client Hello* message that contains the client random value, a list of supported ciphersuites and the client's part of the key share. The server responds with a *Server Hello* message, that contains the server random value, the selected protocol version, the chosen ciphersuite, the server's certificate encrypted using a temporary key unique to the handshake and the Finished message encrypted with the same temporary key. Finally, the client concludes the handshake by transmitting the encryption of its Finished message, also encrypted with the temporary key.

2.4.2 OCSP Stapling

OCSP Stapling is a TLS extension that must be specified in TLS handshake and it works as follows: the TLS server pre-fetches the OCSP responses necessary to validate the entire certificate chain and provides these responses to the client during the handshake, as part of the server's certificate message, so the OCSP responses are stapled together with the certificates. In this way, the certificate and the proof that the certificate is valid at the current moment are provided. A benefit of this extension is privacy, because OCSP server does not know client's identity, but there is the disadvantage of the pre-generation of OCSP responses - they are not fresh and there is the risk of fast attacks.

Typically, OCSP Stapling is an automatic process managed by the server. Nonetheless, the client has the option to explicitly request OCSP Stapling within the Client Hello message during the TLS handshake.

2.5 Hardware Security Module

A Hardware Security Module (HSM) is a cryptographic accelerator for servers, i.e., a co-processor used to perform efficiently cryptographic operations that are computationally

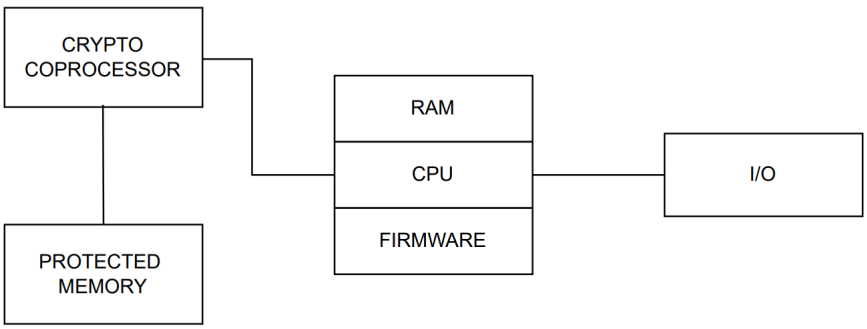

Figure 2.3. Hardware Security Module structure.

intensive. It is extensively used as a secure framework for identification and authentication: it permits to perform encryption, decryption, signature, signature validation, hashing, secure cryptographic-key management and trusted random number generation - used to create encryption keys.

HSMs are available in various form factors, including PCI boards, external devices (e.g., USB or SCSI), or even IP network devices (i.e., netHSM). There are not bandwidth and power limitations so HSMs have high speeds, but in case of netHSMs, they typically store multiple keys for various servers, introducing challenges related to authentication and authorization requests, determining which servers are authorized to access specific keys.

As depicted in Figure 2.3, HSMs incorporate a protected nonvolatile memory where cryptographic key material is stored and private keys are tamper-resistant because in case of tampering, the device is designed to erase its content. The computational efficiency is due to a crypto coprocessor, which is a hardware circuit that implements the required cryptographic functions (e.g., RSA or symmetric algorithms) directly in hardware. The crypto coprocessor is the only one allowed to access the protected memory and use the stored private keys. HSMs hinder side-channel attacks - attacks based on leakage of information after attacker's observation of physical properties of the hardware device - because it is not possible to analyse noise, power consumption, electronic leaks and the time used to generate keys within the chip.

Chapter 3

Post-Quantum Cryptography

The widespread availability of quantum computers poses a significant threat to many cryptographic systems in use today. Encryption, digital signatures, key exchanges, and random number generators will all be vulnerable to quantum attacks. Consequently, there is an urgent imperative to develop and adopt cryptosystems based on Post-Quantum Cryptography, specifically designed to resist quantum attacks.

The current chapter initiates with an introduction to quantum computing, highlighting the quantum algorithms that pose a challenge to classical cryptography. Subsequently, it provides an analysis of the essential transition towards Post-Quantum Cryptography. The chapter concludes with an in-depth examination of the algorithm chosen by the National Institute of Standards and Technology (NIST) in July 2022 for key establishment, namely Kyber, and KEMTLS-PDK protocol, both of which constitute integral components of the Master's Thesis project.

3.1 Quantum Computing

Quantum mechanics is the field of physics that studies matter and energy at small length scales as electrons, atoms and molecules which have consequences also at macroscopic scale. The uncontrolled interaction between a quantum system and its environment produces the process named *quantum decoherence*, wherein the system loses information to the environment, i.e. the system is no longer able to interact coherently with its surroundings.

Quantum computing is the study of computers that use quantum mechanics features in calculations, to solve complex problems whose resolution would be slower on classical computers. The basic unit of quantum computers is the *qubit*, which can store the values 0, 1 or the linear combination of the two states, named *superposition*, so zero and one can be thought as coexisting and being processed at the same time. Superposition is the ability of a quantum system to be contemporary in multiple states. Quantum computers use superposition and quantum entanglement to speedup the resolution of problems. *Quantum entanglement* consists of two systems that are strongly correlated to each other, even if they are separated by millions of light-years of space, so gaining information about one of them provides immediate information about the other system.

Quantum computing requires high control of the quantum behaviour, to avoid the loss of

information caused by quantum decoherence occurrence. One of the most pressing concerns is the impact of quantum computing on cybersecurity. Many widely-used cryptographic schemes rely on mathematical problems that are considered computationally infeasible for classical computers to solve. For instance, the RSA cryptosystem depends on the difficulty of factoring large numbers, a challenge that can be overcome by quantum computers using Shor's algorithm. Other examples of vulnerable cryptosystems are Diffie-Hellman Key Exchange and Elliptic-Curve Diffie-Hellman Key Exchange. Furthermore, the ability of a quantum computer to search through a solution space of 2^N values is roughly $2^{N/2}$ steps, which means that quantum computers may also weaken symmetric key cryptography by means of algorithms like Grover's one.

The following subsections will provide insights into Shor's and Grover's algorithms, shedding light on their implications for classical cryptography.

3.1.1 Shor's algorithm

It is widely believed that factoring large numbers and finding discrete logarithms using classical computers increases with the exponential size of the key. Shor's algorithm [29], developed by the American mathematician Peter Shor in 1994, is a quantum computing algorithm for finding the prime factors of an integer N in polynomial time of $logN$, i.e. in $O((logN)^2(log\ logN)(log\ log\ logN))$. If quantum computers do not succumb to quantum noise or decoherence, Shor's algorithm can break various cryptographic schemes, including RSA, Finite Field DH key exchange, and ECDH key exchange.

It reduces factorisation problem to order-finding problem: for any instance of factorisation problem, it is possible to construct in polynomial time an instance of order-finding problem such that from its solution, it is always possible to derive the solution of factorisation problem in polynomial time. This reduction is made possible by randomization and can be performed on classical computers. So, rather than directly attempting the factorization of an integer N, the quantum algorithm finds the order of an element x in the multiplicative group $(mod\ N)$. The multiplicative group $(mod\ N)$ is the group of integers coprime to N that belong to the set $\{0,1,...,N-1\}$. The order of x is the smallest integer r for which the following congruence holds:

$$x^r \equiv 1\ (mod\ N) \tag{3.1}$$

Firstly, an integer $x\ (mod\ N)$ such that $0 < x < N$ is randomly chosen. The greatest common divisor $gcd(x,\ N)$ is computed, for example using the Euclidean algorithm, and if the result is equal to 0, a new random x must be computed in order to find a non-trivial factor of N.

To determine the period r of the number N that needs to be factored, quantum computing becomes indispensable. Given N, its period is the integer r that satisfies:

$$N^r = e \tag{3.2}$$

where e is the identity element.

A sequence of positive integers k is defined such that:

$$f(k) = x^k\ (mod\ N) \tag{3.3}$$

where x^k $(mod\ N)$ is the reminder of the division of x^k by N. In this sequence, one of the terms is equal to one, and the following terms are periodic:

$$f(k+t) = f(k) \tag{3.4}$$

Here, t is the variable used to indicate the period. The integer r is the smallest positive integer for which:

$$f(r) \equiv x^r \equiv 1\ (mod\ N) \tag{3.5}$$

The algorithm proceeds by choosing a suitable Q which is a power of 2, i.e., $Q = 2^L$, such that $N^2 \leq Q \leq 2N^2$. Then, a random integer p, such that $gcd(p,\ N) = 1$, is chosen. Two quantum registers are created, one for the input and the other for the output, and they are entangled, ensuring that if one collapses, so does the other. The Fourier Transform is applied to the input register, a measurement is made on the first register to obtain y and the period r is obtained via continued fractions for $y/2^L$.

The result of the quantum step specifies if it is necessary to test another positive random integer x or if the factor of N has been found. If r is odd or the common factor is trivial, it is necessary to repeat the first steps choosing a new positive integer x. On the other hand, if r is even, the Euclidean algorithm is used to check if $x^{r/2} + 1$ is equal to 0 $mod\ N$ because from the formula 3.5 it is known that $x^r \equiv\ 1\ (mod\ N)$, so

$$\left(x^{r/2}\right)^2 - 1 \equiv\ x^r - 1 \equiv\ 1 - 1\ \equiv\ 0\ (mod\ N) \tag{3.6}$$

and formula 3.7 follows:

$$(x^{r/2} + 1)(x^{r/2} - 1) \equiv\ 0\ (mod\ N) \tag{3.7}$$

If $x^{r/2} + 1 \neq 0\ mod\ N$, it exists a positive integer $x^{r/2} + 1$ that multiplied to the co-prime $x^{r/2} - 1$ produces 0 $mod\ N$, so $x^{r/2} - 1$ is co-prime with N.
If $x^{r/2} + 1 = 0\ mod\ N$, the first step of the algorithm must be recomputed.
If $x^{r/2} + 1 \neq 0\ mod\ N$, the solution of the algorithm is $gcd(x^{r/2} - 1,\ N)$.

3.1.2 Grover's algorithm

It is generally believed that although quantum computers bring fatal threats to asymmetric cryptography such as RSA, they do not fatally threat symmetric cryptography such as AES. Unfortunately, this is not true. Grover's algorithm [9], developed in 1996 by Indian-American computer scientist Lov Kumar Grover, is a quantum computing algorithm that has a significant impact on symmetric cryptography. Grover's algorithm reduces the computational complexity of certain symmetric key problems from 2^N to $2^{N/2}$, making it a potent tool against symmetric cryptography that uses keys shorter than 256 bits. It applies to tasks of finding a preimage, i.e., the set of all elements of the domain mapped into a given subset of the co-domain. Knowing the *(plaintext, ciphertext)* pairs, it can find the symmetric key of a symmetric algorithm such as AES in the square root of the time that a normal exhaustive search would take. E.g., finding an AES-128 secret key in about 2^{64} steps instead of the 2^{128} steps required for a classical computer.

Grover's algorithm solves the task of function inversion: given a function $y = f(x)$, which can be evaluated on quantum computers, it can calculate x given y. This algorithm offers significant asymptotic speed-ups for a range of brute-force attacks on symmetric-key cryptography, including collision attacks and preimage attacks.

The algorithm operates entirely on a quantum computer, utilizing qubits, superposition and a quantum oracle that accesses the function f. Grover's algorithm is useful in many other areas than cryptography (e.g. creation of medicines by speeding up complex problems that involve how proteins fold) but its main problems are the overhead due to quantum operations because the achieved quadratic speedup is too modest to overcome the large overhead of quantum computers. Moreover, the computations have not been paralleled in quantum computers, meaning that the 2^{64} steps required to find an AES-128 secret key must be performed sequentially, which demands a considerable amount of time.

3.2 Transition to Post-Quantum Cryptography

Post-quantum cryptography has been designed to be secure against the quantum threat. However, as outlined in [15], transitioning to quantum-safe cryptography is an arduous and delicate process because it is necessary to establish standards, migrate legacy systems, develop and deploy hardware and software solutions. Any organization that wants to complete the transition to quantum-safe cryptography for a particular cyber-system must consider three parameters:

1. $T_{shelf-life}$: number of years in which the information must be protected by the cyber-system. This is a business decision that the company must take.

2. $T_{migration}$: number of years required to properly migrate the system to a quantum-safe solution.

3. T_{threat}: number of years before there is the possibility to break the quantum-vulnerable system. It is difficult to assume.

If the sum of the migration time and of the shelf-life time is greater than the threat timeline, the organization may not be able to protect its assets for the entire shelf-life time against quantum threats. So the difference between T_{threat} and $T_{shelf-life}$ is the maximum $T_{migration}$ possible, i.e., the maximum time in which organizations must organize the transition.

3.3 CRYSTALS-Kyber

In July 2022, NIST announced which post-quantum algorithms they will standardize by 2024 [16]. CRYSTALS-Kyber algorithm has been chosen for public-key encryption and key establishment, while CRYSTALS-Dilithium, FALCON and SPHINCS+ have been chosen for digital signature. This section provides an in-depth exploration of Kyber in the context of key establishment.

Kyber [1] belongs to lattice-based family of post-quantum algorithms and is designed to be a post-quantum secure Key Encapsulation Mechanism (refer to section 2.1), based

on solving the Learning-With-Errors-and-Rounding problem in module lattices (MLWER). Kyber employs a Public Key Encryption scheme in which the plaintext cannot be specified because it is generated as a random key during the encryption process, and it is interactive because encapsulation can be performed only after having received the input from the other party. The term "CRYSTALS" stands for "CRYptographic SuiTe for Algebraic Lattices", and represents the package submitted to NIST for post-quantum standardisation.

Kyber can be effectively integrated into TLS protocol (refer to section 2.4), operating in hybrid mode, coexisting with pre-quantum cryptographic methods.

3.3.1 Key Encapsulation Mechanism functions

Consider the message m that is 256-bits long, the corresponding ciphertext ct, the 256-bits long shared secret ss result of the key agreement, the public key pk and the private key sk, two random oracles H and G, respectively SHA256 and SHA512 and the KDF SHAKE-256. The size of the ciphertext depends on the specific Kyber version in use. The following pseudocodes of Kyber functions follow the IETF "Kyber Post-Quantum KEM" draft [28].

The Pseudocode 1 is related to KeyGen function, used to generate a Kyber keypair. Starting from the first 32 bytes of a seed of 512 bits named *seed*, the function *InnerKeyGen* deterministically produces a public key *pk* - which will be provided as the output of *KeyGen* function - and a private key *cpaSk*. The private key output of the algorithm is *sk* and it is the concatenation of the private key *cpaSk*, the public key, the SHA3-256 digest of the public key and the last 32 bytes of the seed.

Algorithm 1 KeyGen

```
seed: 64 bytes
z = seed[32:]
(pk, cpaSk) = InnerKeyGen(seed: seed[:32])
h = H(pk)
sk = cpaSk || pk || h || z
return (pk, sk)
```

In Pseudocode 2, *Encaps* algorithm is described. It receives in input the public key *pk* generated with KeyGen function. Firstly, the seed of 256 bits is randomly generated, then

Algorithm 2 Encaps

```
seed: 32 random bytes
m = H(seed)
KBar = G(m || H(pk))[:32]
cpaSeed = G(m || H(pk))[32:]
ct= InnerEnc(message: m, key: pk, coins: seed)
ss = KDF(KBar || H(ct))
return (ct, ss)
```

the message m to encrypt is obtained as the SHA3-256 digest of the seed. The ciphertext ct is computed with the function named *InnerEnc*, which takes the seed and deterministically

encrypts the message m using the key pk. Then, the shared secret key ss is the output of the KDF that receives the concatenation of $KBar$ and the SHA3-256 digest of the ciphertext ct. $KBar$ is the first 256 bits of the SHA3-512 digest of the concatenation of m with the SHA3-256 digest of the public key pk.

The *Decaps* function (pseudocode 3) receives in input the private key sk and the ciphertext ct, and produces the shared secret key ss. The Kyber private key sk has been computed with *KeyGen*, so it is the concatenation of $cpaSk$, pk, h and z. In order to perform decapsulation, it is necessary to extract these values from sk, and this is made possible by their fixed lengths, which depend on the specific Kyber version. On this purpose, the parameters n and k, defined in Table 3.1, are used. Once $cpaSk$, pk, h and z variables are set, the function *InnerDec* decrypts the received cyphertext with the private

Algorithm 3 Decaps

```
cpaSk = sk[: 12*k*n/8]
pk = sk[12*k*n/8 : 24*k*n/8+32]
h = sk[24*k*n/8+32 : 24*k*n/8+64]
z = sk[24*k*n/8+64 :]
m = InnerDec(ciphertext: ct, key: cpaSk)
KBar = G(m || h)[:32]
cpaSeed = G(m || h)[32:]
ct2 = InnerEnc(message: m, key: pk, coins: cpaSeed)
ss = { KDF(KBar||H(ct))   if ct == ct2
     { KDF(z||H(ct))      otherwise
return ss
```

key $cpaSk$, producing the message m. Then, $KBar$ and $cpaSeed$ and the ciphertext $ct2$ are computed in the same way as the other peer did through *Encaps* function. If the received ciphertext is equal to the computed one, also the message in clear m is equal to the one owned by the other peer, and it follows that also $KBar$ is the same, so the current peer can compute the shared secret key ss with the same formula used by the other peer, being sure that the key will be the same. Otherwise, if the ciphertexts differ, the KEM failed and the computed ss will be different to the one owned by the other peer.

3.3.2 Key exchange

Kyber key exchange process is depicted in Figure 3.1:

1. In the initial step, the first peer $P1$ generates a fresh key-pair and transmits the resulting public key pk to the second peer $P2$.

2. Upon receipt of the public key, $P2$ encapsulates pk to derive the ciphertext and the shared secret key.

3. Subsequently, $P2$ sends the ciphertext to $P1$, who, in order to extract the shared secret key, decapsulates the ciphertext, effectively decrypting it using the private key generated during the first step.

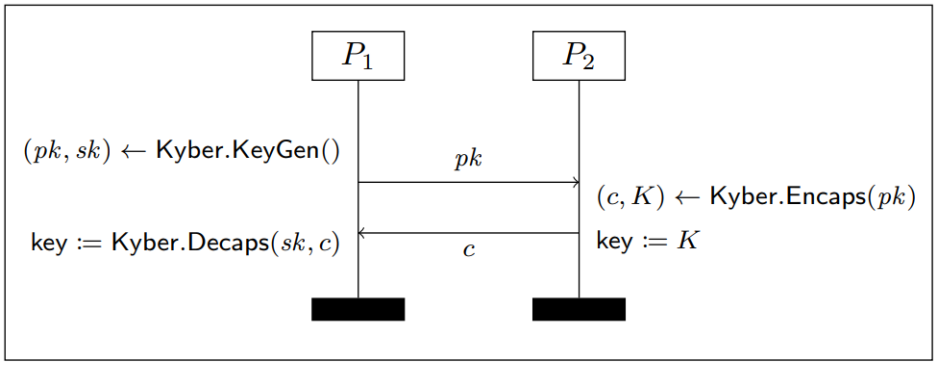

Figure 3.1. KYBER key exchange. Source: [1]

4. At this point, both peers possess the shared secret key, enabling them to securely exchange encrypted data.

3.3.3 Authenticated key exchange

Kyber key exchange scheme shown in Figure 3.1 is unauthenticated, so it protects against passive adversaries because they cannot deduce anything about the traffic they see, but a MITM (Man-In-The-Middle) attack is still possible. As depicted in Figure 3.2, when the first peer transmits the public key pk, an attacker is able to intercept and block it in reaching the second peer.

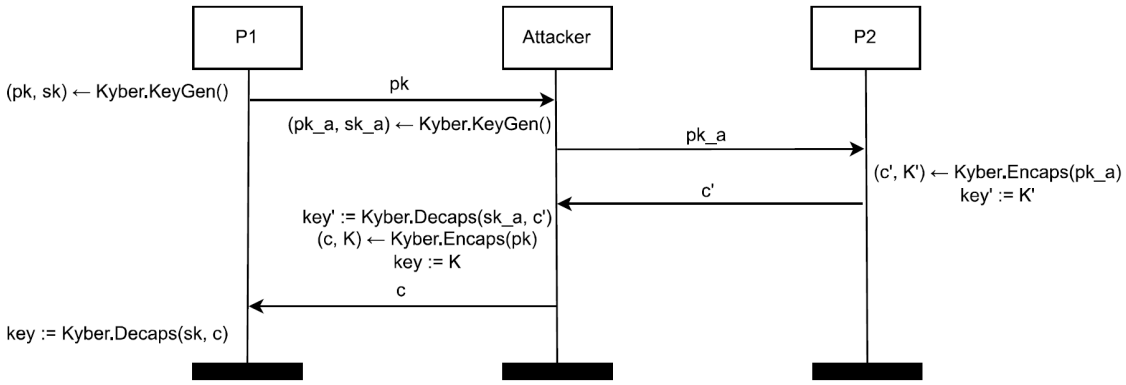

Figure 3.2. Kyber key exchange - MITM attack.

The attacker may generate and then transmit a public key pk_a to P2. P2 uses the received public key to compute the ciphertext and the shared secret key via *Encaps* function and sends back the ciphertext c' to P1. Again, the attacker intercepts the ciphertext, blocks

it in reaching the first peer and computes the ciphertext and the shared secret key using the public key pk intercepted from P1. Then, the attacker sends the computed ciphertext c to P1, P1 decapsulates the received ciphertext with the computed secret key and obtains the key. Obviously, the keys known by P1 and P2 are different and the attacker in the middle will be able to read the traffic during the transmission.

A solution to protect against MITM attacks is the *authenticated key agreement*, shown in Figure 3.3, where both peers have their long term keypair used for authentication. They are static keys employed for a relatively long period of time and in many instances of the cryptographic key-establishment scheme. The static keys are meant to be in long term certificates, so they are not exchanged out-of-band (OOB) but they are present within Public Key Certificates (PKCs). Thanks to static keys, MITM attacks are not possible because

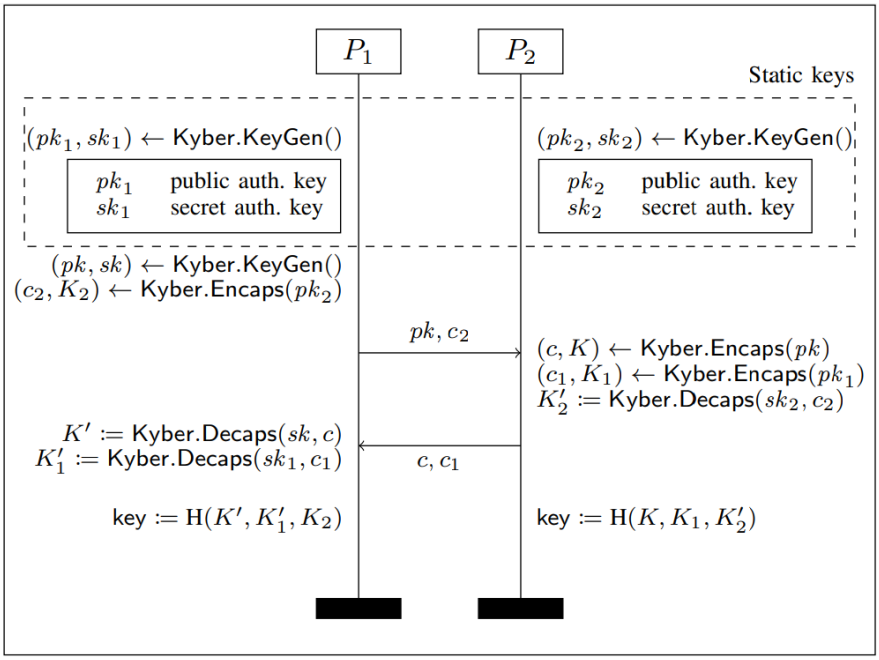

Figure 3.3. Kyber authenticated key exchange. Source: [1]

both shared secret keys K_1 and K_2 - respectively computed performing the encapsulation using the static public key of P1 and P2 - would be different if an attacker were in the middle. Moreover, the hashes would be different and the symmetric encryption/decryption would not lead to the right result. All the keys $(K_1, K_2, K'_1, K'_2, K, K')$ are necessary to guarantee that the two peers are really who they claim to be and to guarantee integrity of the exchanged data (pk, c_2, c, c_1) because only having K equal to K', K_1 equal to K'_1 and K_2 equal to K'_2, authentication and integrity are guaranteed and the two peers $P1$ and $P2$ will be able to encrypt and decrypt the exchanged application data without anyone else being able to read their communications.

3.3.4 Parameters

In Table 3.1, there are Kyber parameters. Kyber512 has a NIST security level 1 and a classical equivalent resistance of AES-128. The integer n indicates the number of bits of entropy of the keys to encapsulate, i.e. the length of the plaintext provided in input to the encryption function. The integer k is a value selected to fix the lattice dimension as a multiple of n. The main mechanism in Kyber to scale security and, as a consequence, efficiency, to different levels consists of changing k value. In Kyber variant, the number that follows the word "Kyber" is the result of the multiplication between n and k. The secret keys, from Kyber512 to Kyber768 and from Kyber768 to Kyber1024 differ by 768 bytes and the corresponding public keys differ by 384 bytes.

Table 3.1. Kyber parameters

	secret key (bytes)	**public key (bytes)**	**ciphertext (bytes)**	**n (bits)**	**k**
Kyber512	1632	800	768	256	2
Kyber768	2400	1184	1088	256	3
Kyber1024	3168	1568	1568	256	4

As many post-quantum KEMs, Kyber has the problem of the size of keyshares, that compared to classical KEMs is big. The keyshare is the public key sent by the first peer to the second one during the key exchange, in order to generate the shared secret key. For example, in Kyber512, the data over the wire is the ciphertext and the public key, i.e., 1568 bytes, while in X25519 it is twice the Public key, i.e., 64 bytes. In the hybrid versions of Kyber, both public keys of Kyber and of X25519 are taken. For example, in the hybrid Kyber512, the keyshare is 832-bytes long because it takes the public key of Kyber512 (i.e., 800 bytes) and the public key of X25519 (i.e., 32 bytes). The length of the keyshare is a serious problem because a *ClientHello* message will just barely fit in a network packet so some TLS implementations may crash on the larger keyshare messages which contain bigger post-quantum keys. Another serious problem is fragmentation because fragmenting a *ClientHello* of TLS handshake into two initial packets would lead to performance degradation because putting packets together is not free, it requires to keep track of the partial messages around.

3.4 KEMTLS-PDK

Web applications secured with the TLS 1.3 protocol (refer to section 2.4) face quantum threats. The vulnerabilities arise from the protocol's mechanisms for confidentiality and integrity, which involve Authenticated Encryption with Associated Data (e.g., ChaCha20_Poly1305 or AES128GCM). As discussed in section 3.1, quantum attacks based on Grover's algorithm become feasible when the symmetric key is shorter than 256 bits. The key exchange in TLS 1.3 relies on the X25519 Elliptic Curve Diffie-Hellman protocol, which is based on the discrete logarithm problem for elliptic curves. Unfortunately, this choice is susceptible to quantum attacks based on Shor's algorithm. Moreover, the digital signature

algorithms RSA or ECDSA, integral to TLS certificates for website authentication, are also vulnerable to quantum attacks based on Shor's algorithm. These foundational aspects of TLS 1.3 make it susceptible to compromise in a quantum computing environment.

The KEMTLS-PDK protocol [26], an evolution of the KEMTLS protocol [27], represents a post-quantum solution with pre-distributed keys. It adopts a signature-free approach, employing a post-quantum Key Encapsulation Mechanism (KEM)-based key exchange and authentication. The primary objectives of KEMTLS-PDK are:

- Provide quantum-resistant TLS, ensuring confidentiality and authentication even in the presence of quantum threats.
- Optimize computational costs, reducing the number of round trips, and minimizing bandwidth requirements.

Implemented by modifying *Rustls* TLS library [24], KEMTLS-PDK enhances the round-trip efficiency compared to its predecessor, KEMTLS. In the protocol, both peers possess a static KEM keypair, and the client has prior knowledge of the server's static KEM public key. The long-term KEM keys used by the protocol are authenticated by CAs digital signatures (refer to section 2.3.1).

Noteworthy aspects of KEMTLS-PDK include:

- *Implicit Client Authentication*: client authentication is implicit, meaning that the computed shared secret is only known by the intended client.
- *Explicit Server Authentication*: server authentication is explicit, ensuring the participation of the intended server. This authentication occurs one round trip later than the first application data sent by the client.
- *Downgrade Resilience*: KEMTLS-PDK is designed to resist downgrade attacks, where adversaries attempt to force the use of a vulnerable protocol version by sending fake server responses. Full downgrade resilience is achieved at the end of the handshake, providing the client with assurance regarding the selected algorithms, as explicit server authentication occurs before the client transmits application data.

3.4.1 Handshake with proactive authentication

In Figure 3.4, KEMTLS-PDK handshake with key derivation schedule and proactive client authentication is presented. Proactive client authentication is possible when the client already knows that the server requires mutual authentication, so the client sends its certificate in advance. It is important to note that the provided Figure 3.4 differs from the one presented in "More efficient post-quantum KEMTLS with pre-distributed public keys" Paper [26] because during the study of the protocol, an error was identified in the handshake and communicated to the researchers working on KEMTLS-PDK. They provided the corrected version, incorporated into the current Master's Thesis and taken from Thom Wiggers' PhD Thesis [41].

The protocol unfolds through multiple stages to establish a session, with each stage establishing a shared secret indistinguishable from a random key. The keys generated to encrypt subsequent parts of the handshake serve internal purposes, and the final session

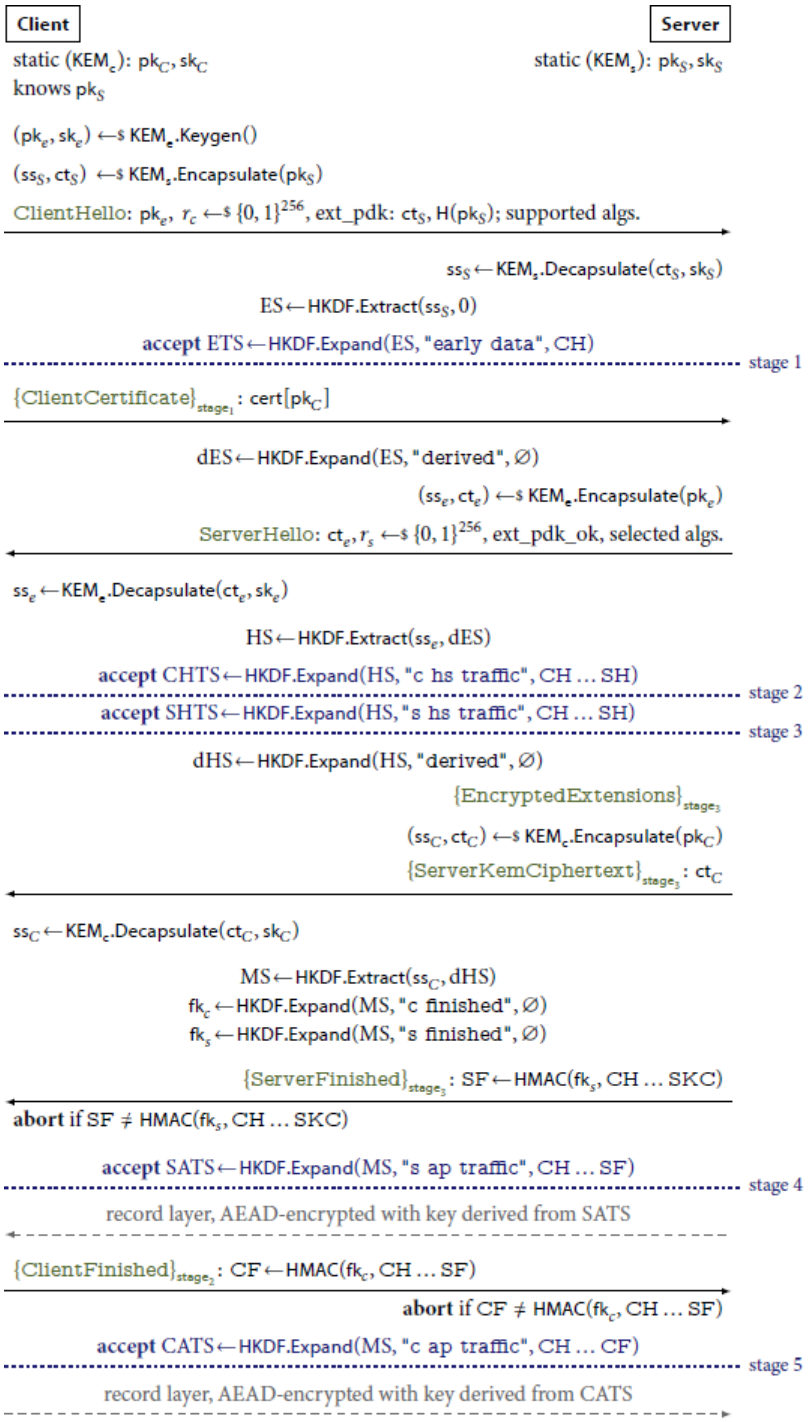

Figure 3.4. KEMTLS-PDK handshake. Source: [41]

key is employed for authenticated encryption of application data. Initiating the handshake after the exchange of TCP SYN and TCP SYN-ACK, the client generates the ephemeral keypair using the secure KEM with ephemeral key exchange KEM_e to provide *forward secrecy*. Subsequently, the client encapsulates the server's long-term public key pk_s using KEM_s, a secure KEM with implicit authentication. The algorithms utilized by the two KEMs can be the same, and KEMs can be hybrid, combining post-quantum cryptography and classical cryptography.

The first client-to-server handshake message is the *ClientHello*. It contains the ephemeral public key pk_e, the client's nonce r_c used for freshness, the ciphertext ct_s generated through the encapsulation performed in the previous step, and the supported algorithms for key exchange and authenticated encryption.

Upon receiving the ciphertext, the server uses its owned long-term KEM_s private key sk_s to perform decapsulation, obtaining the first shared secret ss_s. Both peers execute the *Extract* and *Expand* HKDF functions. The former, a randomness extractor, takes a salt (used for the current secret state) and the input key material (used for new shared secrets), while the latter, a pseudorandom function with variable length, receives in input the secret key, a label, the string of the hash of the transcript of messages, and the desired length of the output key (which is omitted in Figure 3.4). A transcript is the concatenation of consequent TLS messages and the $\emptyset$ symbol is used to indicate an empty value. The hash function used by the HKDF is collision-resistant, ensuring pseudorandomness in both salt and input keying material arguments.

The computed shared secret of the first stage is ETS, enabling implicit authentication of the server, as its computation relies on the server's long-term keypair, allowing only the legitimate server to decrypt messages encrypted with ETS.

The client sends its certificate, which contains its long-term public key, encrypted using ETS. Then, both peers compute dES with the HKDF *Expand* function; the server encapsulates the received ephemeral public key pk_e and transmits the *ServerHello* message, that contains the computed ephemeral ciphertext ct_e, server's nonce r_s and the algorithms selected from the client's proposal.

The client computes the ephemeral shared secret ss_e by decapsulating the received ct_e with the ephemeral private key sk_e computed in the first step. Following these procedures, both peers can compute the shared secrets of stages 2 and 3, namely the client handshake traffic secret $CHTS$ and the server handshake traffic secret $SHTS$, along with the derived handshake secret dHS.

The final part of the handshake depends on client's long-term keypair. The server transmits a message containing protocol extensions encrypted with $SHTS$ and the encryption, performed with $SHTS$, of the ciphertext computed with the encapsulation of client's long-term public key.

The client computes the shared secret ss_c decapsulating the received ct_c with its long-term private key. These steps achieve mutual authentication in a single round trip, with implicit authentication for both peers.

Both peers compute the master secret MS, along with the finished keys fk_c and fk_s, which are employed to authenticate the handshake. The server sends the *ServerFinished*, which contains the Hash-based Message Authentication Code (HMAC) used to authenticate the handshake transcript, computed with fk_s and encrypted with $SHTS$. The client verifies the ServerFinished decrypting it, computing the same HMAC and comparing the

received value with the computed one. If the computed HMAC differs from the received one, the client aborts the handshake. Otherwise, the client is assured that both peers have the same transcript (which includes negotiation messages), guaranteeing that KEMs for ephemeral key exchange, public key authentication, authenticated encryption and hash function algorithms are the wanted ones. The same procedure is followed for *ClientFinished* message, and if verification is successful, the session key is computed with HKDF *Expand* function, utilizing MS key, "c ap traffic" label and CH..CF transcript.

Replay attacks are possible for Stage 1 keys because an attacker could replay the same *ClientHello* message several times, but the subsequent stages are replay-protected.

3.4.2 Comparison with TLS 1.3

When comparing TLS 1.3 with KEMTLS and KEMTLS-PDK, several considerations arise:

- *Authentication, Integrity, and Confidentiality*: both KEMTLS and KEMTLS-PDK, like TLS 1.3, provide robust authentication, integrity, and confidentiality.
- *Bandwidth Optimization*: KEMTLS and KEMTLS-PDK stand out by requiring less than half of the bandwidth compared to TLS 1.3. This reduction results from minimizing the amount of data transmitted during the handshake process.
- *Trusted Code Base*: KEMTLS and KEMTLS-PDK contribute to a reduced trusted code base, enhancing overall security.
- *Speed* : the number of server CPU cycles spent on asymmetric cryptography is significantly reduced, approaching a remarkable 90% reduction [27].
- *Offline Deniability*: both KEMTLS and KEMTLS-PDK have offline deniability, making it impossible to distinguish between a genuine and a forged transcript. This property means that the protocols lack the *non-repudiation* security property present in TLS 1.3.
- *Transmission of Encrypted Data*: while TLS 1.3 allows the transmission of encrypted and authenticated data from the server to the client starting with the first response message of the handshake, KEMTLS and KEMTLS-PDK initiate the transmission of the first application data in the client-to-server message flow. This design choice reduces the time taken for the handshake before the client can send application data.

Chapter 4

Cryptography in Space Systems

In order to establish a comprehensive foundation for the current Master's thesis, this chapter provides a contextual overview of the Space domain. The discussion begins with a broad exploration of the Space domain, laying the groundwork for a deeper understanding of the challenges involved in securing space-based systems. Following the contextual introduction, the state-of-the-art in cryptography specifically tailored for space systems is analysed, highlighting the current cryptographic techniques and technologies employed in space systems. Building upon the examination of the current state-of-the-art, the cryptographic goals for the future are outlined. Understanding these objectives is crucial for placing the thesis work within the broader framework of European initiatives and aspirations in the realm of Space security. To provide a practical example of ongoing efforts to address cryptographic challenges in the Space domain, the last section of the chapter offers a high-level overview of an active ESA project, focused on securing ground-to-space communications.

4.1 Cybersecurity for Space Systems

The Space domain encompasses the essential elements required for the functioning of space systems and the delivery of space-based services. A *space system* is composed by vehicles and infrastructure designed to operate within the space environment. A *satellite*, as a fundamental operational asset, is composed of a platform, known as the bus, and one or more payloads. The *bus* represents the physical infrastructure of the satellite, incorporating mechanisms and subsystems for various functions, including attitude determination and control, power systems, propulsion, thermal control, telemetry, tracking and command communications, and processing. On the other hand, the *payload* comprises mission-specific components that are distinct from the overall satellite operations.

4.1.1 High-Level Architecture of a Space System

Figure 4.1 illustrates the primary components of the high-level architecture of a space system. The *Space segment* encompasses space infrastructure not hosted on Earth [8], housing satellites, probes (small spacecraft without crew), and space stations, such as the International Space Station (ISS). Conversely, the *Ground segment* constitutes the space architecture situated on Earth, encompassing ground stations, operations centers (e.g., the European Space Operations Centre), and the ground network. While not explicitly illustrated in Figure 4.1, the *Link segment* plays a crucial role in the space system architecture.

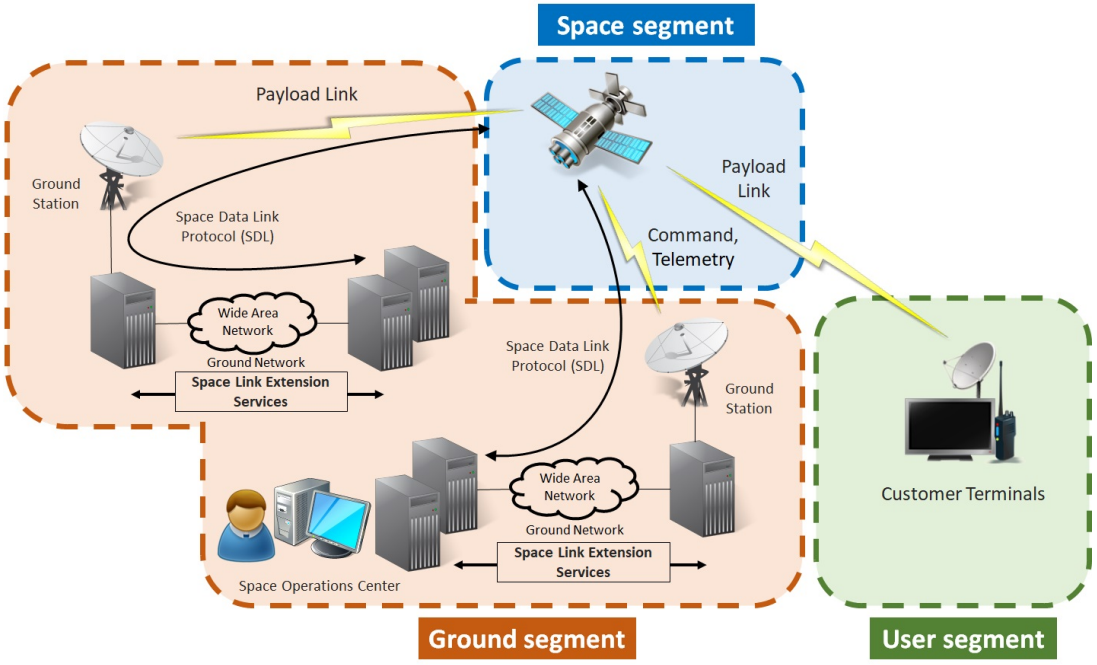

Figure 4.1. High-level architecture of a space system. Source: [22]

This segment encompasses communication links between the ground and space segments. These links can be categorized as follows:

- *Uplink*: Communications originating from the ground and directed towards satellites.
- *Downlink*: Communications originating from satellites and directed towards the ground.
- *Crosslink*: Communications occurring between satellites.

These communication links operate using radio frequency or free-space optical networks.

The *User segment* constitutes all interfaces and infrastructures that expose space system services to consumers.

4.1.2 Transformative Applications of Space Systems

Satellites, initially designed to support national security and telecommunications, have undergone a profound evolution. In the contemporary landscape, the utilization of space-based systems extends far beyond their traditional roles. The increasing affordability of deploying satellites for enterprise use cases has sparked a technological revolution. Various stakeholders, including governments, international organizations, technical authorities, and service consumers/providers, are steering the Space domain towards heightened strategic significance in defense and security contexts [40]. Space systems now play pivotal roles in border and maritime surveillance, humanitarian operations, telecommunication networks, verification of military agreements, and support of defense operations [20]. As the reliance on space systems grows, so does the concern for their cybersecurity. The absence of industry-wide standards poses a significant challenge, given that satellite systems operate in one of the most hostile environments. Threats range from natural events such as disasters, space debris, space weather, and atmospheric disturbances to cyber-attacks conducted by humans.

4.1.3 Cyber-attacks against Space Systems

As many civilian space systems lack adequate protection, cyber-attacks targeting them often do not require high levels of expertise. When a space system becomes a target, the attack may focus on the spacecraft itself, its ground infrastructure, the user segment, or the data links between spacecraft and ground segment. Potential targets encompass the hardware and embedded software of the satellite, including onboard software, transmitted data, and the ground network.

Various types of cyber-attacks against satellites have been identified [12]:

- *Side Channel Attack*: a passive attack that does not physically damage the spacecraft. Instead, it observes electromagnetic radiation from the device, attempting to deduce the secret key.
- *Jamming Attack*: an electronic attack challenging to execute but easy to trace. The attacker sends de-authentication signals in an effort to disrupt communication between the transmitter and receiver. Targets may include sensor data or guidance control.
- *Spoofing Attack*: a popular and easy attack where the attacker intercepts data packets, seeking to decipher their content. The targets may encompass sensor data or guidance control.
- *Distributed Denial Of Service (DDoS) Attack*: in this attack, a large volume of data is sent to the victim, overloading it and compromising its operational performance. Targets may include guidance or sensors.
- *Hijacking Attack*: the attacker infiltrates the satellite ground station to gain full control over the satellite.
- *Replay Attack*: the attacker intercepts valid transmitted data packets containing authentication or access control information, subsequently repeating or delaying their transmission to gain unauthorized access or generate unauthorized effects.

The Mission Control Software (MCS) is typically the primary target of cyber-attacks, as gaining control over spacecrafts is a key objective. The repercussions of a successful cyber-attack against a space system can range from reversible to irreversible, causing damage, disruption, or the destruction of parts or of the entire system. Confidential and sensitive data may be exposed, the attacker might gain physical control of the satellite through remote intrusion of ground stations, and the satellite's trajectory could be altered, leading to collisions with other satellites. Additionally, malicious code injection and corruption of sensor systems, data breaches, and in extreme cases, loss of human lives in crewed missions, are potential consequences. Notably, spoofing is a highly effective attack, being undetectable and based on weak encryption. Therefore, the implementation of stronger encryption algorithms is imperative.

4.1.4 Countermeasures

Space systems, including commercial entities, are deemed critical infrastructures susceptible to vulnerabilities that could lead to the exposure of sensitive data. To mitigate the risks posed by vulnerabilities, implementing robust security measures is of utmost importance. Various cybersecurity measures for space systems include:

- *Authentication and Access Control Mechanisms*: implementing mechanisms to authenticate actors and data and control access helps safeguard against unauthorized intrusions.
- *Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs)*: these systems play a crucial role in identifying and preventing unauthorized access or malicious activities within the space systems.
- *Secure Communication Protocols and Encryption Techniques*: fundamental for securing data, technologies, and human lives, especially in the case of crewed space systems.
- *Encryption and Protection Systems*: implementing strong encryption and protection systems is essential to provide data confidentiality and integrity.
- *Hardened and Continuously Monitored Infrastructures*: infrastructures should be fortified and subject to continuous monitoring to detect and counteract potential threats.
- *Onboard Software and On-Ground Software Continuously Updated*: regular updates to both onboard and on-ground software are critical to patch vulnerabilities and enhance overall system security.

Ensuring the security of the Ground segment is paramount as it serves as the sole interface between space assets and Earth. The Space segment, being physically challenging to repair if compromised, must be protected. Additionally, securing the Link segment is crucial to guarantee confidentiality, integrity, and availability.

Europe is urgently prioritizing the enhancement of space asset and data security to prevent the weaponization and geopoliticization of satellite systems. This focus arises from the understanding that security on Earth is intertwined with security in Space, particularly with satellite communications emerging as a valid alternative to terrestrial Internet data

transmission. While military satellites already heavily employ encryption, authentication mechanisms, and integrity checks to resist cyber-attacks, civilian satellites, unfortunately, lag behind in proper security measures. In many cases, they are inadequately secured or lack security altogether. Consequently, the integration of cybersecurity practices with technological progress is indispensable.

4.2 State-of-the-Art in Cryptography for Space Systems

As emphasized in section 4.1.3, the use of weak encryption techniques poses a significant risk, facilitating data interception and decryption for potential attackers. In addition, insecure protocols leave systems vulnerable to data modification and theft.

Space agencies, including the European Space Agency, depend on cryptographic research and standardization from the civilian domain for implementing cryptographic algorithms in future civilian space missions. Existing civilian space systems with cryptography heavily rely on symmetric cryptographic primitives and hashes. For space missions requiring data, voice, and video confidentiality, integrity, and authenticated encryption, cryptographic measures are crucial in the following areas:

- *Telecommand (forward space link)*: used for communications from the ground to the spacecraft.
- *Telemetry (return space link)*: used for communications from the spacecraft to the ground segment.
- Across the *ground data network*.

Depending on the specific system, cryptography can be applied at various layers, including the Application layer (e.g., IETF protocol TLS, see section 2.4), Network layer (e.g., IPSec), Data Link layer (e.g., CCSDS Space Data Link Security, i.e., SDLS protocol), and even the physical layer (e.g., bulk encryption, i.e., combined transmissions from a multiplexer encrypted all together). Cryptography can be applied at one or many of these layers.

However, many systems lack cryptography due to challenges in updating, long lifecycles, and difficulty keeping pace with technological advancements. This presents an urgent problem as the absence of data confidentiality and security disruptions in terms of data integrity, authentication, and confidentiality can have severe consequences during transmission, leading to data manipulation, corruption, unauthorized access, disclosure of sensitive data, mission failures, and even loss of human lives in crewed missions.

The Consultative Committee for Space Data Systems (CCSDS) published Blue Book [31], offering recommended standards for security protocols at physical layer (Proximity-1 Space Link Protocol) and data link layer (SDLS protocol). Authenticated encryption algorithms are recommended for space systems by CCSDS and ESA due to their ability to provide confidentiality, integrity, authentication, and high-speed communications.

According to CCSDS standard for Cryptographic Algorithms [32]:

- Existing CCSDS implementations that have cryptography use AES-128 algorithm.

- Future implementations are expected to adopt AES-256, with Galois Counter Mode (GCM) if additional data integrity is required.

Most space systems favor symmetric encryption for its computational efficiency and memory usage. Key sharing through secure channels is feasible because space systems are often isolated.

For authentication, CCSDS recommends solutions such as HMAC with SHA-256 for hash message-based authentication, Cipher-based Message Authentication Code (CMAC) with AES-128/192/256 for cipher-based authentication, Galois Message Authentication Code (GMAC) with AES-128/192/256 for cipher-based authentication requiring authenticated encryption only for authentication, and RSA-2048/4096, DSA, ECDSA for digital signature-based authentication.

CCSDS standards also foresee the use of asymmetric cryptographic primitives to enhance flexibility for federated operations, adopt digital signature based authentication, improve scalability (essential in new space, i.e. private space industry, and for projects with large constellations, which are networks of several satellites similar in functions and interconnected to each other working for the same mission), and foster interoperability with terrestrial systems.

Since data encryption and authentication are necessary, key establishment is crucial. For ground-to-satellite communications, keys are often pre-deployed, with a good practice of refreshing keys for each session. An alternative approach is key exchange offering forward secrecy. For satellite-to-satellite communications, key establishment can occur indirectly via ground stations, but a preferred method involves using inter-satellite links for key exchange, providing better scalability.

4.3 Goals in Cryptography for Space Systems

As detailed in section 3.1, Shor's algorithm poses a threat to nearly all classical public key cryptography, including RSA and elliptic curve cryptography. Additionally, Grover's algorithm can compromise symmetric cryptography which utilises keys shorter than 256 bits, such as AES-128 keys. This threat extends not only to current space systems but also to future ones, as they are anticipated to use algorithms like RSA for key agreement, encryption, and signatures that lack quantum resistance (refer to section 4.2). With the expected advent of widely available quantum computers within the next 10-15 years, it is crucial to recognize them as a serious threat and initiate the migration of systems, considering the enduring nature of space systems. Given all the needs identified in this chapter, it is necessary to start integrating Post Quantum Cryptography in space systems, adopting hybrid solutions that incorporate both classical algorithms to ensure resilience against conventional computers and PQC algorithms which will provide quantum resilience in the long term. The European Space Agency (ESA) has set the objective of securing all the markets depicted in Figure 4.2, aiming to reduce vulnerabilities and enhance cyber-resilience. An ESA goal is to actively support the development of secure satellite communication products and systems. This support extends to providing best practices, facilities, and a robust cyber framework. ESA acknowledges the strategic value of cybersecurity, leveraging it as a fundamental business asset for the space industry.

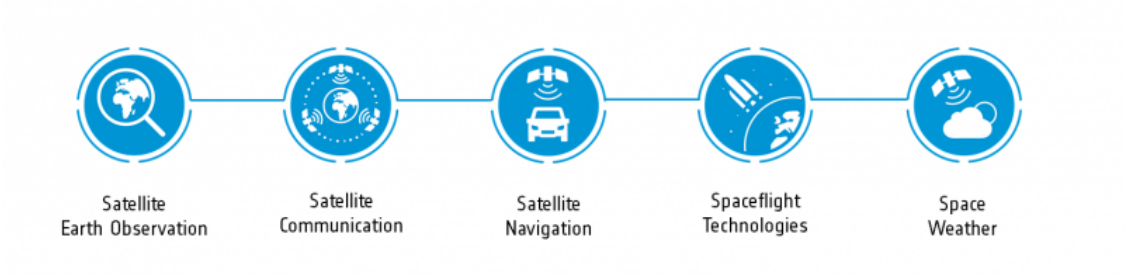

Figure 4.2. ESA space markets that require cybersecurity. Source: [5]

4.4 Overview of an active ESA Project: Hardware Security Module As A Service (HSMAAS) - MO

Hardware Security Module As A Service (HSMAAS) - MO [38] is a component of a collaborative project between the European Space Agency, CGI, and Skudo companies. This initiative prototypes end-to-end data encryption for ground-to-space communications at the CCSDS Mission Operations Message Abstraction Layer (MO/MAL) [33], employing a TLS-inspired handshake and encryption mechanism. The MO/MAL layer defines rules regarding the syntax of high-level application MO services, the semantics of MO interaction models, and the synchronization of communications between entities. The MAL specification provides a standard abstract Application Programming Interface (API), making it a Platform Independent Model (PIM) for defining MO services. To materialize concrete services, a language-specific API binding and a technology binding for message protocol encoding/decoding are utilized. MAL ensures standardization of data types, message headers, allowed sequences of message exchange, Quality of Service (QoS), and access control. The implementation of encryption at the MO/MAL layer holds the potential to enhance the security of a broad spectrum of space missions.

Following the TLS protocol concept (refer to section 2.4), asymmetric encryption is utilized solely in key agreement, while symmetric encryption is performed with derived symmetric data encryption keys to encrypt session data. To validate the implementation, a ground-to-space demonstration involving the OPS-SAT satellite was successfully conducted [37].

Entities involved in the HSMAAS MO work include Ground end nodes (consumers) and the Satellite (provider). Specifically, Mission Operations (MO) applications, responsible for operating spacecraft and their payloads, seek secure interactions. MO services encompass end-to-end operations on the ground or in a spacecraft, handling tasks such as monitoring and controlling spacecraft and payloads, delivering mission data, managing onboard software, analyzing spacecraft performance, determining orbit and attitude, planning and executing mission operations, and preparing predictions and maneuvers. MO services framework is a software platform used to experiment with ground-to-space PKI, it is standardized by CCSDS [34] and places into Application and Network layers of the

OSI model.

The project's output includes the development of Secure MAL, the software responsible for handshake and encryption at the Message Abstraction Layer (MAL). Secure MAL contains two new Java components [36]:

- *Secure Access Control* (SAC): an Access Control interface for MO applications that initiates key agreements and performs MAL message payload encryption. This encryption ensures secure communications, irrespective of the underlying transport technology. SAC queries the security module to determine the existence of a secure session, triggering an automatic handshake if necessary.
- *MO Security Service Consumer/Provider*: a MO service specification in terms of MAL that does the handshake using MO Security Service Interface.

Secure MAL, packaged as a JAR, becomes a dependency for MO applications and can be utilized in both MO consumer and MO provider applications.

One notable security enhancement is the applicability of Secure MAL to any MO application, including existing ones, with any MO transport binding. As MAL messages are transformed to transport technologies like TCP/IP, there is no need to alter the MO framework, and key agreements can be performed over unsecured MO services.

Chapter 5

Design and implementation of the KEMTLS-PDK-based architecture

Imagine a scenario of communication between a Ground node, representing a Mission Operations (MO) application on Earth, and a Space node, which is a MO application aboard a spacecraft. The exchange of sensitive data lacks confidentiality as it is transmitted without encryption, relying on the assumption that nobody would attempt to intercept the traffic. Enter Kat, a curious individual turned attacker, who successfully intercepts this unencrypted traffic, posing potential threats and consequences, as discussed in Chapter 4.

Consider a subsequent scenario, where the Ground and Space nodes decide to enhance their security by adopting the widely used TLS 1.3 protocol. However, in this evolving landscape, Kat has upgraded her capabilities and now possesses a quantum computer, raising new challenges for the security of their communications.

The Master's thesis work addresses the latter scenario by proposing a solution. Conducted during the internship as a cybersecurity engineer at the European Space Operations Centre (ESOC) of the European Space Agency in Darmstadt, the project builds upon an existing project undertaken by ESA in collaboration with CGI and Skudo (see section 4.4). Trying to leverage the existing infrastructure and architecture, the objective is to replace the TLS 1.3-based handshake with a Post-Quantum Cryptography solution, in order to assess the feasibility of implementing a PQC protocol in the context of space systems. The current chapter begins by providing a detailed description of the designed architecture. Subsequently, it delves into the KEMTLS-PDK-based handshake, highlighted key aspects of the implementation. At the end, the significant challenges and technical difficulties encountered throughout implementation phase are examined.

5.1 Architecture

In Figure 5.1, the designed architecture is depicted. It involves three entities that are the Ground end node, which roles as the consumer, the Space end node, which acts as the

provider, and the Ground Public Key Infrastructure. They are described in the following subsections.

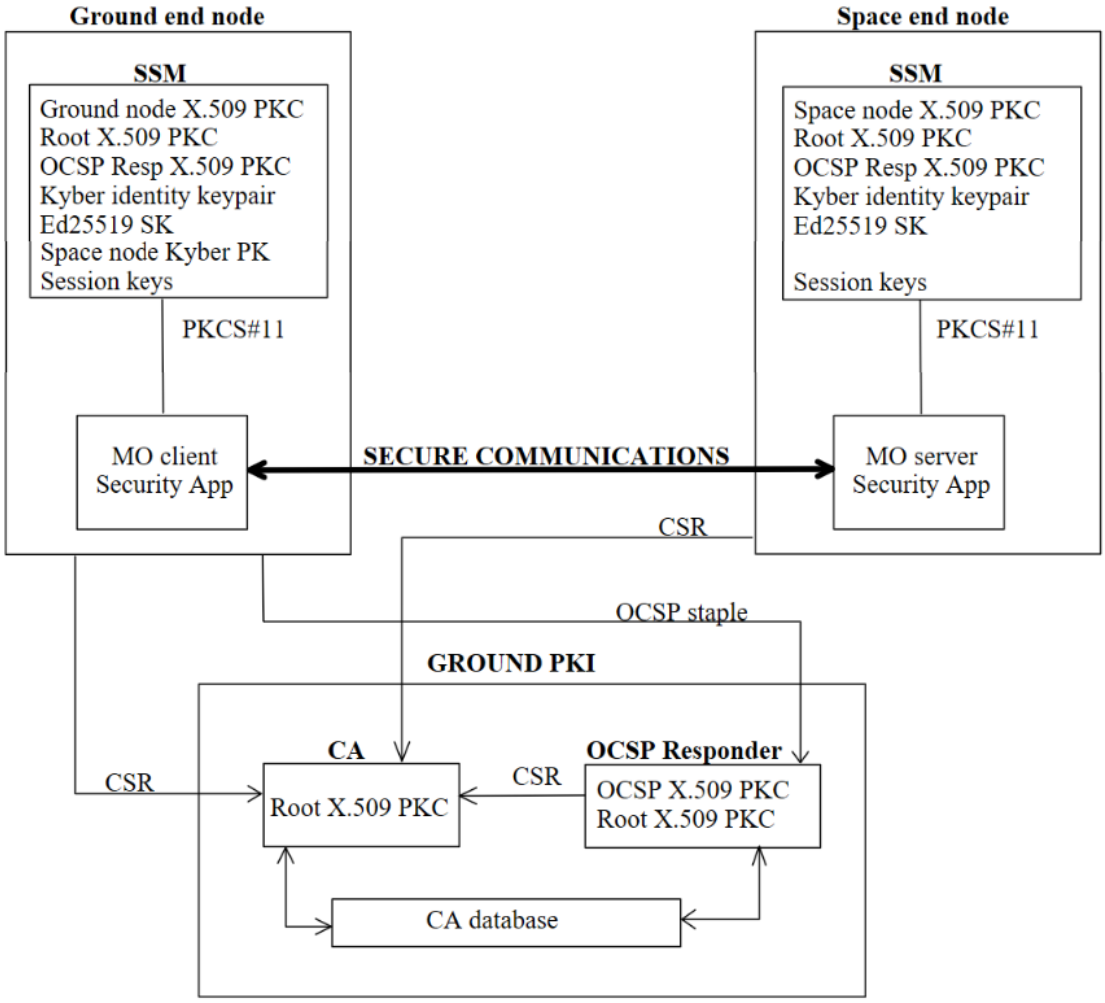

Figure 5.1. Designed architecture

5.1.1 Ground end node

The *Ground end node* is composed of one or more *Mission Operations client Security App*, where the word "Security" has been already added by the HSMAAS - MO project after the implementation of the Secure MAL (see section 4.4). This MO application interacts with node's Software Security Module (SSM) via PKCS#11 interface [21]. PKCS#11 is used to create and manipulate cryptographic tokens like Security Modules. The SSM contains:

- *Ground node X.509 PKC*: it is the node's public key certificate which contains the Ed25519 public key used by whoever wants to verify a digital signature performed by the node.
- *Root X.509 PKC*: it is the public key certificate of the Root Certification Authority that generated the Ground node X.509 PKC.
- *Kyber identity keypair*: as described in section 5.2.2, the PQC protocol adopted for establishing secure sessions uses Kyber512 algorithm for key establishment. The Kyber identity keypair is the long-term keypair that is used with KEM_s (refer to section 3.4.1) during the handshake for implicit authentication of the consumer.
- *Ed25519 SK*: it is the Ed25519 private key associated to the public key contained in Ground node X.509 PKC and used to digitally sign documents.
- *Space node Kyber PK*: KEMTLS-PDK protocol is used, so the consumer already knows the Kyber long-term public key of the provider, and it stores it into the SSM, using it during the handshake.
- *OCSP Resp X.509 PKC*: it is the public key certificate of the OCSP Responder that signs the OCSP staples for client's certificate.
- *Session keys*: it indicates all the 256-bits long keys that have been established after the successful conclusion of the handshakes to establish secure session with MO applications on the Space end node. They are used to encrypt/decrypt data with AES-256-GCM-no padding algorithm.

5.1.2 Space end node

The *Space end node* provides *MO server Security Apps* to consumers on ground. These MO applications interact with the SSM that is onboard, using PKCS#11 interface. While in HSMAAS - MO project the space end node uses a HSM (refer to section 2.5), for this project SSMs were adopted for both peers because no known HSMs implement Kyber algorithm. As the Ground end node, the Space end node SSM encompasses the node's X.509 public key certificate, which contains the Ed25519 public key, the Certification Authority public key certificate, the PKC of the OCSP Responder which signed the OCSP staples that the Space end node has to verify, the node's long-term Kyber keipar, used for identifying the node during the handshake, the private key corresponding to the public one contained in the node's PKC and the session keys established with KEMTLS-PDK-based handshake.

5.1.3 Ground Public Key Infrastructure

When the Ground end node and the Space end node want to ask for a X.509 public key certificate, they send a *Certificate Signing Request* (CSR) to a Certification Authority. This Certification Authority (CA) is part of the Ground Public Key Infrastructure (PKI), that manages the generation, revocation and verification of certificates. The CA has its own certificate (Ground and Space node keep a copy of it within their SSM) that is self-signed because the CA is a root CA, and has access to the CA database, which contains the

certificates managed by the PKI. The PKI encompasses another entity that is the *OCSP Responder*, which has its own PKC generated by the CA after having received a valid CSR, and stores a copy of CA's PKC. The OCSP Responder receives requests of *OCSP staple* by Ground nodes, in order to attest the current validity of their certificate, to be sent together with their certificates during the handshake. In this way, the provider does not have to contact the CA to verify client authentication. OCSP Responder has access to the CA database to verify the status of a certificate.

5.2 Implementation

The designed architecture aims to allow Mission Operations (MO) applications interaction using encryption upon the successful establishment of a secure session. Building upon the existing foundation of CCSDS MO services and the CCSDS Message Abstraction Layer (MAL), implemented in Java by the European Space Agency [13], the Secure MAL implementation (derived from the Secure MAL developed in the HSMAAS - MO Project) has also been realized in Java. The Public Key Infrastructure (PKI) is implemented using Docker containers for both Certification Authority (CA) and Online Certificate Status Protocol (OCSP) Responder. These containers are executed during the setup phase on ground, ensuring availability whenever an entity requests a certificate. They also handle tasks such as revoking invalid certificates and verifying the validity of a certificate. The Kyber functions for Key Encapsulation Mechanism (KEM) are executed using the official implementation of Kyber [25].

5.2.1 Software behaviour

The software behavior is elucidated through a high-level sequence diagram presented in Figures 5.2 and 5.3. In Figure 5.2, the MO Security Service Consumer and MO Security Service Provider, situated at the two endpoints, engage in communication over an untrusted network, performing the handshake in order to mutually authenticate and establish a session key via MO operations. These MO operations, shown in Table 5.1, are *openKEMTLSSecureSession_KeyExchange* and *clientFinished*, and are performed using MAL messages. In MO context, there are six Interaction Patterns (IP) and the following three are used within the implementation:

- *SendIP*: utilized for unidirectional data transmission without requiring a response or acknowledgment. This IP is employed for secure exchange of encrypted data.
- *SubmitIP*: it sends data without receiving a response, but an acknowledgment is provided. This pattern is used for *clientFinished* message, where the acknowledgment is crucial for the client to confirm the successful completion of the handshake.
- *RequestIP*: data is sent, and a response is received. It is used for *openKEMTLSSecureSession_KeyExchange* message and its corresponding response.

In Table 5.1, *IN* encompasses the input arguments of the operation, while *OUT* represents the produced output. The format of their values is expressed as *<name of argument> : (<object type>)*.

Table 5.1. MO operations

Operation identifier	openKEMTLSSecureSession_KeyExchange	
Interaction Pattern	REQUEST	
Pattern Sequence	Message	Body Type
IN	REQUEST	clientHello_AuthenticationData : (ClientHello_AuthenticationData)
OUT	RESPONSE	serverHelloChallengeServerFinished : (ServerHelloChallengeServerFinished)
Operation identifier	clientFinished	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body Type
IN	SUBMIT	clientFinished : (ClientFinished)

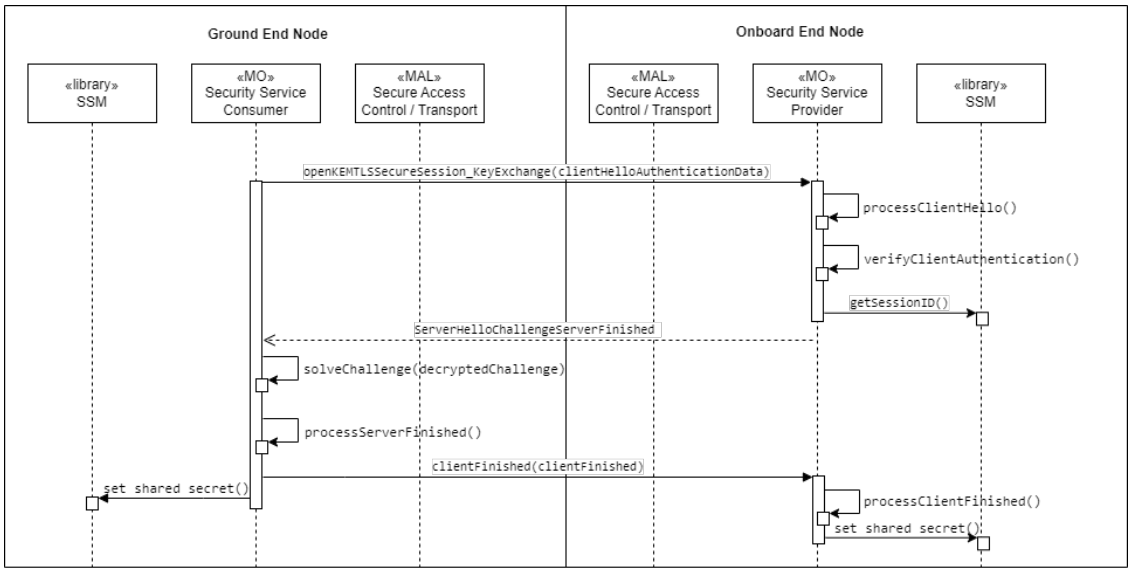

Figure 5.2. Sequence diagram of the handshake.

The initiation of the handshake is an automated process triggered by the consumer with the transmission of the first message and handled by the *Security Service* code. Upon successful verification of the consumer's finished message by the provider, they can start exchanging MAL messages whose bodies are encrypted using AES-256-GCM-no padding and the established shared secret. The changes introduced to the handshake are deeply detailed in section 5.2.2.

Subsequently, the software behavior aligns with that of HSMAAS - MO: as depicted in Figure 5.3, MO applications exchange encrypted MAL messages using MO Security

Service *sendEncryptedData* operation. The encryption/decryption of MAL message bodies is performed by the Secure Access Control component that is provided by Secure MAL. When a peer decides to terminate its execution, all the session keys are removed from its SSM.

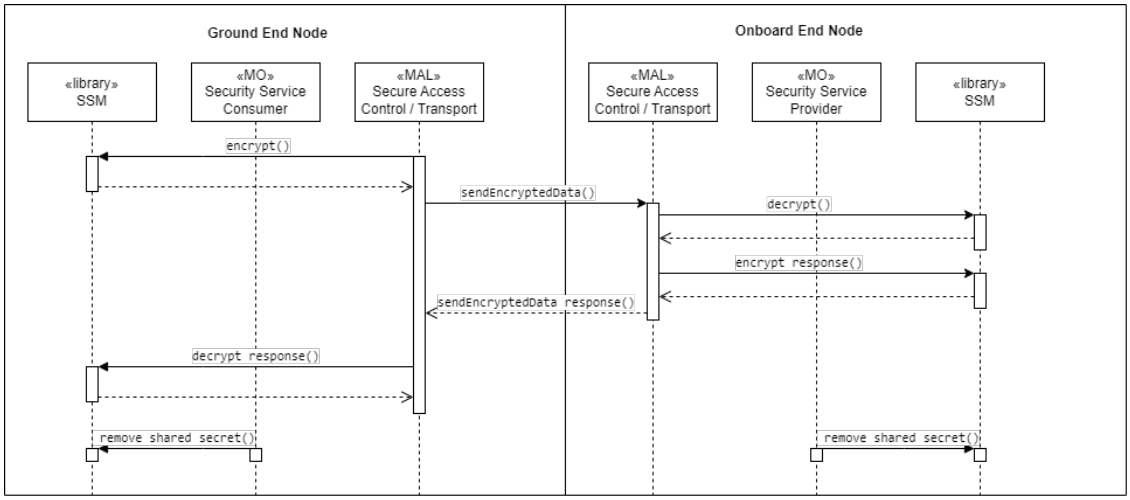

Figure 5.3. Software behaviour after session establishment.

5.2.2 KEMTLS-PDK-based handshake

When a MO client Security Application wants to communicate with a MO server Security App, it starts the handshake. Session resumption mechanism is provided: if a session has been previously established with the same peer, it is possible to directly exchange application data encrypted with the session key established in a prior handshake. Otherwise, the entire handshake must be executed. This handshake is built upon KEMTLS-PDK handshake (refer to section 3.4.1), with the introduction of steps that are necessary for the specific architecture. These additional steps are highlighted in bold in Figures 5.4 and 5.5. Notably, there is no negotiation of algorithms, as they are predetermined by the spacecraft. Consequently, the client must adhere to the algorithms chosen by the spacecraft. In this context, the selected key establishment algorithm is Kyber512. The changes introduced to the handshake are:

- *clientHello*: it is the concatenation of the ephemeral public key pk_e computed with KeyGen function, the ciphertext ct_s output of the Encaps function execution over the static Kyber512 public key pk_s of the provider. Then the concatenation encompasses the *sessionId* and *sessionUri* associated with the specific session. If *sessionId* is 0, then the client wants to establish a new session, otherwise it is attempting to resume an already established session.

- *authenticationData*: it is composed by concatenating several elements, all encrypted using the symmetric key ETS, which was computed in the preceding step by the client. The included fields consist of the client's public key certificate, the OCSP staple associated to client's certificate, client's long-term Kyber public key and the signature of the latter, computed using Ed25519 private key associated to the public key contained in client's certificate.
- *clientHelloAuthNData*: it is the concatenation of *clientHello* and *authenticationData*.
- *openKEMTLSSecureSession_KeyExchange(clientHelloAuthNData)*: it represents the initial message transmitted by the client to the server during the handshake. It contains the *clientHelloAuthNData*, and serves as an indication to the server that the client intends to establish a session.
- *processClientHello*: after receiving the first message by the client, the server decrypts all its fields and computes the digest of the received client's static Kyber public key with the cryptographic hash function chosen at setup time (i.e. SHA3-512). Then, it decrypts the received signature using the public key contained in client's certificate and compares the two digests. If they are the same, then verification is successful, otherwise the handshake is aborted. The server verifies if the received *sessionId* is associated to a valid session key already established.
- *verifyClientAuthentication*: the server checks client's certificate and OCSP staple. If verification fails, the server aborts the handshake.
- *serverHello*: it is the concatenation of the ephemeral ciphertext ct_e computed in the previous step by the server, and the *sessionId* chosen by the server. This *sessionId* is equal to the one sent by the client if the server accepted to do session resumption, otherwise it is associated to a new session that they will open at the end of the successful execution of the handshake.
- *serverHelloChallengeServerFinished*: it is the content of the message that the server will send in response to the received client's message. It is the concatenation of *serverHello*, *challenge* (i.e. ciphertext ct_c encrypted with the key *SHTS* established earlier in the handshake) and *serverFinished* (i.e. HMAC of the transcript CHAD, where CHAD stands for ClientHelloAuthNData, computed with the key fk_s).
- *openKEMTLSSecureSession_KeyExchange(serverHelloChallengeServerFinished)*: it is the response to client's message and contains *serverHelloChallengeServerFinished*.
- *processServerHello(serverHelloChallengeServerFinished)*: received server response, the client understands if session resumption is possible by comparing the received *sessionId* contained in the *serverHello* with the *sessionId* contained in the *clientHello*. When there is session resumption, the challenge is null.

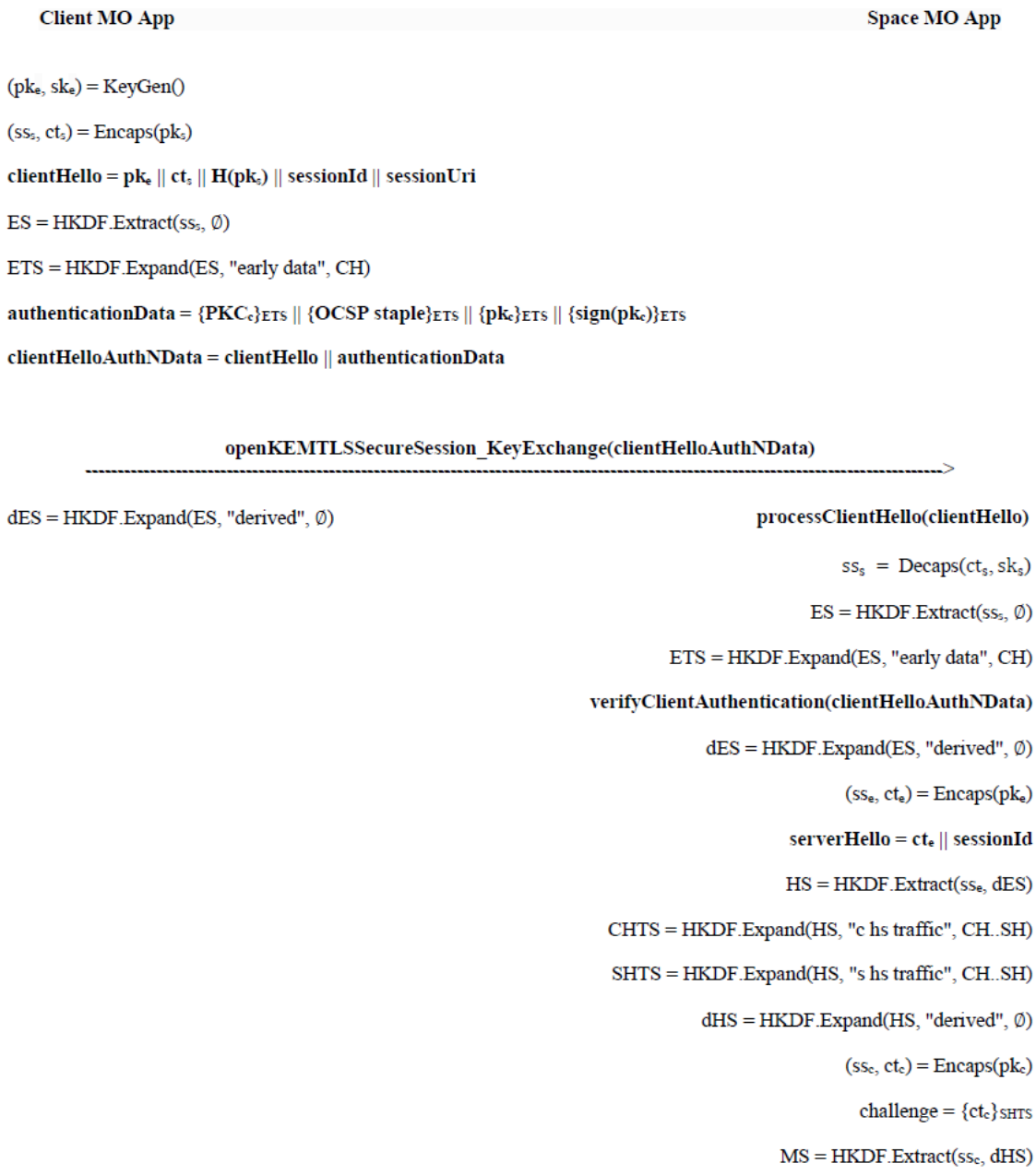

Figure 5.4. First part of KEMTLS-PDK-based handshake.

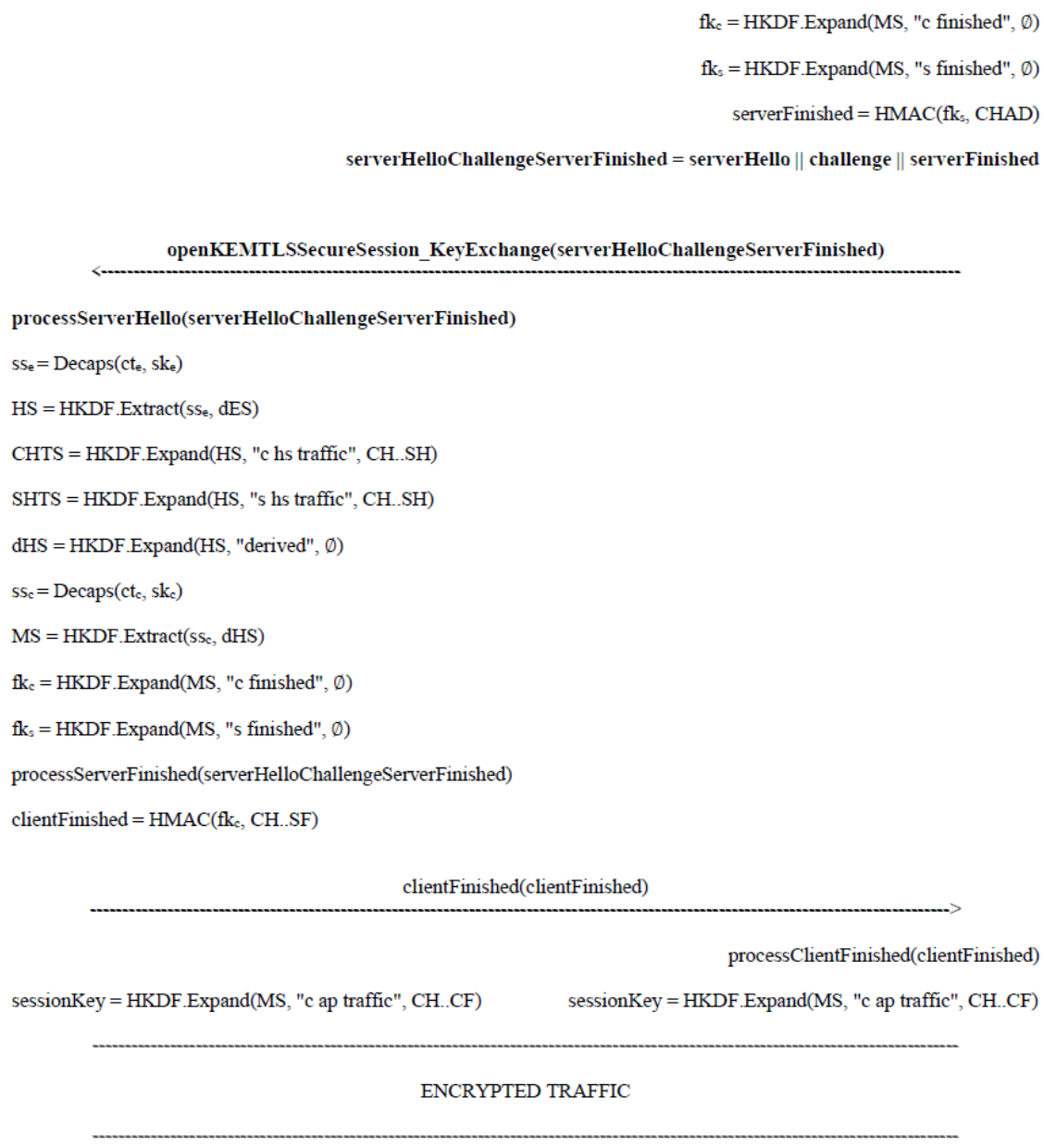

Figure 5.5. Second part of KEMTLS-PDK-based handshake.

In the scenario of session resumption, after processing the *serverHello*, the client retrieves the session key associated to the session ID contained in the *serverHello* and transmits the *clientFinished* message directly. The *clientFinished* contains the HMAC of the transcript of messages exchanged, computed using dES, which is the last key generated by both peers during the handshake. Then, the server processes the received *clientFinished* checking it with the same procedure described for the verification of the *serverFinished*. Once this verification is successful, the exchange of application data, encrypted using AES-256-GCM-No padding, can commence.

Upon successful establishment of the secure session, the session key is securely stored within the software security modules of both peers. Subsequent retrieval of the session key is facilitated by referencing the corresponding session ID.

5.2.3 Secure Message Abstraction Layer

The Message Abstraction Layer (MAL) constitutes an integral part of the Mission Operations (MO) stack [34]. It provides language and message transport independence, along with generic service patterns for MO. All messages traverse the Secure Access Control *check* method, a method previously implemented in HSMAAS - MO, which is reused in the current implementation. Depending on message type, *check* method establishes if the body of the MAL message must be encrypted or not. Encryption is applied to outgoing messages destined to a receiver that is outside the host and it is performed by using session keys established during the implemented KEMTLS-PDK-based handshake. Incoming messages are decrypted if the sender is external, with all internal messages exempt from encryption and decryption.

The encryption process involves the following steps [36]:

1. Run any access control check implementation, if present.
2. Verify the existence of a secure session by checking if a symmetric key associated with a session ID is stored within the Software Security Module. If a session key exists, the client requests session resumption during the handshake; otherwise, the full handshake is performed.
3. Obtain the session ID, which is 0 if the client intends to open a new session; otherwise, it takes the value of the session ID associated with a previously opened session with the desired server.
4. Encode the message body by serializing it into a list of blobs, as illustrated in Figure 5.6.
5. Encrypt the message body using AES-256-GCM with no padding.
6. Set the *authenticationId* header field to *<sessionId>_<timestamp>*.
7. Transform the message header fields *area*, *service* and *operation*.
8. Run Secure Access Control check().

In Figure 5.6, the initial MAL message is depicted on the left with its header containing the fields *authenticationId*, *area*, *service*, *operation*, and the body encompassing one or more MAL elements. During encoding, the message undergoes transformation, and the field *authenticationId* assumes the specific format required by Secure MAL, i.e., *<sessionId>_<timestamp>*. The *area* field contains the security value, the service is *SecurityService*, and the operation is *sendEncryptedData*, indicating to the receiver that the client is equipped with the secure version of MAL and intends to encrypt/decrypt the exchanged

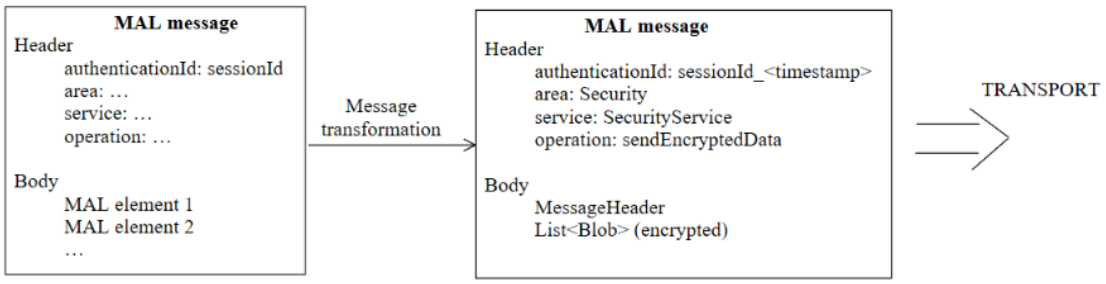

Figure 5.6. Secure MAL message encoding.

data. The body of the encoded message contains the original message header and a list of blobs, which represent the bytes of the MAL elements in the original message body. This list of blobs is encrypted using the session key associated with the session ID contained in the *authenticationId* field. With the foundational support provided by the PKI and SSMs, existing MO application can transit to MAL message encryption by repackaging them with Secure MAL instead of MAL.

5.2.4 Secure Sessions

As shown in Figure 5.7, the same MO consumer can have multiple secure sessions with different MO providers that can be on the same host. When a message goes through the Secure Access Control with an address, e.g. malspp:247/100, the encryption key to use can be found using this address and the session ID attached to the message. Basically, SecureAccessControl is an interceptor for all messages sent and received to/by the consumer or provider.

All the providers on the same host share the same SSM, using different slots, retrieving their session keys via session IDs and employing the retrieved keys to decrypt messages.

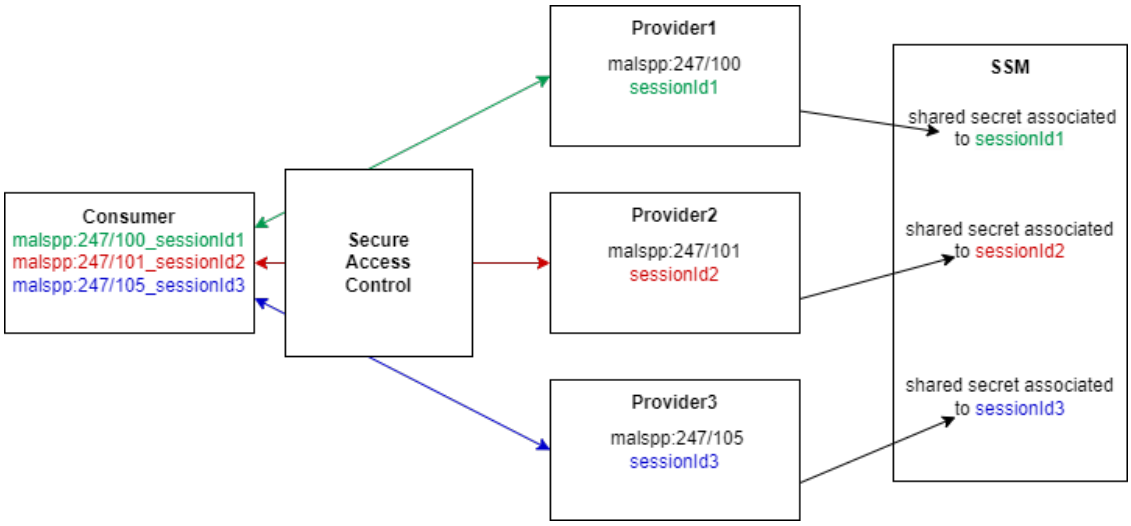

Figure 5.7. Multiple secure sessions for the same consumer.

5.2.5 Software Security Modules

SoftHSMv2 library was used for peers' software security modules [30], supported by OpenSC pkcs11 [18] and Java SunPKCS11 provider [19]. The latter behaves as a bridge between SoftHSMv2 and Java Cryptography Architecture and Java Cryptography Extension. The implemented software delegates key storage and cryptographic functionalities to Software Security Modules not to extract private keys and session keys, keeping all the cryptographic computation within the modules themselves. Actually, this objective is partially reached because currently SoftHSMv2 does not implement Kyber512 algorithm and Post-Quantum Cryptography in general, so Kyber512 keys are extracted from the SSMs. The objective of the architecture is to be reused in the future with security modules, hardware or software-based (depending on the specific requirements), that will support PQC. An example of the content of a SSM is provided in Figure 5.8: the peer is the Consumer Test Tool (refer to section 6.1), which uses the slot 1 of the SSM. The consumer stores provider's Kyber public key, which is an object of type *data* labeled with "ops-sat_public_key". Then the consumer stores its X.509 certificate, its own Kyber static keypair composed by "consumer-test-tool_kyber_public_key" and "consumer-test-tool_kyber_private_key" and its Ed25519 keypair. The two objects of type *secret key* refer to the same session key established with the provider: the one that contains "server-traffic" in the label is used to decrypt the incoming messages, the one that contains "client-traffic" is used to encrypt the outgoing messages.

```
INFO: Using slot with index 1 (0x4e131b94)
Data object 2
  label:          'ops-sat_public_key'
  application:    ''
  app_id:         <empty>
  flags:           modifiable
Certificate Object; type = X.509 cert
  label:      consumer-test-tool
  subject:    DN: C=DE, L=Darmstadt, O=ESA, CN=ConsumerTestTool
  ID:         b39527c22785361f5ed5c0e0ef2b8e07f84a86ee
Data object 4
  label:          'consumer-test-tool_kyber_public_key'
  application:    ''
  app_id:         <empty>
  flags:           modifiable
Data object 5
  label:          'consumer-test-tool_kyber_private_key'
  application:    ''
  app_id:         <empty>
  flags:           modifiable
Secret Key Object; Generic secret length 32
  VALUE:      54332a5a0d877b32c22c670a0dc89959a2979448337f96398ba0ff3c120e8f1b
  label:      malspp:247/100_server-traffic_sessionid_1698696382805
  Usage:      encrypt, decrypt, verify, wrap, unwrap
  Access:     extractable
Secret Key Object; Generic secret length 32
  VALUE:      54332a5a0d877b32c22c670a0dc89959a2979448337f96398ba0ff3c120e8f1b
  label:      malspp:247/100_client-traffic_sessionid_1698696382805
  Usage:      encrypt, decrypt, verify, wrap, unwrap
  Access:     extractable
Private Key Object; unknown key algorithm 64
  label:      consumer-test-tool
  ID:         b39527c22785361f5ed5c0e0ef2b8e07f84a86ee
  Usage:      sign, derive
  Access:     sensitive, always sensitive, never extractable, local
Public Key Object; unknown key algorithm 64
  label:      consumer-test-tool
  ID:         b39527c22785361f5ed5c0e0ef2b8e07f84a86ee
  Usage:      derive
  Access:     local
```

Figure 5.8. Example of the content of a Software Security Module.

5.3 Implementation challenges

In section 5.1, the implemented architecture has been described. However, the initial architecture was different. Initially, the plan revolved around adopting the new Kyber certificates in place of the standard X.509 Public Key Certificates (PKCs), thereby obviating the need for Ed25519 key pairs for peers. As illustrated in Figure 5.9, the Ground end node's SSM stored its Kyber certificate, standard X.509 Root CA and OCSP Responder PKCs. Additionally, it stored the provider's Kyber static public key and the established session keys. The Space end node's SSM stored only the peer's Kyber certificate, Root CA and OCSP Responder X.509 PKCs and session keys. This architecture allowed to reduce the amount of data sent by the client within the first handshake message during the handshake.

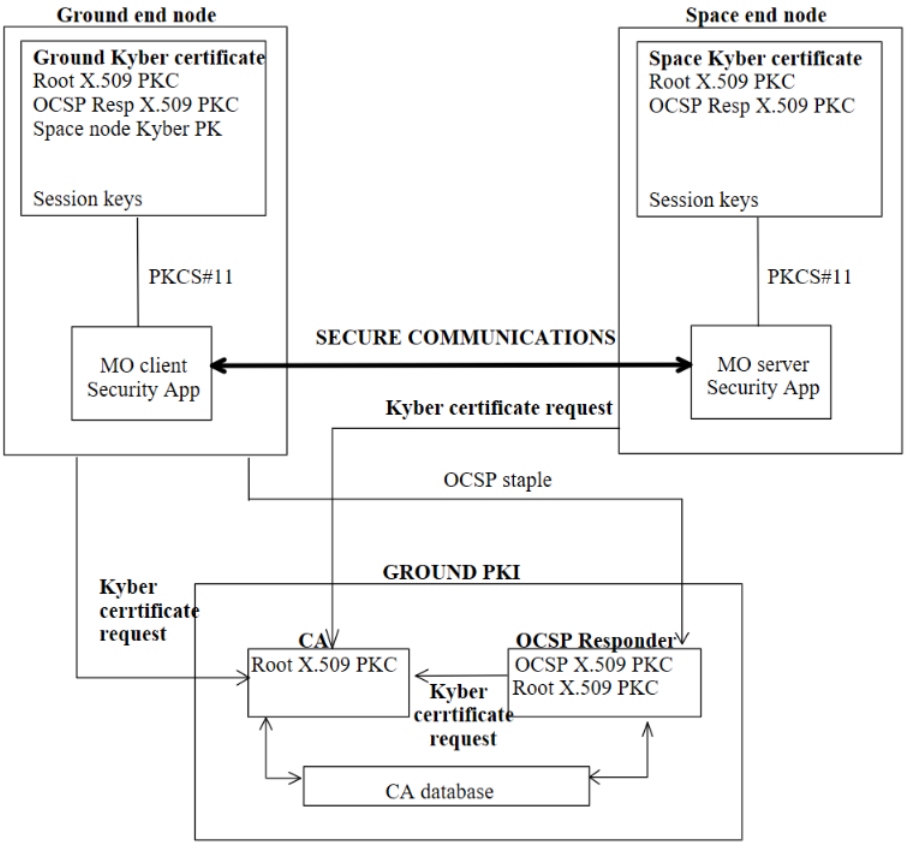

Figure 5.9. Initial implemented architecture.

As shown in Figure 5.10, instead of transmitting the encryption of its standard X.509 PKC, Kyber public key and signature of Kyber public key in the initial handshake message, the client sent the encryption of its Kyber certificate and OCSP staple computed by the OCSP responder over the client's Kyber certificate. Although the generation of

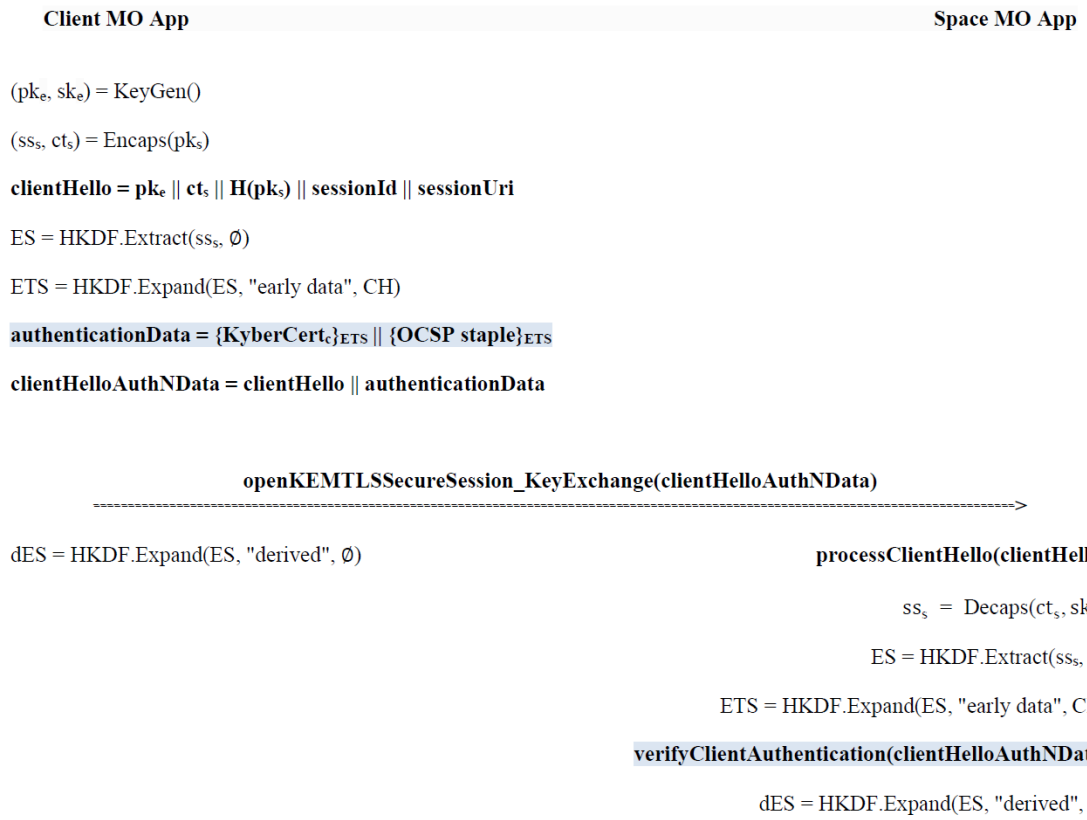

Figure 5.10. First part of initial KEMTLS-PDK-based handshake.

Kyber certificates posed challenges, overcoming them led to the emergence of two subsequent challenges: devising a method to request a Kyber certificate and enabling the OCSP Responder to validate its authenticity. The details of each challenge are elaborated upon in the following subsections. In order to solve all these three challenges, the architecture was modified, introducing X.509 PKCs for the nodes, instead of Kyber certificates. These X.509 PKCs contain Ed25519 public keys, and the corresponding private keys are stored within the software security modules of the owners. When a node wants to ask for a X.509 certificate, it creates a Certificate Signing Request using OpenSSL and sends it to the Certification Authority. Moreover, each node stores its Kyber keypair into the SSM and when a client wants to initiate the handshake, it transmits its Kyber public key together with the signature of this key, performed using the Ed25519 private key contained in the X.509 certificate.

5.3.1 Kyber certificate

The Kyber certificates implemented in this study are founded on the specifications outlined in the Internet Engineering Task Force (IETF) draft entitled 'Internet X.509 Public Key

Infrastructure - Algorithm Identifiers for Kyber' [39]. Adhering to the X.509 standard, the structure of Kyber certificates encompasses the following fields: the *version*, denoting the X.509 standard version; the *serial number*, serving as a unique identifier for the certificate; the *issuer*, containing the Relative Distinguished Name (RDN) of the entity that generated the certificate; the *Validity*, delineating the temporal validity of the certificate; and the RDN of the *Subject*, which is the owner of the certificate. The *Public Key Algorithm* field of the sequence *SubjectPublicKeyInfo* contains Kyber512 identifier and the *SubjectPublicKey* value is the string of the byte stream of Kyber512 public key. Given Kyber's role of key establishment algorithm, the only extension available is *Key Usage*, marked as critical and exclusively containing *keyEncipherment* to reflect the purpose of Kyber public keys. As well as the corresponding Kyber private keys, Kyber certificates were encoded using Distinguished Encoding Rules (DER) encoding of Abstract Syntax Notation One (ASN.1).

```
Certificate:
    Data:
        Version:  <VERSION>
        Serial Number:
            <SERIAL NUMBER>
        Issuer: <ISSUER RDN>
        Validity:
            NotBefore: <DATE & TIME>
            NotAfter: <DATE & TIME>
        Subject: <SUBJECT RDN>
        SubjectPublicKeyInfo:
            Public Key Algorithm: id-alg-kyber-512
            SubjectPublicKey: <BIT STRING>
        Extensions:
            CERT-KEY-USAGE: critical
                keyEncipherment
```

The generation of Kyber certificates posed significant challenges, primarily stemming from the intricacies of manual implementation using the post-quantum cryptography provider of Bouncy Castle [2] Java cryptographic library. Noteworthy difficulties included the recognition of the Kyber OID (Object Identifier) and complications arising during the reading of both the generated certificates and their associated private keys within the Java environment, leveraging the capabilities of Bouncy Castle. Navigating these challenges was imperative to establish a functional and recognizable version of a Kyber certificate. Once this goal was achieved, the focus shifted to the implementation of a Certificate Signing Request.

5.3.2 Generation of Certificate Signing Requests

To obtain an X.509 certificate, an applicant must submit a PKCS#10 Certificate Signing Request (CSR) [17] to the Certification Authority (CA). A CSR is a message containing certification request information, including the public key the requester wants to be signed, the signature algorithm to use, and the digital signature of the certification request information. Upon receiving a CSR, the CA verifies that the provided information is valid and

verifies the signature to ensure that the applicant truly owns the private key corresponding to the public key chosen for inclusion into the certificate. An inherent challenge arises when dealing with Kyber certificates, as the applicant lacks a means to sign the certificate request information using the Kyber private key since Kyber is not a digital signature algorithm. A potential solution, proposed in "Proof-of-possession for KEM certificates using verifiable generation" [10], involves simultaneous key generation and proof of possession. However, the objective of this thesis project was to retain much of the original architecture and infrastructure while utilizing OpenSSL. Although the Open Quantum Safe provider for OpenSSL [35] offers PQC functionalities, it currently does not implement certificates uniquely designed for post-quantum key encapsulation mechanisms.

5.3.3 Certificate verification with OpenSSL

The third challenge pertained to the verification of Kyber certificates generated using Bouncy Castle. To generate an Online Certificate Status Protocol (OCSP) staple, the OCSP responder must verify the status of the certificate, determining its validity. The OCSP Responder, managed using OpenSSL, encountered difficulties because the certificates were manually generated in Java. Despite being signed using the private key of the OpenSSL Certification Authority, the OCSP Responder verification consistently failed.

```
1  V  260818195614Z  01  unknown /C=IT/L=Turin/O=PoliTO/CN=MO Root CA
2  V  241028195616Z  02  unknown /C=IT/L=Turin/O=PoliTO/CN=OCSPResponder
3  V  241028195617Z  03  unknown /C=DE/L=Darmstadt/O=ESA/CN=GroundMOProxy
4  V  241028195617Z  04  unknown /C=DE/L=Darmstadt/O=ESA/CN=ConsumerTestTool
5  V  241028195617Z  05  unknown /C=FR/L=Space/O=ESA/CN=OPSSAT
```

Figure 5.11. OpenSSL index.txt file content.

This failure was attributed to the OpenSSL OCSP Responder's reliance on checking the content of a file named *index.txt*, as depicted in Figure 5.11. This file contains an entry for each certificate generated by the CA, featuring details such as status, generation date, a certificate-associated number, and the distinguished name of the certificate holder. Since certificate generation occurred externally to OpenSSL, no entries were added. Attempting manual modifications to the *index.txt* file to facilitate successful verification and adding the generated certificates to the CA database resulted in OpenSSL crashing. Faced with these challenges, a reevaluation of the design phase was undertaken, leading to the adoption of the solution described at the beginning of this chapter.

Chapter 6

Integration and Testing with OPS-SAT Satellite

The implemented KEMTLS-PDK-based handshake, detailed in Chapter 5, underwent rigorous testing, evolving from a standalone application to a real-world scenario involving OPS-SAT satellite. The progression included successful tests with local Mission Operations Applications, *EchoGround* and *EchoSpace*, exchanging encrypted echo messages. Subsequently, the architecture was adapted to the unique challenges posed by OPS-SAT, a pioneering spacecraft with an experimental nature. This choice was led by the ease of integration, coupled with the author's familiarity gained during the internship at European Space Operations Centre, where participation in the OPS-SAT Mission Control Team was a key responsibility. OPS-SAT tests involved firstly the Engineering Model of the spacecraft, then the satellite itself.

Before delving into the experiment architecture, an overview of OPS-SAT satellite is provided, to outline its innovative features and the experimental objectives that make it an ideal candidate for the proposed study. The chapter proceeds with an in-depth exploration of the experiment architecture tailored for OPS-SAT. To facilitate a comprehensive understanding, the chapter offers a detailed account of the setup configuration for the OPS-SAT experiment. Step-by-step instructions are provided to guide readers through the execution process, ensuring clarity in implementation. Finally, a thorough analysis and interpretation of the obtained results conclude the chapter.

Through this chapter, the thesis aims to showcase the adaptability of the KEMTLS-PDK-based handshake in a practical, mission-critical setting, emphasizing its relevance in advancing secure communication protocols for space missions.

6.1 OPS-SAT

OPS-SAT is the first 3U CubeSat directly owned and operated by the European Space Agency [6]. Launched on 18th December 2019 from Kourou, in French Guiana, OPS-SAT is an experimental nanoSatellite stationed in low Earth orbit. Operated from the European Space Operations Centre (ESOC) in Darmstadt, it is the first satellite mission in the world designed to test satellite control technology in orbit. Figure 6.1 provides a visual

representation of OPS-SAT. Functioning as an open software/hardware innovation platform, the satellite enables testing of new protocols and standards. Moreover, it facilitates European industry, institutions, and individuals in conducting in-flight experiments by running software and firmware, transmitting commands to the spacecraft through the Internet. This process is swift, cost-free, and non-bureaucratic. Currently, OPS-SAT boasts over 260 registered experiments, encompassing areas such as telemetry, image compression algorithms, and experimental IP-cores on the FPGA. Experimenters communicate with their flight experiments by accessing the space segment via the Satellite Experimental Processing Platform (SEPP). As the heart of the mission payload system, SEPP is integral to OPS-SAT's experimental nature, processing all major experiments. SEPP comprises an Altera Cyclone V SX System-on-Chip (SoC) digital logic device and a Cyclone V Field Programmable Gate Array (FPGA). The SEPP system image runs Linux Ångström and incorporates the *NanoSatMO Framework* (NMF) [14], written in Java, which interfaces with all the OPS-SAT payloads systems.

Figure 6.1. OPS-SAT Satellite Source: ESA.

NMF provides services to experimenter applications, allowing NMF applications to run without direct access to the real satellite hardware, leveraging the simulator based on the NMF SDK [3]. The NMF infrastructure hosts Mission Operations (MO) provider applications, housekeeping MO services, and a Supervisor responsible for controlling the life cycle of end-user MO applications. The NMF SDK simulates core OPS-SAT payloads (e.g., camera, GPS, iADCS) and supports the execution of simulation scenarios using the *Consumer Test Tool* (CTT). This tool enables experimenters to test and manually verify their experiments. CTT is also used to interact with end-user MO applications running on OPS-SAT. Communication between experimenters and SEPP can occur offline, through file transfer, or live, receiving and sending space packets in real-time via MO services over the Internet [11].

6.2 Experiment architecture

An high-level representation of the architecture utilized to run the experiment with OPS-SAT satellite is depicted in Figure 6.2. It is coherent with the architecture used by HS-MAAS MO Project (refer to section 4.4) for in-orbit demonstration. On ground, there is the dockerized Public Key Infrastructure, composed of the CA and the OCSP Responder. The ground end node has a Software Security Module, which contains session keys, node's X.509 PKC, Ed25519 private key, Kyber512 static keypair, OPS-SAT Kyber512 static public key, CA and OCSP Responder PKCs. The CTT is the Consumer Test Tool, graphical

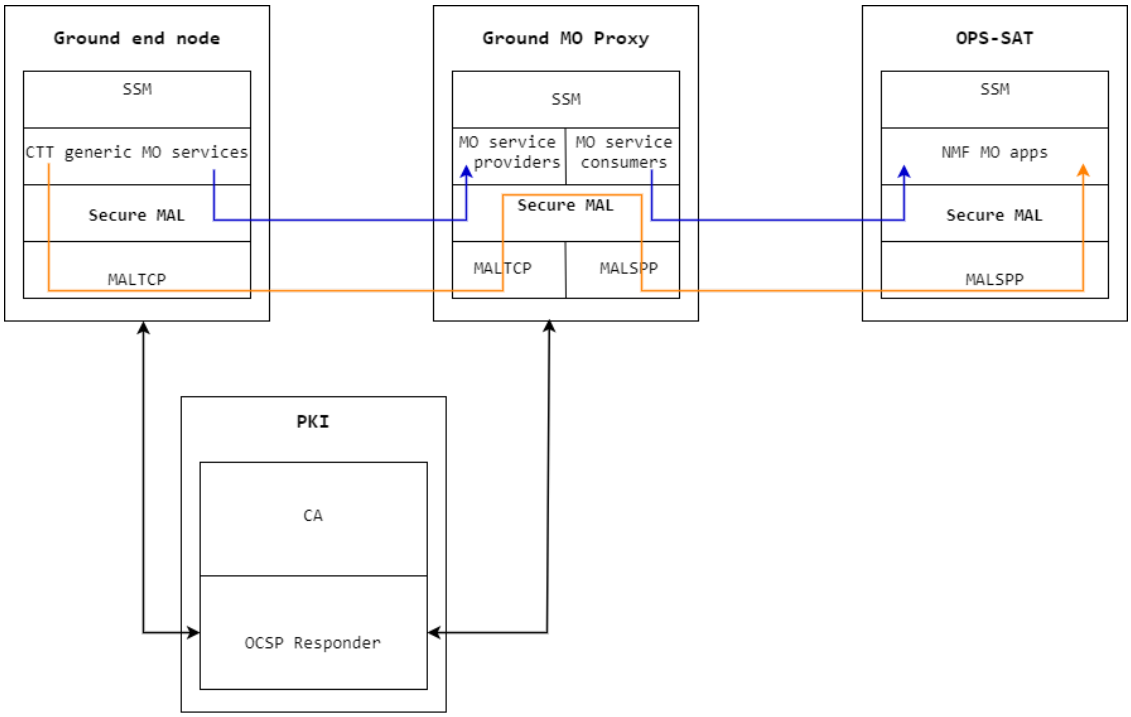

Figure 6.2. High-level experiment architecture.

MO services client used to ask for Mission Operations (MO) services. CTT can connect to any MO service provider app, send commands to it and display responses. It is used to ask for *PushClock* MO application, which simply sends the current time every second. The *PushClock* is run by the NanoSat MO Framework. The Ground MO Proxy translates MALTCP protocol into MALSPP without being able to decrypt the messages, sends them to the NMF MO applications and caches the space side state. A deeper view is given by Figure 6.3, in which continuous arrows represent non-encrypted exchange of messages and dotted arrows represent encrypted application data. Communications between CTT and Ground MO Proxy are not encrypted, messages sent by the CTT to OPS-SAT apps that go through the Ground MO Proxy are encrypted. Both CTT and Ground MO Proxy

share the same Software Security Module, using the former the slot 1, the latter the slot 0. Considering that CTT is the client and the proxy serves as a server for CTT and as a client for OPS-SAT, both can interact with OCSP Provider to ask for OCSP staples. After having translated the transport protocol of the messages, the Ground MO Proxy sends them to ESOC-1 antenna, through a SSH channel MALSPP over TCP. Then the antenna uplinks the messages to the satellite during passes, via MALSPP. A pass is the period of time when the satellite is above the local horizon and it is able to communicate with the ground station. All involved entities (CTT, NMF Supervisor, the specific app *exp263* that runs PushClock and Ground MO Proxy) have the secure version of MAL, so whenever a client sends the initial message, the KEMTLS-PDK-based handshake is initiated. The NMF Supervisor and exp263 share the same SSM and slot, thereby they use the same session keys.

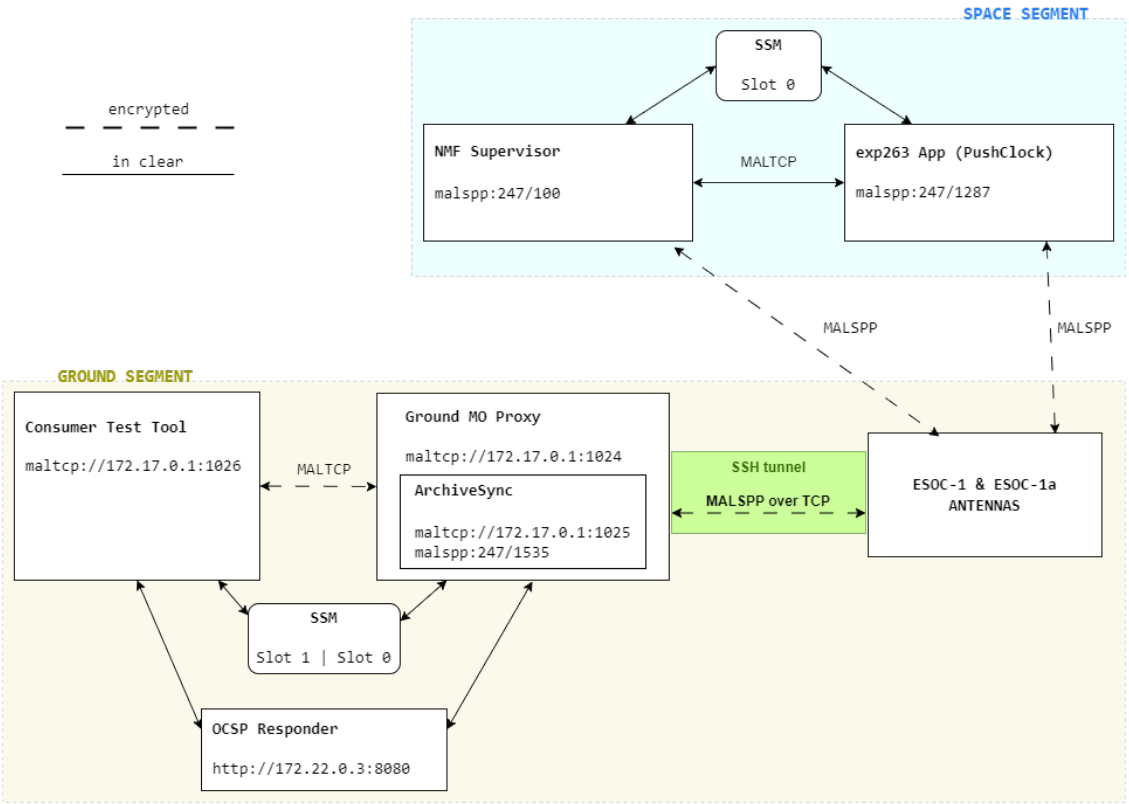

Figure 6.3. Interactions within the architecture.

6.3 Setup configuration of the experiment

Considering that a satellite pass happens in few minutes, to run the experiment it is necessary that everything that can be anticipated is done at setup time. Firstly, it is necessary that the Docker containers of CA and OCSP Responder are deployed. During build process of CA container, a self-signed certificate is generated for the CA and during build process of OCSP Responder, OCSP Responder certificate is generated and signed by the CA. A copy of CA and OCSP Responder certificates is stored into each node's SSM. Consumer Test Tool and Ground MO Proxy are patched with Secure MAL dependency. They run on the same machine, where a Software Security Module is installed, and they are configured to use two different slots of the SSM. CTT and Ground MO Proxy ask to CA for having a X.509 PKC, sending a CSR. The CA receives their CSRs, verifies them and after successful verification, it generates and returns their X.509 PKCs. These certificates are installed into the SSM. OPS-SAT follows the same procedure to request a certificate, during a step that precedes the one in which the experiment is run. At setup time, Ground MO Proxy, CTT and OPS-SAT node generate their Kyber512 keypairs and store them into SSMs.

The algorithms to use are set according to the needs of the spacecraft. For Consumer Test Tool client, these algorithms, together with the configuration at setup time, are:

```
Environment
OCSP_RESPONDER_URI = http://172.22.0.3:8080
IDENTITY_KEY_LABEL = consumer-test-tool
PKCS11_CONFIG_PATH = ground/nmf/consumer-test-tool/pkcs11.cfg
PKCS11_PASSWORD = aj$Q78dnSr!r=
CA_CERT_PATH = ground/nmf/consumer-test-tool/pki/rootCA.crt
OCSP_CERT_PATH = ground/nmf/consumer-test-tool/pki/ocsp.crt
IDENTITY_CERT_PATH = ground/nmf/consumer-test-tool/pki/consumer-t
est-tool.crt
IDENTITY_KYBER_PRIV_KEY_PATH = ground/nmf/consumer-test-tool/pki/
consumer-test-tool_kyber_private_key.txt
IDENTITY_KYBER_PUB_KEY_PATH = ground/nmf/consumer-test-tool/pki/c
onsumer-test-tool_kyber_public_key.txt
SERVER_KYBER_PUBLIC_KEY_PATH = ground/nmf/consumer-test-tool/pki/
ops-sat_kyber_public_key.txt
KEYGEN_SCRIPT_PATH = ground/nmf/bin/generate_kyber512_keypair
ENCAPS_SCRIPT_PATH = ground/nmf/bin/kyber_encapsulation
DECAPS_SCRIPT_PATH = ground/nmf/bin/kyber_decapsulation

Handshake
esa.mo.pki.random.algorithm = PKCS11
esa.mo.pki.random.bytes.size = 128
esa.mo.pki.hashing.algorithm = SHA-512
esa.mo.pki.signature.algorithm = SHA512withECDSA

Encryption
esa.mo.pki.hmac.algorithm = HmacSHA256
```

```
esa.mo.pki.cipher.algorithm = AES/GCM/NoPadding
esa.mo.pki.encryptionkey.algorithm = AES
esa.mo.pki.encryptionkey.size = 256
esa.mo.pki.ivseed.size = 16
esa.mo.pki.gcm.authtag.length = 128
esa.mo.pki.encryption.host.internal = false
esa.mo.pki.encryption.cleartext.areas = 7,105
```

For OPS-SAT, the software is encapsulated into an opkg package and uplinked by OPS-SAT Mission Control Team. The binaries of Kyber512 encapsulation, decapsulation and key generation are cross-compiled in order to run on the spacecraft. A SSM is installed onboard.

Initially, the chosen HMAC algorithm was HmacSHA512 but it was replaced with HmacSHA256 because HmacSHA512 did not work. The client's Ed25519 private key is used to sign using the algorithm SHA512withECDSA. Kyber algorithm, in every variant (i.e., Kyber512, Kyber768 and Kyber1024), produces a session key of 256 bits, that is why esa.mo.pki.encryptionkey.size is 256. There is not internal encryption, i.e., messages exchanged within the same host are sent in clear. The value of the parameter named esa.mo.pki.encryption.cleartext.areas indicates that encryption is turned off for Software Management (i.e., 7) and Platform (i.e., 105) MAL Areas.

6.4 Run the in-orbit demonstration

During the in-orbit demonstration with OPS-SAT satellite, the onboard NanoSat MO Framework (NMF) Supervisor, which is the one that runs onboard MO applications, has to be replaced with the version that supports Secure MAL, in order to let the Supervisor establish secure sessions through the KEMTLS-PDK-based handshake. The in-orbit demonstration can be summarized in the following steps:

1. A SSH tunnel between the Ground MO Proxy and ESOC-1 antenna is opened with the command:

   ```
   ssh -vvv NoemiT@opssat1.esoc.esa.int -N -R0.0.0.0:16256:localhost:4096
   ```

   This SSH tunnel is necessary to let Ground MO Proxy and NMF Supervisor establish a session.

2. Establish a secure session between Ground MO Proxy and NMF Supervisor to encrypt exchanged data, sent via uplink and downlink through antenna.

3. Start the Virtual Network Computing (VNC) server on the machine that hosts CTT and Ground MO Proxy, to let the VNC client connect to it showing a window with the desktop of the remote machine on which the VNC server runs and taking control of the remote machine. This is necessary because the Consumer Test Tool, which plays the role of the consumer in the designed architecture, is a graphical interface. The VNC client runs CTT, and its Graphical User Interface is started, as shown in Figure 6.4. The Directory Service URI of Ground MO Proxy is inserted in CTT GUI

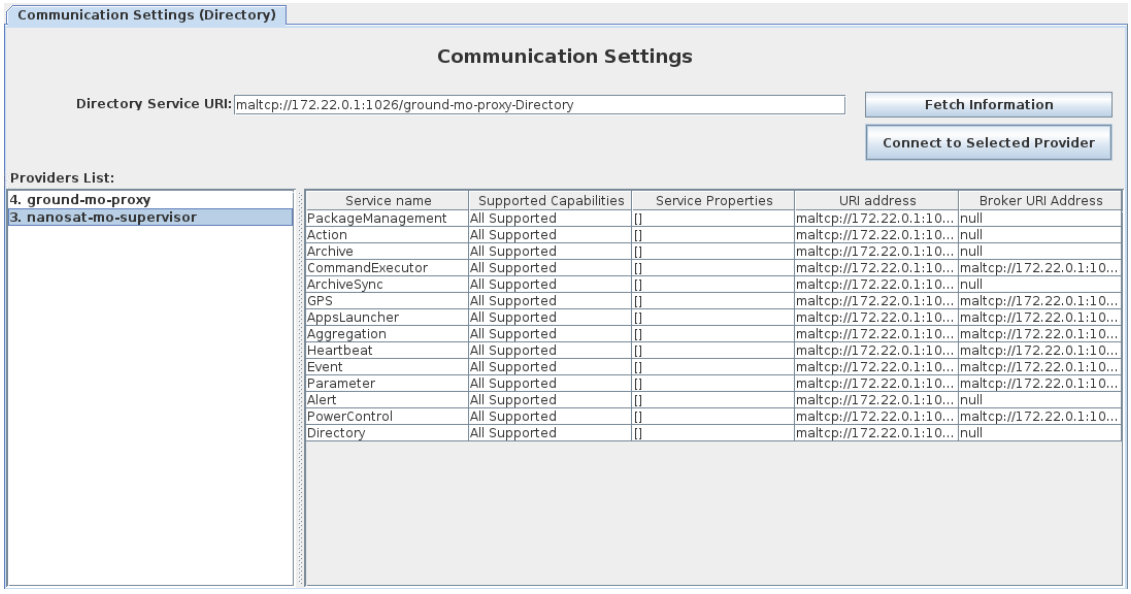

Figure 6.4. Connecting CTT to NMF Supervisor Provider.

and then, from the sidebar named "Providers List", *nanosat-mo-supervisor* is selected to let CTT connect to NMF Supervisor.

4. Select exp263 application in the list of NMF Mission Operations Applications and ask NMF Supervisor to run it. This triggers the initiation of the handshake between CTT and NMF Supervisor, and the establishment of a secure session at the end of the successful handshake process.

5. To establish a secure session between CTT and exp263, exp263 is selected from Providers List and the session establishment leads to the storage of the session key shown in Figure 6.5. This key is used for encryption and decryption of messages exchanged between CTT and exp263 end nodes. The CTT logs of the hanshake with exp263 are provided in Figure 6.7, while exp263's part of the handshake is shown in Figure 6.8. They report exactly the steps of the handshake described in section 5.2.2. Then, in Figure 6.9, there is a perspective of the outgoing messages sent by CTT and of the incoming messages received by CTT. It is evident that the messages destined to and coming from NMF Supervisor and exp263 application are encrypted. The logs provide a summary of MO services, "count" column contains the number of messages encrypted so far and "Encrypted size" column represents the number of encrypted bytes of the message.

The code of exp263 app starts PushClock application, opening a new tab related to PushClock. Telemetry, i.e. onboard time sent by PushClock, is sent encrypted by exp263 and is decrypted and displayed by CTT, as shown in Figure 6.6.

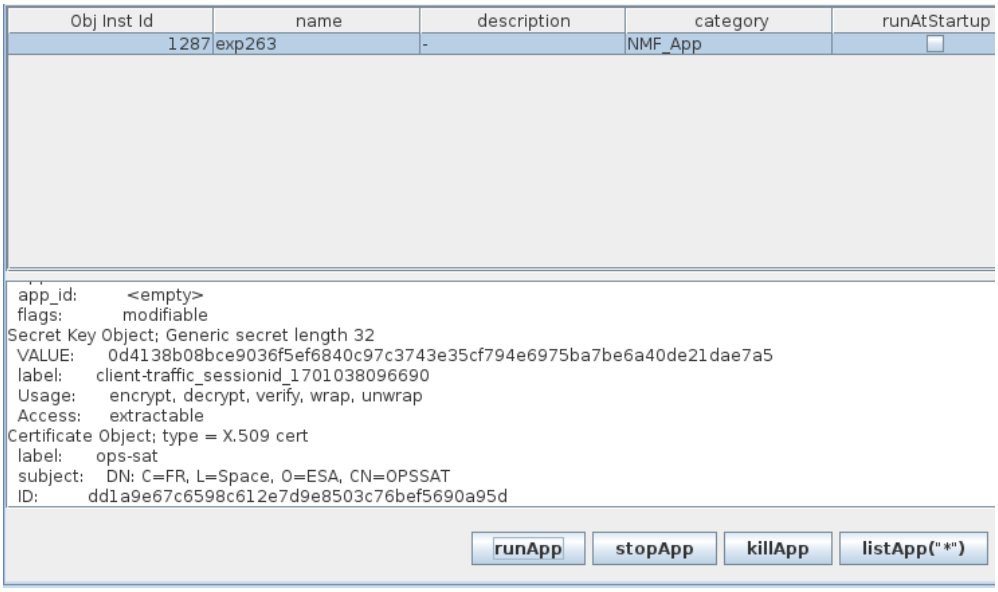

Figure 6.5. Store session key.

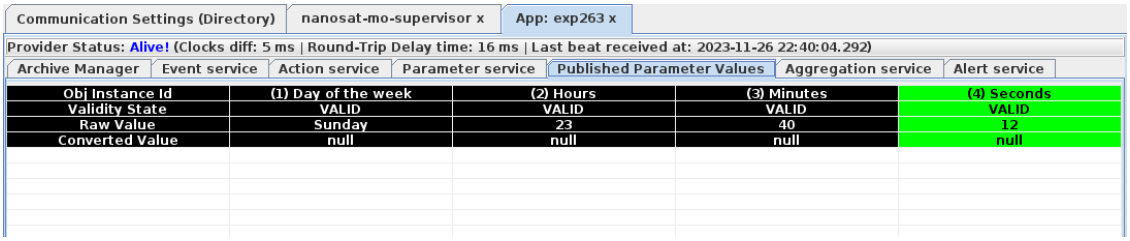

Figure 6.6. Onboard time decrypted by CTT.

```
315  INFO: CONSUMER [consumer-test-tool] - Opening secure session...
316  2023-11-26 23:39:31.673 esa.mo.pki.securityservice.SecurityServiceClient createSecurityConsumer
317  INFO: Security Consumer URI      : maltcp://172.22.0.1:1027/SecurityConsumer
318  2023-11-26 23:39:32.082 esa.mo.pki.securityservice.SecurityServiceClient openSecureSession
319  INFO: CONSUMER [consumer-test-tool] - Sending ClientHello and AuthenticationData...
320  2023-11-26 23:39:32.085 esa.mo.pki.accesscontrol.MALSecureAccessControlImpl check
321  INFO: Handshake message: maltcp://172.22.0.1:1027/SecurityConsumer -> maltcp://172.22.0.1:1025/ProtocolBridge@malspp:247/1287/0
322  2023-11-26 23:39:32.743 esa.mo.pki.accesscontrol.MALSecureAccessControlImpl check
323  INFO: Handshake message: maltcp://172.22.0.1:1025/ProtocolBridge@malspp:247/1287/0 -> maltcp://172.22.0.1:1027/SecurityConsumer
324  2023-11-26 23:39:32.744 esa.mo.pki.securityservice.SecurityServiceClient openSecureSession
325  INFO: CONSUMER [consumer-test-tool] - Received ServerHello, Challenge and ServerFinished. There is NOT SESSION RESUMPTION. Session with ID=sessionid_1701038372162 is going to be opened.
326  2023-11-26 23:39:32.791 esa.mo.pki.securityservice.SecurityServiceClient openSecureSession
327  INFO: CONSUMER [consumer-test-tool] - Solving the challenge...
328  2023-11-26 23:39:32.798 esa.mo.pki.securityservice.SecurityServiceClient openSecureSession
329  INFO: CONSUMER [consumer-test-tool] - Processing ServerFinished...
330  2023-11-26 23:39:32.847 esa.mo.pki.encryption.implementation.ClientHandshakeImpl processServerFinished
331  INFO: CONSUMER - ServerFinished successfully processed.
332  2023-11-26 23:39:32.963 esa.mo.pki.securityservice.SecurityServiceClient openSecureSession
333  INFO: CONSUMER [consumer-test-tool] - Sending ClientFinished...
334  2023-11-26 23:39:32.964 esa.mo.pki.accesscontrol.MALSecureAccessControlImpl check
335  INFO: Handshake message: maltcp://172.22.0.1:1027/SecurityConsumer -> maltcp://172.22.0.1:1025/ProtocolBridge@malspp:247/1287/0
336  2023-11-26 23:39:33.219 esa.mo.pki.accesscontrol.MALSecureAccessControlImpl check
337  INFO: Handshake message: maltcp://172.22.0.1:1025/ProtocolBridge@malspp:247/1287/0 -> maltcp://172.22.0.1:1027/SecurityConsumer
338  2023-11-26 23:39:33.220 esa.mo.pki.securityservice.SecurityServiceClient openSecureSession
339  INFO: CONSUMER [consumer-test-tool] - Setting session key for sessionUri=malspp:247/1287 and sessionID=sessionid_1701038372162.
340  2023-11-26 23:39:33.221 esa.mo.pki.encryption.SecureSessionKeyStore setSecretKey
341  INFO: Storing session secret key with alias: malspp:247/1287_client-traffic_sessionid_1701038372162
342  2023-11-26 23:39:33.266 esa.mo.pki.encryption.SecureSessionKeyStore setSecretKey
343  INFO: Storing session secret key with alias: malspp:247/1287_server-traffic_sessionid_1701038372162
```

Figure 6.7. Logs of CTT's handshake with exp263.

```
336  2023-11-26 23:39:32.092 esa.mo.pki.accesscontrol.MALSecureAccessControlImpl check
337  INFO: Handshake message: malspp:247/1463/204 -> malspp:247/1287/0
338  2023-11-26 23:39:32.160 esa.mo.pki.securityservice.SecurityServiceServer openKEMTLSSecureSession_KeyExchange
339  INFO: PROVIDER - Starting key exchange... with malspp:247/1463/204
340  2023-11-26 23:39:32.161 esa.mo.pki.securityservice.SecurityServiceServer openKEMTLSSecureSession_KeyExchange
341  INFO: PROVIDER - Processing ClientHello... coming from malspp:247/1463/204
342  2023-11-26 23:39:32.162 esa.mo.pki.securityservice.SecurityServiceServer openKEMTLSSecureSession_KeyExchange
343  INFO: PROVIDER - ClientHello processed. SESSION ID to use: sessionid_1701038372162.
344  2023-11-26 23:39:32.173 esa.mo.pki.securityservice.SecurityServiceServer openKEMTLSSecureSession_KeyExchange
345  INFO: PROVIDER - Client authentication... of malspp:247/1463/204
346  2023-11-26 23:39:32.402 esa.mo.pki.encryption.implementation.ServerHandshakeImpl verifyClientAuthentication
347  INFO: PROVIDER - Client authentication correctly done.
348  2023-11-26 23:39:32.420 esa.mo.pki.securityservice.SecurityServiceServer openKEMTLSSecureSession_KeyExchange
349  INFO: SERVER - No session resumption.
350  2023-11-26 23:39:32.738 esa.mo.pki.securityservice.SecurityServiceServer openKEMTLSSecureSession_KeyExchange
351  INFO: PROVIDER - Sending ServerHello, Challenge and ServerFinished...
352  2023-11-26 23:39:32.739 esa.mo.pki.accesscontrol.MALSecureAccessControlImpl check
353  INFO: Handshake message: malspp:247/1287/0 -> malspp:247/1463/204
354  2023-11-26 23:39:32.968 esa.mo.pki.accesscontrol.MALSecureAccessControlImpl check
355  INFO: Handshake message: malspp:247/1463/204 -> malspp:247/1287/0
356  2023-11-26 23:39:32.970 esa.mo.pki.securityservice.SecurityServiceServer clientFinished
357  INFO: PROVIDER - Processing ClientFinished... of malspp:247/1463/204
358  2023-11-26 23:39:33.078 esa.mo.pki.encryption.implementation.ServerHandshakeImpl processClientFinished
359  INFO: PROVIDER - ClientFinished successfully processed.
360  2023-11-26 23:39:33.083 esa.mo.pki.securityservice.SecurityServiceServer clientFinished
361  INFO: PROVIDER - Setting session key for sessionUri=malspp:247/1287 and sessionID=sessionid_1701038372162.
362  2023-11-26 23:39:33.084 esa.mo.pki.encryption.SecureSessionKeyStore setSecretKey
363  INFO: Storing session secret key with alias: client-traffic_sessionid_1701038372162
364  2023-11-26 23:39:33.136 esa.mo.pki.encryption.SecureSessionKeyStore setSecretKey
365  INFO: Storing session secret key with alias: server-traffic_sessionid_1701038372162
```

Figure 6.8. Logs of exp263's handshake with CTT.

6.5 Results

The experiment can be deemed successful as it fulfills all the project requirements. Specifically, it demonstrates the feasibility of employing a Post-Quantum Cryptography protocol, KEMTLS-PDK, to establish secure sessions between Mission Operations (MO) applications on ground and MO applications aboard a spacecraft that implements the MO stack. The established secure sessions enable peers to exchange encrypted messages using a quantum-resistant symmetric cryptographic protocol, namely AES-256-GCM-no padding. Rigorous testing has been conducted, confirming the successful establishment of all expected secure sessions through the KEMTLS-PDK-based handshake. Notable instances include the Ground MO Proxy and NMF Supervisor, CTT and Supervisor, as well as CTT and exp263. Following a successful handshake, the system permits the transmission of encrypted telecommand to the satellite and the reception of encrypted telemetry from OPS-SAT. The examination of Software Security Modules content confirms the correct establishment of session keys, ensuring that involved peers possess identical session keys. Furthermore, the implemented symmetric encryption and decryption mechanisms exhibit accurate functionality, as illustrated in Figure 6.6, where the readability of the onboard time received as part of telemetry is evident. Both the CTT and onboard statistics tables reflect encrypted messages and the number of bytes encrypted. Notably, the spacecraft's onboard behavior remains in accordance with requirements: the NMF Supervisor interacts with other onboard applications without encrypting and decrypting messages, as stipulated by the necessity to omit encryption within the same host. This successful implementation showcases the viability and effectiveness of the proposed cryptographic protocol in a real-world space mission context.

Outgoing messages

Service operation	Type	URI To	Count	Encrypted size (diff)
COM::Archive.retrieve	internal	maltcp://172.22.0.1:1026	13	N/A
SoftwareManagement::Heartbeat.getPeriod	cleartext	malspp:247/100	5	N/A
COM::Event.monitorEvent	encrypted	malspp:247/100	3	131 bytes (+57.8%)
Common::Directory.lookupProvider	internal	maltcp://172.22.0.1:1026	2	N/A
MC::Action.listDefinition	internal	maltcp://172.22.0.1:1026	2	N/A
MC::Aggregation.monitorValue	encrypted	malspp:247/100	1	38 bytes (+72.7%)
SoftwareManagement::PackageManagement.findPackage	cleartext	malspp:247/100	1	N/A
MC::Aggregation.listDefinition	encrypted	malspp:247/100	1	21 bytes (+320.0%)
SoftwareManagement::AppsLauncher.runApp	cleartext	malspp:247/100	1	N/A
Security::SecurityService.openKEMTLSSecureSession_KeyExchange	handshake	malspp:247/1287	1	N/A
Security::SecurityService.clientFinished	handshake	malspp:247/100	1	N/A
MC::Alert.listDefinition	encrypted	malspp:247/100	1	21 bytes (+320.0%)
SoftwareManagement::AppsLauncher.listApp	cleartext	malspp:247/100	1	N/A
SoftwareManagement::Heartbeat.beat	cleartext	malspp:247/100	1	N/A
MC::Alert.listDefinition	encrypted	malspp:247/1287	1	21 bytes (+320.0%)
SoftwareManagement::AppsLauncher.monitorExecution	cleartext	malspp:247/100	1	N/A
COM::Event.monitorEvent	encrypted	malspp:247/1287	1	38 bytes (+72.7%)
MC::Parameter.listDefinition	encrypted	malspp:247/1287	1	21 bytes (+320.0%)
Security::SecurityService.openKEMTLSSecureSession_KeyExchange	handshake	malspp:247/100	1	N/A
SoftwareManagement::Heartbeat.beat	cleartext	malspp:247/1287	1	N/A
MC::Aggregation.listDefinition	encrypted	malspp:247/1287	1	21 bytes (+320.0%)
SoftwareManagement::Heartbeat.getPeriod	cleartext	malspp:247/1287	1	N/A
MC::Parameter.monitorValue	encrypted	malspp:247/100	1	38 bytes (+72.7%)
MC::Parameter.listDefinition	encrypted	malspp:247/100	1	21 bytes (+320.0%)
MC::Parameter.monitorValue	encrypted	malspp:247/1287	1	38 bytes (+72.7%)
Security::SecurityService.clientFinished	handshake	malspp:247/1287	1	N/A
MC::Aggregation.monitorValue	encrypted	malspp:247/1287	1	38 bytes (+72.7%)

Incoming messages

Service operation	Type	URI From	Count	Encrypted size (diff)
COM::Event.monitorEvent	encrypted	malspp:247/100	62	6797 bytes (+16.1%)
MC::Parameter.monitorValue	encrypted	malspp:247/100	32	2325 bytes (+27.1%)
COM::Archive.retrieve	internal	maltcp://172.22.0.1:1026	26	N/A
SoftwareManagement::AppsLauncher.monitorExecution	cleartext	malspp:247/100	24	N/A
COM::Event.monitorEvent	encrypted	malspp:247/1287	19	1782 bytes (+19.3%)
MC::Parameter.monitorValue	encrypted	malspp:247/1287	19	1296 bytes (+28.6%)
SoftwareManagement::Heartbeat.beat	cleartext	malspp:247/100	17	N/A
SoftwareManagement::Heartbeat.getPeriod	cleartext	malspp:247/100	5	N/A
SoftwareManagement::Heartbeat.beat	cleartext	malspp:247/1287	3	N/A
Common::Directory.lookupProvider	internal	maltcp://172.22.0.1:1026	2	N/A
MC::Action.listDefinition	internal	maltcp://172.22.0.1:1026	2	N/A
MC::Aggregation.monitorValue	encrypted	malspp:247/100	1	0 bytes (+0%)
SoftwareManagement::PackageManagement.findPackage	cleartext	malspp:247/100	1	N/A
MC::Aggregation.listDefinition	encrypted	malspp:247/100	1	18 bytes (+800.0%)
SoftwareManagement::AppsLauncher.runApp	cleartext	malspp:247/100	1	N/A
Security::SecurityService.openKEMTLSSecureSession_KeyExchange	handshake	malspp:247/1287	1	N/A
Security::SecurityService.clientFinished	handshake	malspp:247/100	1	N/A
MC::Alert.listDefinition	encrypted	malspp:247/100	1	18 bytes (+800.0%)
SoftwareManagement::AppsLauncher.listApp	cleartext	malspp:247/100	1	N/A
MC::Alert.listDefinition	encrypted	malspp:247/1287	1	18 bytes (+800.0%)
MC::Parameter.listDefinition	encrypted	malspp:247/1287	1	30 bytes (+114.3%)
Security::SecurityService.openKEMTLSSecureSession_KeyExchange	handshake	malspp:247/100	1	N/A
MC::Aggregation.listDefinition	encrypted	malspp:247/1287	1	18 bytes (+800.0%)
SoftwareManagement::Heartbeat.getPeriod	cleartext	malspp:247/1287	1	N/A
MC::Parameter.listDefinition	encrypted	malspp:247/100	1	57964 bytes (+0.0%)
Security::SecurityService.clientFinished	handshake	malspp:247/1287	1	N/A
MC::Aggregation.monitorValue	encrypted	malspp:247/1287	1	0 bytes (+0%)

Figure 6.9. Logs of CTT outgoing and incoming messages.

Chapter 7

Conclusions and Future Work

The master's thesis main objective was to present the architecture implemented to secure ground-to-space communications between Mission Operations applications that run on ground and on spacecrafts. Using KEMTLS-PDK protocol on top of CCSDS Message Abstraction Layer, sessions have been established with a process that is quantum-resistant, allowing the exchange of data encrypted using AES-256-GCM-no padding algorithm, which is secure against quantum attacks. The process that led to the implementation and then to in-orbit demonstration of the architecture with OPS-SAT satellite was challenging, starting from delving into Post-Quantum Cryptography, to considering the intricacies of space systems.

The project started as an extension of HSMAAS MO project, realised by European Space Agency in collaboration with Skudo and CGI, so the initial phase involved a thorough analysis of the HSMAAS MO project. It required a meticulous examination of project documentation and source code to understand, modify, and integrate Post-Quantum Cryptography. Firstly, the TLS 1.3-based handshake was studied, followed by an examination of its integration into the Message Abstraction Layer. Subsequently, the study extended to both standalone testing and evaluation within the framework of two dockerized Mission Operations applications operating on ground. After delving into the initial project during the intensive understanding phase, the focus shifted towards discerning the components to retain, those requiring modification, and devising strategies for their adaptation while integrating Post-Quantum Cryptography without causing disruptions to the existing framework. The implementation of KEMTLS-PDK-based handshake as a standalone application was succeeded by a meticulous testing phase, during which the tests were adapted from the initial project to align with the current implementation. The next step was comprehending what Mission Operations are, and how Message Abstraction Layer works, in order to understand how to integrate the handshake on top of it. Numerous individuals played pivotal roles in navigating these phases: colleagues from ESA and from CGI were instrumental in understanding the source code, providing documentation, sharing knowledge, offering valuable guidance during the implementation, and providing crucial assistance in troubleshooting errors encountered during both implementation and testing stages. The most challenging part of this work was the initial implementation, as detailed in Section 5.3, because of the innovative nature of incorporating Kyber certificates, a concept entirely new and derived solely from the IETF draft [39]. This implementation required an

in-depth analysis of the IETF draft, of the structure of an X.509 public key certificate and utilization of the Bouncy Castle Java cryptographic library. The solution to overcome problems related to Kyber certificate request and verification was to use both PQC and classical cryptography. After being able to find this solution, the next challenge was related to adopting the architecture to a specific scenario that encompasses specific entities: Consumer Test Tool is the MO consumer application, NMF Supervisor and PushClock MO app are the providers, Ground MO proxy is a consumer for NMF Supervisor and is needed to forward the MAL messages from CTT to NMF Supervisor/PushClock just changing the transport protocol and without being able to decrypt the message bodies.

The tests conducted with the Engineering Model of the satellite proved to be both incredibly interesting and complex. Configuring the environment, ensuring accurate parameter settings, and orchestrating the configuration in the correct order and with precise timing constituted a remarkable lesson about space systems operation. Moreover, the internship at ESA allowed the involvement in other tasks beyond the thesis, including contributing to OPS-SAT onboard software development, which provided additional context for understanding issues that, while not directly related to implemented architecture, surfaced during the project.

This thesis work not only constituted a significant challenge, but also carried a profound responsibility to demonstrate that not only space systems require encryption for communications, but that this encryption must be robust enough to withstand emerging threats as quantum ones, because spacecrafts have a long life cycle that lets them staying in orbit for many years. The collaborative efforts and perseverance led to the successful implementation of the proposed cryptographic solution, which showcases the feasibility and effectiveness of integrating advanced cryptographic techniques into the complex domain of space systems. While the journey was undoubtedly demanding, the ultimate goal was successfully realized, contributing to the advancement of secure communication methods in Space. Contributing to the evolution of cryptography for space systems was both an honor and a captivating experience. Integrating knowledge from Master's Degree studies, delving into the state-of-the-art literature, designing inventive solutions, conducting research, and implementing novel concepts provided a comprehensive learning journey. The culmination of the project with real satellite testing, initially with the Engineering Model and later with the actual spacecraft, added a practical dimension to the academic exploration.

As a prospect for future work, enhancing the presented implementation could involve a transition to the initial idea of adopting Kyber certificates instead of X.509 PKCs. This would entail sending the encryption of Kyber certificates instead of the encryption of standard X.509 PKCs, Kyber long-term public key, and its signature. This solution would reduce the amount of data sent by the client to the server, being conceptually correct because in KEMTLS-PDK handshake, digital signature is not needed, so Kyber certificates would be sufficient. However, implementing such a solution necessitates support for Kyber certificates by tools like OpenSSL: this would include commands for generating, verifying their validity, and creating certificate requests comparable to Certificate Signing Requests. An advantageous aspect of the current architecture is its adaptability beyond the OPS-SAT satellite; it can be readily reused for other satellites based on the CCSDS MO stack. Shifting towards using only Kyber certificates could enable the establishment of secure sessions based only on Post-Quantum Cryptography. Looking ahead, it would be intriguing to conduct additional tests that were not feasible during the internship. These

tests could focus on performance metrics such as computation, memory usage, and time efficiency. Furthermore, conducting experiments with both KEMTLS-PDK and TLS 1.3 would provide insights into their comparative effectiveness when deployed aboard a spacecraft.

Bibliography

[1] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. «CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM». In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2018, pp. 353–367. DOI: 10.1109/EuroSP.2018.00032.

[2] *Bouncy Castle Cryptography Library*. URL: https://www.bouncycastle.org/java.html.

[3] C. Coelho, D. Marszk, et al. «NanoSat MO Framework». In: URL: https://github.com/esa/nanosat-mo-framework.

[4] Sandro Coretti, Ueli Maurer, and Björn Tackmann. «A Constructive Perspective on Key Encapsulation». In: 8260 (Jan. 2013). DOI: 10.1007/978-3-642-42001-6_16.

[5] Laurence Duquerroy. *CYBER SECURITY AND SPACE BASED SERVICES Webinar*. 2020. URL: https://business.esa.int/sites/business/files/Cybersecurity%20and%20Space%20based%20service_Webinar_Slides.pdf.

[6] David Evans. «OPS-SAT: Operational Concept for ESA'S First Mission Dedicated to Operational Technology». In: May 2016. DOI: 10.2514/6.2016-2354.

[7] XML Security Working Group F2F. *Key Encapsulation: A New Scheme for Public-Key Encryption*. May 2009. URL: https://lists.w3.org/Archives/Public/public-xmlsec/2009May/att-0032/Key_Encapsulation.pdf.

[8] Gregory Falco, Wayne Henry, Marco Aliberti, Brandon Bailey, Mathieu Bailly, Sebastien Bonnart, Nicolò Boschetti, Mirko Bottarelli, Adam Byerly, Joseph Brule, Antonio Carlo, Giulia Rossi, Gregory Epiphaniou, Matt Fetrow, Daniel Floreani, Nathaniel Gordon, Duncan Greaves, Bruce Jackson, Garfield Jones, and Mattias Wallen. «An International Technical Standard for Commercial Space System Cybersecurity - A Call to Action». In: Oct. 2022. DOI: 10.2514/6.2022-4302.

[9] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: quant-ph/9605043 [quant-ph].

[10] Tim Güneysu, Philip Hodges, Georg Land, Mike Ounsworth, Douglas Stebila, and Greg Zaverucha. *Proof-of-possession for KEM certificates using verifiable generation*. Cryptology ePrint Archive, Paper 2022/703. https://eprint.iacr.org/2022/703. 2022. DOI: 10.1145/3548606.3560560. URL: https://eprint.iacr.org/2022/703.

[11] M. Henkel, P. Romano, and R. Zeif. «OPS-SAT Phase B2/C/D/E1 - Experimenter ICD». In: 2016.

[12] Shivam Lohani and Rinki Joshi. «Satellite Network Security». In: *2020 International Conference on Emerging Trends in Communication, Control and Computing (ICONC3)*. 2020, pp. 1–5. DOI: 10.1109/ICONC345789.2020.9117553.

[13] D. Marszk, C. Coelho, et al. *ESA's Java implementation of the CCSDS MO services*. URL: https://github.com/esa/mo-services-java.

[14] D. Marszk, C. Coelho, et al. «NanoSat MO Framework - mission tailoring for OPS-SAT». In: URL: https://github.com/esa/nmf-mission-ops-sat.

[15] Michele Mosca and Marco Piani. «Quantum Threat Timeline Report 2022». In: 2022. URL: https://globalriskinstitute.org/mp-files/2022-quantum-threat-timeline-report-dec.pdf/.

[16] NIST. «Post-Quantum Cryptography - Selected Algorithms 2022». In: 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[17] Magnus Nystrom and Burt Kaliski. *PKCS #10: Certification Request Syntax Specification Version 1.7*. RFC 2986. Nov. 2000. DOI: 10.17487/RFC2986. URL: https://www.rfc-editor.org/info/rfc2986.

[18] *OpenSC - PKCS#11 API*. URL: https://github.com/OpenSC/libp11.

[19] Oracle. *JDK 8 PKCS#11 Reference Guide*. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/p11guide.html.

[20] Massimo Pellegrino and Gerald Stang. *Space security for Europe*. Tech. rep. European Union Institute for Security Studies (EUISS), 2016. URL: http://www.jstor.org/stable/resrep07091 (visited on 11/11/2023).

[21] *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40*. OASIS. URL: https://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html.

[22] François Quiquet. *Description of the Elements of a Satellite Command and Control System*. 2019. URL: https://www.spacesecurity.info/en/description-of-the-elements-of-a-satellite-command-and-control-system/.

[23] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

[24] «Rustls library». In: URL: https://docs.rs/rustls/latest/rustls/.

[25] P. Schwabe, G. Seiler, et al. *Kyber*. URL: https://github.com/pq-crystals/kyber.

[26] Peter Schwabe, Douglas Stebila, and Thom Wiggers. «More efficient post-quantum KEMTLS with pre-distributed public keys». In: *Computer Security – ESORICS 2021*. Ed. by Elisa Bertino, Haya Shulman, and Michael Waidner. Lecture Notes in Computer Science. Cham: Springer International Publishing, Sept. 2021, pp. 3–22. ISBN: 978-3-030-88418-5. DOI: 10.1007/978-3-030-88418-5_1. URL: https://kemtls.org/publication/kemtlspdk/.

[27] Peter Schwabe, Douglas Stebila, and Thom Wiggers. «Post-Quantum TLS Without Handshake Signatures». In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, 1461–1480. ISBN: 9781450370899. DOI: 10.1145/3372297.3423350. URL: https://kemtls.org/publication/kemtls/.

[28] Peter Schwabe and Bas Westerbaan. *Kyber Post-Quantum KEM*. Internet-Draft draft-cfrg-schwabe-kyber-03. Work in Progress. Internet Engineering Task Force, Sept. 2023. 32 pp. URL: https://datatracker.ietf.org/doc/draft-cfrg-schwabe-kyber/03/.

[29] Peter W. Shor. «Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer». In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172. eprint: https://doi.org/10.1137/S0097539795293172. URL: https://doi.org/10.1137/S0097539795293172.

[30] *SoftHSM version 2*. URL: https://github.com/opendnssec/SoftHSMv2.

[31] Consultative Committee for Space Data Systems. *Blue Books: Recommended Standards*. URL: https://public.ccsds.org/Publications/BlueBooks.aspx.

[32] Consultative Committee for Space Data Systems. «CCSDS Cryptographic Algorithms». In: 2019. URL: https://public.ccsds.org/Pubs/352x0b2.pdf.

[33] Consultative Committee for Space Data Systems. «CCSDS Mission Operations Message Abstraction Layer». In: 2013. URL: https://public.ccsds.org/Pubs/521x0b2e1.pdf.

[34] Consultative Committee for Space Data Systems. «CCSDS Mission Operations Services Concept». In: 2010. URL: https://public.ccsds.org/Pubs/520x0g3.pdf.

[35] D. Stebla, M. Mosca, et al. *oqsprovider - Open Quantum Safe provider for OpenSSL (3.x)*. URL: https://github.com/open-quantum-safe/oqs-provider.

[36] ESA HSMAAS Team. «Hardware Security Module as a Service (HSMaaS) / MO - Software Design Document». In: 2023.

[37] ESA HSMAAS Team. «Hardware Security Module as a Service (HSMaaS) / MO - Technical Note: Cryptographic App Experiment – Results». In: 2023.

[38] ESA HSMAAS Team. «Hardware Security Module as a Service (HSMaaS) / MO - Technical Note: MO Security Analysis». In: 2023.

[39] Sean Turner, Panos Kampanakis, Jake Massimo, and Bas Westerbaan. *Internet X.509 Public Key Infrastructure - Algorithm Identifiers for Kyber*. Internet-Draft draft-ietf-lamps-kyber-certificates-02. Work in Progress. Internet Engineering Task Force, Oct. 2023. 9 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lamps-kyber-certificates/02/.

[40] European Union. «European Union Space Strategy for Security and Defence». In: 2023.

[41] Thom Wiggers. «Post-Quantum TLS». PhD thesis. Nijmegen, The Netherlands: Radboud University, Jan. 9, 2024. URL: https://thomwiggers.nl/publication/thesis/.