HOCHSCHULE
DER MEDIEN

FACULTY OF PRINT AND MEDIA
Hochschule der Medien Stuttgart

Bachelor's thesis

Hybrid Cryptographic Protocols: Measuring the Performance of a Combined ECDH and Kyber Key Exchange Scheme

Submitted by:

Luca Breh
Mat.-Nr.: 41952

to obtain the academic degree of Bachelor of Science for the degree programme "Medieninformatik"

First Examiner	Prof. Dr. Roland Schmitz
Second Examiner	Prof. Dr. Joachim Charzinski

June 4, 2025

Abstract

Advances in quantum computing pose a serious threat to classical key exchange mechanisms. In scenarios such as *Harvest Now, Decrypt Later*, where encrypted data is stored today and decrypted in the future, long-term confidentiality is no longer assured. To mitigate this threat, hybrid approaches combine classical and post-quantum cryptographic schemes, creating a layered defense strategy during the transition to quantum-resistant systems.

This thesis investigates the performance characteristics of a hybrid key exchange protocol that combines a classical (ECDH using Curve25519) with a post-quantum key encapsulation mechanism (Kyber768). A modular test framework was developed to evaluate the key exchange phase both in isolation and in hybrid composition. Each mode was benchmarked with 1,000 repetitions, recording latency, CPU usage, and memory consumption. The system supports native and Docker-based test execution and enables experimentation under emulated, realistic network conditions.

Results show that the post-quantum algorithm (Kyber768) performs comparably to or better than the classical ECDH scheme. While the hybrid combination introduces a small but measurable increase in CPU load and latency, it remains within acceptable operational bounds. Although network impairments affected absolute performance metrics, all key exchange modes are impacted similarly, resulting in no change to their relative performance ranking.

The findings confirm that hybrid key exchange protocols offer a viable and robust path toward post-quantum readiness, balancing modern cryptographic compatibility with future-proof security guarantees.

German Abstract

Der Fortschritt im Bereich des Quantencomputings stellt eine ernsthafte Bedrohung für klassische Schlüsselaustauschverfahren dar. In Szenarien wie *Harvest Now, Decrypt Later*, bei denen verschlüsselte Daten heute abgefangen, gespeichert und in der Zukunft entschlüsselt werden, ist die langfristige Vertraulichkeit nicht mehr gewährleistet. Um dieser Bedrohung zu begegnen, kombinieren hybride Ansätze klassische mit quantenresistenten Algorithmen und schaffen so eine mehrschichtige Verteidigungsstrategie für den Übergang zu quantensicheren Systemen.
Diese Arbeit untersucht die Leistungsmerkmale eines hybriden Schlüsselaustauschverfahrens, das ein klassisches Verfahren (ECDH mit Curve25519) und ein postquantenkryptografisches Schlüsselkapselungsverfahren (Kyber768) kombiniert. Ein modulares Testframework wurde entwickelt, um die Schlüsselaustauschphase sowohl isoliert als auch in hybrider Zusammensetzung zu evaluieren. Jeder Modus wurde mit 1.000 Wiederholungen getestet, wobei Latenz, CPU-Auslastung und Arbeitsspeicherauslastung gemessen wurden. Das System unterstützt sowohl native als auch Docker-basierte Testausführungen und ermöglicht Experimente unter emulierten, realitätsnahen Netzwerkbedingungen.
Die Ergebnisse zeigen, dass der postquantenkryptografische Algorithmus (Kyber768) eine vergleichbare oder bessere Leistung als das klassische ECDH-Verfahren erzielt. Obwohl die hybride Kombination zu einem minimalen, aber messbaren Anstieg der CPU-Last und Latenz führt, bleibt sie im Rahmen betrieblicher Anforderungen.
Netzwerkbeeinträchtigungen beeinflussen zwar die absoluten Leistungswerte, betreffen jedoch alle Schlüsselaustauschmodi in ähnlicher Weise, sodass sich deren relative Leistungsbewertung nicht verändert.
Die Ergebnisse bestätigen, dass hybride Schlüsselaustauschmechanismen einen praktikablen und robusten Weg zur Postquantensicherheit darstellen, indem sie aktuelle kryptografische Standards mit zukunftssicherer Post-Quantum-Technologie verbinden.

Contents

1 Introduction **3**
- 1.1 Motivation . . . 3
- 1.2 Problem Statement . . . 3
- 1.3 Objective . . . 4
- 1.4 Research Questions . . . 4
- 1.5 Thesis Overview . . . 5

2 Related Work **6**
- 2.1 Classical Key Exchange Mechanisms . . . 6
- 2.2 Post-Quantum Cryptography . . . 6
- 2.3 Hybrid Key Exchange Standardization Approach . . . 7
- 2.4 Existing Implementations and Experiments . . . 8
- 2.5 Position of this Work . . . 9

3 Theoretical Background **10**
- 3.1 Classical Cryptographic Foundations . . . 10
  - 3.1.1 Fundamentals of Public-Key Cryptography . . . 10
  - 3.1.2 The Discrete Logarithm Problem and its Elliptic Curve Variant . . . 10
  - 3.1.3 Elliptic Curve Diffie-Hellman . . . 11
  - 3.1.4 Limitations in the Context of Quantum Computing . . . 12
- 3.2 Post-Quantum Cryptography . . . 13
  - 3.2.1 Design Principles and Security Goals . . . 13
  - 3.2.2 Hardness Assumptions in Post Quantum Cryptography . . . 13
  - 3.2.3 The Learning With Error Problem and its Module Variant . . . 14
  - 3.2.4 Key Encapsulation with Kyber . . . 15
- 3.3 Hybrid Key Exchange Concepts . . . 17
  - 3.3.1 Motivation for Hybrid Key Exchange . . . 17
  - 3.3.2 Conceptual Model . . . 18
  - 3.3.3 Security Properties . . . 19
  - 3.3.4 Limitations and Considerations . . . 20

4 Implementation **21**
- 4.1 System Architecture . . . 21
  - 4.1.1 System Overview . . . 21
  - 4.1.2 Cryptographic Libraries . . . 21
  - 4.1.3 Logging and Metrics . . . 21
- 4.2 Key Exchange Strategies . . . 22
- 4.3 Network Emulation . . . 23
- 4.4 Execution Environments . . . 24
- 4.5 Automation . . . 25
  - 4.5.1 Configuration and Scripts . . . 25
  - 4.5.2 Determinism and reproducibility . . . 25

5 Experiments and Evaluation **26**
5.1 Objectives . . . 26
5.2 Experiments . . . 26
5.3 Measurement Perspective and Execution Environment . . . 28
5.3.1 Logging Perspective Evaluation . . . 28
5.3.2 Performance Across Environments . . . 29
5.4 Mode Comparison . . . 31
5.4.1 Key Exchange Duration Across Modes . . . 31
5.4.2 System Resource Utilisation Across Modes . . . 32
5.4.3 Summary of Cryptographic Cost Differences . . . 34
5.5 Network Profile Impact . . . 34
5.5.1 Key Exchange Duration Across Network Profiles . . . 34
5.5.2 Resource Consumption Across Network Profiles . . . 35
5.6 Robustness Summary . . . 37

6 Conclusions **39**

1 Introduction

1.1 Motivation

The continuous advancement of quantum computing poses a significant threat to the security of widely deployed cryptographic protocols. Quantum computers are expected to break algorithms such as RSA and Elliptic-curve Diffie–Hellman (ECDH). These algorithms underpin many public-key infrastructures today and rely on mathematical problems like integer factorization or the discrete logarithm. These problems are believed solvable in polynomial time on a sufficiently powerful quantum computer using Shor's algorithm [1]. Although such machines are not yet available at scale, recent advancements in quantum hardware research underscore the accelerating progress toward quantum-capable adversaries. [2, 3]

In response to these emerging threats, organizations and infrastructure providers must adopt a strategy of cryptographic agility to ensure long-term data confidentiality. This is particularly urgent in light of the so-called *Harvest Now, Decrypt Later* scenario, in which encrypted data is intercepted and stored today with the aim of decrypting it once quantum capabilities become accessible. In many cases, it is necessary to ensure the confidentiality of data for extended periods of time to protect it against future decryption attempts — for instance, information such as medical records or diplomatic communications. This creates a pressing need to act today, even though quantum attacks are not yet practical. [4, 5]

To address these challenges, global standardization efforts have made significant progress. As part of its post-quantum cryptography (PQC) standardization program, the U.S. National Institute of Standards and Technology (NIST) has selected *CRYSTALS-Kyber* — a quantum-resistant algorithm — as the first standard for post-quantum key exchange, marking a major milestone in the shift toward quantum-secure infrastructure. [6, 7]

Since key exchange (KEX) mechanisms are a central component of secure communication protocols such as Transport Layer Security (TLS), evaluating the practical performance properties of quantum-resistant KEX mechanisms is essential to understanding their real-world applicability. [8]

However, post-quantum algorithms are still relatively new and are often viewed with caution. The mathematical problems they are based on have not been tested to the same extent as established classical cryptographic problems. This uncertainty has motivated the development of hybrid KEX mechanisms, which combine classical and post-quantum schemes to maintain security even if one problem is compromised. [8, 7]

Finally, the integration of PQC into existing systems poses significant performance and engineering challenges. PQC schemes often involve different failure probabilities, new implementation constraints, or additional hardware requirements. It is therefore important to empirically evaluate their impact on handshake performance, resource consumption, and robustness to support well-informed deployment decisions. [5, 7]

1.2 Problem Statement

Despite the growing importance of PQC, there is still limited publicly available performance data on quantum-safe and hybrid KEX mechanisms in realistic deployment scenarios. While classical schemes like ECDH are well established and Kyber has recently been standardized as a post-quantum key-encapsulation mechansim (KEM), most evaluations of these algorithms focus on their security properties or analyze them in the context of full protocol stacks [5]. As a result, the performance implications of using these mechanisms — partic-

ularly considering the numerous possible hybrid combinations — under varied system and network conditions are not yet comprehensively understood. [7]

Hybrid KEX mechanisms are designed to maintain security even if one of their cryptographic components is broken, making them an attractive solution during the migration phase. However, these constructions inherently require the parallel execution of at least two KEX operations, which raises concerns regarding their computational efficiency, latency overhead, and compatibility with resource-constrained platforms. [8, 9]

Although several studies have examined the integration of post-quantum algorithms into full TLS protocol stacks, such evaluations typically measure the performance of the entire handshake process. [5, 10] As a result, it is difficult to isolate and attribute performance characteristics specifically to the KEX component.

1.3 Objective

The objective of this thesis is to design and implement a lightweight, modular framework for the empirical evaluation of KEX mechanisms. The implementation supports three configurations:

- a classical KEX using ECDH with x25519 as a performance baseline,
- a post-quantum KEM based on Kyber768, as specified in Federal Information Processing Standards (FIPS) 203 [6],
- a hybrid configuration combining both ECDH and Kyber768 to establish a joint shared secret.

The framework aims to provide isolated performance measurements for each configuration by focusing exclusively on the KEX phase, decoupled from other protocol components such as authentication and session management. Furthermore, the evaluation is intended to yield statistically reliable insights into how these configurations perform under varying, realistic network conditions. The goal is to contribute empirical data that supports the assessment of hybrid and quantum-safe KEX protocols with regard to their deployability in practical systems.

1.4 Research Questions

With respect to the outlined challenges and knowledge gaps, this thesis seeks to answer the following research questions:

1. *How does a hybrid KEX mechanism perform compared to classical (ECDH) and post-quantum (Kyber768) approaches with regard to:*
   - *KEX latency,*
   - *CPU utilization,*
   - *and memory consumption?*
2. *What are the practical trade-offs involved in combining classical and post-quantum KEX algorithms within a hybrid construction?*
3. *How do varying network conditions — such as increased latency, packet loss, and limited bandwidth — affect the robustness and efficiency of the chosen post-quantum and hybrid KEX mechanisms compared to the classical ECDH method?*

1.5 Thesis Overview

In the following Chapter *Related Work*, existing research on classical, post-quantum, and hybrid KEX mechanisms is reviewed. The chapter outlines foundational approaches to secure key establishment and discusses how emerging post-quantum algorithms aim to address the threat posed by quantum adversaries. Furthermore, current standardization efforts are examined, along with an overview of available implementations and experimental studies that explore the practical deployment of hybrid KEX schemes.

Chapter 3 *Theoretical Background* introduces the cryptographic and mathematical foundations relevant to hybrid KEX. It begins with the basic principles of public-key cryptography and elaborates on classical hardness assumptions that form the basis for the experiments conducted in this thesis. Additionally, important principles of post-quantum cryptography are introduced, covering the Learning with Errors (LWE) problem, its module variant, and the process of establishing a shared secret using Kyber. Finally, hybrid KEX is analyzed from both a conceptual and a security-theoretic perspective, including a discussion of design motivations, compositional models, and known limitations.

After the groundwork is covered, Chapter 4 *Implementation* describes the practical realization of the KEX framework developed in this thesis. The setup used to implement and evaluate the KEX mechanisms is outlined, including the integration of cryptographic libraries, the configuration of test environments, and the automation of benchmark execution. This implementation serves as the foundation for the evaluation presented in the next chapter.

Chapter 5 *Experiments and Evaluation* presents the results of performance measurements conducted for various KEX configurations. The chapter compares classical, pure post-quantum schemes, and hybrid schemes under different network conditions and evaluates their impact on latency and computational footprint. The findings are used to assess the practical feasibility of a hybrid deployment.

Chapter 6 *Conclusion* summarizes the main findings of the thesis and reflects on the implications of hybrid KEX in the context of the post-quantum transition.

2 Related Work

2.1 Classical Key Exchange Mechanisms

Classical KEX mechanisms form the foundation of modern cryptographic protocols and are widely used in secure communication systems such as TLS and Secure Shell (SSH). [11, 12] Historically, two main approaches have been established: Diffie-Hellman (DH) and RSA-based KEX. Both algorithms rely on mathematical problems that are considered hard for classical computers. [13, Chapter 7]

Static DH and RSA-based protocols, while used in earlier versions of TLS, have been deprecated in TLS 1.3 as KEX algorithms due to their lack of forward secrecy and known weaknesses [11]. Traditional DH, based on the discrete logarithm problem in a multiplicative group, has largely been replaced by elliptic-curve-based variants because of their improved efficiency and stronger security properties [14].

A widely used classical mechanism today is ECDH, particularly in its ephemeral form (Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)). In modern protocols like TLS 1.3, ECDHE with the Curve25519 parameter set — in this work referred to as x25519 — is the default KEX method [11]. x25519 is favored due to its relatively small key sizes and well-understood security properties [15][13, Chapter 8]. It is widely supported in cryptographic libraries such as OpenSSL and is used in a wide range of real-world applications based on modern cryptographic protocols. [16, 11]

Despite their widespread adoption and proven performance in classical computing environments, these schemes are vulnerable in the presence of sufficiently powerful quantum computers. Shor's algorithm enables efficient factorization of large integers and computation of discrete logarithms, effectively compromising both RSA and ECDH. As such, while classical KEX mechanisms remain dominant today, their long-term security is fundamentally threatened in a post-quantum threat model. [13, Chapter 3][1]

2.2 Post-Quantum Cryptography

As discussed in the previous section, classical KEX mechanisms such as RSA and ECDH are vulnerable to attacks by quantum computers. In response to this threat, the field of PQC has emerged, aiming to develop cryptographic algorithms that remain secure even in the presence of quantum adversaries. Unlike classical approaches, PQC schemes do not rely on problems such as integer factorization or the discrete logarithm, but are instead based on alternative hardness assumptions, such as those derived from lattice problems, which will be examined later in this thesis. Over the past years, PQC has evolved from a theoretical research area into a field of practical relevance, driven by active standardization efforts [17] and real-world implementations [18] — most notably the NIST standardization process and the resulting FIPS 203 [6] specification. [7, 19]

To identify suitable candidates for post-quantum standardization, the NIST launched a public competition in 2016 [17] that attracted a wide range of proposals based on different hardness assumptions. After several rounds of evaluation, *CRYSTALS-Kyber* emerged as the most promising KEM. Its selection was based on a combination of strong theoretical security guarantees, implementation efficiency, and broad applicability across use cases. These properties make Kyber particularly attractive for integration into existing protocols and performance-sensitive environments — an aspect that is especially relevant for this thesis. [19, 20]

The Kyber algorithm has already been integrated into several test implementations, including those conducted by Cloudflare [10] and the Technical University of Munich [5]. It has frequently been evaluated in hybrid configurations that combine Kyber with classical mechanisms. These studies have demonstrated highly competitive latency characteristics, comparable to elliptic-curve-based schemes. [5, 9]

In this thesis, Kyber768 is used as the post-quantum KEX algorithm under evaluation. This specific configuration of the Kyber family — defined by its target security level and key size — is described in more detail in Section 3.2.4. Its selection is motivated by its standardized status [6], the availability of stable implementations [21], and its practical relevance in current PQC-TLS test environments such as those at TUM [5].

2.3 Hybrid Key Exchange Standardization Approach

Hybrid KEX has evolved from a theoretical research idea into a viable engineering approach and is already being deployed in experimental and early-stage production environments [10]. This section summarizes the current state of a standardization effort that aims to enable secure and interoperable hybrid KEX in practice.

The current focus of standardization efforts is the Internet-Draft *Hybrid Key Exchange in TLS 1.3* [8] by the Internet Engineering Task Force (IETF). The document is in the *Publication Requested* stage within the TLS Working Group, and its planned expiration in July 2025 indicates that broad technical consensus has been reached on its core design.

The draft follows a deliberately minimalist design approach:

- **Negotiation:** Hybrid KEX methods are advertised as paired `NamedGroup` codes within the existing `supported_groups` extension, avoiding the need for an additional TLS-level extension.
- **Data transport:** Public keys and — in the case of KEMs — ciphertexts from both component algorithms are transmitted sequentially within the existing `key_share` payloads. This preserves the one-round-trip handshake model of TLS 1.3.
- **Key derivation:** The shared secrets from both algorithms are concatenated and processed using the standard TLS key schedule. The design relies on the existing HMAC-based Key Derivation Function (HKDF) layers for entropy mixing, avoiding the need for a custom combiner.
- **Scope management:** Decisions regarding algorithm selection, parameter sets, and FIPS compliance are considered out of scope. The draft is limited to defining the wire format and negotiation procedures.

In summary, the draft aims to achieve hybrid KEX that is cryptographically robust and practically deployable. Its primary security goal is to ensure that the resulting shared secret remains secure as long as at least one of the combined KEX mechanisms is unbroken. Beyond cryptographic strength, the draft also emphasizes compatibility with non-hybrid-aware endpoints, low computational and latency overhead, no additional round trips during the handshake, and minimal duplication of transmitted information. [8]

By reusing existing TLS 1.3 message structures without introducing additional message types or protocol extensions, the underlying procedure of this draft minimizes the risk of interoperability issues and and implementation complexity. This design enables a low-risk migration path toward hybrid KEX. [8]

The IETF draft defines the reference wire format that serves as the basis for the hybrid experimental setup described in Chapter 4 and evaluated in Chapter 5.

2.4 Existing Implementations and Experiments

The implementation of classical elliptic-curve-based KEX in this thesis relies on the `PyNaCl` library, which serves as a Python binding to the well-established `libsodium` cryptographic library. Internally, it provides access to the `crypto_box` API of the Networking and Crypthography Library (NaCl) framework, which is designed to offer high-speed, high-security cryptographic operations through a minimalistic interface. [22, 23]

To enable post-quantum KEX within this thesis, the Kyber algorithm is accessed through the `oqs-python` library, a Python wrapper around the C-based `liboqs` framework developed by the Open Quantum Safe project. `liboqs` offers standardized interfaces for a range of quantum-resistant algorithms, including Kyber, and supports runtime selection of parameter sets such as Kyber512, Kyber768, and Kyber1024. In this work, the Kyber768 configuration is used, as it aligns with current NIST recommendations [6]. The library exposes a key encapsulation interface that allows encapsulation and decapsulation operations to be carried out with minimal overhead, making it suitable for hybrid protocol integration and controlled performance evaluations. [21]

A comprehensive performance analysis of PQC in TLS 1.3 was presented at the *CoNEXT Companion '23* conference [5]. The study focuses on practical deployability and evaluates handshake latency, resource usage, and robustness under emulated network conditions. The authors presented a measurement framework for handshake metrics, combining both black-box and white-box methodologies. Black-box measurements capture end-to-end handshake latency and data volume under varying cryptographic configurations, while white-box profiling analyzes computational costs. The experimental setup uses a three-node architecture consisting of a client, a server, and a timestamper. All measurements were performed using a modified version of OpenSSL with post-quantum support, executing repeated TLS 1.3 handshakes.

The evaluation covers a range of algorithmic configurations across three NIST security levels (1, 3, and 5). The study benchmarks both classical schemes (e.g., x25519 and NIST P-curves, i.e., standardized prime-order elliptic curves) and post-quantum candidates such as Kyber [20], HQC [24], and BIKE [25]. Furthermore, hybrid KEX mechanisms are included by pairing P-curves with post-quantum KEMs — e.g. p384_kyber768 — executed in parallel and combined. In addition, constrained network environments were introduced (using `tc netem`) such as high loss (10 %), low bandwidth (1 Mbit/s) or long delay (1 s Round Trip Time (RTT)).

The evaluation results reveal that post-quantum KEX mechanisms — particularly Kyber and HQC — exhibit handshake latencies comparable to, and in some cases even better than, classical elliptic-curve counterparts at the same NIST security level. At NIST level 1, Kyber512 achieved median handshake durations below 2 ms, closely matching x25519. Hybrid key agreements showed no significant performance penalty at lower security levels. On higher levels, latency was mainly influenced by the classical component, effectively making the hybrid as fast — or as slow — as the slower constituent. These findings support hybrid KEX as a viable transitional strategy.

White-box measurements further confirmed the efficiency of Kyber-based handshakes: CPU costs per handshake remained low on both client and server sides. Even in hybrid

configurations, resource usage remained well within practical limits.

Under constrained network conditions, Kyber-based and hybrid handshakes remained resilient. Due to Kyber's relatively compact key and ciphertext sizes, its performance degraded less than that of alternatives such as HQC or BIKE.

Overall the results validate Kyber — both on its own and in hybrid form — as a post-quantum-ready mechanism for TLS 1.3, offering low overhead and strong performance across a wide range of deployment contexts. The authors explicitly recommend the use of hybrid protocols during the transition phase towards PQC. [5]

Another major deployment initiative evaluates hybrid post-quantum KEX in a real-world production setting [10]. In October 2022, Cloudflare — a major content delivery network and internet security provider used by approximately 19.4% of all websites [26] — rolled out a hybrid key agreement combining ECDH (x25519) with Kyber768 as a beta service across its global infrastructure. The feature is enabled by default for all customers, requiring no configuration changes or additional cost, thereby facilitating large-scale compatibility testing under real traffic conditions.

Cloudflare reports several technical challenges encountered during deployment. Protocol ossification, which can be caused by outdated middleboxes or client implementations, may prevent the acceptance of novel key formats or larger payloads. Similarly, the increased size of key material may trigger compatibility issues with legacy systems, especially in constrained environments.

Despite these concerns, initial performance measurements show only marginal increases in handshake latency when using the hybrid x25519_Kyber768 variation. Using Kyber768 only leads to a significant performance improvement compared to the classical mechanism.

Finally, Cloudflare stresses the importance of early deployment — not only to strengthen forward security against quantum threats, but also to surface protocol-level issues early enough for collaborative resolution with browser vendors and client developers. [10]

2.5 Position of this Work

This work contributes a focused, implementation-driven perspective to the study of cryptographic KEX mechanisms. While prior studies — such as the one conducted by TUM [5] or the global deployment initiative by Cloudflare [10] — analyze hybrid and post-quantum mechanisms within the full TLS 1.3 handshake, this work focuses specifically on the KEX component itself. This design decision allows for precise attribution of performance metrics to the underlying cryptographic primitives, free from confounding factors such as certificate verification, session resumption, or TLS handshake overhead.

In contrast to Cloudflare's large-scale deployment of hybrid KEX in real production networks, which mainly focused on compatibility and rollout challenges, this thesis concentrates on understanding the performance characteristics of the cryptographic operations themselves. While Cloudflare tested hybrid handshakes in the real world, this work uses a controlled environment to measure exactly how much time and how many resources various configurations actually consume.

The framework developed in this thesis uses lightweight Python components and relies on trusted cryptographic libraries (`pynacl` [22] and `liboqs` [21]). Additionally, this work follows the current IETF draft for hybrid KEX in TLS 1.3 [8]. The implementation uses the same format and key combination method as the draft, providing a practical check of how this design performs in a controlled setting.

3 Theoretical Background

3.1 Classical Cryptographic Foundations

3.1.1 Fundamentals of Public-Key Cryptography

Public-key cryptography is a fundamental component of modern secure communication and is synonymously used with asymmetric cryptography. Unlike symmetric schemes, which have been in use for over 4000 years, public-key cryptography is relatively new. A first approach was publicly presented in 1976 by Whitfield Diffie and Martin E. Hellman, today known as *Diffie–Hellman key exchange protocol* [27]. In contrast to symmetric schemes, which rely on a shared secret key, public-key cryptography employs a mathematically related key pair: a public key, which may be distributed openly via a public medium, and a private key, which must remain confidential. This concept enables secure key establishment, encryption, and digital signatures without requiring prior shared secrets. As such, it is an essential component of modern protocols like TLS and SSH. [13, Chapter 7][28]

One of the primary applications of public-key cryptography is secure key exchange over an untrusted channel. In this context, public-key mechanisms serve as the initial step for deriving symmetric keys used for subsequent encryption and decryption. Classical KEX protocols rely on the hardness of underlying mathematical problems to ensure confidentiality against adversaries. These number-theoretical problems are considered not to be solvable efficiently on classical computers. *Efficiently* refers to the ability to solve a computational problem within polynomial time relative to the size of the input, making it practically feasible on modern computers. [13, Chapter 7][28]

The most prominent problems used for public-key cryptography are the factorization problem and the discrete logarithm problem (DLP), which underpins widely used schemes such as DH and ECDH. The assumption that the DLP cannot be solved efficiently on classical computers forms the basis of their cryptographic strength. In the following section, the DLP is introduced, including its elliptic-curve variant, which offers improved efficiency and shorter key sizes while maintaining equivalent security levels. [13, Chapter 7 & 8][28, 14]

3.1.2 The Discrete Logarithm Problem and its Elliptic Curve Variant

To understand how the classical KEX used in this thesis operates, it is essential to examine the underlying hardness assumption on which its security is based. The ECDH protocol relies on the assumed intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP). The following section introduces the discrete logarithm problem in its traditional form and then examines its elliptic-curve variant, which underpins modern protocols such as TLS 1.3. This theoretical foundation enables a deeper understanding of the structural vulnerabilities exposed by quantum computing — a topic addressed in subsequent sections. [13, Chapter 8][28]

The DLP is defined over a finite cyclic group G, typically the multiplicative group of integers modulo a prime p, denoted $\mathbb{Z}_p^*$. Let $g \in G$ be a generator of the group, and let $h \in G$ be another element. The DLP is the problem of finding an integer $x \in \mathbb{Z}$ such that

$$g^x \equiv h \pmod{p}. \tag{1}$$

While computing $h = g^x \pmod{p}$ is efficient even for large primes, recovering x from g and h is believed to be computationally infeasible for sufficiently large p. This asymmetry

forms the basis for many classical cryptographic primitives, including the original DH KEX [27]. The security of such schemes relies on the assumption that there exists no algorithm capable of solving the DLP in polynomial time on a classical computer. [13, Chapter 8][28]

A more efficient instantiation of the DLP is the ECDLP: instead of working in multiplicative groups of integers modulo a prime, this variant is defined over the group of points on an elliptic curve.

Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$, and let $P \in E(\mathbb{F}_q)$ be a point of large prime order. Given another point $Q \in E(\mathbb{F}_q)$, the ECDLP asks for the integer $k \in \mathbb{Z}$ such that

$$Q = kP, \tag{2}$$

where kP denotes scalar multiplication of the point P by the scalar k. Similar to the classical DLP, this operation is easy to compute in one direction but believed to be hard to reverse, provided that the elliptic curve and field parameters are properly chosen. [13, Chapter 9]

Compared to traditional DLP-based schemes, elliptic curve groups allow equivalent security levels with significantly shorter key sizes — for instance, a 256-bit elliptic curve offers comparable security to a 3072-bit modulus in classical DH[14]. This efficiency gain has led to the widespread adoption of elliptic-curve-based cryptographic schemes in practice. [13, Chapter 8]

3.1.3 Elliptic Curve Diffie-Hellman

Building on the hardness of the ECDLP introduced in the previous section, the ECDH protocol offers a practical method for establishing a shared secret between two parties over an insecure channel. ECDH translates the abstract hardness assumptions of elliptic curves into a concrete and efficient KEX protocol that is widely used in modern cryptographic systems. [13, Chapter 8]

The core steps of the ECDH protocol are illustrated in Figure 1. The diagrmm shows how two parties independently generate key pairs and exchange public keys to derive a shared secret using the ECDH protocol. The following elements are involved:

- E: An elliptic curve defined over a finite field $\mathbb{F}_q$ that Alice and Bob agree on.
- $P \in E(\mathbb{F}_q)$: A publicly known base point of large prime order that Alice and Bob agree on.
- $\alpha, \beta \in \mathbb{Z}$: Randomly selected scalars chosen by Alice and Bob, respectively.
- $A = \alpha P; B = \beta P$: The corresponding public keys derived by Alice and Bob.
- $K_A = \alpha B = K_B = \beta A$: The shared secret computed by both parties. Due to the algebraic properties of elliptic curves, it holds that $K_A = K_B$.

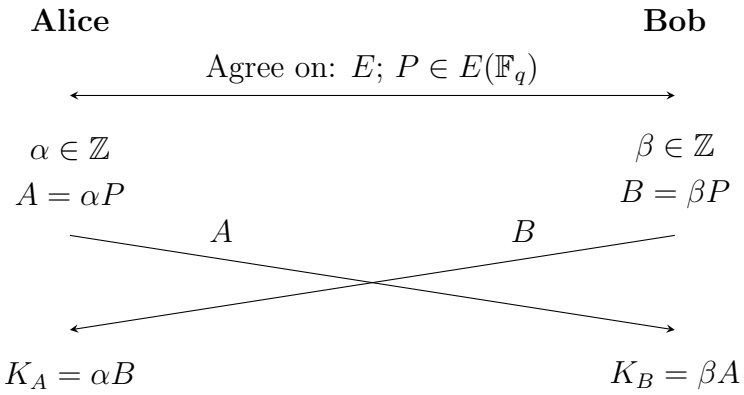


Figure 1: ECDH KEX: key generation, exchange, and shared secret derivation [13, Chapter 8]

The security of the ECDH protocol relies entirely on the computational hardness of the ECDLP. An adversary observing the communication between Alice and Bob can access the public parameters E, P, and the public keys A and B, but not the private scalars α and β. Without the ability to efficiently compute the discrete logarithm on elliptic curves, the adversary is unable to derive the shared secret K — assuming trustworthy parameter selection and resistance to side-channel attacks. [13, Chapter 8]

However, when static key pairs are reused across sessions, ECDH does not inherently provide forward secrecy. To address this limitation, protocols such as TLS 1.3 employ the ephemeral variant, ECDHE, where key pairs are generated for each session. This ensures that compromise of long-term keys does not expose past session data — an essential property in many security applications. [28][13, Chapter 7]

In this thesis, the elliptic curve *x25519* is used as the basis for all experiments, involving a classical algorithm. *x25519* is a specific instantiation of the ECDH protocol defined over the Montgomery curve *Curve25519*, as standardized in *RFC 7748* [29]. It defines a deterministic and well-specified KEX mechanism over the 255-bit prime field $\mathbb{F}_{2^{255}-19}$. It takes as input a scalar and a public point (typically the standard base wpoint with x = 9) and returns a 32-byte shared secret. The protocol abstracts away many of the complexities involved in elliptic curve cryptography, making it suitable for both low-level cryptographic libraries and high-level applications. Its adoption in modern protocols — including TLS 1.3 — reflects its status as a standard for elliptic curve KEX. [13, Chapter 8][29]

3.1.4 Limitations in the Context of Quantum Computing

While classical KEX schemes such as ECDH — particularly its x25519 instantiation — are currently considered secure under traditional computational models, their long-term viability is fundamentally threatened by the progress of quantum computing. Quantum algorithms introduce a new complexity class: bounded-error quantum polynomial-time (BQP), which is lies between the classes P and NP [13, Chapter 3]. Problems in the class BQP are solvable by quantum computers in polynomial time and are therefore considered efficiently coputable in the quantum setting[30]. In particular, Shor's algorithm [1] allows a sufficiently powerful quantum computer to solve both the factorization problem and the DLP in polynomial time.

This directly undermines the security that classical public-key cryptographic algorithms

are based on. As a result, any cryptographic protocol relying on the difficulty of the DLP — including the elliptic curve variant, such as ECDH — is theoretically vulnerable in a quantum setting. Although such quantum computers do not yet exist at a scale required to launch real-world attacks, the implications for future-proof security are severe. [13, Chapter 3 & 8]

One especially relevant concern is the so-called *Harvest Now, Decrypt Later* threat model — sometimes also referred to as *Steal Now, Decrypt Later*. In this scenario, an adversary passively records encrypted traffic today with the intention of decrypting it once a sufficiently large quantum computer becomes available. This poses a particular problem for data with long-term confidentiality requirements, such as diplomatic communications, industrial secrets, or medical records. [4]

In light of these developments, the cryptographic community and standardization bodies — such as NIST — have initiated efforts to develop and deploy PQC. The remainder of this chapter introduces a quantum-resistant alternative, on which the experiments in this thesis are based.

3.2 Post-Quantum Cryptography

3.2.1 Design Principles and Security Goals

PQC aims to provide cryptographic mechanisms that are secure even in the presence of quantum adversaries. Unlike classical public-key cryptography, which relies on the hardness of problems such as factorization or the discrete logarithm, PQC is built on alternative mathematical assumptions that are believed to be resistant to known quantum algorithms. The overarching goal is to ensure long-term confidentiality and authenticity in a post-quantum world, without requiring fundamentally new hardware. Accordingly, PQC schemes are designed to run efficiently on classical computers, while offering robustness against both classical and quantum attacks. [31]

To achieve these goals, post-quantum algorithms are based on hard mathematical problems that are believed to be difficult for both classical computers and quantum computers — such as lattice problems. A key advantage of these problems is that they often allow worst-case to average-case reductions, meaning that breaking a randomly chosen instance implies the ability to solve the hardest instance as well [32].

For KEX, the most common structure is a KEM, which enables the secure transmission of symmetric keys even against active attackers. This means achieving a strong notion of security called IND-CCA, which ensures that attackers cannot distinguish real keys and random ones, even when they can manipulate ciphertexts. [32][13, Chapter 15] At the same time, practical aspects like runtime performance, key and ciphertext sizes, and ease of implementation also play an important role — especially in the context of real-world protocols like TLS. [33]

3.2.2 Hardness Assumptions in Post Quantum Cryptography

This section provides a high-level overview of the main hardness assumptions underpinning PQC, including code-based, multivariate, hash-based, and lattice-based approaches. The last category, encompassing the LWE problem and its variants, is particularly relevant for this thesis and will be examined in more detail in the following section.

Code-based schemes are built on problems from coding theory, particularly on the hardness of decoding linear error-correcting codes. One of the most prominent examples is the McEliece [34] encryption scheme, proposed in 1978, which is still considered secure even in the presence of quantum adversaries. While encryption and decryption in McEliece are computationally efficient, the scheme suffers from very large public key sizes. [35][36, Chapter 4]

Multivariate cryptography is based on the hardness of solving multivariate quadratic equations over finite fields. Schemes like *UOV* [37] achieve fast signing and verification but also suffer from large key sizes and unclear long-term security. While efficient in practice, many multivariate schemes have been broken or lack formal security proofs. [35][36, Chapter 6]

Hash-based cryptographic schemes, for example the Merkle signature scheme (MSS) and its more efficient successor XMSS [38], rely solely on the security of cryptographic hash functions. They offer strong post-quantum resistance and minimal assumptions but require stateful key management due to their use of many one-time keys. While efficient in practice and standardized by the IETF, their main challenges lie in key tracking and parameter selection. [35][36, Chapter 3]

Lattice-based cryptography builds on the idea that certain computational problems become easy when a reduced basis of a lattice is known, but remain hard with a random or unstructured basis. This principle can be used to design public-key schemes by setting the reduced basis as the secret key and an unstructured basis as the public key. Encryption involves adding a small error vector to a lattice point making the closest vector problem hard for attackers but solvable with the secret key. Modern lattice-based constructions rely on problems like the LWE and Shortest Integer Solution (SIS), whose hardness is derived from worst-case lattice problems. These problems allow for strong worst-case to average-case reductions, offering a form of theoretical security that is unique among post-quantum approaches. Although these reductions come at the cost of efficiency, practical schemes such as optimized variants of LWE-based encryption achieve good performance. In addition, lattice-based schemes support advanced functionalities like fully homomorphic encryption, which are not known to be achievable using other cryptographic foundations. [35][36, Chapter 5]

The following section explores LWE problem and its structured variant — called Module Learning with Errors (Module-LWE) — which form the mathematical foundation for Kyber.

3.2.3 The Learning With Error Problem and its Module Variant

The LWE problem constitutes one of the most important hardness assumptions in modern PQC. Intuitively, it asks whether it is possible to recover a secret vector $s \in \mathbb{Z}_q^n$, given a number of linear equations, to which small random errors have been added. More formally, the adversary is given samples of the form $(A, b = A^T s + e) \mod q$, where $A \in \mathbb{Z}_q^{n \times m}$ is a random matrix, $e \in \mathbb{Z}_q^m$ is a noise vector sampled from a certain error distribution, and q is a modulus. Without the error e, the system would be trivially solvable via linear algebra; however, the presence of noise makes it computationally hard to distinguish such instances from random ones or to recover the secret s. This fundamental problem provides the security foundation for numerous post-quantum encryption schemes and KEMs. [39]

To improve the efficiency of LWE-based cryptosystems while maintaining strong security guarantees, the Module-LWE problem was introduced. It generalizes the standard LWE problem by replacing vectors and matrices over integers with elements and tuples in a structured polynomial ring. Module-LWE is defined over the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, where operations are performed on polynomials modulo both a fixed modulus q and a fixed polynomial $X^n + 1$ of degree n. Each problem consists of equations of the form

$$b_i = \langle a_i, s \rangle + e_i \in R_q, \tag{3}$$

where $a_i \in R_q^k$ is a public vector of polynomials, $s \in R_q^k$ is the secret vector and $e_i \in R_q$ is a small random error. The task is to recover the secret s given multiple such equations.

Compared to Ring-LWE, Module-LWE introduces less algebraic structure, which comes with advantages from a security perspective. At the same time, it offers significantly improved efficiency and compactness compared to plain LWE. The Module-LWE assumption balances performance and security and forms the basis of the Kyber KEM used in this thesis. [20]

3.2.4 Key Encapsulation with Kyber

Building upon the Module-LWE assumption introduced in the previous section, the Kyber cryptographic scheme implements a practical and efficient KEM suitable for post-quantum secure communication. As discussed in Section 2.2, it was selected as the standard post-quantum KEM by NIST and formalized in the FIPS 203 specification [6]. Its strong theoretical foundation, efficient design, and broad applicability make it a central reference point in the context of post-quantum secure KEX. In this thesis, Kyber768 is used as the post-quantum algorithm undere evaluation and comparison. [19]

Kyber uses structured polynomial arithmetic over the ring $R_q = \mathbb{Z}_q[x]/(x^n+1)$, where all operations are performed modulo both a prime $q = 7681$ and the cyclotomic polynomial with $n = 256$, resulting in $x^{256} + 1$. This ring structure enables efficient arithmetic operations. Kyber operates on vector polynomials in R_q^k, where the dimension parameter $k \in \{2, 3, 4\}$ determines the security level of the scheme [20, 6]:

	k	Security Level
Kyber512	2	1
Kyber768	3	3
Kyber1024	4	5

Table 1: Overview of Available Kyber Variations [19, 20]

Kyber defines a clear and modular interface consisting of three core algorithms: key generation (`KeyGen`), encapsulation (`Encaps`), and decapsulation (`Decaps`)[1]. Together, these functions implement the complete lifecycle of a KEX mechanism based on the Module-LWE problem.

[1]Descriptions are based on the official Kyber specification submitted to NIST [20].

The key generation process consists of the following steps:

1. **Seed generation:** Two uniformly random 256-bit seeds ρ and σ are generated.
2. **Matrix generation:** Using the seed ρ, the public matrix $A \in R_q^{k \times k}$ is deterministically generated.
3. **Sampling secret and error vectors:** The secret and error vectors $(s, e) \in R_q^k$ are sampled from a centered binomial distribution β_η.
4. **Computing the public key:** The value $t = A \cdot s + e$ is computed and compressed to reduce transmission size.
5. **Output:** The function returns the public key pk — consisting of the compressed t and the seed ρ (for deterministic reconstruction of A) as well as the secret key s.

The encapsulation procedure in Kyber securely generates a ciphertext and a shared secret from a public key. It proceeds as follows:

1. **Message sampling:** A random 256-bit message m is sampled.
2. **Derivation of internal values:** The public key $pk = (t, \rho)$ is hashed together with the message m, and a cryptographic hash-based function G derives two internal values: a preliminary key $\hat{K}$ and a random seed r:
$$(\hat{K}, r) := G(H(pk), m). \tag{4}$$
3. **CPA encryption:** The message m is encrypted under the CPA-secure Kyber encryption scheme using public key pk and seed r, which results in a ciphertext $c = (u, v)$ encapsulating m.
4. **Shared key derivation:** The final shared key K is derived by applying a key derivation function (e.g., SHAKE-256) to the pair $(\hat{K}, H(c))$, ensuring robustness against adaptive chosen-cipher attacks.
5. **Output:** The function provides Ciphertext c and the shared secret K, which can be used as a symmetric key.

On the receiver side, the decapsulation procedure mirrors the logic of encapsulation to reconstruct the same shared secret from the received ciphertext. Using the secret key, the encapsulated message is recovered and used to deterministically recompute the randomness and shared key candidate. The following steps outline the decapsulation process in detail:

1. **Ciphertext parsing:** The ciphertext $c = (u, v)$ is received and split into its components u and v, which represent compressed polynomial vectors.
2. **Reconstruction of the message:** Using the secret key sk, which contains the private vector s, the receiver decrypts the ciphertext under the CPA-secure decryption algorithm to obtain a candidate message m'.
3. **Regeneration of encapsulation randomness:** The decrypted message m' and the hash of the public key are passed through a cryptographic derivation function G to recompute the values $(\hat{K}', r')$. If these values match those generated during encapsulation, the ciphertext is considered valid.

4. **Re-encryption**: As a consistency check, the receiver re-encrypts M' using r' and the public key to produce the ciphertext c'. If c' is equal to the received c, the ciphertext is considered valid.

5. **Shared key derivation:** If the ciphertext is valid, the shared key is computed as $K = H(\hat{K}', H(c))$. If the check fails, a fallback key is used instead, which ensures robustness against chosen-ciphertext attacks.

6. **Output:** The function returns the shared secret K, which — provided that the ciphertext remains unaltered — matches the sender's shared secret.

Figure 2 provides a high-level overview of the key encapsulation process between two parties: Alice and Bob. The diagram illustrates the sequential exchange of public parameters and ciphertexts, resulting in the derivation of a shared secret on both ends. The depicted structure relfexts the typical flow of modern KEMs — such as Kyber.

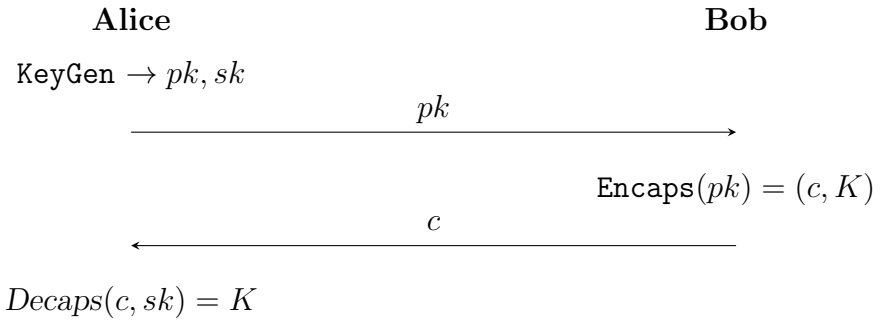


Figure 2: KEM Computation of a Shared Secret. [6]

Compared to classical KEX mechanisms such as x25519, Kyber is based on a hardness problem, that is believed to remain secure even in the presence of quantum computers, thereby offering quantum security by design. Thanks to its modular interface and compatibility with existing cryptographic protocols, Kyber can be integrated with minimal adjustments into frameworks originally built for pre-quantum schemes. This becomes particularly relevant in the context of hybrid KEX mechanisms, which aim to combine classical and post-quantum primitives to ensure both immediate and long-term security. The following section introduces this concept in detail. [6]

3.3 Hybrid Key Exchange Concepts

3.3.1 Motivation for Hybrid Key Exchange

The transition to PQC marks a fundamental shift in the design of secure communication protocols. While post-quantum algorithms offer strong theoretical security against quantum adversaries, they remain relatively new and have not yet undergone the same long-term security and operational deployment as the most classical schemes. On the other hand, classical schemes like x25519 are well-established and widely trusted, but they offer no resistance against quantum attacks — creating a security gap that hybrid KEX aims to close by combining the strengths of both approaches. [8]

Another important motivation is the protection against future cryptanalytic breakthroughs. Neither classical nor post-quantum algorithms can guarantee absolute long-term

security — advances in mathematics or computing could compromise either class. The hybrid approach mitigates this risk by the combination of two independent hardness assumptions. Even if one of the algorithms is broken at some point, the security of the KEX can still rely on the strength of the other. This layered design increases robustness and provides a safety net in case one algorithm turns out to be weaker than expected. [8]

In some deployment scenarios, the use of classical cryptographic algorithms is required by regulation or certification standards — such as FIPS compliance in the United States. This creates a challenge for organizations that want to adopt post-quantum security but are still attached to existing requirements. Hybrid KEX offers a practical solution by allowing classical algorithms to remain part of the protocol while gradually integrating post-quantum components. This parallel use supports compliance and compatibility during the migration phase without sacrificing future readiness. [8]

3.3.2 Conceptual Model

Hybrid KEX mechanisms are structurally characterized by the parallel execution of multiple independent algorithms. Each of them operates autonomously and produces a shared secret. These secrets are subsequently combined — typically through a key derivation function (KDF) — such as the in this thesis implemented function, HKDF — to derive a single session key. The key exchange components are decoupled in design and do not rely on each other during computation or transmission, which allows for flexible integration into existing protocols. This modularity also enables support for multiple different kinds of hybrid configurations (e.g., KEM/KEM, DH/KEM) and ensures that the overall design remains adaptable to evolving cryptographic standards and deployment contexts. The selection of the configuration influences the length of messages, performance and the general complexity.

In this work, the focus is on the DH/KEM configuration. This hybrid model is widely considered the most practical approach for transitional deployments, as it balances compatibility with existing infrastructure and post-quantum security enhancements. The following diagram illustrates the message flow and local computations involved in the DH/KEM hybrid KEX, as specified in the TLS hybrid design draft by the IETF [8]:

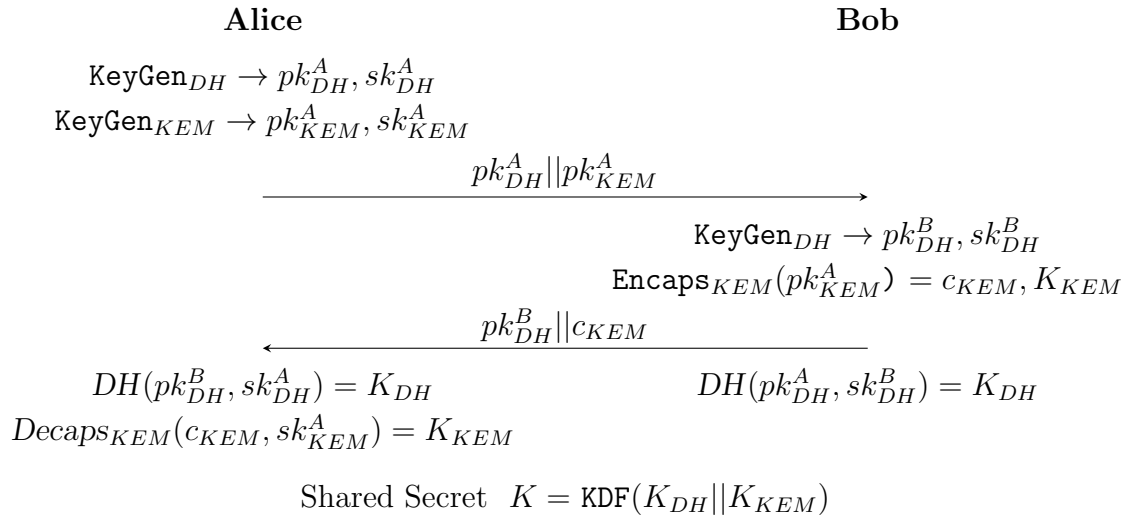


Figure 3: DH/KEM Hybrid Exchange of a Shared Secret[12]

After generating its key material, Alice sends both her classical DH public key pk^A_{DH} and her public KEM key pk^A_{KEM} to Bob. Bob, upon receiving these values generates his own DH key pair (pk^B_{DH}, sk^B_{DH}) and performs two operations independently: he computes the shared secret K_{DH} using his DH private key sk^B_{DH}. independently, he performs KEM encapsulation using Alice's public key pk^A_{KEM}, which provides the ciphertext c_{KEM} and a shared secret K_{KEM}. [8, 20]

Bob sends the ciphertext c_{KEM} together with his own DH public key pk^B_{DH} back to Alice. Alice can now recover the two components of the hybrid exchange: she decapsulates the ciphertext c_{KEM} using her secret key sk^A_{KEM} to recover the shared secret K_{KEM}. Additionally, she computes the DH shared secret by using her secret key sk^A_{DH} and Bob's public key pk^B_{DH}. Due to the properties of the DH-KEX, both parties compute an identical K_{DH}. [8, 20]

Alice and Bob now share the same secrets, consisting of K_{DH} and K_{KEM}. Both values are then concatenated and passed into a KDF to produce a single, cryptographically strong session key. This construction ensures that the resulting key remains secure as long as at least one of the underlying assumptions holds. Moreover, the structure remains fully compatible with the TLS 1.3 handshake through extensions to the `key_share` and `key_exchange` fields. [8, 9]

3.3.3 Security Properties

The primary security objective of hybrid KEX constructions is to ensure that the resulting session key remains cryptographically secure even if one of the underlying algorithms is compromised, thereby extending the overall robustness beyond the guarantees of any individual component. The compositional approach leverages the cryptographic independence of the constituent algorithms to achieve robustness through redundancy. If one of the mechanisms is later found to be insecure, the session key can nevertheless remain confidential. The final key is derived by concatenating both shared secrets, which are then processed by a KDF. It is therefore essential that the KDF ensures the cryptographic key is computationally indistinguishable from a random string. This requirement aligns with the formal notion of a robust KDF, which ensures that the derived key remains unpredictable as long as at least one input maintains sufficient entropy — even if the other input is adversarially controlled. [40, 8]

The security of a hybrid KEX construction builds upon well-established models for its individual components. For the classical DH part, security is typically analyzed within the extended Canetti-Krawcyzk Model (eCK) model, which considers adversaries capable of compromising ephemeral keys and formalizes key indistinguishability and forward secrecy. The post-quantum component used in this thesis is proven secure under the Indistinguishability Under Chosen Ciphertext Attack (IND-CCA) model. This model ensures that an adversary cannot distinguish the shared secret from random, even when given access to a decapsulation oracle. Each of these models provides strong but separate guarantees. The challenge in hybrid settings lies in keeping these properties under composition. [41, 20, 8]

[1] Following the procedure described in [8]

[2] In this Figure, superscripts A and B denote the origin of the key material (Alice or Bob), while the subscripts indicate the cryptographic primitive (DH- or KEM-based).

Forward secrecy is a desirable property in long-lived protocols, ensuring past session keys remain secure even if long-term secrets are compromised [13, Chapter 3]. However, since this thesis evaluates individual key exchange instances in isolation and does not maintain session state, forward secrecy is not applicable in the experimental context.

3.3.4 Limitations and Considerations

Despite their promise, hybrid KEX mechanisms face several conceptual and practical limitations that must be considered in their design and deployment.

First, the theoretical foundations of hybrid constructions are still in an early stage and are not fully explored yet. While initial frameworks and combiners have been proposed, there is currently no broadly established formal model that captures the security of hybrid KEX algorithms against quantum-capable adversaries in a general setting. This limits the ability to derive strong and transferable security guarantees. [9]

Second, the standardization of hybrid KEX is also in an early stage, and needs further exploration. Protocol drafts proposal such as the TLS hybrid design proposal (see [8]) remain under active development, and there is no uniform agreement on implementation practices or interoperability standards across different platforms. [8, 9]

A further practical limitation lies in the performance characteristics of hybrid mechanisms. Since two cryptographic primitives must be executed, the computational overhead depends directly on the choice of algorithms. This variability can affect latency and resource consumption in constrained environments. [8, 9]

Finally, protocol ossification presents an additional challenge. Network infrastructure and middleboxes often enforce rigid expectation of protocol structure and message size. Deviations from familiar formats — such as larger or split messages — can result in handshake failures or compatibility issues, as observed in early experiments conducted by Cloudflare. [10]

4 Implementation

4.1 System Architecture

To enable a controlled evaluation of hybrid KEX, a modular framework was implemented in Python. The primary objective of the architecture is to analyze KEX strategies in isolation, without the overhead and complexity introduced by a full TLS stack. This implementation allows for precise and independent comparisons between classical, post-quantum, and hybrid schemes under reproducible conditions.

By avoiding protocol-level complexity, this system architecture enables targeted measurements highlighting the performance of the different KEX schemes. The controlled environment — including variable network constraints and interchangeable execution modes - ensures that the impact of each KEX strategy can be seen with high granularity.

4.1.1 System Overview

The implementation follows a classical client-server model. Both components communicate over a direct Transmission Control Protocol (TCP) socket connection without additional abstraction layers, ensuring that measurements are not influenced by unrelated protocol overhead.

Two execution modes are supported by the architecture:

- **Native Execution:** Experiments are executed directly on the host machine, serving as a baseline without virtualization overhead.
- **Docker-Based Execution:** Client and server run in separate containers connected via a custom Docker bridge network. This setup enables the use of `tc netem` to emulate a variety of realistic network conditions.

4.1.2 Cryptographic Libraries

The cryptographic operations required for KEX, key derivation and hybrid composition are implemented using the following libraries:

- `pynacl` [22] — for classical ECDH using the `x25519` curve.
- `liboqs` [21] — for post-quantum key encapsulation using the `Kyber768` parameter set.
- `hashlib` [42] and `hmac` [43] — Python's standard libraries for computing the hybrid shared secret via HKDF, in accordance with the recommendations of the IETF draft [8].

4.1.3 Logging and Metrics

During each KEX cycle, performance metrics are recorded on both client and server sides using identical logic. The following validalues are extracted and written to structured `.csv` files:

- **Duration of KEX (`duration_sec`):** The elapsed time is measured via wall-clock timestamps taken immediately before and after the KEX logic. The Python `time.time()` function is used for this purpose, capturing the total duration in seconds from the local perspective of the respective endpoint (client or server) [44]:

- **CPU Usage (`cpu_percent`):** The CPU load of the running process is estimated using the `psutil` [45] library. First, the user and system CPU times are read from `psutil.Process.cpu_times()` before and after the KEX. The difference is divided by the elapsed wall-clock time and normalized over the number of logical CPU cores to compute a per-core usage estimate in percent. [46]

$$\mathrm{CPU}_{\mathrm{percent}} = \frac{(\Delta t_{\mathrm{user}} + \Delta t_{\mathrm{kernel}})}{t_{\mathrm{wall}} \cdot N_{\mathrm{cores}}} \cdot 100 \tag{5}$$

  where:

  - Δt_{user} Difference in user CPU time before and after KEX [45]
  - $\Delta t_{\mathrm{kernel}}$ Difference in system CPU time before and after KEX [45]
  - t_{wall} Wall-clock duration of the KEX
  - N_{cores} Number of logical CPU cores

- **Memory Consumption (`ram_mb`):** Memory usage is determined from the Resident Set Size (RSS) field of `psutil.Process.memory_info()`. This value reflects the actual physical memory occupied by the Python process at the end of the KEX. It is reported in megabytes and includes code, data, and stack memory currently resident in RAM. [45]

Figure 4 illustrates the sequence of measurement phases during a single KEX run. The process is divided into three distinct stages: First, baseline values for CPU time and wall-clock time are recorded. This is followed by the execution of the KEX logic, depending on the selected mode. Upon completion, end timestamps and the resident memory usage are captured. These metrics are collected consistently across all configurations, ensuring a uniform basis for performance evaluation.

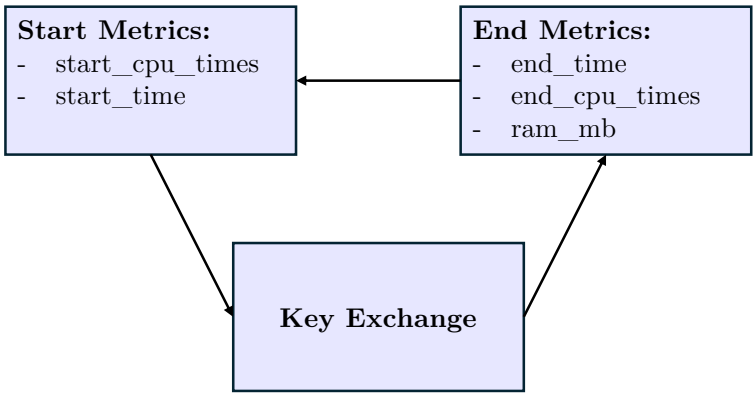


Figure 4: Measurement phases and metrics captured per KEX run

4.2 Key Exchange Strategies

The system supports three distinct KEX strategies — classical, post-quantum (KEM), and hybrid. Each mode is implemented as an encapsulated Python class, exposing a uniform interface for key generation and shared secret computation. This design ensures that different cryptographic mechanisms can be evaluated under identical conditions without modifying the surrounding system architecture.

The classical strategy implements a traditional public-key mechanism. Upon instantiation, a key pair is generated and stored. During the KEX, each peer shares its public key with the other party. A shared secret is then derived from the combination of the received public key and the locally held private key. The class provides two main functions: key generation and shared secret computation. These operations are performed by both parties according to the steps described in Figure 1.

In contrast, the post-quantum class implements the operations required to perform a KEM. It handles key pair generation as well as the encapsulation and decapsulation of shared secrets. One party transmits a public key, and the other returns a ciphertext that encapsulates the shared key material. This structure aligns with the procedure outlined in Figure 2.

The hybrid strategy combines the functionalities of both components within a unified interface. Internally, it manages instances of both modules and invokes them in parallel. After computing individual shared secrets, the results are fused using HKDF to produce the final shared key. The overall process is depicted in Figure 3.

4.3 Network Emulation

To assess whether performance differences emerge under adverse network conditions, the system defines several predefined network conditions. The goal is to enable reproducible execution of real-world scenarios in order to enable direct comparison between KEX strategies under varying levels of network stress. The setup facilitates targeted investigation into the robustness, efficiency, and stability of each KEX variant when exposed to constraints such as delay, packet loss or limited bandwidth.

Network emulation is performed using `tc netem` inside the server-side Docker container. This Linux-based traffic control utility allows the injection of network constraints. The following parameters are used to configure the network profiles, combining `netem` and `tbf` to simulate realistic constraints:

- **Delay:** Introduces artifical latency by adding a fixed delay to each outgoing packet [47] [48].
- **Packet loss:** Simulates an unreliable network by randomly dropping a predefined percentage of packets before being queued [47] [48].
- **Rate:** Limits the average transmission rate [49].
- **Burst:** Defines the maximum amount of data that can temporarily exceed the specified rate limit before shaping is enforced again [49].
- **Latency:** Specifies the maximum amount of time a packet may wait in the token bucket queue before transmission [49].

The various network profiles are designed to represent a range of realistic scenarios in which delay, loss, rate, and buffering parameters are adjusted. The exact parameters are stored in a common configuration file and are activated dynamically depending on the current experiment. Table 2 lists the network profiles used throughout the evaluation, along with their respective parameter setting.

Profile Name	Delay	Loss	Rate	Burst	Latency	Scenario
`netem_mobile`	70 ms [1]	0.5% [2]	12 Mbit [1]	30 kbit [3]	80 ms [3]	Cellular Network (4G, Typical Conditions) [50]
`netem_public_WIFI`	40 ms [4]	1% [4]	2 Mbit [4]	15 kbit [3]	100 ms [3]	Shared Wi-Fi (High Contention, Moderate Interference) [51]
`netem_packet_loss`	–	10%	–	–	–	Simulated Link with High Packet Loss
`netem_satelite`	600 ms [5]	1% [5]	20 Mbit [5]	30 kbit[3]	100 ms[3]	Geostationary Satelite Link [52]

Table 2: Overview of predefined network profiles

By applying these profiles consistently, performance comparisons can be made reliably across all modes and experiments.

4.4 Execution Environments

The framework supports execution in separate environments to isolate protocol performance from deployment-related effects. Both variants rely on the same code base, configuration, and logging format, enabling direct comparison of the recorded metrics.

In the default setup, client and server run inside lightweight Debian-based Python 3.10 containers. A dedicated Docker bridge network isolates all traffic from the host and from other processes. Cryptographic dependencies — PyNaCl [22] for ECDH and the Open Quantum Safe toolchain [18] for post-quantum Kyber — are compiled during the image build, ensuring deterministic library versions across environments.

Since each containers operates their own network namespace, Linux' `tc netem` [47] and `tbf` [49] can be invoked inside the server container without special privileges on the host. This design enables per-experiment injection of delay, loss or rate limiting while keeping the client untouched. All log files are written to a bind-mounted directory so that results needed for the evaluation of the KEX algorithms persist after the containers shut down.

For baseline measurements independent of containerization, the same application can be executed directly on the host system using Python 3.11 within a virtual environment. Client and server communicate via a local TCP socket. Dependency versions are pinned in a requirements file that matches those used in the containerized setup, ensuring library

[1] Based on ERRANT study using 100k speed tests in 4G networks [50].
[2] Derived from Codaval Performance Report (USA avg. 0.5%) [53].
[3] Burst and latency values follow practical guidelines from the Linux `tc-tbf` manual [49] and represent conservative, practical estimates.
[4] Based on analytical thorughput modeling for Wi-Fi under high interference [51].
[5] Based on performance evaluation of QUIC over GEO links [52].

parity. Log files are written to the same directory structure and follow the same naming scheme as their container-based counterparts.

4.5 Automation

A set of Bash scripts automates the complete experimental workflow — covering image rebuilding, container orchestration, configuration management, and structured log collection — ensuring consistent and reproducible execution without manual intervention.

4.5.1 Configuration and Scripts

All run-time parameters are centralized in a single configuration file for each execution environment. The file specifies, among other values, the number of KEX repetitions, the currently active KEX mode, and the selected network profile. Because both server and client import the same JSON configuration file at start-up, experiments remain synchronized and environment variables do not need to be passed into the containers.

Three Bash scripts orchestrate the lifecycle of a complete experiment series:

`rebuild.sh` performs a clean Docker rebuild whenever cryptographic libraries in the base image are modified. To avoid stale artifacts, the script terminates all running containers, removes the intermediate build cache, and reinitializes the client-server setup.

`restart-experiment.sh` stops the client container, launches a new instance, and waits for all repetitions to complete. The server container remains active, preserving its previously configured parameters.

`run_all_enc_modes.sh` serves as the main driver for a measurement batch. The script iterates over the three KEX modes and invokes `restart-experiment.sh`. Once the loop terminates, the logging directory contains six CSV files — client and server logs for each mode.

4.5.2 Determinism and reproducibility

Because the automation layer is limited to plain Bash and a single configuration file, the entire test suite can be executed without additional tooling; no CI pipeline or external task runner is required. As a result, every experiment is:

- Deterministic: By minimizing manual interaction, the automation layer ensures that each experiment is executed under consistent conditions with reduced risk of operator-induced variation.
- Stateless: Containers are recreated or restarted for every mode, ensuring that no residual key material or network state can leak into the next run.
- Portable: Except for Docker and Bash, no host-side dependencies are required and the native execution script relies solely on the system's Python interpreter.

This minimal automation concept guarantees that the performance measurements presented in the following chapter can be reproduced on demand and without manual intervention on any machine that satisfies Docker's hardware requirements.

5 Experiments and Evaluation

This section quantifies the practical impact of the three implemented KEX modes - Classic (ECDH x25519), PQC (Kyber768) and their Hybrid combination under conditions that approximate real-world deployment.

5.1 Objectives

The Objectives of this evaluation aim to answer the research questions posed in Section 1.4 by examining the protocol along three orthogonal axes. Each axis is mapped to a concrete objective that will be revisited in the subsequent subsections on *Methology*, *Mode Comparison*, *Network-Profile Impact*, and the final *Robusness Summary*.

Latency efficiency Determine how the three KEX modes influence *KEX duration*. The focus lies on end-to-end time as perceived by the client, because this value directly affects user-visible connection set-up.

Resource footprint Quantify the additional *CPU utilisation* and *memory consumption* introduced by post-quantum and hybrid primitives, relative to the classical baseline. This objective targets deployment on resource-constrained systems such as mobile or embedded devices.

Resilience to network impairments Evaluate whether the observed performance scales gracefully when realistic link degradations-latency, loss and bandwidth limits are emulated. The goal is to identify any dispropartionate degradation that would hinder adoption in wide-area or wireless environments.

5.2 Experiments

This section describes how the three objectives from Section 5.1 are translated into concrete repeatable experiments. it summarises the hardware platform, defines the experimental matrix and outlines the data-collection workflow that feeds the later evaluation subsections.

Hardware baseline. All measurements were obtained on a single workstation to avoid cross-machine variance:

- **Device:** MacBook Pro (Model Mac16,8), Apple M4 Pro, 12-core CPU
- **Memory:** 24 GB unified memory
- **OS:** macOS 15.4

Table 3 lists every experiment that is evaluated in Chapter 5. Each experiment consists of 1.000 consecutive reduced handshake cycles per KEX mode and execution environment.

ID	Network Profile	Docker Setup	Native Setup
experiment_01	✗	✓	✓
experiment_02	netem_mobile[1]	✓	✗
experiment_03	netem_public_WIFI[1]	✓	✗
experiment_04	netem_package_loss[1]	✓	✗
experiment_05	netem_satelite[1]	✓	✗

Table 3: Overview of experiments their execution and setup

The first experiment, free of any network constraints, is performed and is executed on both the native setup and the Docker-based environment. It is therefore suitable to address several foundational aspects of the evaluation:

First, it enables an analysis of potential discrepancies between KEX durations recorded independently by the client and the server. Since both peer log their own timing, comparing these values helps determine whether network latency is accurately reflected in the measurements, whether consistent offsets are introduced, and whether such differences vary across KEX modes. If the results reveal a stable shift between client- and server-side durations, the evaluation may be simplified by focussing exclusively on one side, reducing analytical complexity without loss of accuracy.

Second, the experiment serves as a baseline comparison between the two execution environments. It reveals whether the Docker-based setup introduces any measurable overhead in KEX duration or system resource usage, thereby validating the reliability of containerised measurements for later experiments. This baseline comparison is conducted not only for reduced handshake time, but also includes CPU and RAM usage across modes.

Additionally, the experiment supports a direct comparison of system resource consumption between the three KEX modes under ideal conditions. By isolating cryptographic performance from external variables, it becomes possible to identify differences in computational cost that are solely attributable to the cryptographic primitives.

Experiment 02 to 05 extend the evaluation by introducing network constraints as outlined in Table 2. Each experiment corresponds to one specific emulation profile inside the Docker-based server container. Experiment 02 uses the `netem_mobile` profile to replicate typical conditions of a 4G connection; Experiment 03 applies the `netem_public_wifi` profile to simulate a congested shared wireless environment; Experiment focuses solely on high packet loss without additional delay or rate shaping; and Experiment 05 emulates a geostationary satellite link with substantial round-trip latency.

The objective of these experiments is to analyse how each KEX mode performs under degraded network conditions that introduce latency, bandwidth limitations, and random packet loss. By observing both KEX duration and resource consumption across these profiles, it becomes possible to assess the robustness and efficiency of each approach in less-than-ideal environments. These results are evaluated and compared against the clean baseline established in Experiment 01. This comparative analysis highlights whether certain modes degrade more gracefully than others and helps identify potential performance bottle necks or vulnerabilities that may only emerge under real-world constraints.

[1]Network profile parameters are defined in Table 2.

5.3 Measurement Perspective and Execution Environment

The validity of performance evaluations depends not only on what is measured, but also on how and where measurements are taken. This section explores two orthogonal factors that can influence recorded results: logging perspective (i.e., whether metrics are collected on the client or server side) and the execution environments (native or Docker-based execution). Both dimensions may introduce systematic biases that must be understood before interpreting the performance of different KEX modes. The subsequent subsections analyze these effects using consistent metrics and controlled experiments to determine whether either factor significantly affects the comparibility or representativeness of the measurements.

5.3.1 Logging Perspective Evaluation

As mentioned in Section 4.1.3 during each KEX cycle, both the client and server indepentently log their performance metrics (duration, CPU usage and memory consumption). The objetive of this analysis is to assess whether logging side significantly affects the recorded values and if one perspective yields more stable or representative results. Figure 5 compares the KEX durations logged by each endpoint. The overall distributions are similar in spread and shape, indicating consistent measurement behaviour across both sides. However, the client-side durations are systematically higher. This offset likely results from the client's inclusion of connection setup and the inherent delay until the first response is received. These client-side timestamps begin before any cryptographic computation occurs on the server and end after the final transmission has beed sent, thus containing the duration of the entire KEX.

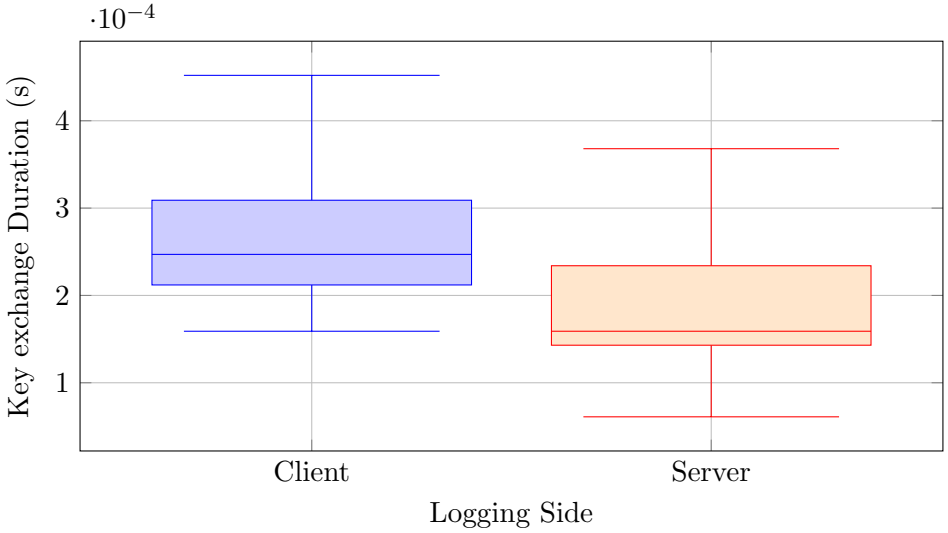
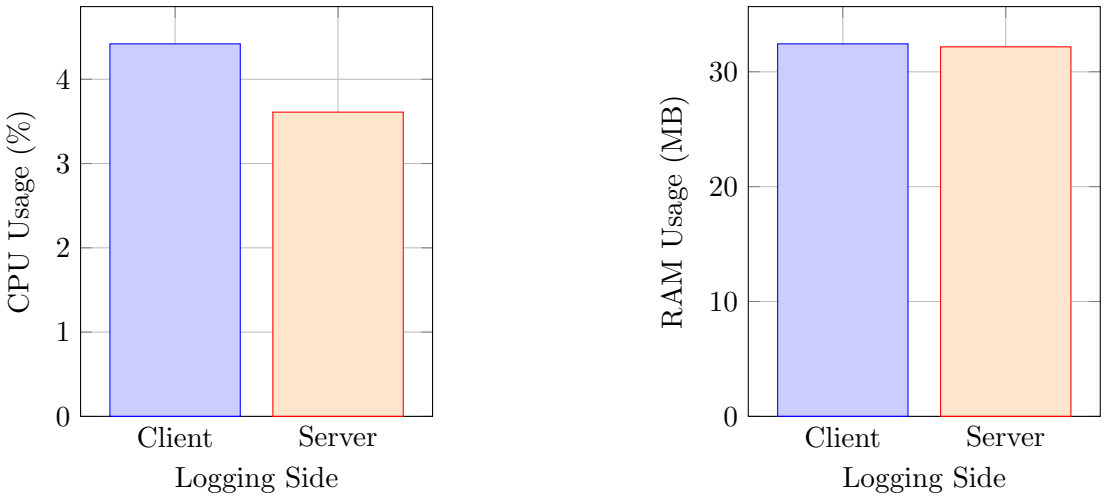


Figure 5: Key exchange duration per logging side

Figures 6a and 6b compare mean CPU and RAM usage per logging side. The client sided CPU usage is slightly higher on average, which aligns with its additional responsibilities during session initiation. RAM usage is virtually identical on both sides, suggesting symmetric memory allocation behaviour across the reduced KEX logic.



(a) Mean CPU usage per logging side

(b) Mean RAM usage per logging side

Figure 6: System resource usage per logging side

Both perspectives exhibit comparable consistency and similar variability, indicating that neither side introduces significant measurement noise. Consequently, either logging side could be used for further evaluation. However, to maintain analytical clarity and avoid asymmetric timing artifacts introduced by split perspectives, all subsequent analysis in this chapter focusses exclusively on the client-side metrics.

5.3.2 Performance Across Environments

To assess whether the execution environment affects protocol performance, this section compares KEX measurements between native and Docker-based deployments. All values are collected from the client perspective, as motivated in Section 5.3.1.

Figure 7 shows the measured durations per KEX cycle in both environments. The distributions are similar in shape with Docker showing higher variance. The medians are nearly identical, suggesting that containerization does not introduce measureable overhead in terms of latency. This finding indicates that the Dockerized client is able to perform cryptographic operations and network interactions with comparable efficiency to its native counterpart.

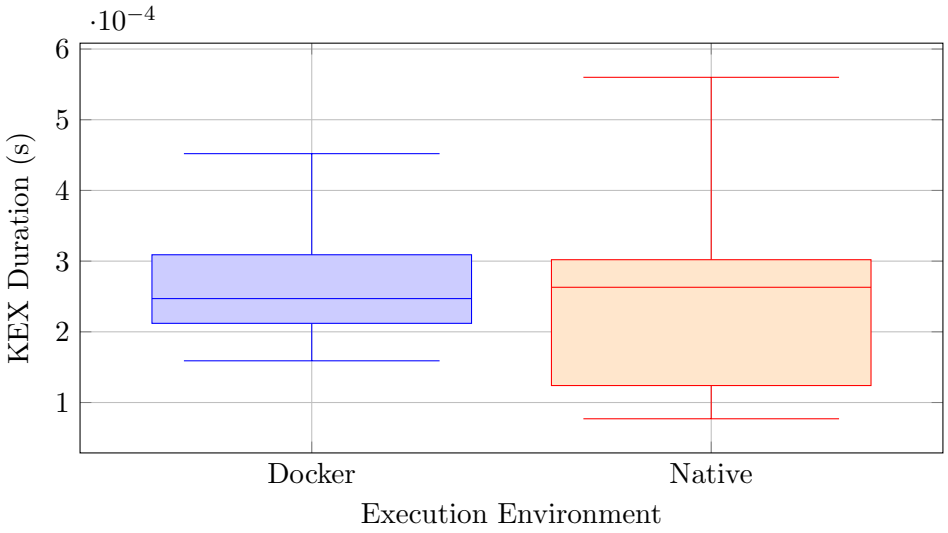
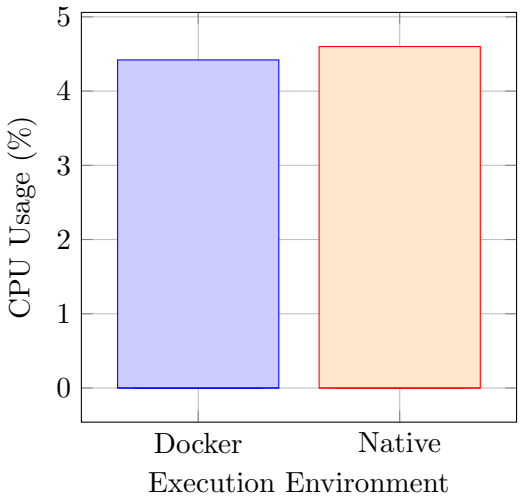


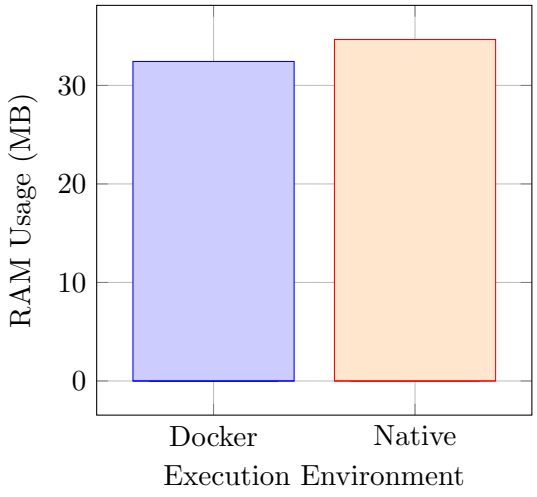
Figure 7: KEX duration by Execution Environment (Client)

CPU usage, illustrated in Figure8a, is marginally higher in the native setting. This slight increase may result from reduced process isolaation or scheduling behavior in the container runtime. However, the fidderence remains within the bounds of natural system variability and does not indicate any substantial computational disadvantage of caintainer-based execution.

Similarly, RAM usage (Figure 8b) is minimally higher in the native environment. This discrepancy is negligible and may stem from variations in memory allocation strategies or concurrent background processes out side the controlled KEX logic.



(a) CPU usage per Execution Environment



(b) RAM usage per Execution Environment

Figure 8: System resource usage per Execution Environment: (a) CPU and (b) RAM. (Client)

In summary, the Docker-based setup incurs no measurable penalties in terms of latency, CPU, or memory usage. These results validate the use of containerized infrastructure

for reproducible benchmarking in this work. Docker provides a consistent and sufficiently lightweight environment for evaluating for cryptographic performance.

5.4 Mode Comparison

Test1234

5.4.1 Key Exchange Duration Across Modes

This section compares the runtime performance of the three implemented KEX variants - Classic (ECDH - x25519), PQC (Kyber768), and the Hybrid composition of both - based on client-side timing data captured in both Docker and native environments.

Figure 9 presents the distribution of KEX durations in a containerized (Docker) setting. Both x25519 and Kyber768 exhibit compact and comparable timing profiles. The PQC variant shows a slightly higher median duration that the classical baseline but remains within the same general performance band. In contrast, the hybrid mode demonstrates a increase in duration, with a higher median and an wider spread, reflecting the computational overhead of performing both KEX sequentially and combining the resulting secrets.

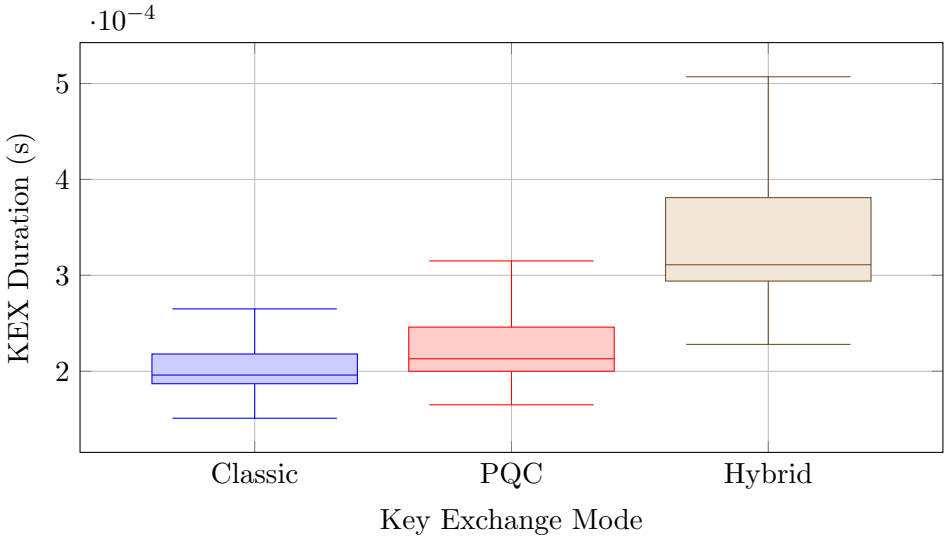


Figure 9: KEX duration per key exchange mode (Docker, Client)

Figure 10 shows the same measurements executed natively on the host system. Here, Kyber768 outperforms x25519 in terms of latency, with a significantly lower median an tighter distribution. This suggests that the implementation of Kyber768 interacts more efficiently with the host system's CPU and memory hierarchay than the curve-based x25519 operations. The hybrid mode again incurs the highest cost, both in terms of median duration and variance.

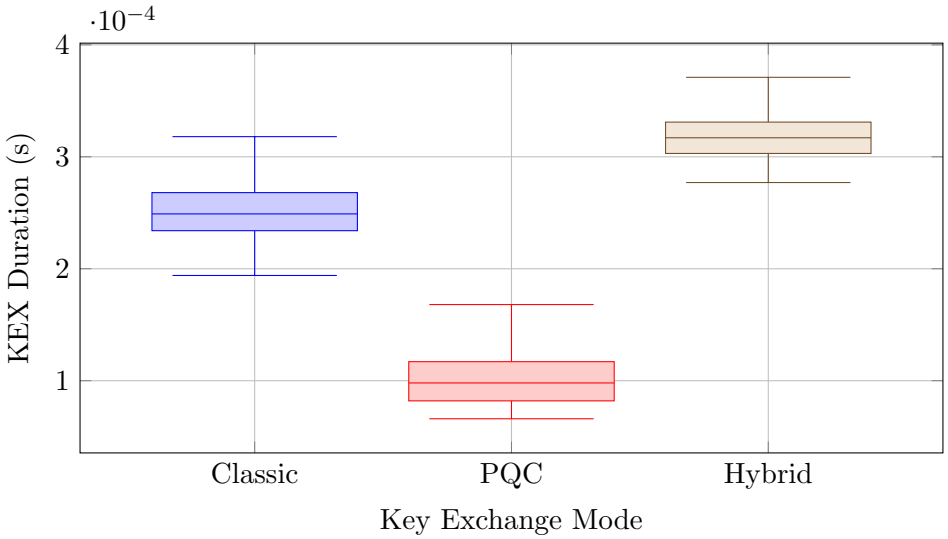


Figure 10: KEX duration per key exchange mode (Native, Client)

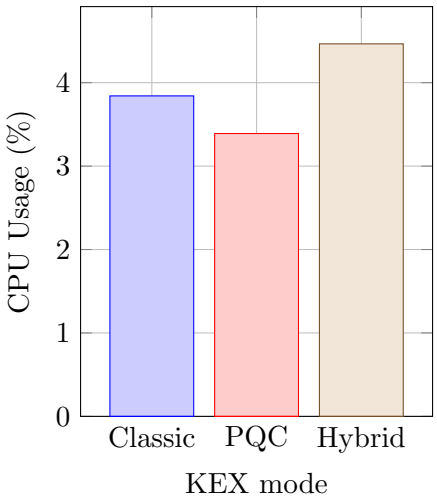
Across both environments, a consistent performance hierarchy emerges. In the Docker environment, the x25519 and Kyber768 variants exhibit comparable latency characteristics, while the hybrid mode incurs a clearly higher computational cost. In contrast, native executions reveals a more differentiated picture: Kyber768 significantly outperforms x25519 in terms of median KEX duration. In both settings, however, the hybrid mode introduces the highest overhead due to the sequential execution and combination of classical and post-quantum primitives.

5.4.2 System Resource Utilisation Across Modes

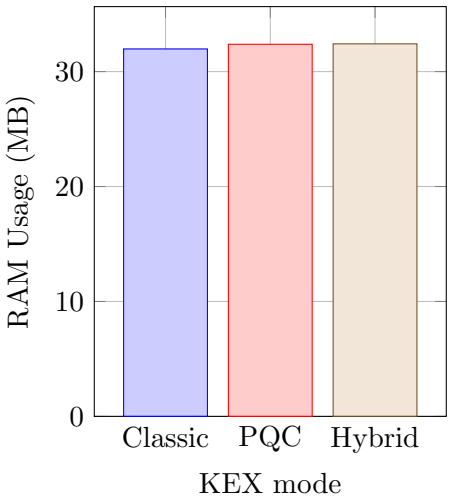
Figures 11 and 12 present the mean CPU and RAM utilisation for each KEX mode, measured from the client's perspective under both Docker-based and native execution environments.

In the Docker setup (Figure 11), CPU usage varies across modes, with the hybrid configuration exhibitng the highest mean load at 4.5%, followed by the classic x25519-based mode at 3.8%, and the Kyber768-based post-quantum mode at 3.4%. This increase in the hybrid case is expected, as it involves both classical and post-quantum operations being executed within a single exchange cycle.

RAM consumption remains more stable across modes in Docker, with classic consuming an average of 32.0 MB, and both PQC and hybrid modes showing slightly higher but identlical values of 32.4 MB. The minor variation suggests that memory allocation remains largely unaffacted by the cryptographic backend withing containerized environments.



(a) Mean CPU usage per KEX mode

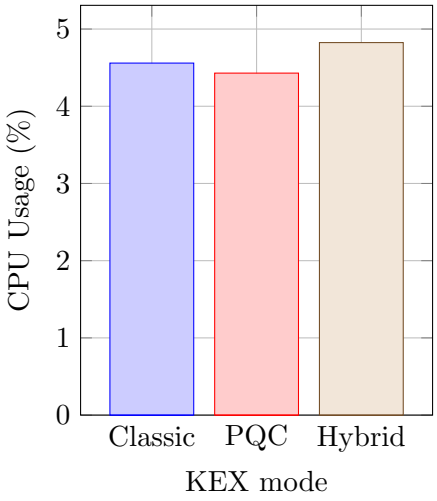


(b) Mean RAM usage per KEX mode

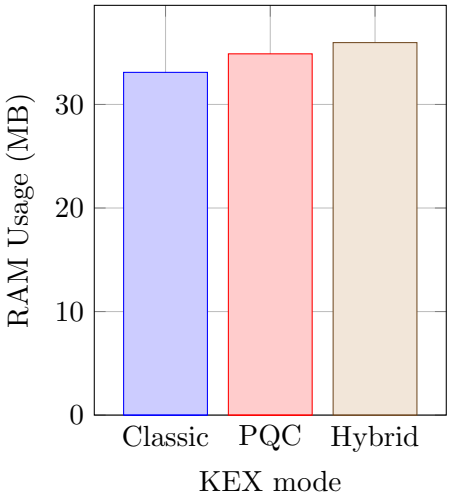
Figure 11: System resource usage per KEX mode (Docker, Client)

Under native execution (Figure 12), the CPU usage distribtion retains the same order as in Docker: hybrid (4.8%) > classic (4.6%) > PQC (4.4%). However, the differences are smaller, indicating more constant resource utilisation outside the container context across the KEX modes.

Considering the memory consumption on native systems, the observed increase across modes may reflect an increase in cryptographic complexity. The classic mode averages 33.1 MB, PQC rises to 34.9 MB, and the hybrid reaches 36.0MB. This progression reflect the additional memory overhead introduced by Kyber 768 and the compounded requirements of hybrid operations. (?????????)



(a) Mean CPU usage per KEX mode



(b) Mean RAM usage per KEX mode

Figure 12: System resource usage per KEX mode (Native, Client)

5.4.3 Summary of Cryptographic Cost Differences

The results presented in Sections 5.4.1 and 5.4.1 enable a comparative summary of the resource and timing characteristics across the evaluated KEX variants. Classic ECDH(x25519) serves as the baseline, with Kyber768 and its hybrid combination representing the post-quantum and transitional configurations.

Across both execution environments, the hybrid KEX consistently exhibits the highest computational cost in terms of duration, CPU usage, and memory consumption. This result is expected, as the hybrid mode involves executing both x25519 and Kyber 768 operations sequentially and combining their shared secrets. The accumulated resource requirements result in increased latency and a higher runtime footprint.

In contrast, Kyber768 demonstrates competitive performance, especially in native execution. Its runtime efficiency surpasses x25519 under host-native conditions, while keeping comparable or slightly lower CPU utilisation. Memory consumption is moderately higher than for the classical mode.

Overall, the cryptographic cost differences are most pronounced in hybrid operation, while Kyber768 shows favorable characteristics that support its viability as a post-quantum alternative to classical elliptic-curve schemes. The result suggest that Kyber768 can be integrated without reasonable performance penalties, and that the transition to hybrid or PQC-ready deployments will require trade-offs primarily in CPU and on strictly memory constrained systems.

5.5 Network Profile Impact

This section examines how varying network conditions influence the performance of different KEX modes. The analysis focuses - similarly to Section 5.4 - on latency, CPU, and memory usage across five predefined network profiles (see Figure 2), inlcuding realisitic scenarios such as mobile and satelite links.

All measurements were collected on the client side in a Docker-based environment, ensuring consistent runtime isolation while reflecting practical deployment conditions in virtualized containers. The data originates from Experiment 01 (baseline, no network constraints) and Experiment 02-05, each applying a specific network profile. For a structured overview of the experiment configurations, refer to Table 3, as discussed in Section 5.2.

5.5.1 Key Exchange Duration Across Network Profiles

Figure 13 shows the KEX duration across the four network profiles compared to the baseline scenario without any network constraints disaggregated by cryptographic mode. While absolute latencies naturally vary between profiles this analyis does not aim to contrast the network conditions themselves. Instead, the foucs lies on validating whether cryptographic performance differences between modes remain stable under diverse network configurations (see Figure 2).

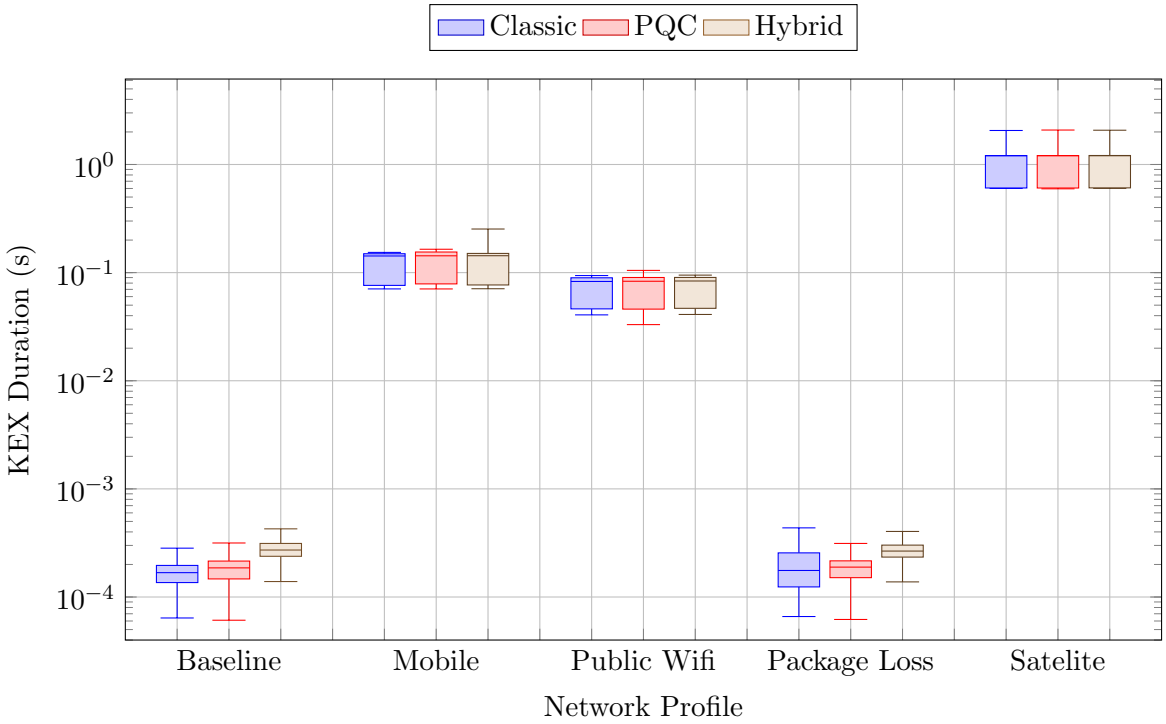


Figure 13: KEX duration per network profile and key exchange mode

The baseline experiment (Figure 9) serves as a reference point, measured without additional network constraints. Here, PQC (Kyber768) exhibits nearly identical performance differences to the classical baseline (x25519), while the hybrid mode incurs the highest delay. This pattern is consistently reflected across al netem-configurated profiles.

For instance, under both Mobile and Public WiFi conditions, the PQC mode remains closely aligned with or marginally above the Classic basline. The Hybrid variant consistently shows elevated durations - demonstrating that its overhead originates from the dual cryptographic computations rather than network sensitivity. Even in high-latency or lossy environments (e.g. Satelite or Package Loss), the relative ordering of Classic, PQC, and Hybrid remains intact.

This consistency across profiles strengthens the interpretation that KEX mode is the dominant facrot in performance variation, not the surrounding network.

5.5.2 Resource Consumption Across Network Profiles

Figures 14 and 15 illustrate the mean CPU and memory consumption for each KEX mode across the five evaluated network profiles. This analysis aims to determine whether different network conditions impose an additional burden on cryptographic resource requirements.

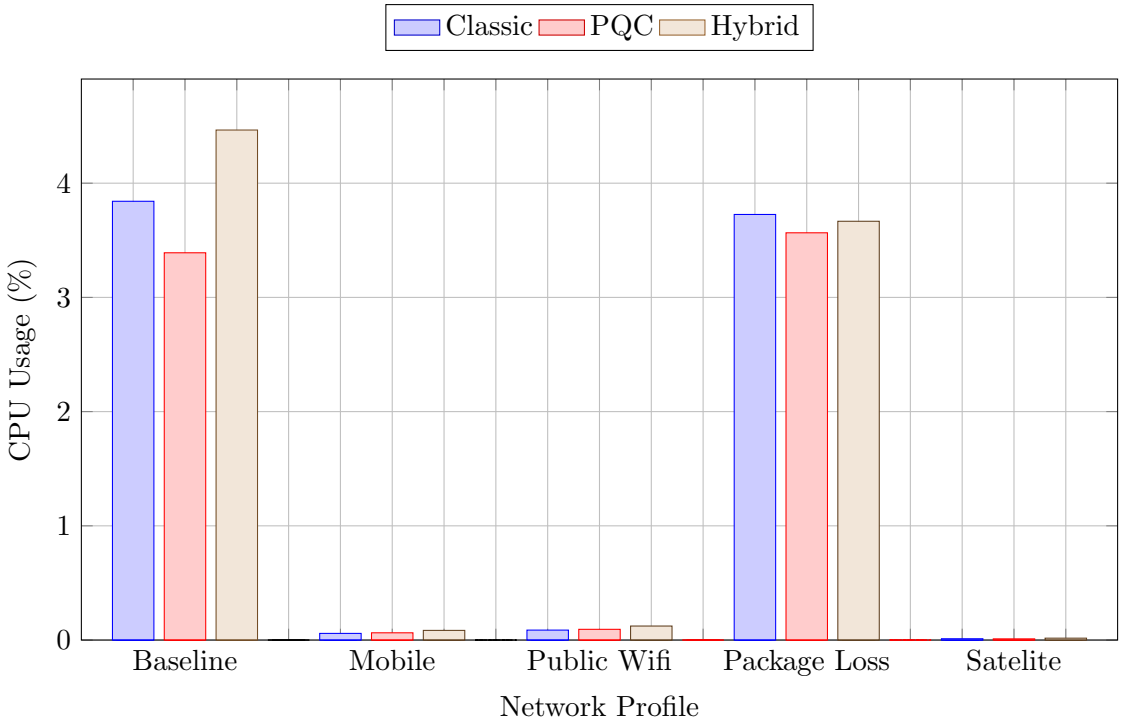


Figure 14: Mean CPU usage per KEX mode and network profile

In terms of CPU usage (Figure 5), the results reveal a clear pattern. Under fast network conditions, such as the *baseline* and *package loss* profiles, cryptographic computations dominate the runtime, leading to relatively high CPU utilisation - ranging from approxiamately 3.4 to 4.5%. In contrast, profiles that introduce greater latency, such as *mobile*, *public WiFi*, and especially *satelite*, yield significantly lower CPU usage. This drop occurs because the system spends a substantial protion of the KEX duration waiting for network responses, resulting in idle or Input/Output (I/O)-bound states where CPU demand is minimal.

This observation aligns with the CPU usage metric (`cpu_percent`) defined in Equation 5: while the numerator - representing actual CPU time in user and kernel space - remains largely constant across profiles, the wall-clock duration t_{wall} increases under high-latency conditions. Since t_{wall} appears in the denominator, resulting CPU percentage decreases accordingly.

A consistent ranking among the three KEX modes is preserved across all network settings: the hybrid variant shows the highest CPU usage, followed by the classical x25519-based mode, with Kyber768-based PQC mode exhibiting the lowes CPU load. This is in line with the computational design of each mode and confirms that these relative costs remain stable under varying network conditions.

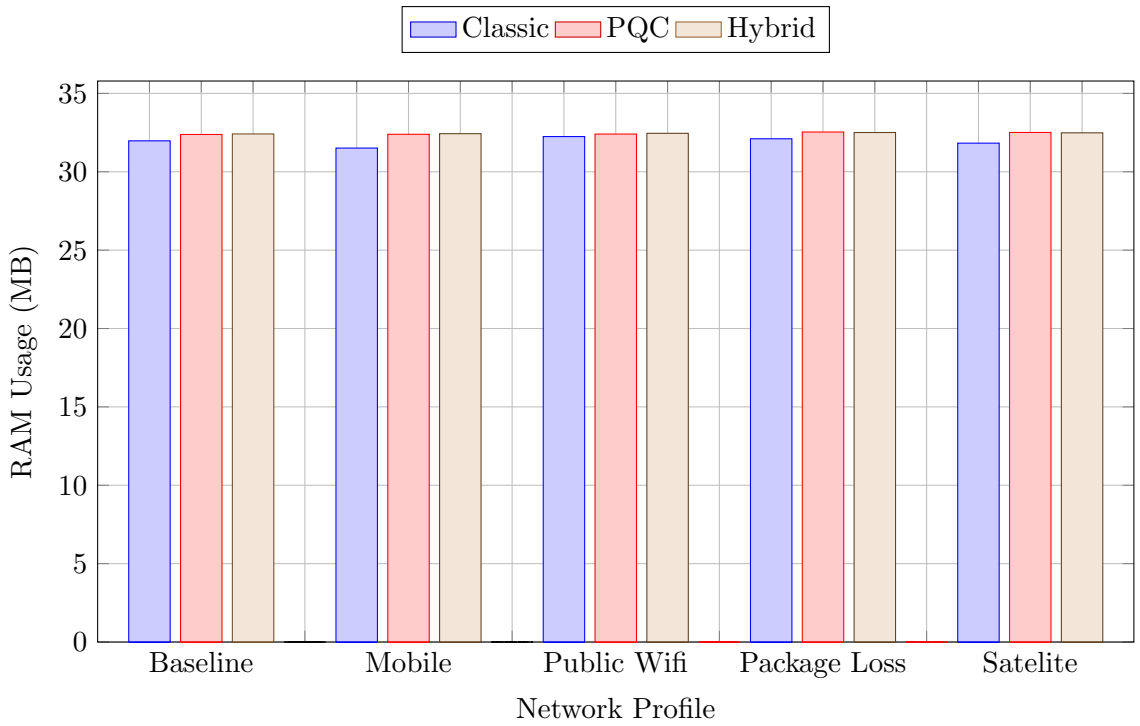


Figure 15: Mean RAM usage per KEX mode and network profile

Memory usage (Figure 15) appears largely unaffacted by network quality. Across all network profiles, RAM consumption is stable within a narrow range. As observed in previous sections, PQC and hybrid modes require marginally more memory compared to the classical implementation. However, differences between the network profiles themselves are minimal ($<$ 1MB), suggesting that memory allocation during the KEX phase is primarily governed by the cryptographic backend rather than external transmission delays or losses.

In summary, resource usage patterns are robust across all evaluated network scenarios. While CPU ustilisation may drop under high-latency conditions due to increased idle time, the relative computational cost of each KEX mode remains unchanged. RAM consumtion, meanwhile, is effectively invariant. These findings reinforce the conclusion that post-quantum and hybrid KEX designs are resilient to adverse network environments in terms of system resource demands.

5.6 Robustness Summary

The results presented in the previous sections demonstrate the robustness and resilience of all evaluated KEX modes under varying network conditions. Across all emulated profiles - including high-latency, low-bandwidth, and high-loss scenarios - the protocols remained stable and completed successfully without failures or critical outliers. No mode exhibited runtime instabilities or operational breakdowns, underscoring their baselne sutability for deployment across a range of environments.

Importantly, the relative performance characteristics between the modes were preserved. Both in clean and impaired conditions, the ordering of runtime performance remained consistent:classical x25519 performed fastest, Kyber768 followed closely, and the hybrid approach exhibited the highest latency. This consistency suggest that each mode scales proportionally

with the quality of the network, and that post-quantum and hybrid schemes do not degrade disproportionately under stress.

CPU and memory consumption also remained stable. In particular, RAM usage showed minimal variation across both network profiles and KEX modes. As discussed in Section 5.5.2, mean CPU utilisation decreased in high-latency condition due to increased I/O wait times. This behaviour affected all modes similarly and does not indicate ineffiency of the xryptographic primitives themselves.

Taken together, the results confirm that all three KEX modes - including the post-quantum and hybrid approach - remain performant, consistent, and operational under real-world network constraints. Their robustness, combined with predictable scaling behaviour, makes them viable candidates for deployment in latency-sesnitive, bandwidth-constrained, or mobile environments.

6 Conclusions

This thesis investigated a hybrid KEX protocol that combines classical elliptic curve cryptography (ECDH using x25519) with a post-quantum secure mechanism (Kyber768). The primary objective was to analyze the practical performance characteristics of hybrid KEX schemes in comparison to their classical and post-quantum counterparts.

In order to achive this objective, a modular prototype was implemented in Python, enabling isolated execution and measurement of KEX operations without the complexity of a full TLS stack. The architecture follows a client-server model based on raw TCP sockets and supports flexible switching between execution environments (Docker and native). Furthermore, realistic network conditions such as latency, bandwidth limitations, and packet loss were emulated using *tc netem* within containerized setups.

Three KEX modes - classical (ECDH), post-quantum (Kyber768), and hybrid (ECDH + Kyber768) - were systematically evaluated across six distict network scenarios. Each configuration was tested with 1,000 reduced handshakes iterations, during which metrics including KEX duration, CPU utilization, and RAM usage were recorded independently on both client and server sides.

The collected data provide a comprehensive basis for assessing performance, resource efficiency, and robustness of hybrid KEX protocols under emulated real-world deployment conditions. The findings contribute to a deeper understanding of the practial trade-offs involved in adopting hybrid and PQC cryptographic schemes in communication systems.

The experimental evaluation yielded several key insights regarding the performance and behavior of the examined KEX modes across various conditions: Across both execution environments, the classical x25519 and post-quantum Kyber768 modes demonstrated similarly low KEX latencies under ideal conditions. In native execution, Kyber768 even outperformed the classical mode, suggesting high runtime efficiency of lattice-based encapsulation on modern hardware. The hybrid mode, as expected, introduced additional latency due to the sequential execution and combination of two cryptographic methods, although the increase remained within acceptable bounds for most deployment scenarios. Considering the CPU usage, the hybrid mode consistently exhibited the highest load, followed by the classical and post-quantum variants. However, the differences were minor in absolute terms and did not suggest much higher computational overhead. RAM consumption remained largely stable across al modes and network profiles, with the classical baseline showing slightly reduced memory usage compared to the quantum-attack resistent choices. Under constrained network conditions - including high latency, bandwidth throttling, and packet loss - all three KEX modes remained operationally stable. The relative performance ordering (x25519 $\approx$ Kyber768 $<$ hybrid) was preserved consistently across all emulated profiles. Neither Kyber768 nor the hybrid mode exhibited disproportionate decline highlighting their robustness and suitability for deployment in adverse or mobile environments.

The research questions formulated in the introductory section of this thesis aimed to guide the evaluation of hybrid KEX protocols with regard to their practical viability. Based on the empirical findings the following ansers can be derived.

First, with respect to the comparative performance of hybrid KEX mechanisms, the results confirm that the hybrid schmeme provides a measureable, but manageable, performance ovoerhead. In terms of KEX latency, the hybrid mode exhibited longer durations

than its standalone counterparts, reflecting the additional comutational cost of executing two key exchanges and combining their outputs using a hash function. Similarly, CPU usage was highest in the hybrid configuration. However, these increases were modest and remained within operationally acceptable limits. Memory consumption, in contrast, showed no significant disadvantage. In some configurations, the hybrid and post-quantum modes even required slightly less RAM than the classical baseline, suggesting favorable memory characteristics of the Kyber implementation.

Second, the trade-offs associated with hybrid KEX protocols primarly concern performance versus security. While hybrid schemes introduce higher computational xomplexity, they offer the combined benefits of forward secrecy (through ECDH ??????) and quantum resistance (through Kyber768). This dual-layered security model ensures that the resulting session key remains secure even if one of the underlying algorithms were to be broken in the future. Moreover, hybrid approaches support graceful migration paths, allowing compatibility with existing infrastructure while extending cryptographic assurances into the post-quantum era. This makes them particulary as transitional mechanisms in scenarios where long-term data confidentiality must be guaranteed today.

Third, the impact of network conditions on protocol robustness was systematically examined through a range of emulated scenarios, including high-latency satelite links, mobile networks with packet loss, and constraint wireless environments. Across all tested profiles, the KEX protocols remained operationally stable. The duration of the simulated reduced handshakes scaled proportionally with the severity of network constraints, but the relative performance differences between modes remained consistent. Notably, no mode exhibited disproportionate degradation under adverse conditions. This observation supports the conclusion that both hybrid and post-quantum KEX schemes are robust and reliable even in challenging real-world network environments.

Taken together, these results demonstrate that hybrid KEX protocols can offer a compelling balance between enhanced security and practical deployability. Their moderate resource requirements, consistent performance, and resilience under network stress osition them as viable candidates for adoption in latency-sensitive or resource-constrained systems preparing for the post-quantum transition.

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[2] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[3] H. Y. Wong, *Shor's Algorithm*, pp. 289–298. Cham: Springer International Publishing, 2024.

[4] M. Barenkamp, "Steal now, decrypt later," *Informatik Spektrum*, vol. 45, no. 6, pp. 349–355, 2022.

[5] M. Sosnowski, F. Wiedner, E. Hauser, L. Steger, D. Schoinianakis, S. Gallenmüller, and G. Carle, "The performance of post-quantum tls 1.3," in *Proc. International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, (New York, NY, USA), Association for Computing Machinery, 2023.

[6] National Institute of Standards and Technology, "Fips 203: Module-lattice-based key-encapsulation mechanism standard." `https://csrc.nist.gov/pubs/fips/203/final`, 2024. Accessed: May 17, 2025.

[7] M. Mosca, "Cybersecurity in an era with quantum computers: Will we be ready?," *IEEE Security & Privacy*, vol. 16, no. 5, pp. 38–41, 2018.

[8] D. Stebila, S. Fluhrer, and S. Gueron, "Hybrid key exchange in TLS 1.3," Internet-Draft draft-ietf-tls-hybrid-design-12, Internet Engineering Task Force, Jan. 2025. Work in Progress.

[9] N. Bindel, J. Brendel, M. Fischlin, B. Goncalves, and D. Stebila, "Hybrid key encapsulation mechanisms and authenticated key exchange," in *Post-Quantum Cryptography* (J. Ding and R. Steinwandt, eds.), (Cham), pp. 206–226, Springer International Publishing, 2019.

[10] S. Ahmad, L. Valenta, and B. Westerbaan, "Cloudflare now uses post-quantum cryptography to talk to your origin server." `https://blog.cloudflare.com/post-quantum-to-origins/`, 2023. Accessed: May 20, 2025.

[11] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, Aug. 2018.

[12] C. M. Lonvick and T. Ylonen, "The Secure Shell (SSH) Transport Layer Protocol." RFC 4253, Jan. 2006.

[13] P. Duplys and R. Schmitz, *TLS Cryptography In-Depth: Explore the intricacies of modern cryptography and the inner workings of TLS*. Packt Publishing, 2024.

[14] E. Barker, "Recommendation for key management: Part 1 – general," NIST Special Publication 800-57 Part 1 Revision 5, National Institute of Standards and Technology (NIST), May 2020. Chapter 5.

[15] A. Langley, M. Hamburg, and S. Turner, "Elliptic Curves for Security." RFC 7748, Jan. 2016.

[16] The OpenSSL Project, "X25519 — openssl 3.2 manual page." `https://docs.openssl.org/3.2/man7/X25519/`, 2024. Accessed: May 20, 2025.

[17] National Institute of Standards and Technology (NIST), "Post-quantum cryptography standardization," 2025. Accessed: 06/03/2025.

[18] Open Quantum Safe, "Open quantum safe project," 2025. Accessed: 06/03/2025.

[19] G. Alagic *et al.*, "Status report on the third round of the nist post-quantum cryptography standardization process," tech. rep., NIST, 2022.

[20] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "Crystals - kyber: A cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 353–367, 2018.

[21] O. Q. S. Project, "Kyber key encapsulation mechanism (kem)." `https://openquantumsafe.org/liboqs/algorithms/kem/kyber.html`, 2023. Accessed: April 25, 2025.

[22] "Pynacl: Python binding to the libsodium library - public key encryption." `https://pynacl.readthedocs.io/en/latest/public/`, 2025. Accessed: May 22, 2025.

[23] D. J. Bernstein, T. Lange, and P. Schwabe, "The security impact of a new cryptographic library," in *Progress in Cryptology – LATINCRYPT 2012*, pp. 159–176, Springer Berlin Heidelberg, 2012.

[24] C. A. Melchor *et al.*, "Hamming quasi-cyclic (hqc) specification – fourth round version," tech. rep., HQC Project, February 2025. Accessed: 06/03/2025.

[25] N. Aragon *et al.*, "Bike: Bit flipping key encapsulation – round 4 submission," tech. rep., BIKE Project, October 2024. Accessed: 06/03/2025.

[26] W3Techs, "Usage statistics and market share of cloudflare, may 2025," May 2025. Accessed: May 22, 2025.

[27] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[28] C. Paar and J. Pelzl, *Introduction to Public-Key Cryptography*, pp. 149–171. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[29] A. Langley, M. Hamburg, and S. Turner, "Elliptic Curves for Security." RFC 7748, Jan. 2016.

[30] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[31] D. J. Bernstein and T. Lange, "Post-quantum cryptography," *Nature*, vol. 549, no. 7671, pp. 188–194, 2017.

[32] M. Kumar and P. Pattnaik, "Post quantum cryptography(pqc) - an overview: (invited paper)," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–9, 2020.

[33] T. Reddy.K and H. Tschofenig, "Post-Quantum Cryptography Recommendations for TLS-based Applications," Internet-Draft draft-reddy-uta-pqc-app-07, Internet Engineering Task Force, Feb. 2025. Work in Progress.

[34] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang, "Classic mceliece: Conservative code-based cryptography – cryptosystem specification," tech. rep., Classic McEliece Team, October 2022. Accessed: 06/03/2025.

[35] J. A. Buchmann, D. Butin, F. Göpfert, and A. Petzoldt, *Post-Quantum Cryptography: State of the Art*, pp. 88–108. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[36] J. Buchmann, E. Dahmen, and M. Szydlo, *Post-Quantum Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[37] W. Beullens, M.-S. Chen, J. Ding, B. Gong, M. J. Kannwischer, J. Patarin, B.-Y. Peng, D. Schmidt, C.-J. Shih, C. Tao, and B.-Y. Yang, "Uov: Unbalanced oil and vinegar – algorithm specifications and supporting documentation version 1.0," tech. rep., National Institute of Standards and Technology (NIST), May 2023. Accessed: 06/03/2025.

[38] A. Hülsing, D. Butin, S.-L. Gazdag, J. Rijneveld, and A. Mohaisen, "Xmss: extended merkle signature scheme." RFC 8391, May 2018. Accessed: 06/03/2025.

[39] Z. Zheng, K. Tian, and F. Liu, *Learning with Error*, pp. 53–98. Singapore: Springer Nature Singapore, 2023.

[40] M. Backendal, S. Clermont, M. Fischlin, and F. Günther, "Key derivation functions without a grain of salt," in *Advances in Cryptology – EUROCRYPT 2025* (S. Fehr and P.-A. Fouque, eds.), (Cham), pp. 393–426, Springer Nature Switzerland, 2025.

[41] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Provable Security* (W. Susilo, J. K. Liu, and Y. Mu, eds.), (Berlin, Heidelberg), pp. 1–16, Springer Berlin Heidelberg, 2007.

[42] P. S. Foundation, "hashlib — secure hashes and message digests." `https://docs.python.org/3/library/hashlib.html`, 2025. Accessed: 06/03/2025.

[43] P. S. Foundation, "hmac — keyed-hashing for message authentication." `https://docs.python.org/3/library/hmac.html`, 2025. Accessed: 06/03/2025.

[44] P. S. Foundation, "time — time access and conversions." `https://docs.python.org/3/library/time.html`. Accessed: May 09, 2025.

[45] "psutil: Cross-platform lib for process and system monitoring in python." `https://psutil.readthedocs.io/en/latest/`. Accessed: May 9, 2025.

[46] B. Gregg, *Systems Performance: Enterprise and the Cloud*, ch. 6, pp. 219–302. Mark L. Taub, 2 ed., 2021.

[47] L. man-pages project, "tc-netem(8) — linux manual page." https://man7.org/linux/man-pages/man8/tc-netem.8.html. Accessed: 04/23/2025.

[48] S. Hemminger, "Network emulation with netem," *Linux Conf Au*, 05 2005.

[49] L. man-pages project, "tc-tbf(8) — linux manual page." https://man7.org/linux/man-pages/man8/tc-tbf.8.html. Accessed: 04/23/2025.

[50] M. Trevisan, A. Safari Khatouni, and D. Giordano, "ERRANT: Realistic emulation of radio access networks," *Comput. Netw.*, vol. 176, p. 107289, July 2020.

[51] V. Khandetskyi, V. Gerasimov, and N. Karpenko, "Performance analysis of wireless computer networks in conditions of high interference intensity," *Radio Electronics, Computer Science, Control*, no. 3, 2023.

[52] S. Endres, J. Deutschmann, K.-S. Hielscher, and R. German, "Performance of quic implementations over geostationary satellite links," 2022.

[53] J. Araújo, "Real-world latency and packet loss." https://blog.codavel.com/performance-report-defining-use-cases, 2019. Accessed: April 24, 2025.

Acronyms

BQP bounded-error quantum polynomial-time

DH Diffie-Hellman

DLP discrete logarithm problem

ECDH Elliptic-curve Diffie–Hellman

ECDHE Elliptic Curve Diffie-Hellman Ephemeral

ECDLP Elliptic Curve Discrete Logarithm Problem

FIPS Federal Information Processing Standards

HKDF HMAC-based Key Derivation Function

I/O Input/Output

IETF Internet Engineering Task Force

KEM key-encapsulation mechansim

KEX key exchange

LWE Learning with Errors

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

Module-LWE Module Learning with Errors

MSS Merkle signature scheme

NaCl Networking and Crypthography Library

NIST U.S. National Institute of Standards and Technology

PQ post-quantum

PQC post-quantum cryptography

RSS Resident Set Size

SIS Shortest Integer Solution

SSH Secure Shell

TCP Transmission Control Protocol

TLS Transport Layer Security

KDF key derivation function

eCK extended Canetti-Krawcyzk Model

IND-CCA Indistinguishability Under Chosen Ciphertext Attack

RTT Round Trip Time

Glossary

adaptive chosen-cipher attacks Test12334. 16

ephemeral Test123. 6, 12, 19

forward secrecy Test12334. 6, 12, 19

homomorphic Test12334. 14

Montgomery curve Test12334. 12

quantum adversaries Test12334. 6

side-channel attacks Test12334. 12

List of Figures

1 ECDH KEX: key generation, exchange, and shared secret derivation) 12

2 KEM Computation of a Shared Secret 17

3 DH/KEM Hybrid Exchange of a Shared Secret 18

4 Measurement phases and metrics captured per KEX run 22

5 Key exchange duration per logging side 28

6 System resource usage per logging side 29

7 KEX duration by Execution Environment (Client) 30

8 System resource usage per Execution Environment: (a) CPU and (b) RAM. (Client) 30

9 KEX duration per key exchange mode (Docker, Client) 31

10 KEX duration per key exchange mode (Native, Client) 32

11 System resource usage per KEX mode (Docker, Client) 33

12 System resource usage per KEX mode (Native, Client) 33

13 KEX duration per network profile and key exchange mode 35

14 Mean CPU usage per KEX mode and network profile 36

15 Mean RAM usage per KEX mode and network profile 37

List of Tables

1 Overview of Available Kyber Variations 15

2 Overview of predefined network profiles 24

3 Overview of experiments their execution and setup 27