Raphaela Maria Costa e Silva

Automated Detection and Attack Analysis for Downgrade Vulnerabilities in Hybrid Post-Quantum TLS 1.3

Belo Horizonte, Minas Gerais
2025

Raphaela Maria Costa e Silva

Automated Detection and Attack Analysis for Downgrade Vulnerabilities in Hybrid Post-Quantum TLS 1.3

Report submitted as a requirement for the course "Guided Project in Computing" of the Bachelor's Degree in Computer Science at UFMG

Universidade Federal de Minas Gerais
Institute of Exact Sciences
Department of Computer Science

Supervisor Prof. Dr. Michele Nogueira Lima

Belo Horizonte, Minas Gerais
2025

Abstract

Quantum computing is developing fast, and it threatens to break classical public-key cryptography that currently secures the internet communications (quantum computers enable the "Harvest Now, Decrypt Later" attack, where encrypted data captured today could be decrypted by future quantum computers). To address this problem, hybrid TLS 1.3 protocols combine classical algorithms with post-quantum key encapsulation mechanisms (KEM), providing security during this critical transition period before quantum computers become practical. However, this hybrid approach introduces new complexity and potential vulnerabilities that existing security analysis tools cannot adequately assess, as they lack support for post-quantum extensions and hybrid configurations.

In light of these limitations in current tooling and the growing complexity of hybrid TLS, the present work builds upon the initial groundwork produced in POC I (Projeto Orientado em Computação (Guided Project in Computing)). The exploratory research conducted in POC I established the theoretical foundation through literature review and validated hybrid handshake functionality through experimental testing, this work (POC II) addresses two critical gaps identified in the first phase: (1) the lack of automated tools for analyzing post-quantum components in hybrid TLS handshakes, and (2) deepen the understanding of downgrade attack vulnerabilities targeting these hybrid configurations during the transition period. To study these problems, this work developed a TLS parser for automated analysis of hybrid configurations and analyzed specifications for downgrade attack scenarios.

The developed parser successfully identifies post quantum groups (ML-KEM-512/768/1024) in real TLS handshakes. Additionally, theoretical specifications for two downgrade attack scenarios targeting hybrid TLS were analyzed: post-quantum component removal, and algorithm weakening. These specifications document potential vulnerabilities that were based on the study of the RFC 8556 protocol documentation and historical precedents including FREAK and Logjam attacks, that exploited similar downgrade vulnerabilities in classical TLS implementations. Therefore, this work contributes both for a functional analysis tool enabling automated offline assessment of hybrid TLS configurations and educational documentation of security challenges that requires attention during the post-quantum cryptographic transition.

Key-words: Post-Quantum Cryptography, Hybrid TLS 1.3, TLS Parser, Kyber, Educational Security Research

Lista de tabelas

Lista de abreviaturas e siglas

AES	Advanced Encryption Standard
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
JSON	JavaScript Object Notation
KEM	Key Encapsulation Mechanism
ML-KEM	Module-Lattice-Based Key-Encapsulation Mechanism
NIST	National Institute of Standards and Technology
OQS	Open Quantum Safe
PCAP	Packet Capture
POC	Projeto Orientado em Computação (Guided Project in Computing)
PQC	Post-Quantum Cryptography
RSA	Rivest–Shamir–Adleman
TLS	Transport Layer Security

Sumário

1 Introduction

The security of modern internet communications relies fundamentally on cryptographic protocols, in particular the Transport Layer Security (TLS), which protects data transmitted on the transport layer of the network. For example, every online transaction, from baking operations to confidential communications, depends on the integrity of these cryptographic systems. However, the rapid development of quantum computing poses a threat to the classical public-key cryptography that is currently used on TLS security communication over the internet.

The threat posed by quantum computers arises from the fact that classical public-key cryptography relies on mathematical problems that are hard to solve using classical computers (integer factorization, discrete logarithms). However, quantum computers are able to execute algorithms that solve these mathematical problems on which Rivest–Shamir–Adleman (RSA) and elliptic curve cryptography depend, effectively breaking these widely deployed systems. This implies the "Harvest Now, Decrypt Later"threat, where adversaries capture encrypted traffic today with the intention of decrypting it later, once they have sufficient quantum power available. It is noticeable that data that require long-term confidentiality, such as medical records, state secrets, and financial information, are extremely vulnerable in this context.

In response to this situation, the National Institute of Standards and Technology (NIST) has standardized several post-quantum cryptography (PQC) algorithms, those algorithms are designed to resist quantum attacks. However, the transition to PQC cannot occur immediately due to challenges such as legacy system compatibility, performance of the new cryptographic system, and the need for gradual deployment strategies. In view of these challenges, a prolonged migration period becomes necessary, during which hybrid approaches are essential.

This gradual approach can be exemplified by observing the combination used in Hybrid TLS 1.3: post-quantum mechanisms alongside classical algorithms, ensures security even if one component is compromised [1]. The session key of this combination derives from both classical and post-quantum key exchanges, providing protection against current classical attacks and future quantum attacks simultaneously. However, this added complexity introduces new attack surfaces that are not adequately addressed by existing security methodologies.

Security analysis tools and frameworks are fundamental to maintaining the integrity of the cryptographic protocol in production environments. Organizations rely on these tools to validate that implementations align with security specifications, detect misconfigurations that could weaken protection, and monitor for anomalous behavior indicating potential attacks. Without automated analysis capabilities, organizations must rely on manual

inspections of network traffic, which are time-consuming, error prone, and impractical on the scale of modern enterprise networks.

Given the importance of analysis tools discussed above, the central problem addressed by this project is the current gap in security analysis methodologies when it comes to assessing the risks posed by quantum computing in classical and hybrid cryptographic systems. Many contemporary security analysis tools and frameworks overlook threat models that include quantum-capable adversaries, as well as the unique vulnerabilities that can arise in hybrid cryptographic configurations. This gap highlights the need for updated analysis mechanisms capable of evaluating hybrid TLS deployments within threat models that include quantum-enabled adversaries.

1.1 General Objective

The primary objective of this project is to develop an educational TLS analysis tool able to automatically detect post-quantum components in hybrid TLS 1.3 handshakes, and to analyze potential downgrade attack scenarios for educational purposes. This work aims to provide both practical analysis capabilities and theoretical security analysis to support the secure deployment of hybrid protocols during the post-quantum cryptographic transition. In order to explain the objectives, the following sections present the specific objectives of the work, describe the scope limitations of the proposed tool, and outline the overall organization of the text.

1.2 Specific Objectives

This project is divided in two main projects, POC I and POC II.

POC I establishes the theoretical foundation through a literature review and conducts preliminary exploratory tests validating that hybrid handshakes include both classical and post-quantum components. These topics are detailed below:

Literature Review: Comprehensive review of scientific literature on quantum computing threats (Shor‘s and Grover‘s algorithms), post-quantum cryptographic algorithms (particularly Kyber/ML-KEM), hybrid TLS 1.3 protocols, and documented downgrade attack methodologies (FREAK, Logjam) [2]. This review in recent literature (NIST SP 900-208 [1], Open Quantum Safe (OQS) project [3]).

Experimental Validation: Deployment of a test environment on Ubuntu 24.04.2 using the OQS project's OpenSSL fork [4] to implement hybrid TLS handshakes combining Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) with Kyber512 (p256_kyber512 hybrid group). Three experiments are conducted: First, comparison of classical (p256) vesus hybrid (p256_kyber512) handshakes, revealing significant size differences, sencond, validation of hybrid group negotiation through forced parameters and Wireshark analysis, confirming

presence of both classical P-256 and post-quantum Kyber512 components with group identifier 0xfe30, and third, evaluation of common TLS analysis tools (testssl.sh, sslyze, nmap) against hybrid configurations. These experiments validate the technical feasibility of hybrid protocols in the test environment and provided baseline understanding of handshake structure.

Gap Identification: Analysis using standard tools, such as Wireshark, OpenSSL s_client, and common security scanners (testssl.sh, sslyze, nmap), faces a limitation: current TLS analysis tooling either displays post-quantum extension fields as raw or minimally parsed data without explaining the hybrid structure or fails to detect or validate hybrid post-quantum configurations. As documented in the POC I report, all three tested security scanners fails to recognize the hybrid handshake on the experimental post-quantum hybrid implementation environment, with testssl.sh reporting the server is not recognized as a TLS endpoint, sslyze failing to establish a handshake, and nmap detecting an open port but with no protocol information.

The second part focuses on putting into practice what was learned in the first phase. That is, to develop an educational analysis tool. Specific objectives include:

1. Develop a modular parser architecture in Python using Scapy, implementing three independent components (packet reader, extension parser, post-quantum detector) with comprehensive error handling and validation logic to enable automated detection of ML-KEM variants (512/768/1024-bit).

2. Implement a mapping of post-quantum group identifiers mapping numeric codes registered in IANA TLS parameters to algorithm names, security levels, and expected key share sizes, enabling semantic interpretation of post-quantum extensions.

3. Design and execute validation testing in an isolated Docker environment using OQS-OpenSSL, developing test suite covering hybrid detection, algorithm identification, key share validation, and edge cases, measuring parser accuracy and performance.

4. Analyze technical specifications for two downgrade attack scenarios: post-quantum component removal exploiting fallback mechanisms, and algorithm weakening downgrading security levels while maintaining appearance of post-quantum protection.

1.3 Important Scope Limitations

This work is educational and demonstrative. The developed parser operates only on pre-captured PCAP files (offline analysis), and it is designed for controlled laboratory environments only.

1.4 Text Organization

This text is structured as follows. Chapter 2 presents the theoretical background necessary to understand hybrid post-quantum TLS security, including TLS 1.3 protocol mechanisms, PQC fundamentals, and downgrade attack vulnerabilities. Chapter 3 describes the research methodology employed to develop the TLS parser and attack specifications, detailing the development environment, infrastructure, and validation procedures. Chapter 4 presents the experimental results and analysis, including parser validation outcomes, performance characteristics, and detailed attack scenario specifications. Finally, Chapter 5 concludes the work, presenting the key findings, lessons learned, and directions for future research.

2 Theoretical Background and Related Work

This chapter establishes the theoretical foundation necessary to understand the security challenges addressed in this work. The transition from classical to PQC represents a significant shift in information security history, requiring fundamental changes to protocols that currently protect internet communications. To contextualize this transition and the vulnerabilities it introduces, this chapter is organized into four main sections: Overview of TLS 1.3 Protocol, PQC and Hybrid Protocols, Downgrade Attacks, and Previous Research.

2.1 Overview of TLS 1.3 Protocol

The TLS protocol is a fundamental cryptographic protocol that provides confidentiality, integrity, and authentication for network communications on the modern internet. Furthermore, every Hypertext Transfer Protocol Secure (HTTPS) connection, secure email transmission, and Virtual Private Network tunnel relies on TLS to protect data from eavesdropping and tampering during transmission. Therefore, understanding TLS operation is fundamental to this work, as hybrid post-quantum protocols extend TLS mechanisms to incorporate new cryptographic algorithms while maintaining backward compatibility with existing infrastructure.

In order to understand how TLS achieves its security objectives, it is necessary to examine the three cryptographic primitives that underpin its operation, each serving a distinct but complementary purpose. Initially, public-key cryptography handles the initial establishment of secure communication channels by enabling two parties to agree on shared secret keys without prior arrangement. Once these shared secrets are established, symmetric encryption then uses them to efficiently encrypt the data transmitted during the session. Finally, cryptographic hash functions complement these two mechanisms by generating message authentication codes that detect any tampering with transmitted data. Together, these three components form a layered defense that protects against multiple threats, and this layered approach is important to understand how hybrid protocols must coordinate multiple key exchange mechanisms.

Building on these foundational cryptographic primitives, TLS 1.3 introduces a fundamental redesign of the protocol with focus on eliminating legacy vulnerabilities. In order to eliminate such vulnerabilities, the specification RFC 8446 [5] removes support for outdated algorithms that existed in previous TLS versions for backward compatibility. These removed algorithms include static RSA key transport, which is vulnerable to key compromise and retrospective decryption, as well as static Diffie-Hellman exchanges that lack forward secrecy, and SHA-1 hashing that is vulnerable to collision attacks. In their place, TLS 1.3 uses ephemeral Diffie-Hellman key exchanges for all connections, thereby

ensuring forward secrecy. This design is relevant for hybrid deployments, once it ensures that even if classical key exchanges are eventually broken by quantum computers, the ephemeral nature of the keys still provides some protection.

After establishing the protocol's cryptographic foundations, the following paragraphs explain how TLS 1.3 operationalizes them through its streamlined handshake mechanism.

The handshake initiates when the client sends a ClientHello message containing three pieces of data: a list of supported cipher suites, supported groups specifying acceptable key exchange parameters, and ephemeral key shares for one or more of these groups. The server then responds with a ServerHello message selecting a specific cipher suite and group and providing its own ephemeral Diffie-Hellman computation to derive a shared secret, which is passed through a key derivation function to produce symmetric encryption keys. As illustrated in Figure 1, encryption begins immediately after this exchange, protecting all subsequent authentication credentials and application data. This handshake structure is very important in hybrid configurations, where supported groups must include both classical elliptic curves and post-quantum KEM.

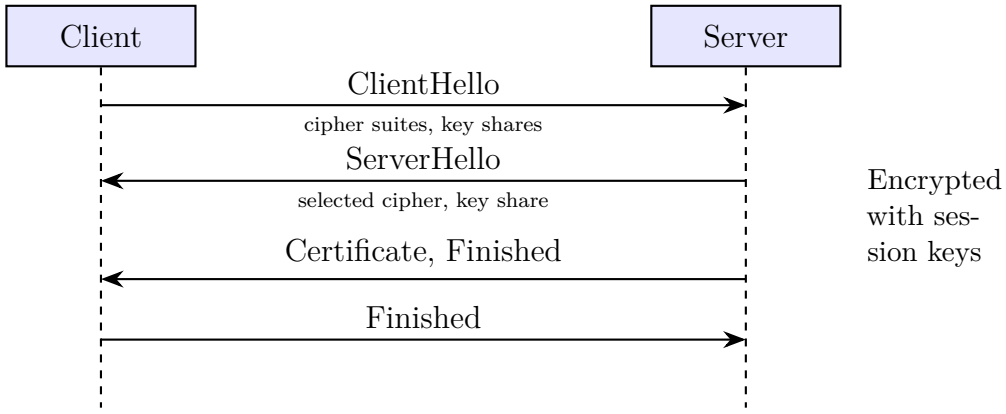

Figura 1 – TLS 1.3 handshake message flow showing ClientHello, ServerHello, and encrypted messages

Understanding these TLS 1.3 mechanisms is essential for this work because hybrid post-quantum protocols extend this handshake structure by adding post-quantum key shares to the existing framework. This increased complexity of hybrid handshakes creates new attack surfaces that require specialized analysis. Without tools capable of interpreting these hybrid configurations, organizations wound need to manually verify whether their TLS deployments successfully negotiate post-quantum protection or remain vulnerable to downgrade attacks, motivating the parser development and threat modeling presented in subsequent chapters.

2.2 PQC and Hybrid Protocols

Despite the strong protections in TLS 1.3 protocol, TLS 1.3's and classical cryptographic security depend on computational hardness assumptions. Specifically, the security of ECDHE (the mechanism used on TLS 1.3) depends on the computational hardness of solving the elliptic curve discrete logarithm problem. These assumptions have secured the internet transmission layer for decades, however, advances in quantum computing challenge this foundation, as quantum computers are capable of executing algorithms that are able to solve discrete logarithm problems in polynomial time (such as Shor's algorithm). This advances in quantum computing permits the "Harvest Now, Decrypt Later"threat, where encrypted data intercepted today may be decrypted once quantum computers become practical. Consequently, organizations that deal with sensitive long-term data such as medical records or state secrets face an immediate risk, since information intercepted today could be decrypted within its sensitivity lifetime.

In response to this threat, NIST selected ML-KEM in 2024 for use in securing communications against quantum-capable adversaries [6]. ML-KEM bases its security on lattice problems that remain computationally hard even for quantum computers, providing quantum-resistant key establishment suitable for TLS deployment. This makes it particularly relevant for securing communications during the transition from classical to PQC.

Unlike traditional key exchange protocols, where both parties interactively compute a shared secret, key encapsulation mechanisms adopt a non-interactive asymmetric approach. In this approach, the recipient first generates a public-private key pair and transmits the public key. Then, the sender generates a random shared secret, encapsulates it using the recipient's public key to produce ciphertext, and sends this ciphertext back. Finally, the recipient decapsulates the ciphertext to recover the same secret. This non-interactive design simplifies security analysis and allows for efficient implementations.

To further illustrate its application in practice, ML-KEM is specified in three parameter sets corresponding to NIST security levels, which indicate computational hardness, as shown in Table 1. ML-KEM-512 provides Level 1 security equivalent to Advanced Encryption Standard (AES)-128 (2^{128} operations), suitable for most commercial applications. It is also possible to look for ML-KEM-768 in the Table 1 and perceive that it provides a LEVEL 3 security equivalent to AES-192 (2^{192} operations), recommended for high-security and long term protection. Lastly, ML-KEM-1024 provides Level 5 security equivalent to AES-256 (2^{256} operations), intended for the most sensitive applications. Understanding these security levels is essential when analyzing hybrid protocols, as adversaries may attempt to downgrade connections from higher security levels to lower ones.

The practical implementation of hybrid key exchange is reflected in the ML-KEM + ECDHE hybrid combination illustrated in Figure 2. In this setup, client and server

Tabela 1 – NIST PQC Security Levels

Level	**Equivalent**	**Description**
Level 1	AES-128	At least as hard to break as AES-128 (128-bit security)
Level 2	SHA-256	At least as hard as collision search on SHA-256
Level 3	AES-192	At least as hard to break as AES-192 (192-bit security)
Level 4	SHA-384	At least as hard as collision search on SHA-384
Level 5	AES-256	At least as hard to break as AES-256 (256-bit security)

perform two independent key establishments in parallel during TLS handshake. This parallel execution enables the protocol to integrate classical and post-quantum mechanisms, providing layered security while maintaining compatibility with existing TLS infrastructure. Understanding this setup is essential before examining the detailed operations of each component.

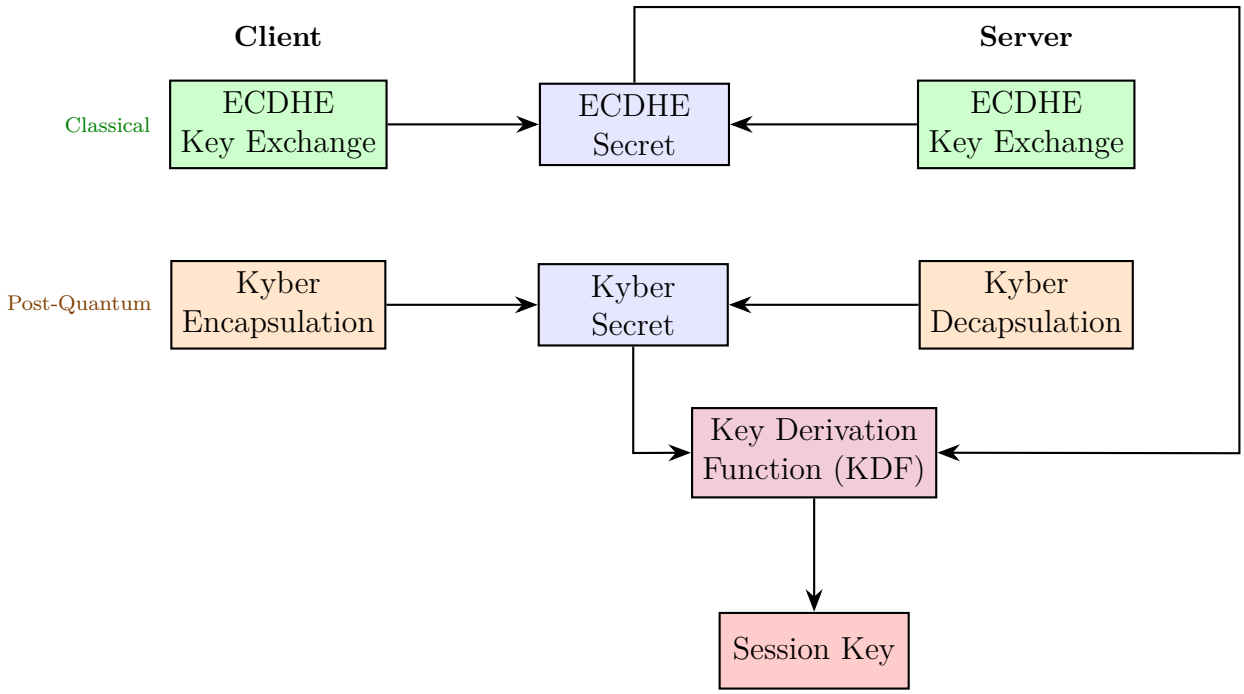

Figura 2 – Hybrid key exchange structure combining classical ECDHE and post-quantum Kyber to generate session key

For the classical component, both parties generate ephemeral ECDHE key pairs, exchange public keys through the key_share extension, and compute the shared secret. Meanwhile, for the post-quantum component, the client generates an ML-KEM pair and includes the public key in the ClientHello message. The server then generates a

random shared secret, encapsulates it using the client's ML-KEM public key, and sends the resulting ciphertext to recover the post-quantum shared secret. Once both key exchanges are complete, the two independent secrets are combined and passed through the TLS 1.3 key derivation function to produce the final session keys. Having detailed how each component operates, it is important to highlight the security implications of this design.

Overall, this design preserves forward secrecy through ephemeral ECDHE keys, while simultaneously ensuring quantum resistance via ML-KEM. As a result, even if one component is compromised, the session still has some protection by the other component, providing robust security against both classical and emerging quantum threats. Thus, the hybrid TLS handshake exemplifies a practical and effective approach for transitioning to post-quantum security while maintaining compatibility and performance in real-world deployments.

Finally, understanding the hybrid protocol mechanisms is crucial, because the increased complexity introduces multiple attack surfaces once adversaries can exploit this complexity to perform downgrade attacks, manipulating handshake messages to remove post-quantum components from the supported_groups list and thereby forcing a fallback to classical-only exchanges. Such attacks highlight that hybrid protocols, while providing enhanced security, simultaneously create new ways for exploitation that demand specialized analysis. This need for specialized analysis capabilities underscores the necessity of developing the parser and attack specifications that form the central contributions of this work, enabling accurate evaluation of hybrid TLS deployments.

2.3 Downgrade Attacks

Downgrade attacks represent a critical vulnerability class where adversaries exploit protocol flexibility to force communicating parties to use weaker security parameters than they would normally negotiate. This attack category happened in practice, with attack such as FREAK and Logjam, these attacks compromised millions of servers. Given the scalability of these attacks, understanding how they work is essential for analyzing TLS hybrid security because hybrid protocols must negotiate both classical and post-quantum components while maintaining backward compatibility. Thereby, creating new opportunities for downgrade manipulation.

2.3.1 Downgrade Protection in TLS 1.3

TLS 1.3 was specifically designed to eliminate many downgrade vulnerabilities through fundamental restructuring: removing insecure legacy algorithms, mandating forward secrecy via ephemeral Diffie-Hellman, and implementing explicit downgrade detection mechanisms. These improvements reflect historical attacks and represent efforts to reduce

attack surface by limiting protocol flexibility. In this subsection, it will be explained how TLS 1.3 eliminate each one of these vulnerabilities.

First, consider the following protocol downgrade scenario: A client supporting TLS 1.3 sends a ClientHello message advertising TLS 1.3 support and modern cipher suites. An active man-in-the middle adversary intercepts this ClientHello, modifies the version field to indicate only TLS 1.2 support, removes strong cipher suites from the list, and forwards the modified message to the server. The server, examining only the modified ClientHello, believes this represents the client's true capabilities and responds with a ServerHello negotiating TLS 1.2 with weaker cipher suites. Without additional protection, this attack would succeed in downgrading the connection, exposing it to attacks that TLS 1.3 was designed to prevent.

To counter such attacks, the TLS 1.3 specification (RFC 8446) [5] includes explicit downgrade detection through special values embedded in the ServerHello message. Consequently, when a TLS 1.3 capable server is forced to negotiate the TLS 1.2 version or below (such as in the scenario above), it embeds 'downgrade sentinel' values in the received ServerHello. The result is that if the sentinel is present, the client immediately aborts the handshake, because the sentinels indicate the server supports TLS 1.3 bust has been forced to downgrade, revealing the manipulation.

Second, consider cipher suite downgrade attacks that exploit the presence of weak cipher suites in earlier TLS versions. An adversary could intercept a ClientHello advertising strong cipher suites such as those using AES-GCM, remove them from the cipher suite list, leaving only weak options like export-grade RSA or RC4, and forward the modified message. The server would then select from the remaining weak ciphers, enabling attacks like FREAK that successfully decrypted connections protected by deliberately weakened 512-bit RSA keys.

TLS 1.3 eliminates cipher suite downgrade by fundamentally redesigning the cipher suite structure and removing all weak algorithms from the specification entirely. The protocol permits only Authenticated Encryption with Associated Data ciphers that provide strong authenticated encryption, such as AES-GCM and ChaCha20-Poly1305, completely eliminating export-grade ciphers, RC4, 3DES, and other weak algorithms that enabled historical attacks. Consequently, even if an adversary modifies the cipher suite list in a ClientHello, the server can only select from strong ciphers, as weak alternatives simply do not exist in the TLS 1.3 specification.

Third, consider algorithm downgrade attacks targeting the key exchange mechanism. In earlier TLS versions, adversaries could manipulate handshake messages to force connections to use static RSA key transport instead of ephemeral Diffie-Hellman key exchange. For example, an adversary intercepts a ClientHello advertising support for both ECDHE and RSA key transport, removes the ECDHE cipher suites, and forwards only those using static RSA. The server would negotiate RSA key transport, which lacks forward secrecy,

in other words, if the server's RSA private key is later compromised, all past sessions using that key become vulnerable to retrospective decryption.

TLS 1.3 eliminates algorithm downgrade by mandating ephemeral Diffie-Hellman key exchange for all connections and completely removing static RSA key transport from the protocol specification. All TLS 1.3 cipher suites exclusively use (EC)DHE for key exchange, ensuring forward secrecy is a mandatory property rather than an optional feature. Therefore, adversaries cannot force downgrade to non-ephemeral key exchange because such mechanisms simply do not exist in TLS 1.3.

Despite these comprehensive protections against traditional downgrade attacks, TLS 1.3 mechanisms were designed for only for classical cryptographic deployments. This creates practical downgrade paths where post-quantum protection is eliminated without triggering existing detection mechanisms, making the attack difficult to detect without specialized monitoring. Understanding these limitations is essential because hybrid protocols, despite building upon TLS 1.3's strong foundation, introduce new vulnerabilities surface that existing protection mechanisms do not adequately address, as explored in the following subsection.

2.3.2 Challenges and Vulnerabilities in Hybrid TLS Setups

Hybrid TLS protocols introduces a complexity that creates new downgrade vulnerabilities possibilities. According to NIST SP 800-208 [1], hybrid schemes must carefully handle the negotiation and the validation of both components to ensure that the security is not compromised through the combination mechanism itself. The fundamental challenge stems from coordinating two parallel cryptographic mechanisms with different security assumptions while maintaining backward compatibility.

The primary vulnerability introduced is that individual components can be selectively stripped during negotiation without detection, unlike classical TLS where attacks must target the entire protocol. In classical TLS, negotiation involves selecting a single algorithm from an ordered list, where either the connection uses strong algorithms or weak algorithms, making downgrade attempts relatively straightforward to detect through the mechanisms described in the previous section. However, hybrid protocols fundamentally change this model by concatenating keys from both classical (ECDHE) and post-quantum (ML-KEM) schemes through key derivation, creating security through redundancy where the protocol remains secure as long as at least one component is uncompromised. At the same time, this structure introduces new attack surfaces, as adversaries can attempt to compromise individual components without necessarily affecting the entire hybrid protocol.

For example, consider a component removal attack targeting the post-quantum element. An adversary intercepts a ClientHello that advertises hybrid support by including both classical groups and post-quantum groups in the `supported_groups` extension. The adversary modifies this extension to remove all post-quantum group identifiers, leaving

only classical groups, and forwards the modified ClientHello to the server. The server, examining the modified list, sees only classical options and selects classical ECDHE for key exchange, negotiating pure classical operation. Critically, the protocol version remains TLS 1.3, all TLS 1.3 security mechanisms remain active including forward secrecy and encryption, and from the protocol's perspective negotiation has succeeded normally, consequently the downgrade sentinel mechanism is not triggered because protocol version has not changed. Only the post-quantum protection has been silently eliminated, and unless endpoints specifically track whether post-quantum algorithms were successfully negotiated and alert on their absence, the attack remains undetected. The system falls back to classical-only operation as designed for backward compatibility.

Beyond component removal, hybrid deployments are also vulnerable to parameter downgrade attacks. In these cases, the adversary allows the hybrid configuration to proceed but manipulates the supported_groups extension to force weaker parameters. For example, ML-KEM-512 (Level 1 security) may be chosen instead of ML-KEM-768 (Level 3 security), reducing security margins while maintaining the appearance of post-quantum protection. Similarly, weaker classical curves may be forced. These attacks are challenging to detect because both components remain present, but the overall security level is silently weakened, requiring a protocol analysis to identify.

Another vector arises from key share manipulation attacks, which exploit the substantially larger message sizes inherent to hybrid handshakes. ML-KEM public keys and ciphertexts significantly increase ClientHello and ServerHello sizes compared to classical handshakes. Adversaries can selectively corrupt portions of the larger key share, triggering decapsulation failures, or exploit middleboxes and firewalls with size limits that fragment or drop oversized messages, potentially causing denial-of-service conditions. Additionally, malformed key share data can be injected to trigger implementation errors in post-quantum code while leaving classical shares intact, effectively disabling the post-quantum component without altering the protocol itself.

Understanding these hybrid-specific vulnerabilities is essential because they represent the primary security challenges this work addresses. The parser developed in Chapter 3 provides automated capability to detect and validate hybrid configurations in captured network traffic, identifying whether connections successfully negotiated post-quantum protection or were downgraded to classical-only operation, what specific parameter sets were selected for each component, and whether both components are present with appropriate security levels. By doing so, this parser bridges the gap between the theoretical security benefits of hybrid protocols and the practical challenges of deploying and validating them.

2.4 Previous Research

Downgrade attacks on TLS have been extensively documented in academic literature. Adrian et al. (2015) [2] demonstrated how the FREAK attack exploited export-grade RSA cipher suites, forcing connections to use deliberately weakened 512-bit keys. The same authors later revealed the Logjam attack, which targeted Diffie-Hellman key exchange by downgrading to weak parameters. These attacks highlighted fundamental vulnerabilities in protocol negotiation and backward compatibility mechanisms.

The transition to PQC introduces new challenges for hybrid protocols. NIST Special Publication 800-208 (2024) [1] addresses the security considerations for stateful hash-based signature schemes and emphasizes the fragility of hybrid deployments during the post-quantum transition. The document warns that hybrid schemes combining classical and post-quantum components must carefully validate both parts to prevent selective weakening attacks.

Recent work on hybrid TLS configurations has focused primarily on performance and compatibility rather than security analysis. While projects like OQS have enabled practical experimentation with hybrid handshakes, comprehensive security testing methodologies remain underdeveloped. Many existing TLS testing tools predate post-quantum extensions and lack the capability to recognize or validate hybrid configurations.

This gap motivates the need for specialized analysis tools that can automatically detect post-quantum components in TLS handshakes and for detailed specifications of potential downgrade attacks targeting hybrid protocols. Understanding these vulnerabilities is essential for developing robust implementations and testing procedures during the post-quantum transition.

2.5 Summary of the chapter

This chapter established the theoretical foundation necessary for understanding the security challenges addressed in this work. Initially the TLS 1.3 protocol architecture was examined, demonstrating how it achieves security through three cryptographic primitives (public-key cryptography for key establishment, symmetric encryption for data protection, and cryptographic hash functions for integrity verification). The analysis also highlighted that TLS 1.3 security fundamentally depends on computational hardness assumptions such as the elliptic curve discrete logarithm problem, which remain secure against classical adversaries but become vulnerable to quantum computers executing Shor's algorithm, creating the "Harvest Now, Decrypt Later"threat that motivates the transition to PQC.

To address this quantum threat, PQC fundamentals were presented, specifically the ML-KEM algorithm that bases its security on lattice problems remaining computationally hard even for quantum computers. The concept of hybrid protocols was detailed, explaining

how they combine classical ECDHE with post-quantum ML-KEM through parallel key exchanges to ensure security through defense-in-depth, where the protocol remains secure as long as at least one component is uncompromised. The analysis revealed that hybrid handshakes substantially increase message sizes, making hybrid connections distinguishable through passive traffic analysis and creating new attack surfaces.

Beyond protocol mechanisms, TLS 1.3's protection features were analyzed, including downgrade sentinel values for detecting version manipulation, the removal of weak cipher suites, and mandatory ephemeral Diffie-Hellman. The analysis, however, showed that these protections were designed for classical configurations: while sentinels can detect protocol version manipulation, they cannot identify the removal of post-quantum components within TLS 1.3. Hybrid-specific vulnerabilities were further detailed, such as component removal attacks that strip post-quantum groups without altering the protocol version, parameter weakening attacks that reduce security levels while preserving a hybrid appearance, and key share manipulation attacks that exploit large message sizes.

Finally, related tools and prior research were presented, documenting that hybrid TLS implementations exist through OQS and historical downgrade attacks are studied. This theoretical foundation establishes the basis on which this work was developed. The next chapter details the methodology, describing how the TLS parser was implemented to address the interpretation gap and how attack scenarios were systematically documented to specify the vulnerabilities identified in this theoretical analysis.

3 Methodology

This chapter describes the research design and methodological approach employed to develop the TLS parser and attack specifications that constitute the core contributions of this work. The methodology follows an applied research approach focused on tool development and threat modeling, structured in three sequential phases that build from implementation through specification to validation. Understanding this methodological framework is essential because it establishes how the parser achieves automated detection of hybrid configurations.

3.1 Research Design

This work adopts an applied research methodology centered on developing practical security analysis capabilities for hybrid post-quantum TLS deployments. The research addresses two complementary objectives: first, creating an automated tool that can detect and validate hybrid configurations in captured network traffic, and second, documenting detailed threat models that specify how adversaries could exploit vulnerabilities in hybrid protocols during the post-quantum transition. These objectives reflect the dual nature of security research, which requires both defensive capabilities (the parser for detection and validation) and offensive understanding (attack specifications).

The research methodology is structured in three sequential phases:

- **Tool Development:** Design and implementation of a modular TLS parser in Python using Scapy that automatically detects and validates hybrid post-quantum configurations in offline PCAP files. This phase includes architecture design with three independent components (packet reader, extension parser, post-quantum detector), implementation of comprehensive group identifier mapping for ML-KEM variants, and development of validation logic for key share sizes and protocol correctness.

- **Attack Specification:** Analisys of two downgrade attack scenarios targeting hybrid TLS deployments. Each specification includes threat models describing adversary capabilities, step-by-step modification procedures detailing which protocol fields to manipulate and how, impact analysis quantifying security degradation, and detection difficulty assessment. Specifications are validated against RFC 8446 protocol documentation and compared with historical precedents (FREAK, Logjam).

- **Validation:** Testing of parser functionality through a test suite covering hybrid detection, algorithm identification, key share validation, and edge cases, the tests were executed in isolated Docker environment using OQS-OpenSSL to generate

authentic hybrid handshake samples. Validation measures parser accuracy, and robustness (handling of malformed inputs).

All work was conducted in controlled laboratory environments to ensure both safety and reproducibility of results. The parser development and testing utilized isolated Docker containers with no external network connectivity, preventing any possibility of accidental interaction with external systems. The OQS-OpenSSL instances used to generate hybrid handshake samples operated in isolated networks, ensuring that experimental traffic remained contained within the test environment.

3.2 Parser Development

The parser development process incorporate the complete implementation lifecycle from initial infrastructure setup through testing and validation. This section describes how the modular TLS parser was constructed to address the automated detection requirements identified in the research objectives. The development followed a systematic approach beginning with establishment of the controlled test environment that generates authentic hybrid handshakes, progressing through component-by-component implementation of the parser architecture, and concluding with validation to confirm functional correctness and performance characteristics.

3.2.1 Infrastructure and Tools

The research environment employs a carefully designed infrastructure that enables safe generation of hybrid TLS handshakes, automated parser development, and reproducible validation of results. This infrastructure combines containerization for isolation, post-quantum cryptographic library support for generating authentic hybrid handshakes, and a development stack for parser implementation and analysis. The complete environment architecture is illustrated in Figure 3, showing how components interact while maintaining network isolation to ensure ethical research practices.

Containerization Platform: Docker and Docker Compose provide the containerization platform that creates isolated testing environments with complete network isolation from external systems. This containerized approach ensures that all hybrid TLS handshake generation occurs within controlled environments that cannot accidentally interact with production infrastructure or external networks. Each container runs a minimal Ubuntu-based image with only the necessary components for its specific role, they either act as a TLS server with hybrid support or as a TLS client capable of negotiating post-quantum algorithms. Also, network isolation is enforced through Docker's built-in networking capabilities, which create isolated virtual networks that prevent containers from accessing

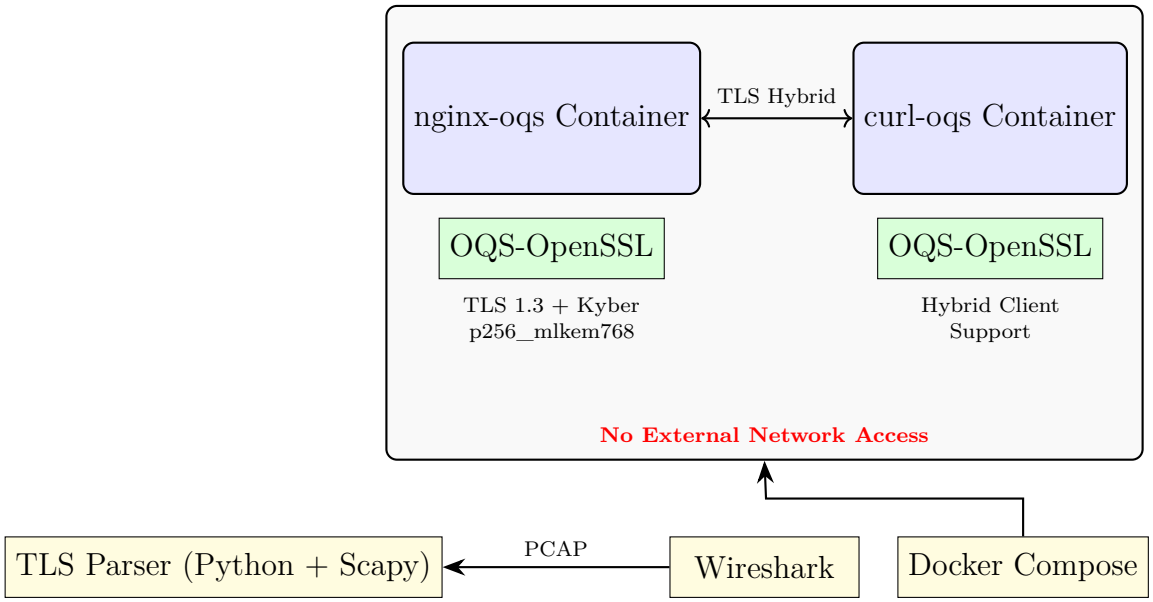

Figura 3 – Docker-based development environment for hybrid TLS testing with isolated network

external resources while allowing them to communicate with each other for handshake generation.

PQC Support: OQS project's OpenSSL fork (liboqs-openssl) [4] provides the cryptographic implementation that enables generation of authentic hybrid TLS handshakes combining classical and post-quantum algorithms. This fork extends standard OpenSSL with implementations of NIST-standardized post-quantum algorithms, including all three ML-KEM parameter sets (ML-KEM-512, ML-KEM-768, ML-KEM-1024) and their hybrid combinations with classical algorithms like ECDHE. The OQS-OpenSSL implementation follows draft IETF specifications for hybrid key exchange in TLS 1.3, enabling negotiation of hybrid groups through standard TLS extensions (supported_groups and key_share) without requiring protocol modifications. Both the server container (nginx-oqs) and client container (curl-oqs) utilize OQS-OpenSSL, allowing them to perform complete hybrid handshakes that include both classical ECDHE and post-quantum ML-KEM key exchanges in parallel, producing the authentic hybrid traffic samples necessary for parser validation.

Development Stack: The development stack comprises the software tools and libraries used for parser implementation, packet analysis, and validation. This stack was selected to balance ease of development, protocol manipulation capabilities, and compatibility with existing security analysis workflows. The development stack of this work is detailed below:

- **Python 3.8+** serves as the implementation language for the TLS parser, chosen for its extensive library ecosystem, straightforward syntax that enhances code readability and maintainability, and widespread adoption in the security research community.

Python's dynamic typing and high-level abstractions enable rapid development of tools while maintaining code clarity.

- **Scapy library** provides the packet manipulation and protocol dissection capabilities that form the foundation of the parser implementation. Scapy's layered architecture allows reading PCAP files, accessing individual protocol layers within captured packets, and creating custom dissectors for protocol extensions. The library's TLS layer support includes built-in understanding of TLS handshake structures, enabling extraction of extensions like supported_groups and key_share that contain the post-quantum negotiation data the parser must interpret.

- **Wireshark** serves dual purposes: capturing network traffic from the Docker environment for parser input, and providing baseline validation by manually inspecting handshake messages to verify that the containerized environment correctly generates hybrid handshakes before automated parser testing. Wireshark displays raw protocol data that confirms handshake structure and presence of extension fields.

3.2.2 Test Environment

The test environment provides the controlled infrastructure necessary for generating authentic hybrid TLS handshakes and validating parser functionality. This environment consists of isolated Docker containers running OQS-enabled TLS implementations that can negotiate hybrid key exchanges, enabling capture of network traffic containing both classical and post-quantum components. Understanding this environment is essential because it establishes how the parser test data is generated and ensures that all experiments occur in isolated conditions without external dependencies or security risks.

The server component consists of an nginx web server compiled with OQS-OpenSSL provider support, deployed in an isolated Docker container to provide controlled TLS endpoints for testing. This server is configured to support hybrid TLS groups combining classical elliptic curves with ML-KEM variants, specifically p256_mlkem768 (combining NIST P-256 curve with ML-KEM-768) and x25519_mlkem768 (combining Curve25519 with ML-KEM-768) covering all three ML-KEM security levels (ML-KEM-512, ML-KEM-768, ML-KEM-1024), enabling generation of handshakes that include both classical and post-quantum key exchange components. The server uses Dilithium3, a NIST-standardized post-quantum signature algorithm, for certificate signatures, ensuring that the entire TLS stack employs PQC rather than only the key exchange mechanism.

The client component utilizes curl compiled with OQS-OpenSSL provider support, deployed in a separate Docker container that can initiate hybrid TLS connections to the nginx-oqs server. This client is configured to advertise support for the same hybrid groups as the server (p256_mlkem768, x25519_mlkem768), enabling successful hybrid handshake negotiation where both endpoints agree on post-quantum key exchange mechanisms. The

curl-oqs container operates within the same isolated Docker network as the server but in a separate container, simulating a realistic client-server architecture while maintaining complete isolation from external networks. This separation enables capture of complete TLS handshakes including both ClientHello and ServerHello messages with their respective key shares, providing the authentic hybrid traffic samples necessary for parser validation.

All packet captures analyzed by the parser were generated exclusively from connections between the nginx-oqs and curl-oqs containers within the isolated Docker network, ensuring controlled test conditions and eliminating external dependencies. Traffic capture occurs through Wireshark monitoring the virtual network interface connecting the two containers, producing PCAP files containing complete handshake sequences with hybrid key exchange components. The isolation from external networks guarantees that captured traffic contains only the intended test scenarios without interference from external connections, malformed packets from the internet, or other unpredictable factors that could affect parser validation.

3.3 Parser Development

The parser development began with requirements analysis identifying the core capabilities necessary for automated hybrid TLS analysis. These requirements include: automatically detecting post-quantum group identifiers in TLS handshake extensions, validating that key share sizes match expected values for negotiated algorithms, distinguishing hybrid configurations (containing both classical and post-quantum components) from classical-only handshakes, and generating structured output in machine-readable format for integration with analysis workflows. Based on these requirements, the parser architecture adopts a modular design with three independent components that implement separation of concerns and enable extensibility for future algorithm support. This component-based architecture allows each module to focus on a specific responsibility (packet reading, extension parsing, or post-quantum detection)facilitating independent development, testing, and maintenance of each component.

The parser accepts PCAP files as input, processes TLS handshake messages contained within captured packets, and produces JavaScript Object Notation (JSON) formatted output documenting detected configurations. The JSON output format provides structured data that can be programmatically processed by other tools, enabling integration into automated security monitoring pipelines or further analysis workflows. The command-line interface follows Unix conventions with standard input/output handling and clear error messages, ensuring the parser integrates smoothly with existing security analysis workflows. An illustration of the parser architecture can be seen in Figure 4.

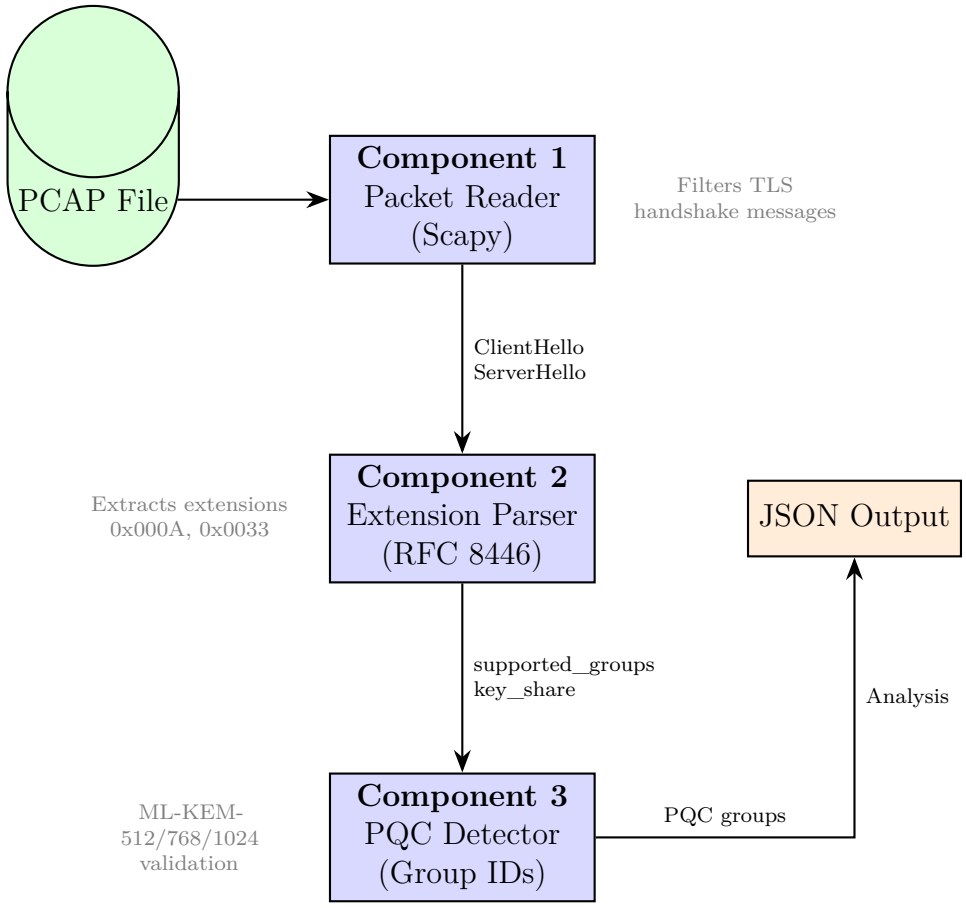

Figura 4 – Modular architecture of the TLS hybrid parser with three independent components

3.3.1 Implementation

The parser implementation employs three independent components that collectively provide complete hybrid TLS analysis capability. This component-based approach ensures modularity, enabling each component to be developed, tested, and modified independently while maintaining clear interfaces between components.

The **Packet Reader** component handles PCAP file loading and TLS message filtering. This component utilizes Scapy's `rdpcap()` function to read captured packets from PCAP format files and applies TLS layer dissectors to identify and extract handshake messages from network traffic. The component implements comprehensive error handling for malformed packet captures, empty PCAP files, and files containing non-TLS traffic, ensuring robust operation across various input conditions. By isolating packet reading functionality in a dedicated component, the parser can easily adapt to different input formats or packet capture libraries without affecting extension parsing or detection logic.

The **Extension Parser** component extracts and interprets TLS extensions from ClientHello and ServerHello messages. This component focuses specifically on extensions relevant to hybrid negotiation: the `supported_groups` extension that advertises which

elliptic curves and post-quantum algorithms the client supports, and the `key_share` extension that contains the actual public key material exchanged during handshake. The component parses extension data structures according to RFC 8446 [5] specifications, correctly handling variable-length fields and nested structures to extract group identifiers and key share data. This specialized parsing enables reliable extraction of the post-quantum negotiation information that subsequent components analyze.

The **Post-Quantum Detector** component provides the semantic interpretation that distinguishes this parser from generic packet analyzers. This component maintains a comprehensive mapping of group identifiers to algorithm names and security properties, translating numeric codes like 0x11EC into meaningful algorithm descriptions such as ML-KEM-768. The component validates that key share sizes match expected values for each algorithm. Most critically, this component identifies hybrid configurations by detecting the presence of both classical groups (X25519 or P-256) and post-quantum groups (Module-Lattice Key Encapsulation Mechanism (ML-KEM) variants) within the same handshake, enabling automated classification of connections as hybrid or classical-only.

In summary, the parser code is organized into three Python modules that correspond to logical functional boundaries. The `tls_parser.py` module contains the main parser logic including ClientHello and ServerHello processing, component coordination, and JSON output generation. The `pqc_groups.py` module encapsulates all post-quantum algorithm knowledge including group identifier mappings, expected key share sizes, and algorithm metadata, centralizing algorithm-specific information that may require updates as new post-quantum algorithms are standardized. The `test_parser.py` module provides test coverage.

3.3.2 Testing

Parser validation employs a testing strategy that combines unit tests, integration tests, and edge case coverage to ensure correct functionality across diverse scenarios. The testing approach validates both functional correctness (does the parser correctly identify algorithms and classify configurations) and performance characteristics (does the parser process handshakes efficiently), establishing confidence in parser reliability for production use.

Unit tests validate individual parser functions in isolation, ensuring that each component correctly implements its specific responsibility. These tests verify group identifier recognition by confirming the parser correctly maps numeric codes to algorithm names for all supported algorithms, key share size validation by testing that the parser accepts correctly-sized key shares and rejects malformed data, and hybrid detection logic by verifying the parser accurately distinguishes hybrid configurations from classical-only handshakes based on group combinations. Each major function has corresponding test

cases that exercise both normal operation and error conditions, achieving comprehensive code coverage.

Integration tests validate end-to-end parser operation by processing real hybrid TLS handshakes captured from the OQS-enabled test environment described in Section 3.3. These tests capture complete handshake sequences containing authentic hybrid negotiations between nginx-oqs and curl-oqs containers, process these captures through the complete parser pipeline from PCAP reading through JSON output generation, and verify that parser output correctly reflects the actual negotiated algorithms and key shares. Integration testing ensures that components correctly interact and that the parser functions properly with real-world protocol data rather than only synthetic test inputs.

Edge case testing addresses unusual or potentially problematic scenarios that may not occur in typical operation but must be handled gracefully. Test cases include empty PCAP files containing no packets, malformed packets with corrupted TLS structures or invalid extension data, PCAP files containing multiple sequential handshakes, and mixed traffic combining hybrid and classical handshakes in the same capture. These tests ensure the parser maintains robust operation and provides meaningful error messages rather than crashing or producing incorrect output when encountering unexpected inputs.

Test data consists of packet captures generated from the nginx-oqs server configured with multiple hybrid groups (p256_mlkem768 combining P-256 with ML-KEM-768, and x25519_mlkem768 combining Curve25519 with ML-KEM-768) covering all three ML-KEM security levels (ML-KEM-512, ML-KEM-768, ML-KEM-1024) as well as classical-only handshakes using X25519 and secp256r1 for comparison. All test data originates from the controlled Docker environment described in Section 3.2.2, eliminating dependencies on external network conditions.

4 Results

This chapter presents the results achieved in POC II, organized into two main deliverables that address the research objectives established in Chapter 1. The first deliverable is the TLS parser implementation, which provides automated capability to detect and validate hybrid post-quantum configurations in captured network traffic. The second deliverable consists of detailed attack specifications that document potential downgrade vulnerabilities targeting hybrid TLS deployments.

4.1 Functional Capabilities

The parser implements three core capabilities that address the limitations in existing TLS analysis tools documented in POC I experimental results.

First, the parser provides **automatic post-quantum detection** by parsing the `supported_groups` extension from ClientHello messages and matching group identifiers against a comprehensive database of ML-KEM variants. The parser recognizes all three NIST-standardized parameter sets (ML-KEM-512 with identifier 0x11EC, ML-KEM-768 with identifier 0x11EB, and ML-KEM-1024 with identifier 0x11EA) as well as their hybrid combinations with classical algorithms, translating numeric codes into meaningful algorithm descriptions that provide semantic interpretation.

Second, the parser implements **key share analysis** by extracting key share data from the `key_share` extension and validating that sizes match expected values for negotiated algorithms. For each detected group, the parser verifies that the associated key share has the correct size. This size validation detects malformed handshakes that may indicate implementation errors or protocol manipulation attempts, providing an additional verification layer beyond simple group identifier recognition.

Third, the parser provides **hybrid configuration detection** by analyzing the combination of groups present in handshake messages to distinguish hybrid configurations from classical-only or pure post-quantum deployments. The parser identifies hybrid handshakes by detecting the presence of both classical groups (such as X25519 or P-256) and post-quantum groups (ML-KEM variants) within the same ClientHello, enabling automated classification of connections based on their cryptographic protection level.

4.2 Test Results

The parser was subjected to testing covering multiple scenarios including post-quantum group detection, hybrid configuration identification, classical-only handshake processing,

and edge cases. Testing employed authentic hybrid handshakes captured from the OQS-enabled Docker environment described in Chapter 3. Table 2 summarizes the complete test suite results, demonstrating that all eight test cases passed successfully.

Tabela 2 – Parser Validation Test Results

Test Case	**Capability Tested**	**Expected**	**Result**
Test 1	PQC group detection	Identify ML-KEM-768	PASS
Test 2	PQC group detection	Identify ML-KEM-512	PASS
Test 3	PQC group detection	Identify ML-KEM-1024	PASS
Test 4	Hybrid detection	Detect mixed configs	PASS
Test 5	Classical detection	Identify x25519 only	PASS
Test 6	Multiple handshakes	Parse 3+ ClientHellos	PASS
Test 7	Group ID mapping	0x11EB → mlkem768	PASS
Test 8	Group ID mapping	0x11EC → mlkem512	PASS
Test 9	Group ID mapping	0x2A01 → mlkem1024	PASS
Test 10	Hybrid group mapping	0x2F39 → x25519_mlkem768	PASS
Test 11	Key share validation	ML-KEM-768	PASS
Test 12	Key share validation	ML-KEM-512	PASS
Test 13	Key share validation	ML-KEM-1024	PASS
Test 14	Protocol version	Identify TLS 1.3	PASS
Success Rate			**100% (14/14)**

The 100% success rate across all test cases demonstrates the parser's ability to correctly identify post-quantum groups through accurate group identifier mapping, validate key share sizes against expected values for each algorithm, and distinguish between hybrid and classical configurations based on the combination of negotiated groups. These results confirm that the parser achieves its primary objective of providing automated semantic interpretation of hybrid TLS handshakes.

4.3 Performance Characteristics

On the tests executed, the parser achieves average processing time of 1.2 milliseconds per ClientHello message when analyzing hybrid handshakes containing both classical and post-quantum components, measured on a standard development laptop with no performance optimization. The parser performance scales linearly with the number of handshakes in a PCAP file.

4.4 Handshake Size Analysis

Analysis of captured hybrid handshakes reveals significant size increases compared to classical-only configurations, with implications for network performance, protocol

distinguishability, and attack surface. Table 3 presents key share size measurements extracted from authentic handshakes in the test environment, demonstrating the magnitude of overhead introduced by PQC.

Tabela 3 – Key Share Size Comparison: Classical vs Hybrid TLS

Configuration	**Size (bytes)**	**Increase**	**Main Component**
Classical (x25519 only)	326	—	ECDHE key share (32 bytes)
Hybrid (p256_mlkem768)	1249	+923	ML-KEM-768 share (1184 bytes)
Hybrid (x25519_mlkem768)	1216	+890	ML-KEM-768 share (1184 bytes)

Note: Size measurements represent the total key share payload extracted from the key_share extension (0x0033), not the complete ClientHello message size. Measurements obtained from captures in the isolated Docker environment.

The measurements demonstrate that hybrid key shares are way larger than classical counterparts, with the increase almost entirely attributable to ML-KEM-768 public key material. This substantial size increase has some security implications:

First, hybrid handshakes become trivially distinguishable from classical handshakes through passive traffic analysis, potentially enabling adversaries to selectively target connections employing post-quantum protection. Second, the larger messages may encounter problems with network middleboxes that impose size limits on initial TLS messages, potentially causing connection failures or forcing fallback to classical algorithms. Third, the larger key shares provide adversaries with more ciphertext material that might facilitate cryptanalysis if weaknesses in ML-KEM are discovered.

These observations inform the attack specifications developed in the following section, which consider how adversaries might exploit these protocol characteristics during downgrade attacks.

4.5 Downgrade Attack Analysis for Hybrid TLS 1.3

This section presents detailed analysis of two downgrade attack scenarios that could potentially compromise hybrid TLS 1.3 deployments during the post-quantum transition period. Unlike the parser implementation, which was empirically validated through testing, this analysis represents theoretical examination of protocol vulnerabilities based on RFC 8446 specifications and historical attack precedents. These scenarios demonstrate how adversaries might exploit the increased complexity of hybrid protocols to force connections back to weaker cryptographic configurations, thereby negating the quantum-resistant protection that hybrid deployments are intended to provide.

Each attack scenario is documented through a structured threat model that includes the conceptual attack approach, step-by-step procedural description of required protocol

modifications, theoretical security impact if the attack succeeds, and detection difficulty assessment. Two attack scenarios were specified:

4.5.1 Attack 1: Post-Quantum Component Removal

This attack scenario exploits the backward compatibility mechanisms in hybrid TLS deployments by removing post-quantum components from handshake negotiation, forcing connections to fall back to classical-only cryptography. The attack targets the fundamental tension between maintaining interoperability with legacy systems and ensuring post-quantum protection, demonstrating how the same fallback mechanisms that enable gradual deployment also create opportunities for adversarial manipulation.

Attack Concept: An adversary with man-in-the-middle network positioning intercepts ClientHello messages during the initial handshake phase and removes all post-quantum group identifiers and key shares, leaving only classical algorithm options. The server, seeing only classical groups in the modified ClientHello, negotiates classical ECDHE key exchange, resulting in a connection that maintains TLS 1.3 protocol level but lacks post-quantum protection. This attack is subtle because it exploits standard TLS fallback behavior, consequently, the protocol operates normally and all validation checks pass, but quantum resistance has been silently eliminated.

Procedural Steps: The attack requires the following protocol modifications to the intercepted ClientHello message. First, the adversary parses the `supported_groups` extension to identify post-quantum group identifiers such as ML-KEM-768 (0x11EB), ML-KEM-512 (0x11EC), or hybrid combinations, and removes all such identifiers from the group list while preserving classical groups like X25519 or P-256. Second, the adversary parses the `key_share` extension (type 0x0033) and removes the corresponding post-quantum key share entries, which are substantially larger than classical key shares. Third, all extension length fields must be recalculated to reflect the reduced data size (the `supported_groups` extension length, the `key_share` extension length, and the total extensions length field in the ClientHello). Fourth, the overall handshake message length field must be updated to account for the removed data. Finally, TCP/IP checksums must be recalculated to ensure the modified packet remains valid at the network layer. The modified ClientHello is then forwarded to the server, which negotiates classical-only key exchange believing this represents the client's true capabilities.

Theoretical Security Impact: Successful execution reduces ClientHello message size by approximately 800-1200 bytes depending on which ML-KEM variant is removed, as the post-quantum key share data constitutes the majority of hybrid handshake overhead. The server negotiates a classical group such as X25519, resulting in a TLS 1.3 session with forward secrecy and authenticated encryption but lacking any post-quantum protection. The connection remains vulnerable to future quantum attacks under the "Harvest Now, Decrypt Later"threat model. From the protocol's perspective, negotiation succeeds nor-

mally with no errors or anomalies, and the downgrade sentinel mechanism in TLS 1.3 is not triggered because protocol version remains TLS 1.3 rather than being downgraded to TLS 1.2 or below.

Detection Difficulty: Detecting this attack requires monitoring that specifically tracks whether post-quantum algorithms were successfully negotiated rather than simply verifying that TLS 1.3 is in use. Organizations must implement monitoring that compares advertised capabilities against negotiated results, alerting when connections that should support post-quantum protection fall back to classical-only configurations.

4.5.2 Attack 2: Post-Quantum Algorithm Weakening

This attack scenario represents a more sophisticated approach where adversaries maintain the appearance of post-quantum protection while reducing security levels, making detection substantially more difficult than complete component removal. The attack exploits the availability of multiple ML-KEM parameter sets with different security levels to force negotiation of weaker variants that may not meet organizational security requirements.

Attack Concept: An adversary with man-in-the-middle positioning intercepts ClientHello messages and replaces strong post-quantum algorithm identifiers with weaker variants from the same algorithm family. For example, ML-KEM-1024 (NIST Level 5, 256-bit equivalent security) is replaced with ML-KEM-512 (NIST Level 1, 128-bit equivalent security). The server negotiates the weaker variant, and both endpoints report successful post-quantum protection, but the actual security level is substantially reduced. This attack is particularly insidious because it does not eliminate post-quantum protection entirely.

Procedural Steps: This attack requires more complex modifications than simple removal because key share sizes differ between ML-KEM variants. First, the adversary parses the `supported_groups` extension to identify strong post-quantum algorithms and replaces their identifiers with weaker variants: ML-KEM-1024 (0x2A01) becomes ML-KEM-512 (0x11EC), ML-KEM-768 (0x11EB) becomes ML-KEM-512, or hybrid combinations are downgraded similarly. Second, the `key_share` extension must be modified more carefully because different ML-KEM variants have different key sizes—ML-KEM-1024 public keys are 1568 bytes while ML-KEM-512 keys are 800 bytes, creating a 768-byte size difference. The adversary must either generate a valid ML-KEM-512 key share to replace the original, or truncate/pad the existing key share while maintaining protocol validity. Third, all extension length fields must be recalculated to account for size differences across multiple affected extensions. Fourth, handshake message length and TCP/IP checksums must be updated. The technical complexity is higher than simple removal, but the difficulty of detection makes it a more viable attack in practice.

Theoretical Security Impact: Successful execution reduces post-quantum security from NIST Level 5 (256-bit equivalent, comparable to AES-256, requiring approximately 2^{256} operations to break) to Level 1 (128-bit equivalent, comparable to AES-128, requiring approximately 2^{128} operations). While Level 1 still provides post-quantum resistance and remains computationally infeasible to break with current or near-term technology, the security margin is substantially reduced compared to organizational security policies that may mandate Level 3 or Level 5 for sensitive communications. The reduction in computational hardness from 2^{256} to 2^{128} represents a decrease of 2^{128} factor in attacker effort. Critically, the connection appears post-quantum protected in all standard logging and monitoring—both client and server negotiate a valid ML-KEM variant and report quantum-resistant key exchange, making detection significantly more difficult than complete PQC removal.

Detection Difficulty: Detection is substantially more difficult than binary PQC presence checks because it requires monitoring the specific security level negotiated. Standard TLS analysis tools and security scanners report "ML-KEM in use"without distinguishing between ML-KEM-512 (Level 1), ML-KEM-768 (Level 3), and ML-KEM-1024 (Level 5) variants. Organizations would need specialized monitoring that parses group identifiers at a granular level, compares negotiated security levels against policy requirements, and alerts when connections use weaker variants than mandated. The parser developed in this work provides the capability to perform such granular detection by mapping group identifiers to specific security levels, but most existing deployments lack this capability.

5 Conclusions and Future Work

This chapter synthesizes the outcomes of POC II, presenting both achievements and critical lessons learned during the development and validation of the TLS parser and attack analysis. Beyond summarizing deliverables, this chapter reflects on the practical insights gained through implementation, identifies limitations encountered during the research, and proposes concrete directions for future work that address these limitations while extending the scope of hybrid TLS security analysis.

5.1 Research Outcomes and Key Findings

POC II successfully achieved its dual objectives through two complementary contributions that together advance understanding of hybrid post-quantum TLS security:

First, a fully functional TLS parser was implemented in Python using Scapy, achieving 100% test success rate (14/14 tests) across diverse scenarios including post-quantum group detection, hybrid configuration identification, key share validation, and edge case handling. The parser demonstrates automated capability to detect ML-KEM groups in hybrid handshakes by parsing `supported_groups` and `key_share` extensions, providing semantic interpretation that existing widely-deployed tools like Wireshark and security scanners lack.

Second, a detailed technical analysis of two distinct downgrade attack scenarios (post-quantum component removal, and algorithm weakening) was developed, each documented with complete threat models, step-by-step procedural descriptions, and theoretical impact assessments. Together, these contributions provide both defensive capabilities (automated detection through the parser) and offensive understanding (attack scenarios that inform defensive strategies).

5.2 Critical Lessons Learned

The development and validation process revealed several important insights about both hybrid TLS protocols and security tool development that extend beyond the specific technical deliverables. These lessons inform future research directions and highlight challenges that must be addressed as hybrid protocols transition from experimental to production deployment.

5.2.1 Parser Development Insights

The modular component-based architecture proved essential for managing implementation complexity and enabling systematic testing. Separating packet reading, extension parsing, and post-quantum detection into independent modules allowed each component to be developed and validated in isolation, substantially reducing debugging difficulty compared to monolithic designs. However, this modularity came with integration complexity, ensuring that components correctly communicated parsed data structures required careful interface design and comprehensive integration testing beyond unit tests. The lesson is that modular architecture provides long-term maintainability benefits but requires upfront investment in interface specification and integration validation.

Reliance on Scapy's TLS dissectors introduced both capabilities and limitations that shaped parser functionality. Scapy provided essential infrastructure for PCAP file handling and basic TLS message parsing, significantly reducing implementation effort compared to writing raw packet parsers. However, Scapy's TLS layer support focused on common protocol elements and lacked built-in understanding of post-quantum extensions, requiring custom parsing logic for `supported_groups` and `key_share` extension data. This experience demonstrates that even well-established packet manipulation libraries lag behind emerging protocol features, and security tools targeting new cryptographic deployments must anticipate implementing custom extension handlers rather than relying solely on library support.

The Docker-based test environment revealed practical limitations in simulating real-world deployment complexity. Isolated containers with OQS-OpenSSL provided controlled generation of authentic hybrid handshakes without external dependencies, enabling reproducible validation across different systems. However, the controlled environment could not replicate several real-world conditions that affect hybrid protocol behavior: network middleboxes that fragment or modify large handshake messages, diverse client and server implementations with varying levels of post-quantum support, and production networks with complex routing and filtering policies.

5.2.2 Attack Analysis Insights

The theoretical attack analysis revealed fundamental tensions in hybrid protocol design that cannot be easily resolved through technical mechanisms alone. The most significant insight is that backward compatibility, an essential feature for enabling gradual post-quantum migration, inherently creates downgrade vulnerabilities. Systems must support classical-only negotiation to maintain interoperability with legacy clients, but this same fallback mechanism enables adversaries to force classical-only operation by removing post-quantum components. Similarly, supporting multiple ML-KEM security levels (512/768/1024) provides deployment flexibility, but also enables parameter weakening

attacks where adversaries force lower security levels while maintaining appearance of post-quantum protection. These are not implementation bugs that can be patched but rather fundamental design tradeoffs between flexibility and security.

The attack scenarios also highlighted critical gaps in current monitoring and validation approaches. Most existing TLS security monitoring treats post-quantum protection as a binary property, but the parameter weakening scenario demonstrates that this binary approach is insufficient. Organizations need granular monitoring that tracks specific security levels (ML-KEM-512 vs 768 vs 1024), validates that both classical and post-quantum components are present in hybrid configurations, and compares negotiated parameters against policy requirements.

5.3 Future Work

Building on the outcomes and insights obtained in POC II, ideias for further research and tool enhancement arise. While the parser and attack specifications developed in this work provide foundational capabilities for analyzing hybrid TLS configurations and understanding potential downgrade vulnerabilities, there remain opportunities to expand functionality, increase coverage, and integrate real-time monitoring and detection mechanisms. The following subsections outline concrete directions that can extend the practical impact of this work and deepen understanding of hybrid post-quantum protocol security.

5.3.1 Parser Enhancement

Extending the parser with live packet capture support would enable real-time protocol monitoring in educational and research environments. Rather than post-hoc analysis of PCAP files, the parser could monitor network interfaces to provide immediate feedback on hybrid negotiation attempts, success rates, and anomalies. This capability would be particularly valuable in educational settings where students learn by observing protocol behavior in real-time rather than analyzing static captures.

5.3.2 Broader Protocol Coverage

Extending analysis to ML-DSA (Dilithium) signatures and SLH-DSA (SPHINCS+) would enable complete hybrid handshake analysis including certificate validation and authentication, not just key exchange. Current parser focuses on KEMs because they dominate handshake complexity and size overhead, but comprehensive hybrid analysis requires understanding how post-quantum signatures interact with post-quantum key exchange.

Incorporating alternative KEM mechanisms would provide comparative insights into different post-quantum approaches. These alternatives offer different security assumptions and performance characteristics than lattice-based ML-KEM, and analyzing their deployment in hybrid protocols could reveal whether attack scenarios generalize across post-quantum families or whether specific algorithms introduce unique vulnerabilities.

5.4 Final Remarks

POC II successfully transitioned from the exploratory research of POC I to practical tool development and comprehensive vulnerability analysis. Building on this progression, the parser provides automated capability for hybrid TLS analysis that existing tools lack, while attack scenarios document vulnerabilities that organizations must consider during post-quantum migration. Together, these deliverables form a foundation for understanding and managing the security of hybrid protocols throughout this critical transition period.

This foundation leads to insight: hybrid TLS protocols do more than simply add post-quantum protection to existing TLS; they fundamentally alter the protocol's complexity, observable behavior, and attack surface. The substantial increase in handshake message sizes, the inherent tension between backward compatibility and downgrade resistance all highlight that hybrid deployments demand new analysis methods, enhanced monitoring capabilities, and adapted defensive strategies. In other words, securing hybrid TLS requires rethinking traditional assumptions and validation techniques rather than merely extending classical TLS security practices.

AI use Acknowledgment

This work was developed with assistance from AI tools for:

- Python code debugging and Scapy library usage guidance.
- LaTeX formatting and document structure assistance.
- Technical writing clarity and grammar checking.

All research design, experimental setup, data collection, parser implementation logic, attack specifications, analysis, and conclusions are the original intellectual work of the author. AI tools served as assistive technology, comparable to spell-checkers or search engines. All AI-suggested content was critically reviewed.

Bibliography

[1] National Institute of Standards and Technology. *Recommendation for Stateful Hash-Based Signature Schemes*. [S.l.], 2024. Disponível em: <https://doi.org/10.6028/NIST.SP.800-208>.

[2] ADRIAN, D. et al. Imperfect forward secrecy: How diffie-hellman fails in practice. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Denver, CO, USA: ACM, 2015. p. 5–17.

[3] Open Quantum Safe Project. *liboqs: C library for quantum-safe cryptography*. 2024. `https://openquantumsafe.org`. Accessed: November 2024.

[4] PROJECT, O. Q. S. *OQS-OpenSSL: Post-Quantum Cryptography in OpenSSL*. 2025. `https://github.com/open-quantum-safe/openssl`. Accessed: 2025-06-29.

[5] RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.3*. 2018. `https://datatracker.ietf.org/doc/html/rfc8446`. RFC 8446.

[6] BOS, J. W. et al. *CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation*. 2022. `https://pq-crystals.org/kyber/`. Finalist in the NIST PQC standardization process.

[7] KHOUZANI, M. H.; MAFFEI, M.; RYAN, M. D. Formal methods for tls penetration testing: A survey. *ACM Computing Surveys*, ACM, v. 53, n. 6, p. 1–30, 2020.

[8] PATEL, K.; DAS, A. Quantum-resistant cryptography: A survey and future directions. *International Journal of Computer Applications*, Foundation of Computer Science (FCS), v. 183, n. 4, p. 21–30, 2021. Disponível em: <https://doi.org/10.5120/ijca2021921250>.

[9] KAUFMAN, C.; PERLMAN, R.; SPECINER, M. *Network Security: Private Communication in a Public World*. 3. ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010. ISBN 978-0137067928.

[10] NIELSEN, M. A.; CHUANG, I. L. *Quantum Computation and Quantum Information*. 10th anniversary. ed. Cambridge, UK: Cambridge University Press, 2010. ISBN 978-1107002173. Disponível em: <https://www.cambridge.org/core/books/quantum-computation-and-quantum-information/3D59DB6ABF5F942A73BA5EAEF3F25B3D>.

[11] STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. 8th. ed. Boston, MA: Pearson, 2023. ISBN 978-0136006638. Disponível em: <http://williamstallings.com/Cryptography/>.

[12] HANKERSON, D.; MENEZES, A.; VANSTONE, S. *Guide to Elliptic Curve Cryptography*. New York, NY: Springer, 2004. ISBN 978-0387952734. Disponível em: <https://link.springer.com/book/10.1007/b97644>.

[13] PAAR, C.; PELZL, J. *Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Germany: Springer, 2010. ISBN 978-3642041006. Disponível em: <https://link.springer.com/book/10.1007/978-3-642-04101-3>.

[14] WASHINGTON, L. C. *Elliptic Curves: Number Theory and Cryptography*. 2nd. ed. Boca Raton, FL: Chapman and Hall/CRC, 2008. ISBN 978-1420071467. Disponível em: <https://www.routledge.com/Elliptic-Curves-Number-Theory-and-Cryptography/Washington/p/book/9781420071467>.

[15] The OpenSSL Project. *OpenSSL Documentation*. 2024. `https://www.openssl.org/docs/`. Accessed: 2025-06-02.

[16] The Wireshark Team. *Wireshark User Documentation*. 2024. `https://www.wireshark.org/`. Accessed: 2025-06-02.

[17] BHARGAVAN, K. et al. Downgrade resilience in key-exchange protocols. In: *IEEE Symposium on Security and Privacy (S&P)*. [S.l.]: IEEE, 2016. p. 506–525.

[18] Scapy Project. *Scapy: Packet crafting for Python*. 2024. `https://scapy.net`. Accessed: November 2024.

[19] Wireshark Foundation. *Wireshark: Network Protocol Analyzer*. 2024. `https://www.wireshark.org`. Accessed: November 2024.

[20] National Institute of Standards and Technology. *Post-Quantum Cryptography Standardization*. 2024. `https://csrc.nist.gov/projects/post-quantum-cryptography`. Accessed: November 2024.

[21] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. [S.l.], 2024. Disponível em: <https://doi.org/10.6028/NIST.FIPS.203>.

[22] SHOR, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, Society for Industrial and Applied Mathematics, v. 26, n. 5, p. 1484–1509, 1997. Disponível em: <https://doi.org/10.1137/S0097539795293172>.

6 Appendices

6.1 Appendix A: Source Code

This appendix presents the complete source code for the TLS hybrid parser implementation, including the main parser module, PQC group definitions, test suite, and Docker configuration.

6.1.1 A.1 TLS Parser (tls_parser.py)

The main parser module implements the ClientHello analysis functionality using Scapy for packet manipulation and TLS layer dissection.

Listing 6.1 – TLS Hybrid Parser - Main Module

```
"""
TLS Hybrid Parser - Parse de ClientHello com extensoes PQC
"""

from scapy.all import *
from scapy.layers.tls.all import TLS, TLSClientHello
from scapy.layers.tls.extensions import (
    TLS_Ext_SupportedGroups,
    TLS_Ext_KeyShare,
    TLS_Ext_SupportedVersions,
    TLS_Ext_SignatureAlgorithms
)
from scapy.layers.l2 import Ether
try:
    from scapy.contrib.linux_sll import LinuxSLL, LinuxSLL2
except ImportError:
    LinuxSLL = None
    LinuxSLL2 = None
from typing import Dict, List, Optional
import logging

# Configurar logging
logging.basicConfig(
    level=logging.INFO,
    format='[%(levelname)s] %(message)s'
)
logger = logging.getLogger(__name__)


class TLSHybridParser:
    """
    Parser para ClientHello TLS 1.3 com suporte a algoritmos hibridos
    """

    GROUP_NAMES = {
        # Classicos
        0x0017: 'secp256r1',
```

```
        0x0018: 'secp384r1',
        0x0019: 'secp521r1',
        0x001D: 'x25519',
        0x001E: 'x448',

        # Pos-Quanticos (ML-KEM / Kyber)
        0x11EB: 'mlkem768',
        0x11EC: 'mlkem512',
        0x2A01: 'mlkem1024',

        # Hibridos
        0x2F39: 'x25519_mlkem768',
        0x2F3A: 'secp256r1_mlkem768',
        0x2F3B: 'x25519_mlkem1024',

        # Frodo
        0x0200: 'frodo640aes',
        0x0201: 'frodo976aes',
        0x0202: 'frodo1344aes',
    }

    PQC_GROUP_IDS = {
        0x11EB, 0x11EC, 0x2A01,  # ML-KEM
        0x2F39, 0x2F3A, 0x2F3B,  # Hibridos
        0x0200, 0x0201, 0x0202,  # Frodo
    }

    def __init__(self, verbose: bool = False):
        self.verbose = verbose
        if verbose:
            logger.setLevel(logging.DEBUG)

    def parse_packet(self, pkt) -> Optional[Dict]:
        """Parse um pacote que pode conter TLS"""
        # Se tem Raw data, tentar interpretar como TLS
        if pkt.haslayer(Raw):
            raw_data = bytes(pkt[Raw].load)

            # Verificar se comeca com byte TLS (0x16 = Handshake)
            if len(raw_data) >= 3 and raw_data[0] == 0x16:
                try:
                    # Forcar parse TLS do raw data
                    tls_pkt = TLS(raw_data)

                    # Verificar se tem ClientHello
                    if tls_pkt.haslayer(TLSClientHello):
                        return self.parse_client_hello(
                            tls_pkt[TLSClientHello])
                except Exception as e:
                    if self.verbose:
                        logger.debug(f"Erro ao parsear TLS: {e}")
                    return None

        # Fallback: tentar metodo original
        if pkt.haslayer(TLS):
            if pkt.haslayer(TLSClientHello):
                return self.parse_client_hello(pkt[TLSClientHello])

        if self.verbose:
            logger.debug("Pacote nao contem TLS")
```

```
        return None

    def parse_client_hello(self, ch: TLSClientHello) -> Dict:
        """Parse ClientHello e extrai informacoes hibridas"""
        info = {
            'tls_version': self._get_tls_version(ch),
            'cipher_suites': self._get_cipher_suites(ch),
            'supported_groups': [],
            'supported_groups_names': [],
            'key_shares': [],
            'is_hybrid': False,
            'has_pqc': False,
            'pqc_algorithms': [],
            'extensions_count': 0,
            'raw_size': len(bytes(ch))
        }

        # Parsear extensoes
        if hasattr(ch, 'ext') and ch.ext:
            info['extensions_count'] = len(ch.ext)

            for ext in ch.ext:
                if isinstance(ext, TLS_Ext_SupportedGroups):
                    self._parse_supported_groups(ext, info)
                elif isinstance(ext, TLS_Ext_KeyShare):
                    self._parse_key_share(ext, info)
                elif isinstance(ext, TLS_Ext_SupportedVersions):
                    self._parse_supported_versions(ext, info)

        # Analise de hibrido
        self._analyze_hybrid(info)

        if self.verbose:
            self._log_summary(info)

        return info

    # Metodos auxiliares omitidos por brevidade
    # Ver codigo completo em repositorio

    def parse_pcap(self, pcap_file: str) -> List[Dict]:
        """Parse arquivo PCAP completo"""
        logger.info(f"Parseando␣arquivo:␣{pcap_file}")

        results = []

        try:
            packets = rdpcap(pcap_file)
            logger.info(f"Total␣de␣pacotes:␣{len(packets)}")

            for pkt in packets:
                parsed = self.parse_packet(pkt)
                if parsed:
                    results.append(parsed)
                    logger.info(f"ClientHello␣#{len(results)}␣parseado")

            logger.info(f"Parsing␣completo:␣{len(results)}␣"
                        f"ClientHellos␣encontrados")
```

```
        except FileNotFoundError:
            logger.error(f"Arquivo␣nao␣encontrado:␣{pcap_file}")
        except Exception as e:
            logger.error(f"Erro␣ao␣parsear:␣{e}")

        return results


if __name__ == "__main__":
    import sys

    if len(sys.argv) < 2:
        print("Uso:␣python␣tls_parser.py␣<arquivo.pcap>␣[--verbose]")
        sys.exit(1)

    pcap_file = sys.argv[1]
    verbose = '--verbose' in sys.argv or '-v' in sys.argv

    results = parse_pcap_file(pcap_file, verbose=verbose)
```

6.1.2 A.2 PQC Group Definitions (pqc_groups.py)

This module defines the database of classical, post-quantum, and hybrid key exchange groups with their security properties.

Listing 6.2 – PQC Group Database

```
from typing import Dict, Set, List
from enum import Enum
from dataclasses import dataclass

class SecurityLevel(Enum):
    """Niveis de seguranca NIST"""
    LEVEL_1 = 1  # ~128 bits (equivalente AES-128)
    LEVEL_2 = 2  # ~192 bits
    LEVEL_3 = 3  # ~192 bits (equivalente AES-192)
    LEVEL_4 = 4  # ~256 bits
    LEVEL_5 = 5  # ~256 bits (equivalente AES-256)


class GroupType(Enum):
    CLASSICAL = "classical"
    PQC_PURE = "pqc_pure"
    HYBRID = "hybrid"


class PQCGroup:
    def __init__(self, group_id: int, name: str,
                 group_type: GroupType,
                 security_level: SecurityLevel,
                 public_key_size: int,
                 ciphertext_size: int,
                 description: str):
        self.group_id = group_id
        self.name = name
        self.group_type = group_type
        self.security_level = security_level
```

```
        self.public_key_size = public_key_size
        self.ciphertext_size = ciphertext_size
        self.description = description


# Base de dados de grupos
PQC_GROUPS_DB: Dict[int, PQCGroup] = {
    # Classicos
    0x0017: PQCGroup(0x0017, 'secp256r1', GroupType.CLASSICAL,
                     SecurityLevel.LEVEL_1, 65, 65,
                     'NIST P-256 (ECDH)'),

    0x001D: PQCGroup(0x001D, 'x25519', GroupType.CLASSICAL,
                     SecurityLevel.LEVEL_1, 32, 32,
                     'Curve25519 (ECDH)'),

    # ML-KEM (Kyber Padronizado NIST)
    0x11EC: PQCGroup(0x11EC, 'mlkem512', GroupType.PQC_PURE,
                     SecurityLevel.LEVEL_1, 800, 768,
                     'ML-KEM-512 (Kyber512) - NIST FIPS 203'),

    0x11EB: PQCGroup(0x11EB, 'mlkem768', GroupType.PQC_PURE,
                     SecurityLevel.LEVEL_3, 1184, 1088,
                     'ML-KEM-768 (Kyber768) - NIST FIPS 203'),

    0x2A01: PQCGroup(0x2A01, 'mlkem1024', GroupType.PQC_PURE,
                     SecurityLevel.LEVEL_5, 1568, 1568,
                     'ML-KEM-1024 (Kyber1024) - NIST FIPS 203'),

    # Hibridos
    0x2F39: PQCGroup(0x2F39, 'x25519_mlkem768', GroupType.HYBRID,
                     SecurityLevel.LEVEL_3, 1216, 1120,
                     'X25519 + ML-KEM-768 Hybrid'),

    0x2F3A: PQCGroup(0x2F3A, 'secp256r1_mlkem768', GroupType.HYBRID,
                     SecurityLevel.LEVEL_3, 1249, 1153,
                     'P-256 + ML-KEM-768 Hybrid'),
}

# Funcoes auxiliares
def is_pqc_group(group_id: int) -> bool:
    return group_id in ALL_PQC_IDS

def get_group_info(group_id: int) -> Optional[PQCGroup]:
    return PQC_GROUPS_DB.get(group_id)
```

6.1.3 A.3 Test Suite (test_parser.py)

Test suite covering unit tests and integration tests for the parser functionality.

Listing 6.3 – Parser Test Suite

```
import unittest
from pathlib import Path
from tls_parser import TLSHybridParser, parse_pcap_file, is_pqc_group
from pqc_groups import PQC_PURE_IDS, HYBRID_IDS, CLASSICAL_IDS
```

```
class TestTLSHybridParser(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        """Setup executado uma vez antes de todos os testes"""
        cls.parser = TLSHybridParser(verbose=False)
        cls.test_captures_dir = Path(__file__).parent / "test_captures"

    def test_is_pqc_group(self):
        # PQC puros
        self.assertTrue(is_pqc_group(0x11EB))  # mlkem768
        self.assertTrue(is_pqc_group(0x11EC))  # mlkem512

        # Classicos
        self.assertFalse(is_pqc_group(0x001D))  # x25519

    def test_parse_mlkem768_capture(self):
        capture_file = self.test_captures_dir / "mlkem768.pcap"

        if not capture_file.exists():
            self.skipTest(f"Captura nao encontrada")

        results = parse_pcap_file(str(capture_file))

        self.assertGreater(len(results), 0)
        first_ch = results[0]
        self.assertTrue(first_ch['has_pqc'])
        self.assertIn('mlkem768', first_ch['supported_groups_names'])

    def test_key_share_sizes(self):
        """Valida tamanhos esperados de key shares"""
        capture_file = self.test_captures_dir / "mlkem768.pcap"

        if not capture_file.exists():
            self.skipTest("Captura nao encontrada")

        results = parse_pcap_file(str(capture_file))

        for share in results[0]['key_shares']:
            if share['group_name'] == 'mlkem768':
                # ML-KEM768 deve ter ~1184 bytes
                self.assertGreater(share['key_length'], 1000)
                self.assertLess(share['key_length'], 1200)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

6.1.4 A.4 Docker Configuration (docker-compose.yml)

Docker Compose configuration for the isolated hybrid TLS testing environment.

Listing 6.4 – Docker Compose Configuration

```
version: '3.8'

services:
  nginx-oqs:
```

```
    image: openquantumsafe/nginx:latest
    container_name: nginx-hybrid-server
    hostname: nginx-server

    command: nginx -c /etc/nginx/nginx.conf -g 'daemon off;'

    ports:
      - "4433:4433"

    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./certs:/opt/nginx/pki:ro
      - ./shared:/opt/nginx/html:ro
      - ./logs:/opt/nginx/logs

    networks:
      - oqs-network

    restart: unless-stopped

  curl-oqs:
    image: openquantumsafe/curl:latest
    container_name: curl-hybrid-client
    hostname: curl-client

    command: tail -f /dev/null

    volumes:
      - ./curl/test-scripts:/opt/test-scripts
      - ./certs/ca.crt:/opt/ca.crt:ro

    networks:
      - oqs-network

    depends_on:
      - nginx-oqs

    restart: unless-stopped

networks:
  oqs-network:
    name: oqs-hybrid-network
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/16
```

6.2 Appendix B: Usage Examples

6.2.1 B.1 Basic Parser Usage

```
# Parse a single PCAP file
python tls_parser.py capture.pcap

# Parse with verbose output
```

```
python tls_parser.py capture.pcap --verbose

# Run test suite
python test_parser.py
```