Post-Quantum Hybrid KEMTLS Performance in Simulated and Real Network Environments

Alexandre Augusto Giron[1,2], João Pedro Adami do Nascimento, Ricardo Custódio, and Lucas Pandolfo Perin

[1] Federal University of Santa Catarina (UFSC), Florianópolis-SC, Brazil
[2] Federal University of Technology - Parana (UTFPR), Toledo-PR, Brazil
[3] Technology Innovation Institute (TII), Abu Dhabi, UAE
alexandregiron@utfpr.edu.br,joao.pedro.nascimento@grad.ufsc.br,
ricardo.custodio@ufsc.br,lucas.perin@tii.ae

Abstract. Adopting Post-Quantum Cryptography (PQC) in network protocols is a challenging subject. Larger PQC public keys and signatures can significantly slow the Transport Layer Security (TLS) protocol. In this context, KEMTLS is a promising approach that replaces the handshake signatures by using PQC Key Encapsulation Mechanisms (KEMs), which have, in general, smaller sizes. However, for broad PQC adoption, hybrid cryptography has its advantages over PQC-only approaches, mainly about the confidence in the security of existing cryptographic schemes. This work brings hybrid cryptography to the KEMTLS and KEMTLS-PDK protocols. We analyze different network conditions and show that the penalty when using Hybrid KEMTLS over PQC-only KEMTLS is minor under certain security levels. We also compare Hybrid KEMTLS with a hybrid version of PQTLS. Overall, the benefits of using hybrid protocols outweigh the slowdown penalties in higher security parameters, which encourages its use in practice.

1 Introduction

Network protocols such as TLS 1.3 rely on traditional cryptography to provide secure communications. Some schemes in use today are known to be insecure under a Cryptographically Relevant Quantum Computer (CRQC) threat [22]. Since Shor's publication [36], a global-scale transition event is expected before a CRQC arrives. Several entities will have to update their applications and systems to protect their assets against the CRQC threat. Such updates will likely include Post-Quantum Cryptography (PQC), designed with mathematical problems that do not have (known) efficient solutions by quantum and non-quantum computing.

Recent research suggests that the transition to PQC is not trivial; it imposes a high cost, mainly caused by larger cryptographic artifacts (e.g., public keys) [42,25]. In anticipation of this event, researchers have started to evaluate PQC in different scenarios and network protocols. Different entities can be involved in the migration process, such as users, implementers, applications, hardware

devices, government services, and Certificate Authorities (CAs). Each might have different security or urgency requirements regarding the PQC adoption. Security should also be transparent for most users, which means that users should not need to know (or configure) the underlying cryptographic mechanisms of their use cases. While transparency eases cryptography adoption, a consequence might be making users unaware that the type of cryptography used today is vulnerable to a (possible) future quantum attacker.

The TLS protocol transparently employs cryptography for users, except for the cases where mutual authentication is required. The configuration often resides on the server side, where system administrators use X.509 digital certificates provided by CAs [31]. Nevertheless, both sides will require a new version that implements PQC in the protocol. Such a version exists already in the Open Quantum Safe project (OQS) [39], currently in an experimental stage.

Due to the challenges imposed by the increased size (or complexity) of PQC, researchers proposed different alternatives for TLS. One of them is the PQTLS, which deploys PQC as a replacement [24,37]. In PQTLS, two PQC components are used: a Key Encapsulation Mechanism (KEM) allowing peers to establish symmetric keys and a Digital Signature scheme for end-entity authentication. Literature suggests that the PQC adoption for TLS often incurs slowdowns in handshake establishment times. A consequence of the PQC impact is the availability of web services, i.e., the number of connections per second the service can hold with PQC and without it.

A different approach for PQC adoption in TLS is called KEMTLS [33]. It uses KEMs for end-entity authentication as they might be smaller in size than PQC signatures, aiming for better protocol performance. KEMTLS was evaluated with different PQC algorithms and compared with pre-quantum current alternatives (RSA, ECDSA). One of the KEMTLS variants is the KEMTLS-PDK, a KEMTLS with pre-distributed public keys. KEMTLS-PDK targets scenarios where the client already knows the server's long-term KEM public key, allowing the client to perform protocol operations in advance, thus reducing the time to complete the handshake fully.

In the context of PQC adoption, Hybrid modes for Key Exchange (KEX) and authentication are recommended. Hybrids allow participants to secure their communications by combining two types of cryptography: "traditional", also called pre-quantum or classical schemes, which are widely used today, and PQC schemes. For instance, one can combine by concatenating the output of the algorithms (called ingredients), in which (at least) one is pre-quantum and the second is a PQC one [39,33]. After the concatenation, a Key-Derivation Method generates the desired symmetric keys for the communication, and therefore both ingredients contribute to the derivation process.

PQC Hybrid modes are recommended for adoption mainly because they preserve the existing confidence in the security of traditional cryptography. Hybrid modes can be used until the confidence in PQC is fully established, as it may take a long time. Newer PQC schemes might have been less scrutinized when compared to classical ones. Should one of the ingredients be "dismantled" by

a recent-discovered attack, the security holds by the other (unbroken) ingredient of the hybrid protocol. The OQS project recommends PQC hybrids [28], and NIST considers that hybrid modes using the concatenation-prior-derivation approach have compliance with their standards [3]. Standard drafts for using hybrids in TLS began to appear, for example, the "Hybrid key exchange in TLS 1.3" proposal [38]. On the other hand, few authors evaluate PQTLS using the "Hybrid Penalty" metric, i.e., the costs of using the hybrid mode compared to the PQC-only approach. Although dependent on several factors, the penalties in selected algorithms in hybrid configurations can be minor [37].

When writing this work, the KEMTLS alternative for PQC adoption has not yet been analyzed in the hybrid mode. It is vital to evaluate KEMTLS with hybrids because they are the anticipated method for moving real-world applications to PQC. In this context, this paper aims to fill this gap. Our main contributions are:

- a Hybrid KEMTLS design and implementation, adding classical cryptography to all of NIST's Round 3 finalist KEM schemes;
- a framework for experimenting our Hybrid KEMTLS, allowing the community to reproduce our results and carry out further evaluations;
- an extensive evaluation of our approach for Hybrid KEMTLS, considering simulated networks and geographically-distant servers; and
- comparison of Hybrids between KEMTLS, KEMTLS-PDK, and PQTLS, under the same network conditions and security levels, but also with different metrics, such as handshake completion time and "time-to-send-application-data".

We provide an additional contribution result at the application level by employing a load test in an HTTPS Server transmitting web content. This practical experiment allows us to estimate the impacts of transitioning to a similar server configuration with Hybrid PQC.

This paper is structured as follows. Section 2 gives the necessary background on KEMTLS and its variants. Section 3 shows our design for Hybrid KEMTLS (server-only and mutual authentication). All of the configurations and evaluation metrics are described in Section 4. Section 5 presents the results of the experiments, and Section 6 gives the conclusions of this study.

2 Background

2.1 Transport Layer Security

Transport Layer Security (TLS) is a network protocol that provides a confidential and authenticated communication channel. TLS is probably the most used protocol for securing Internet connections. Other use cases for TLS include microservice architectures [41] and VPN connections [33], and it is recommended for standardization in SDN and 5G networks [12,43].

An Authenticated Key Exchange (AKE) is performed in every TLS 1.3 full handshake, generally the first interaction between a client and a server. The

handshake protocol [31] starts with an ephemeral Key Exchange (KEX). It shares secret data between the peers, each peer deriving secrets into a set of traffic keys that protects communication using symmetric cryptography. During the handshake, the TLS server authenticates itself to the TLS client using an x509 digital certificate (or a Pre-Shared-Key). The server can, optionally, request client authentication (called mutual authentication).

Generally, a TLS 1.3 handshake is designed to complete in one Round-Trip-Time (RTT), but there is a 0-RTT mode (for resumptions), and when the client authenticates, it adds another RTT to the protocol. The first handshake message (`ClientHello`) comprises a random nonce, protocol versions, a list of symmetric cipher/HKDF hash pairs supported by the client, and other information. The *key_share* is composed of an ephemeral public key for the KEX. Although optional, at least a keyshare or a Pre-shared-Key (PSK) message must be sent. The server then replies with his corresponding information (`ServerHello` message), but with additional (optional) messages, such as the `Certificate` and `CertificateVerify`. These two are part of the TLS authentication using digital signatures. An alternative is the server authentication through PSK, e.g., in a session resumption, where the server does not send the certificate. The authentication ends with the `Finished` message, where an HMAC is computed from the transcript of the handshake context. All of the keys required for the encryptions in TLS 1.3 are derived based on the HKDF function [19].

2.2 Post-Quantum Cryptography

One of the main problems in TLS is that the cryptographic computations in TLS 1.3 AKE do not protect against quantum attackers. All of the cryptography based on the Integer Factorization Problem (IFP), Discrete Logarithm Problem (DLP), and Elliptic Curve Discrete Logarithm Problem (ECDLP) is vulnerable to Shor's algorithm [36]. Although powerful-enough quantum computers are not publicly available (at the time of this writing), this threat is worrisome considering if an attacker is capturing TLS packets to decrypt them in the future. In order to solve this issue, researchers started to study and develop Post-Quantum Cryptography (PQC) and its adoption in TLS (and other protocols) [14,24,37]. PQC includes schemes based on different mathematical problems in which there is no (known) efficient solution by both classical and quantum computers [7].

Public-key cryptography is often used for Key Exchange (KEX) and peer authentication in network protocols. In order to protect network communications from quantum adversaries, PQC schemes must be included. A Key-Encapsulation Mechanism (KEM) and a Digital Signature from PQC are required, or at least one PQC KEM [33]. However, one of the main challenges of selecting PQC schemes is their increased size compared to classical schemes [39]. Table 1 allows comparing sizes of some PQC and classical algorithms used for KEX (for a PQC signature sizes comparison, please refer to [42]). The sizes are different for each NIST security level, where each level means that the scheme is as hard to break (using exhaustive key search) as symmetric AES-128 (level one), AES-192 (level three), and AES-256 (level five) [21]. In this work, we will

Table 1: Classical and PQC schemes comparison

Algorithm Name	Parameter Set Name	Public Key size (bytes)	Claimed NIST Security Level	Quantum-safe?
NIST P256	`secp256r1` (L1)	64	1	✗
NIST P384	`secp384r1` (L3)	96	3	✗
NIST P521	`secp521r1` (L5)	132	5	✗
Kyber	`Kyber512` (KyberL1)	800	1	✓
	`Kyber768` (KyberL3)	1184	3	✓
	`Kyber1024` (KyberL5)	1568	5	✓
Saber	`LightSaber_KEM` (Saber L1)	672	1	✓
	`Saber_KEM` (Saber L3)	992	3	✓
	`FireSaber_KEM` (Saber L5)	1312	5	✓
NTRU	`NTRU-HPS-2048-509` (NTRU L1)	699	1	✓
	`NTRU-HPS-2048-677` (NTRU L3)	930	3	✓
	`NTRU-HPS-4096-821` (NTRU L5)	1230	5	✓

use a naming convention where the algorithm name is followed by an indication of the corresponding security level. For example, we will place "KyberL5" to mean that `Kyber1024` was used.

Currently, the most notorious PQC standardization effort is being conducted by the US National Institute of Standards and Technology (NIST) [23]. The PQC proposals submitted to the process can be classified in the following groups [7]: *lattice-based cryptography*, which uses linear algebra constructions; *code-based cryptography*, which uses error-correcting codes; *multivariate-based cryptography*, using systems of multivariate equations; *isogeny-based cryptography*, based on Supersingular elliptic curve isogenies; and *hash-based cryptography*, using cryptographic hash functions.

The NIST process is divided into "selection rounds". Several schemes were submitted to the first round, some were broken, and others were merged. In the third round NIST announced four finalists for KEMs, namely: Classic McEliece, Kyber, Saber, and NTRU, and three for digital signatures: Dilithium, Falcon, and Rainbow. At the time of this writing, the end of the third round of the process defined which algorithms will be standardized:

- KEM: **Kyber**, based on lattices, used for Key Exchange and Public-Key Encryption; and
- Digital Signatures: **Dilithium** (primary choice), based on lattices; **Falcon** and **Sphincs+**, based on lattices and hash functions, respectively.

Several international agencies are following the NIST process. A fourth round is going on, and other algorithms will be scrutinized. The KEMs selected for the fourth round are BIKE, Classic McEliece, HQC, and SIKE [40] (however, both Rainbow and SIKE are now considered broken [8,10]). Besides, NIST calls for new digital signature schemes, particularly interested in those with a small signature size (compared to other PQC schemes). Even though the process has

yet to be concluded, some protocol implementations like OpenSSH use PQC in their connections by default. This example indicates that network protocols will gradually start adopting PQC schemes in their designs.

2.3 Post-Quantum Adoption in TLS

One of the challenges of adopting PQC for TLS is the size of the protocol's messages, as most prominent PQC algorithms substantially increase the size of cryptographic objects, consequently increasing the protocol's payload sizes. For example, some PQC algorithms cannot be deployed in the TLS protocol "as-is", which means changing the protocol is required to fit the algorithm best. Besides, network control mechanisms on the Internet, such as the TCP Congestion Control, can impose a performance slowdown when message sizes surpass a predetermined threshold. Therefore, their adoption in network protocols can be a problem.

The term "hybrid" means that both PQC and traditional algorithms are used in conjunction, where the security of this construction is combined. Only after the confidence in PQC is fully established is it recommended to abandon the classical cryptography schemes. Here, we refer to classical schemes built on IFP, DLP, and ECDLP-based cryptography. Such schemes have higher confidence, years of utilization, and have been standardized (such as the NIST ones [4,5]). On the other hand, the NIST standardization process for PQC schemes still needs to be completed. Therefore, hybrids are recommended before a complete transition to PQC adoption. Worth mentioning that, in this work, the term hybrid does not include Quantum Key Distribution (QKD) [27] and should not be confused with Hybrid Encryption (HE) [20], which is a combination of Public-Key schemes it with symmetric cryptography.

Therefore, hybrid designs accompanied by their performance and security analysis are essential. Besides, understanding the penalties when transiting to hybrid modes contributes to the PQC adoption. One important consideration regarding the Hybrid PQC design is the *cryptographic combiner* in use [9]. The combiner is responsible for keeping security if one of the combined schemes is not secure anymore. Besides, performance evaluations are necessary to understand the impacts of Hybrid PQC utilization (compared to classical schemes and PQC-only approaches).

Although changing a protocol design to fit a cryptographic algorithm deviates from the purpose of **crypto agility**, the Post-Quantum scenario shows that different protocol designs can improve performance. In this context, KEMTLS [33] arises as a size-optimized alternative for TLS. KEMTLS is designed with PQC KEMs, which have, in general, smaller sizes compared to Digital Signatures. The design includes an ephemeral KEM and a second KEM for long-term usage. This way, the digital signature present in every (full) TLS 1.3 handshake is replaced with KEMs.

2.4 KEMTLS variants

KEMTLS-PDK [34] is a variant of the KEMTLS with pre-distributed keys, making it suitable for scenarios where the client knows the server's long-term public key prior to the communication. In this way, the client can perform an encapsulation against the server's long-term public key in his first round, sending the resultant ciphertext through the `ClientHello` extension `ext_pdk`. Then the server decapsulates the client's ciphertext and plugs the shared secret into the key-derivation schedule at the earliest possible stage when deriving the Early Secret.

In KEMTLS, there are conceptually three phases: Ephemeral key exchange using KEMs, implicitly authenticated key exchange using KEMs, and confirmation/explicit authentication. The client can send application data after 1 round, during the second phase, when the server is implicitly authenticated. However, the handshake is fully completed in 2 rounds, only after the third phase when the server is explicitly authenticated. KEMTLS-PDK improved the number of rounds, where the server is explicitly authenticated after only 1 round. This earlier authentication of the server in the protocol flow means that the handshake is fully completed at the same time the client can send application data. The 1-RTT time of the KEMTLS-PDK makes it more competitive with TLS in terms of time to complete the handshake fully.

Celi et al.[11] provided a Go-lang implementation of KEMTLS, KEMTLS-PDK, and PQTLS through a fork of the Go Standard Library. The server's certificate contains a delegated credential in this implementation, which uses the PQC signature algorithm for PQTLS, or the PQC KEM algorithm for KEMTLS and KEMTLS-PDK. This approach decouples the handshake authentication algorithm from the authentication algorithms used in the certificate chain, allegedly facilitating the cryptographic agility principle in the protocol's design. The algorithms used in this implementation are taken from the CIRCL [13] library, which implements many post-quantum algorithms natively in Go. For the tests, the PQC signature algorithms used were Dilithium3 and Dilithium4, while the PQC KEM algorithms were Kyber512 and Sikep434.

A variant of KEMTLS specifically made for scenarios with mutual authentication is presented in KEMTLS with Delayed Forward Identity Protection [17]. In their approach, the privacy of the client is a primary concern. In their proposal, an adversary that knows the server's private key would not be able to disclose the client's identity, and the approach provides a guarantee that the client's identity will only be recoverable by an authenticated server. In order to do so, the client knows the server's public key beforehand, and the client's certificate must be sent encrypted. However, encrypting it with the server's public key is not enough if its private key is compromised. Therefore, an ephemeral contribution of the server is necessary to generate the key that will encrypt the client's certificate. This contribution happens in the second-round trip of KEMTLS, increasing the handshake latency.

Considering the above problem, Felix Günther et al.[17] proposes a protocol that provides forward identity protection in 1-RTT solely based on KEMs. They

introduced a semi-static server key in KEMTLS, also known in advance by the client. The client will encapsulate against the semi-static and long-term keys, obtaining their respective shared secrets and feeding them to a key derivation function. Lastly, the approach will generate a symmetric key for encrypting the client's certificate. Since semi-static keys are not assumed to be certified, when a new semi-static key is generated, it is sent through a handshake that consists of two round trips. These semi-static keys are refreshed periodically. Thus if the semi-static key is not compromised before it expires, the client's identity remains private even if the server's long-term private key is compromised. In summary, the authors show that introducing semi-static keys has a negligible cost for the handshake.

All of the KEMTLS variants proposed do not consider the hybrid PQC design. A Hybrid PQC design is composed of (at least) two algorithms, a PQC one and a classical one, where the security properties must hold as long as one of the ingredients remains unbroken [9]. However, hybrids are considered the first step in the PQC transition process. Some works evaluate hybrid designs in PQTLS [24,37], but none use KEMTLS as the PQC adoption approach. Next, we present our hybrid design for KEMTLS.

3 Hybrid KEMTLS Design

The Hybrid KEMTLS design is shown in Figure 1, considering a handshake configuration with server-only authentication. A provides the design with mutual authentication. The client and server cryptographic computations are described on each side, where the ones highlighted with square dotted boxes are the modifications to the original KEMTLS. This design adds classical cryptography computations to build the hybrid, but in a way that it keeps the same properties as the original KEMTLS offers, namely: offline deniability, and a smaller trusted codebase. We are not using signatures in the handshake, but signature algorithms are required in the certificate chain used.

The KEMTLS protocol is based on two KEM keypairs, an ephemeral and a static one. Therefore, to "hybridize" it, we added new classic KEM keypairs to the protocol. We instantiated the Hybrid KEMTLS with the following classic KEM's: `KEM_P256_HKDF_SHA256`, a KEM using `P256` curve and HKDF with `SHA-256`; `KEM_P384_HKDF_SHA384`, a KEM using `P384` curve and HKDF with `SHA-384`; and `KEM_P521_HKDF_SHA512`, a KEM using `P521` curve and HKDF with `SHA-512`. All classical KEMs we use are obtained from CIRCL library [13]. This enables us to assess a hybrid at any NIST security level.

The Hybrid KEMTLS algorithm names are composed of two parts. The first part of the name is the classic KEM curve (`p256`, `p384` or `p521`), and the second part of the name is the PQC KEM algorithm name. For example, `P256_Kyber512` stands for a Hybrid KEMTLS instance with the `KEM_P256_HKDF_SHA256` classic KEM and `Kyber512` PQC KEM. Following the naming convention used in this work, we add the letter "H" for instantiations in hybrid mode (e.g., KyberL1 H.). We also integrate the PQC algorithm using all parameters for the available

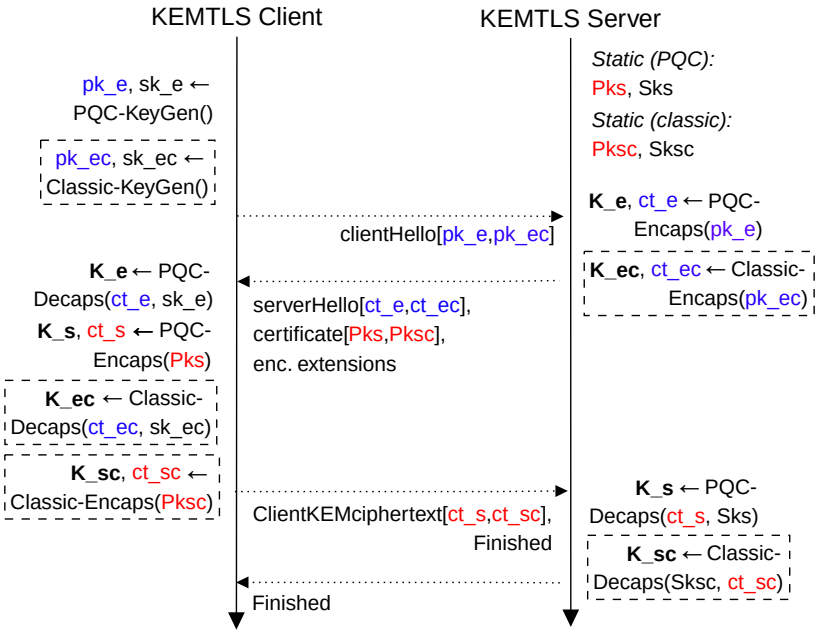

Fig. 1: Proposed Hybrid KEMTLS Handshake (Server-only authentication)

security levels. Then, we had to map each classical algorithm instantiation (with the selected parameters) to the corresponding PQC instantiation, allowing us to combine correctly at the same NIST security level.

The Hybrid KEMTLS protocol works similarly to the KEMTLS, but now each cryptographic operation has its classic counterpart, as the square dotted boxes highlight it. The first step of the protocol, in terms of cryptographic operations, is the key pair generation for both the PQC and Classic ephemeral KEMs. The resultant public keys pk_e and pk_ec are concatenated and transmitted to the server through the `key_share` extension from the `ClientHello` TLS message. Upon reception of these keys, the server performs an encapsulation against each one, generating the ephemeral shared secrets K_e (post-quantum) and K_ec (classic), and ciphertexts ct_e (post-quantum) and ct_ec (classic). After that, both ciphertexts are sent back to the client through the `key_share` extension of the `ServerHello` message. In this same flight, the `EncryptedExtensions` and `Certificate` messages are sent. The `Certificate` message holds an X.509 Certificate composed of two static KEM public keys concatenated (classic and PQC).

Upon receiving the server reply messages, the client decapsulates both the post-quantum ciphertext and the classic ciphertext, obtaining the same two ephemeral shared secrets as the server, K_e, and K_s. Then the client encapsulates against the server's static KEM public keys, resulting in the static shared

secrets K_s and K_sc and ciphertexts ct_s and ct_sc, which are concatenated and sent to the server through the KEMTLS's `ClientKEMCipher text` message along with the Finished Message. The server then decapsulates the received ciphertexts obtaining the same static shared secrets K_s and K_sc as the client. After that, the server can send the Finished message back to the client, finishing the Hybrid KEMTLS Handshake.

The Hybrid KEMTLS proposed Key Schedule follows the KEMTLS Key Schedule, inspired by the TLS 1.3 Key Schedule [31]. The TLS 1.3 Key Schedule is based on a sequence of calls to HKDF extract-and-expand functions. HKDF-Extract takes an Input Keying Material (IKM) and a Salt, generating an output secret, whereas HKDF-Expand function takes a secret, a label, and a transcript hash (described by RFC 8446 [31] - Section 4.4.1). The HKDF-Extract output secret is the input secret for the HKDF-Expand, which generates a protocol traffic secret derived into a TLS 1.3 traffic key, according to RFC 8446 [31] Section 7.3.

KEMTLS slightly modified this Key Schedule by partially removing TLS 1.3's Early Secret stage and adding a new stage between TLS 1.3's Handshake Secret (HS) and Master Secret. This stage produces the Authenticated Handshake Secret (AHS). In KEMTLS, the HS is obtained through an HKDF-Extract of the ephemeral KEM shared secret and the AHS through the static KEM shared secret. In this way, both types of shared secrets contribute to generating the application traffic keys.

In this work, our Hybrid KEMTLS Key Schedule (Figure 2) has the same construction as the KEMTLS Key schedule, differing only in the data used as Input Keying Material to the Handshake Secret and the Authenticated Handshake Secret. Since the Hybrid KEMTLS requires a cryptographic combiner that keeps the security if one of the algorithms is broken, the IKM used to produce the Handshake Secret is the concatenation of the shared secrets K_e and K_ec. At the same time, the IKM used to the Authenticated Handshake Secret is the concatenation of shared secrets K_s and K_sc.

Regarding the concatenation (prior to KDF), we justify this choice based on the NIST standards for key derivation that allows such a construction [3]. Since all shared secrets are used, if a hybrid mode is selected, each K contributes to the generation of the traffic keys, requiring both parties (i.e., client and server) to support all the algorithms of the selected hybrid mode. However, this should not be a significant issue since we add one algorithm (pre-quantum) cryptography to KEMTLS, which is commonly used nowadays. As KEMTLS, our design allows us to select the same algorithm for KEX and authentication, which can simplify the negotiation procedure of the protocol.

3.1 Security Analysis

Schwabe et al. [33] describe KEMTLS security aspects. We did not modify message flows in the KEMTLS protocol, avoiding abrupt changes that would impact their security analysis. In hybrid modes, the security relies mainly on the cryptographic combiner [16,9], which is the focus of our analysis here. We rely on

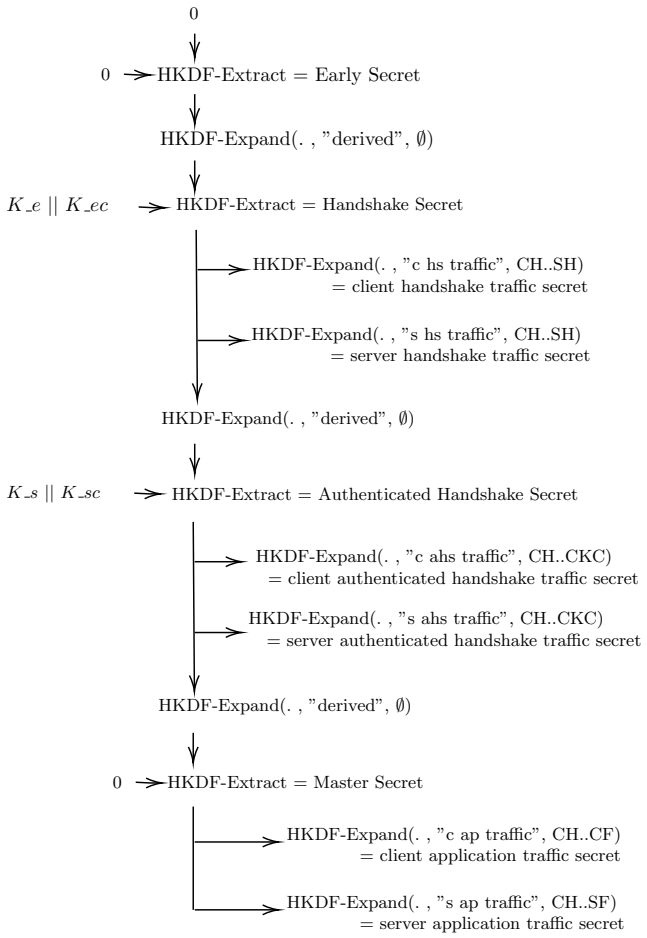

Fig. 2: Hybrid KEMTLS Proposed Key Schedule

the following assumptions: PRF, and Dual-PRF (Definitions 1 [15] and 2 [6], respectively), which corresponds to the assumptions present in Bindel's et al. `dualPRF` combiner (Definition 3) [9].

Definition 1 (Pseudorandom Function). *Let $\mathcal{F} : Dom \longrightarrow Rng$ be a function family from domain $Dom = \{0,1\}^n$ to image set $Rng = \{0,1\}^m$. A function $f : Keys \times Dom \longrightarrow Rng$, where $Keys = \{0,1\}^s$, is a* PRF *if the following properties hold:*

1. *For $x \in \{0,1\}^n$ as an input and $k \in \{0,1\}^s$ as a key, $f(k,x)$ can be computed by a polynomial-time algorithm.*
2. *Let $\mathcal{A}$ be a probabilistic* PRF*-adversary algorithm, bounded by time-complexity t. Then, the* PRF*-advantage of $\mathcal{A}$, is:*

$$\mathbf{Adv}^{prf}_{\mathcal{A},f}(\kappa) := \left| Pr_{K \leftarrow \$\{0,1\}^s}[\mathcal{A}^{f_K}] - Pr_{f \in \mathcal{F}}[\mathcal{A}^f] \right| \leq \text{NEGL}(\kappa)$$

where NEGL() *is a negligible function and* κ *a security parameter.*

Definition 2 (Dual-PRF). *Let* $\mathcal{S} : S_0 \times S_1 \longrightarrow S.Out$ *be a function family. Let* $\mathcal{S}^{\text{swap}} : S_1 \times S_0 \longrightarrow S.Out$*. If both* $\mathcal{S}$ *and* $\mathcal{S}^{\text{swap}}$ *are PRFs, then* $\mathcal{S}$ *is a Dual-PRF.*

Definition 3 (`dualPRF` Combiner). *Let* $\mathcal{K}_1$ *and* $\mathcal{K}_2$ *be two Key Encapsulation Mechanisms (KEMs). Let* k_i *and* c_i *be the output of an encapsulation from* $\mathcal{K}_i$*,* $1 \leq i \leq 2$*. If* ***PRF*** *is a pseudorandom function (Definition 1) and* ***dPRF*** *is a dual-PRF (Definition 2), then* ***dualPRF****[*$\mathcal{K}_1$*,* $\mathcal{K}_2$*,* ***dPRF****,* ***PRF****] combines two KEMs with output* C *as follows:*

$$C = \textit{PRF}(\textit{dPRF}(k_1, k_2), (c_1, c_2)).$$

Bindel et al. [9] showed an example instantiation of `dualPRF` using `HKDF.Extract` and `HKDF.Expand` functions, aiming at combining KEMs securely. In their work, they prove the security of `dualPRF` combiner, i.e., `dualPRF`(K_1,K_2) is IND-CCA secure as long as K_1 or K_2 are IND-CCA secure KEM, and the Dual-PRF assumption (Definition 2) holds. Our approach for combining KEMs in the hybrid mode is based on the `dualPRF` proposal. The main difference is that we have to duplicate `dualPRF` usage. Given that Bindel et al. [9] demonstrated the security of `dualPRF` combiner, we can prove that our Hybrid KEMTLS combiner is also secure.

Theorem 1 (Hybrid KEMTLS combiner security). *Let* $\Pi = [\mathcal{C}_2(Ke_1,Ke_2), \mathcal{C}_1(Ks_3,Ks_4)]$ *be the hybrid KEMTLS combiner, where the cryptographic combiner* $\mathcal{C}_i$ *is instantiated using a secure* ***dualPRF*** *(Definition 3),* Kx_j *is a Key-Encapsulation Mechanism (KEM) with* x *referring to ephemeral KEM or static KEM, an odd* j *to a classical KEM and an even* j *to a PQC KEM. Then, for any adversary* $\mathcal{A}_\Pi$ *against* Π*,*

$$\text{Adv}^{\textit{dualPRF}}_{\mathcal{A},\Pi} := \text{Adv}_{\mathcal{A}_1,\mathcal{C}_1} + \text{Adv}_{\mathcal{A}_2,\mathcal{C}_2} \leq \text{NEGL}(),$$

where NEGL() *is a negligible function.*

Proof. Recall that a dual pseudorandom function is a `PRF` even when one of its inputs is random, but the other is. Mapping this concept to hybrid mode, even if $\mathcal{A}_1$ could (hypothetically) distinguish one of the KEM's output from random, the other KEM holds the security because `dualPRF` output is still random due to our assumption. From Figure 2, the keys K_e, K_{ec}, K_s, and K_{sc} produced by KEMs are concatenated prior to derivation with HKDF, i.e.,

$$\text{Handshake Secret} \leftarrow \texttt{HKDF-Extract}\ (K_e||K_{ec}, \texttt{HKDF-Expand}(.\ , \text{“derived”}, \emptyset)),$$
$$\text{and}$$
$$\text{Authenticated Handshake Secret} \leftarrow \texttt{HKDF-Extract}\ (K_s||K_{sc}, \texttt{HKDF-Expand}(.\ , \text{“derived”}, \emptyset)),$$

where each HKDF call is chained and dependent on the previous call, as shown in the key schedule (Figure 2). Besides, in the protocol, successive HKDF calls for derivation use the handshake transcript hash, which in turn includes KEM's ciphertexts, matching the `dualPRF` definition (Definition 3). The design combines secrets from two ephemeral KEMs (classical and PQC) using `dualPRF`, producing the Handshake Secret. Then it combines the long-term secrets, also using `dualPRF` in the same way, producing the Authenticated Handshake Secret. So, if $\mathcal{A}_1$ breaks $\mathcal{C}_1$ with non-negligible probability, it means that, given the Handshake Secret, $\mathcal{A}_1$ can recover the concatenated output of the KEMs $(K_e||K_{ec})$. Therefore, we can derive an adversary $\mathcal{B}$ that breaks Dual-PRF (Definition 2), but this contradicts our assumption. By symmetry, the same applies to $\mathcal{A}_2$. Hence, our combination has the same security guarantees as `dualPRF`.

3.2 Modelling Cost of the Instantiations

Depending on the algorithms selected for instantiating Hybrid KEMTLS, an associated cost will be added to the protocol in terms of size. The algorithm parameters define the sizes, and then we can model an estimate of the impact in the protocol. In our design, the protocol cost C_{size} can be defined as:

$$C_{size} = \texttt{KEX}_{size} + \texttt{Auth}_{size} + \texttt{Certificates}_{size} + \texttt{Metadata}_{size},$$

where $\texttt{KEX}_{size}$ refers to the size of the protocol's Key Exchange, here including `ClientHello` and `ServerHello` messages; $\texttt{Auth}_{size}$ is the size of KEMTLS server-only or mutual authentication ciphertexts; $\texttt{Certificates}_{size}$ is the sum of sizes of the server certificate plus Intermediate CA certificate; and $\texttt{Metadata}_{size}$ is the sum of other protocol-related data sizes. If mutual authentication is required, two additional messages must be considered: `Server KEMCiphertext` and the client's `Certificate`. Section 4 describes additional certificate chain configurations used in this work.

Table 2 shows the protocol-level cost of hybrid KEMTLS instances in size (in bytes). It considers cryptographic object sizes and additional protocol data (e.g., ECDHE group numbers) compared to a baseline TLS 1.3 configuration. The PQC algorithms are finalists of the NIST process (Round 3). The last two columns show the total size for Server-only authentication (the letter 'S') and Mutual authentication (the letter 'M'). We omitted the metadata column, but it adds approximately 320 bytes to C_{size}, or 540 for mutual authentication. We kept X509 metadata inside the Certificates column. Mutual authentication shows similar result but adds 540 as metadata value, and it approximately doubles the columns 'Auth' and 'Certificates'. Using this table, one can see the cost of using the Hybrid KEMTLS instance we proposed. Besides, it allows estimating impact in networks. For example, all `P521` instances are close to (default) TCP congestion window size (`cwnd`), measured by the number of segments. Typically, the Maximum Segment Size (MSS) is close to MTU size (e.g., 1460, 1500 bytes), and the `cwnd` defaults to 10 MSS. Due to the TCP congestion control, hybrids with greater size will incur additional round-trips.

Table 2: Expected cost (bytes) of hybrid KEMTLS instances

Hybrid Instance	Security Level	KEX	Auth	Certificates	C_{size} S	C_{size} M
`P256` (Baseline)	1	484	74	839	1715	2847
`P256_Kyber512` (KyberL1 H.)	1	2052	833	7807	11012	19872
`P384_Kyber768` (KyberL3 H.)	3	2820	1185	10708	15033	27141
`P521_Kyber1024` (KyberL5 H.)	5	3756	1701	14475	20252	36650
`P256_LightSaber_KEM` (SaberL1 H.)	1	1892	801	7679	10692	19390
`P384_Saber_KEM` (SaberL3 H.)	3	2628	1185	10514	14647	26564
`P521_FireSaber_KEM` (SaberL5 H.)	5	3404	1605	14220	19549	35595
`P256_NTRU_HPS_2048_509` (NTRU L1 H.)	1	1882	764	7707	10673	19363
`P384_NTRU_HPS_2048_677` (NTRU L3 H.)	3	2408	1027	10454	14209	25909
`P521_NTRU_HPS_4096_821` (NTRU L5 H.)	5	3080	1363	14138	18901	34623

We also model computational cost our hybrid KEMTLS design. The focus of this model is on the cost of cryptographic computations since they are the main changing parts of our proposal. In this case, we can define the client's computational cost when using hybrid KEMTLS as $C_{clientops}$:

$$C_{clientops} = \texttt{HKeyGen}_{time} + \texttt{HDecaps}_{time} + \texttt{HEncaps}_{time} + 2 * \texttt{HVerify}_{time},$$

where $\texttt{HKeyGen}_{time}$ and $\texttt{HDecaps}_{time}$ are KEM operations (ephemer-al) and $\texttt{HEncaps}_{time}$ and $2 * \texttt{HVerify}_{time}$ are authentication operations (with long-term keys). Although a KEM is used, two signatures must be verified to complete the authentication (server and Intermediate CA certificates), for each algorithm (PQC and Classical). For the server-side, the cost $C_{serverops}$ has the main difference: it does not have $\texttt{HVerify}_{time}$ involved, nor signing operation, only: $\texttt{HEncaps}_{time}$ for KEX, $\texttt{HDecaps}_{time}$ for authentication. Lastly, the 'H' letter represents the hybrid mode, requiring two operations: one for the PQC and the other for the classical algorithm. Note, however, that are other costs when using the hybrid protocol in practice (e.g., network latency time).

4 Evaluation Characteristics

We use two environments for the experiments: geographical-distant servers and simulated network experiments. The first experiment uses two Google N2 VMs, configured with Intel(R) Xeon(R) 2.80GHz CPU and 8 GiB RAM. They are located in different geographical regions: `europe-central2-a` (Warsaw, Poland) and `southamerica-east1-a` (São Paulo, Brazil), with an average latency of 108ms. The simulated experiment uses NetEm [18] with two parameters: latencies (2ms, 10ms, 100ms, 300ms); and packet loss probabilities (1, 2, 3, 5%). The simulations were executed in an Intel i5-8250U 1.60GHz, with turbo boost technology disabled. In both experiments, 1000 handshakes were evaluated, one at a time.

Unlike TLS 1.3 handshake, KEMTLS requires an additional round-trip to authenticate the server and complete the handshake. Figure 3 shows the differences

considering a Post-Quantum TLS 1.3 and KEMTLS handshakes. In both protocols, the handshake completion time occurs when a peer receives a `Finished` message. Therefore, when comparing the protocols, the `Finished` message completes the handshake at different moments. KEMTLS handshake completion time will be longer than PQTLS due to the added round-trip. However, from a practical perspective, the client can send application data at the exact handshake moment in both protocols. Note that the client is usually the party that initiates a communication (e.g., with an HTTPS request).

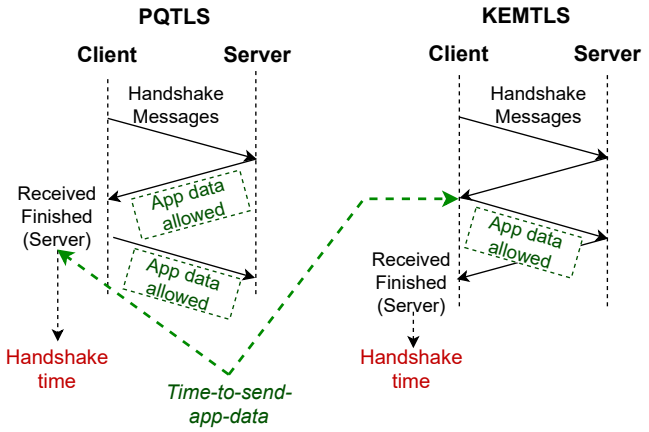

Fig. 3: Comparison of Handshake completion time in KEMTLS and PQTLS

Considering a future adoption scenario for the Hybrids, we also provide a "load test" evaluation of an HTTPS server instantiated. The server is instantiated with both PQTLS and KEMTLS hybrid versions. We use `go-bench` as a web server benchmarking tool [35]. We measure the concurrent requests sent by go-bench to assess how many of them the server can handle (successfully or not). We deploy a webserver that transmits 2 MB of random data for this test, close to the median webpage weight in 2021 [1]. For this benchmark, we also increased the number of maximum open file descriptors in the server (using `ulimit -n 1048576` Linux command).

We integrate our hybrid versions in go-bench to perform this test over 128, 256, 512, and 1024 concurrent client threads. The experiment was allowed to run for 6 minutes for each hybrid instantiation and each number of concurrent client threads. An additional parameter in go-bench is the HTTP Keep-alive connections that we set to `false`, which prevents the server from reusing a previously established connection. We monitor the server's process peak memory usage during the test using `grep ^VmHWM /proc/<PID>/status` Linux command.

Our tests are based on the NIST Round 3 Finalists regarding the algorithm selection for the experiments. We evaluate scenarios with `Kyber`, `Saber`, and

`NTRU` variants, in the three security levels, always in hybrid mode (with NIST curves). `Classic-McEliece` has a large public key that exceeds the TLS message size limits, making it only suitable for scenarios where the public key is not transmitted. For this reason, we evaluated it separately in the Hybrid KEMTLS-PDK in a load test scenario, presented in section 5.3. In KEMTLS, we use a KEM-based ECDH and a PQC KEM to compose our hybrid scheme. A similar configuration is used in PQTLS, but for authentication, we used `Falcon` and `Dilithium`, also in hybrid mode. In both protocols, we fixed the certificate chain to be hybrid and only one hybrid algorithm, chosen to be Dilithium. The choice of a fixed certificate chain allows for comparing KEMTLS and PQTLS at the protocol level (regarding handshake operations). Hence the same chain is used for both protocols. Other options would include a classical certificate chain, mixed-chains [25], or PQC-only chains, but this work focuses on the transition with Hybrid PQC.

Additionally, RFC 8446 allows peers to avoid sending Root CAs certificates [31]. Therefore, the `Certificate` message in the protocol includes two certificates: the server (or client) certificate and the Intermediate CA certificate. Therefore we followed this scenario where only the Intermediate CA certificate is sent. We believe this scenario is more suitable for minimizing the impact of the PQC adoption, though it assumes that both peers have (and trust) the Root CA certificate pre-installed (and trust).

Another design decision made for the experiments is that the same NIST security level is used for all algorithms part of the handshake protocol. Namely, the KEX, the authentication, and the certificate chain are at the same security level. Since the focus here is to analyze performance impact, the results will correspond to the selected security requirement. Therefore, we expect better performance for the minimum security level and the worst performance impact for the maximum (NIST) security level. In this way, the experiments would provide minimum and maximum indicators.

We adapted a Golang repository implementation from Celi et al. [11] for our experiments. We "hybridize" their KEMTLS and PQTLS implementation using NIST curves `p256`, `p384`, or `p521`, accordingly to the security level of each algorithm version. We also implement a new testbed to support different PQC algorithms and allow the reproducibility of our experiments. We use liboqs-go wrapper [30] from the OQS project to integrate all algorithms in KEMTLS.

In the context of hybrid adoption and aiming at a fair comparison, our evaluation has the following metrics:

1. Handshake Completion Time: at the client, it initiates from the start of the protocol until it receives the `Finished` message.
2. *Time-to-Send-App-Data*: the required time for the client send application data to the server.
3. Hybrid Penalty: if H is the measured time in a hybrid instantiation and P is the PQC-only time, we subtract $H - P$, resulting in the hybrid penalty.
4. HTTPS/TLS request successes and failures: the number of successful requests and failures for the server's load test.

5. Server-side Memory load: the process memory peak usage for the server's load test.

In summary, metric 1,2,3 are measured in the client's protocol implementation, whereas metrics 4 and 5 are metrics for evaluating the server. Table 3 summarizes the characteristics and environments of the experiments. The experiments can be reproduced as we provide both the hybrid KEMTLS repository and the test repository [2]. Raw results are also publicly available there.

Table 3: Description of the experiments performed in this work

Experiment	Environment	Metrics used	Protocols Compared
KEMTLS Hybrid Penalty	Geo. Distant server	1,2,3	PQC-Only and Hybrid KEMTLS
	Simulated Network	1,2,3	PQC-Only and Hybrid KEMTLS
Hybrid Comparison	Geo. Distant server	2	Hybrid PQTLS and Hybrid KEMTLS
	Simulated Network	2	Hybrid PQTLS and Hybrid KEMTLS
Load Test	Geo. Distant server	4,5	Hybrid PQTLS and Hybrid KEMTLS

5 Hybrid KEMTLS Evaluation

The evaluation is divided into three parts. Section 5.1 discusses the hybrid penalties found in the experiments. Section 5.2 compares hybrid KEMTLS to hybrid PQTLS performance. Lastly, Section 5.3 discusses the web server load-testing experiment using hybrids. Note that we will use the algorithm naming or parameter set naming for referring to PQC algorithms (and their security levels) as shown in Table 1.

5.1 Hybrid Penalty

The Hybrid Penalty in KEMTLS is the cost of adding a hybrid mode over the PQC-only configuration. This section presents the KEMTLS analysis, starting with the geographical-distant server experiment, followed by the Simulated Environment results.

Geographical-Distant Server This experiment evaluates Hybrid KEMTLS (and KEMTLS-PDK) using geographically separated peers, considering handshake completion times. Figures 4 to 8 present the box plots for the KEMTLS timings of the three NIST security levels, considering server-only authentication. Figures 5 to 9 correspond to the KEMTLS-PDK results.

In the geographical-distant scenario, we do not control network variations or load variations that might occur in the Google VMs. On the other hand, such a test environment better reflects a realistic scenario. In this scenario, we observed different results regarding the hybrid penalties (i.e., the cost of adopting

our hybrid). The box plots for security level 1 show overlapping timings, which means that the penalties are minor, regardless of the tested algorithms. However, increasing the security level, the boxes no longer overlap, increasing the penalty. As seen in Figure 8, with level 5, the hybrid penalties are much more significant than the other levels caused by the classical algorithm.

Due to design differences, KEMTLS-PDK obtained better timings because of its reduced number of RTTs. The RTT conceals most cryptographic operation timings in such a geographical-distant connection. When analyzing the hybrid penalties, KEMTLS and KEMTLS-PDK have similar results. These results favor hybrid versions but not much at higher security levels. B shows results for hybrid KEMTLS using mutual authentication, where the penalties have similar behavior, corroborating these findings.

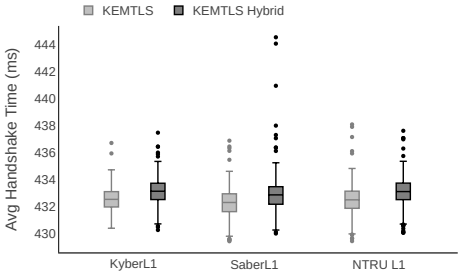

Fig. 4: PQC-Only and Hybrids (L1)

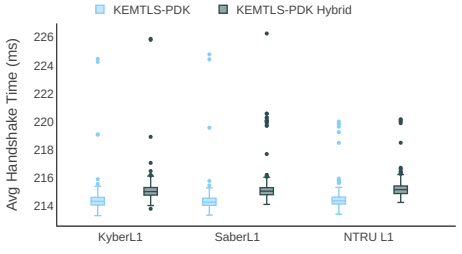

Fig. 5: PDK Instantiations (L1)

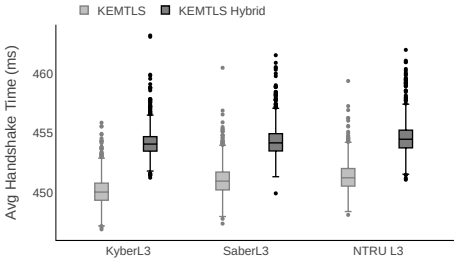

Fig. 6: PQC-Only and Hybrids (L3)

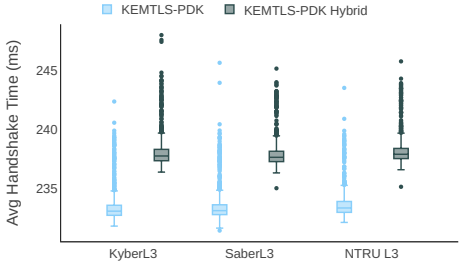

Fig. 7: PDK Instantiations (L3)

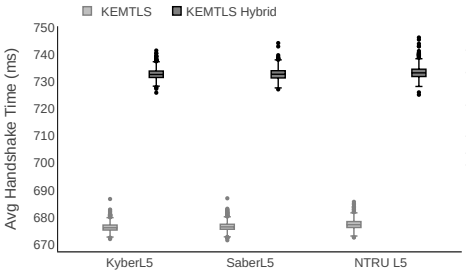

Fig. 8: PQC-Only and Hybrids (L5)

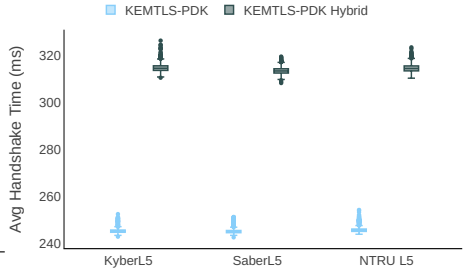

Fig. 9: PDK Instantiations (L5)

Simulated Environment Unlike the geographical-distant experiment, the simulated environment allows controlling the effect of parameter variations such as network latency. Table 4 highlights the hybrid penalties in KEMTLS using the average handshake time (HS) as the metric and increasing the simulated latency. The network latency plays a significant role in the handshake completion time. By configuring 1 ms of link latency, the client and the server will be delayed by 2 ms, and since KEMTLS requires two round trips to complete the handshake, it doubles this number, reaching 4 ms. The same behavior happens in KEMTLS mutual authentication, requiring three instead of two round trips. However, in practice, KEMTLS allows the client to send application data before handshake completion, removing one (additional) round trip and its performance impact.

Table 4: Average Handshake time (HS, in ms) for PQC-Only and Hybrid KEMTLS under different simulated latencies ('L' means security level and 'H' if in hybrid mode).

Algorithm and Security Level	**Latency: 1 ms**			**Latency: 5 ms**			**Latency: 50 ms**			**Latency: 150 ms**		
	HS Time	**Penalty**	**St. Dev.**	**HS Time**	**Penalty**	**St. Dev.**	**HS Time**	**Penalty**	**St. Dev.**	**HS Time**	**Penalty**	**St. Dev.**
KyberL1	6.0	-	0.4	22.3	-	0.3	202.8	-	0.2	602.9	-	0.2
KyberL1 H.	7.0	**1.0**	0.4	23.2	**0.9**	0.3	203.6	**0.9**	0.3	603.7	**0.8**	0.4
KyberL3	38.5	-	0.8	54.8	-	0.8	236.3	-	1.0	636.6	-	1.0
KyberL3 H.	46.8	**8.3**	0.9	62.9	**8.1**	2.3	243.2	**6.9**	1.2	643.9	**7.3**	1.6
KyberL5	63.0	-	0.8	78.4	-	0.8	261.1	-	6.0	659.9	-	1.0
KyberL5 H.	194.6	**131.6**	2.4	211.4	**133.0**	3.7	393.0	**132.0**	4.5	791.6	**131.7**	3.2
SaberL1	6.1	-	0.5	22.3	-	0.3	202.8	-	0.2	602.9	-	0.3
SaberL1 H.	7.1	**1.0**	0.5	23.3	**0.9**	0.4	203.7	**0.9**	0.4	603.8	**0.8**	0.4
SaberL3	38.9	-	0.7	55.5	-	0.8	236.0	-	1.0	637.1	-	2.2
SaberL3 H.	46.1	**7.2**	0.8	62.7	**7.1**	0.8	243.9	**7.9**	1.0	644.6	**7.5**	1.8
SaberL5	62.6	-	1.7	79.0	-	0.8	260.3	-	0.9	661.1	-	2.5
SaberL5 H.	196.5	**133.8**	2.4	212.2	**133.2**	3.2	391.9	**131.6**	3.3	792.7	**131.6**	3.4
NTRU L1	6.1	-	0.3	22.3	-	0.3	202.8	-	0.2	602.9	-	0.2
NTRU L1 H.	7.1	**1.0**	0.3	23.3	**1.0**	0.4	203.6	**0.8**	0.3	603.7	**0.8**	0.3
NTRU L3	39.2	-	0.8	55.8	-	0.8	236.5	-	2.4	636.9	-	1.0
NTRU L3 H.	46.5	**7.3**	0.8	63.0	**7.2**	0.8	243.6	**7.0**	1.0	644.1	**7.2**	1.0
NTRU L5	62.7	-	0.8	78.7	-	0.8	259.4	-	1.0	659.3	-	1.6
NTRU L5 H.	197.1	**134.4**	2.8	211.1	**132.3**	4.5	393.7	**134.3**	2.9	793.3	**134.0**	3.3

Table 4 shows that the hybrid penalty is negligible at lower security levels and significant at level 5. For instance, the largest penalty in security level 1 is 1.0 ms (e.g., KyberL1 H.), for security level 3 is 8.3 ms from KyberL3 H., and for security level 5 is 134.4 ms from NTRU L5 H. The latency variation did not impact the hybrid penalty significantly since they are more affected when changing to different security levels.

We also simulated different packet loss probabilities looking for hybrid penalties. Table 5 shows the results of using the time-to-send-app-data metric for PQC-Only and hybrid versions of KEMTLS. We do not use handshake completion time here because it would double the actual packet loss employed (due to the additional RTT). When analyzing columns from Table 5, we observed sig-

Table 5: Time-to-send-app-data (in ms) considering different packet loss probabilities.

Algorithm and	Packet Loss: 1%		Packet Loss: 2%		Packet Loss: 3%		Packet Loss: 5%	
Security Level	Median	95% percentile	Median	95% percentile	Median	95% percentile	Median	95% percentile
KyberL1	1.6	2.9	1.6	3.3	1.6	207.5	1.7	208.3
KyberL1 H.	2.3	3.4	2.3	7.9	2.3	207.3	2.4	209.4
KyberL3	34.0	36.1	34.3	39.2	34.8	239.6	34.9	242.0
KyberL3 H.	39.9	42.1	39.8	43.4	40.3	246.1	40.7	247.2
KyberL5	58.4	60.9	58.5	63.6	57.6	263.1	58.9	266.3
KyberL5 H.	162.6	166.8	162.0	167.2	161.0	359.2	162.1	368.0
SaberL1	1.6	2.7	1.7	3.4	1.7	206.9	1.7	208.6
SaberL1 H.	2.4	3.6	2.4	8.1	2.4	7.9	2.4	209.8
SaberL3	34.5	37.0	34.8	38.1	34.5	238.7	34.3	241.9
SaberL3 H.	40.0	42.6	40.2	244.2	40.7	245.5	40.8	247.6
SaberL5	58.1	60.8	58.7	64.8	58.1	263.6	58.1	265.5
SaberL5 H.	161.9	167.6	162.4	168.2	162.1	168.5	162.5	368.1
NTRU L1	1.7	2.5	1.7	3.7	1.7	206.3	1.7	209.1
NTRU L1 H.	2.4	3.5	2.4	4.1	2.4	207.3	2.4	209.5
NTRU L3	34.3	36.6	34.2	38.3	34.3	51.1	34.8	242.4
NTRU L3 H.	40.3	43.5	40.4	44.6	40.4	46.3	40.2	246.9
NTRU L5	58.6	61.3	59.0	62.2	58.2	263.9	58.8	266.0
NTRU L5 H.	163.0	167.7	162.2	169.7	162.7	364.4	162.2	368.0

nificant changes in the penalties in medians since the largest difference observed is 1.3 ms, from NTRU L5 H. (median at packet loss probability of 3% minus the one at 2%). When reaching 5% loss probabilities, some connections can slow down significantly, which can be seen at the 95% percentile. This slowdown happens because increasing size the likelihood of losing packets increases. Similar behavior is observed in mutual-authenticated connections (see B, Table 8). However, this increase happens with PQC-only and hybrids, with a larger increase in security level 5 (for example, KyberL5 H. differs near 100 ms to PQC-only, for all packet loss probabilities). Overall, using level 1 hybrid instantiations, we do not anticipate a large penalty if a wireless connection experiences packet loss.

5.2 Hybrid Comparison

This section analyzes the performance of different hybrid approaches (in KEMTLS and PQTLS).

Geographical-Distant Server We selected the time-to-send-app-data metric for the client who starts the protocol (in general) and is the first to send application data. For applications dependent on the handshake completion time, one can compare hybrid PQTLS timings with the KEMTLS timings provided in Figures 4 to 8. In such a case, KEMTLS additional round-trip imposes a delay which is often worse than PQTLS timings.

Figures 10 to 12 compare hybrids (KEMTLS and PQTLS), considering the algorithms at each NIST security level. At level 1, the hybrid's boxes overlap, meaning similar timings. The main difference is that hybrid PQTLS has a dual-signature operation for the handshake transcript data. Hybrid KEMTLS replaces it with two KEM encapsulations (using a classical and a PQC algorithm).

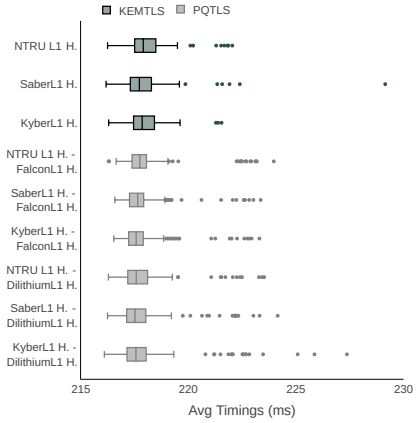

Fig. 10: Hybrids Comparison (L1)

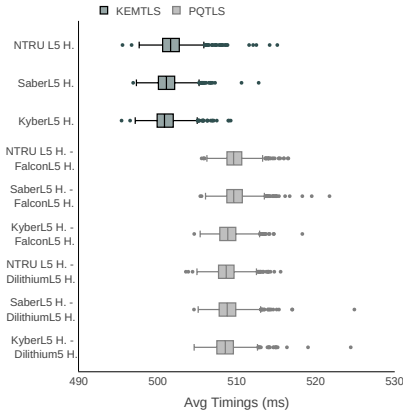

Fig. 11: Hybrids Comparison (L5)

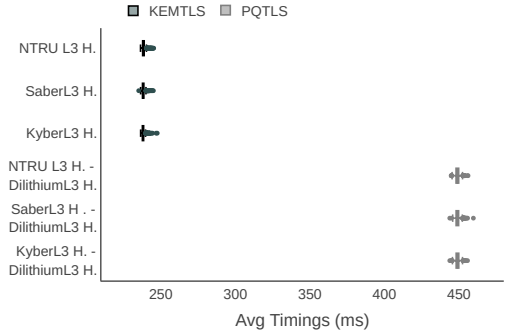

Fig. 12: Hybrids Comparison (L3)

The hybrid approaches achieved similar performance at security level 1. At security level 3, however, we observed an interesting result when comparing them. The hybrid KEMTLS is significantly faster than PQTLS in the time-to-send-app-data metric. Hybrid PQTLS sizes, usually superior to KEMTLS sizes, can easily surpass network thresholds such as TCP Maximum Segment Size. For example, considering the TCP window standard size (10 MSS), if we compare hybrids C_{size}: KEMTLS using KyberL3 H. has 15033 bytes, and it is 16.57% smaller than hybrid PQTLS using KyberL3 and DilithiumL3, both in hybrid mode (18019 bytes). This difference incurs an additional round-trip at the TCP level. If we increase MSS size, the performance can be equated, but such a change can affect network performance negatively, as discussed by Sikeridis et al. [37]. We could not test Falcon in security level 3 (no parameter set available). Lastly, level 5 instantiations also exhibit a performance difference that favors the deployment of hybrid KEMTLS rather than PQTLS.

Simulated Environment Table 6 compares hybrids using simulated network latencies. They are all average values (in ms); on average, hybrid KEMTLS achieved better performance in all security levels tested. We used the "time-to-send-app-data" metric (instead of the handshake completion time) since we consider it a more practical (and fair) comparison.

Table 6: Hybrid KEMTLS time-to-send-app-data (in ms) compared with Hybrid PQTLS under different simulated latencies

Algorithms and Security Level	Timings Latency: 1 ms	Timings Latency: 5 ms	Timings Latency: 50 ms	Timings Latency: 150 ms
KyberL1 H. - DilithiumL1 H.	4.9	13.1	103.5	303.5
KyberL1 H. - FalconL1 H.	5.4	13.6	104.0	304.1
KyberL1 H. (KEMTLS)	4.5	12.5	102.7	302.9
SaberL1 H. - DilithiumL1 H.	5.0	13.1	103.4	303.5
SaberL1 H. - FalconL1 H.	5.4	13.6	104.0	304.1
SaberL1 H. (KEMTLS)	4.5	12.6	102.8	302.9
NTRU L1 H. - DilithiumL1 H.	5.0	13.1	103.4	303.5
NTRU L1 H. - FalconL1 H.	5.4	13.6	103.9	304.1
NTRU L1 H. (KEMTLS)	4.5	12.6	102.7	302.8
KyberL3 H. - DilithiumL3 H.	68.7	76.6	167.0	366.5
KyberL3 H. (KEMTLS)	42.5	50.7	141.0	341.4
SaberL3 H. - DilithiumL3 H.	68.5	76.6	167.5	367.1
SaberL3 H. (KEMTLS)	42.5	50.6	141.2	341.5
NTRU L3 H. - DilithiumL3 H.	68.7	76.2	167.6	367.5
NTRU L3 H. (KEMTLS)	43.1	50.9	141.5	341.6
KyberL5 H. - DilithiumL5 H.	181.7	191.3	279.1	479.8
KyberL5 H. - FalconL5 H.	182.2	190.6	280.3	480.3
KyberL5 H. (KEMTLS)	163.9	171.8	263.6	462.3
SaberL5 H. - DilithiumL5 H.	182.1	190.4	279.4	480.5
SaberL5 H. - FalconL5 H.	182.9	190.3	280.2	480.9
SaberL5 H. (KEMTLS)	163.8	172.6	263.9	462.6
NTRU L5 H. - DilithiumL5 H.	182.3	190.0	280.0	479.3
NTRU L5 H. - FalconL5 H.	182.6	190.7	280.9	480.7
NTRU L5 H. (KEMTLS)	163.9	176.7	263.3	462.9

Using Dilithium (instead of Falcon) in the server certificate improved PQTLS due to the better signing time, but still slower than KEMTLS. The differences in performance between hybrid KEMTLS and hybrid PQTLS are small in security level 1 since the largest difference observed was 1.3 ms. However, it becomes more significant when increasing security parameters, with an average difference of 25.9 ms in level 3 (standard deviation of 0.5 ms) and an average difference of 17.2 ms in level 5 (standard deviation of 1.2 ms).

5.3 Load Test

Using the parameters specified in Section 4, we performed a load testing of a web service providing content using our hybrid instantiations, compared to a baseline configuration (using classical cryptography only). Figure 13 shows the

number of successful requests for algorithms in hybrid modes using the security level 1 parameter set. The hybrids used are KyberL1 H. (`P256_Kyber512`) for KEMTLS KEX and Authentication; KyberL1 H. (KEX) and DilithiumL1 H. (`P256_Dilithium2`) (Auth) for PQTLS; KyberL1 H. (KEX) and Classic-McElieceL1 H. (`P256_Classic-McEliece-348864`) (Au-th) in KEMTLS-PDK; and KyberL1 H. and DilithiumL1 H. for PQTLS Cached-cert configuration. We selected McEliece in PDK since it can be the optimized size configuration (as shown in [34]). Due to recent attacks in Rainbow, we did not evaluate PQTLS using a Rainbow Cached certificate (which would be a size-optimized instantiation for PQTLS). RFC 7924 [32] specifies caching of certificates.

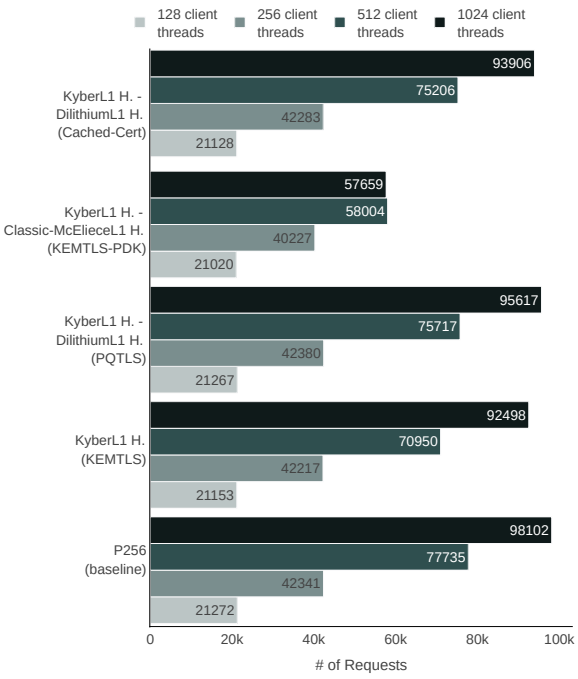

Fig. 13: Load testing with hybrids at level 1

Regarding the number of requests, the four configurations obtained similar results and a low error rate (mostly zero) when 128 client threads were used. Despite KEMTLS additional RTT for the handshake completion, KEMTLS performed similarly to PQTLS at 128 and 256 client threads. This similarity can be explained by the fact that both protocols allow the client to send application data in the same RTT, and the server's resources were not exhausted. On the other hand, when increasing the number of client threads, the differences between the configurations grow significantly. For instance, hybrid KEMTLS connections each have an additional RTT, which might cause the network to become

overloaded early, especially when there are more threads, which leads to fewer successful requests (compared to PQTLS). Additionally, the web page content size (2MB) helps to congest the network because each client thread requests such data. Regarding the number of errors (i.e., failed requests), we omitted them in Figure 13, since in all tests, they were below 25 failures, thus not significant. It is worth mentioning that increasing the number of client threads leads to greater differences in PQ versus baseline (classical).

We expected that the Cached-Cert and KEMTLS-PDK would achieve more successful requests because fewer data was transferred between the peers. However, KEMTLS-PDK with McEliece achieved the server limit at 512 clients. We noted that McEliece has greater memory requirements (please refer to [29]), mainly due to its larger public key. Besides, on the client side, verifying Dilithium signatures is faster than decapsulating McEliece ciphertexts. The gain in sizes did not lead to better performance in this case.

5.4 Summarizing Results

Figure 14 summarizes some experiments performed in this work, adding the memory peak metric (so far not discussed). We present an aggregation of average handshake time, average time-to-send-app-data, both from the simulated environment, handshake sizes, number of successful requests, and memory peak usage (by the server), considering hybrid implementations at security level 1. Initially, we compared hybrid to PQC-only implementation, but now all the data is normalized to the baseline configuration, which uses classical cryptography.

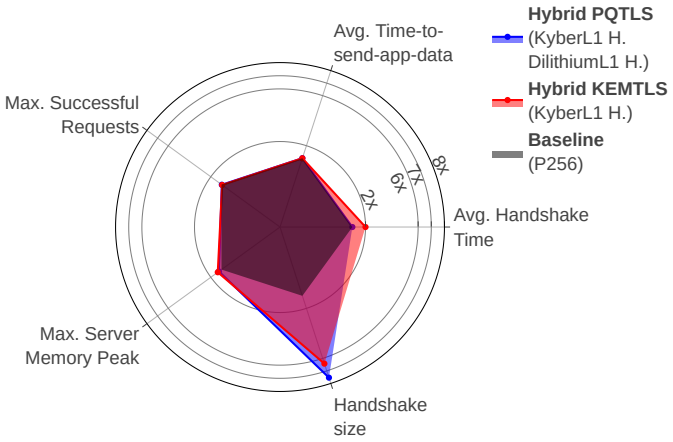

Fig. 14: Summary of performance of hybrids at level 1

The main finding for security level 1, shown in Figure 14, is that the hybrid penalties in performance (now compared to baseline) are small. The handshake

sizes increased almost 7x in our hybrid, using level 1, and the average handshake time takes almost 2x since there is an additional RTT (from KEMTLS' original design). There is no significant penalty for using the time-to-send-application-data at this level. However, regarding penalties, we observed that the behavior changed when increasing security parameters. The penalties start to grow when comparing hybrids to PQC-only alternatives (shown in Section 5.1).

The number of requests is not evident in Figure 14 scale, but the Max. Successful requests decreased by 3% and 6% for hybrids (PQTLS and KEMTLS, respectively, level one). In absolute values, it decreased between 2485 and 5604 requests, which can be significant depending on the application transiting to hybrids. Lastly, memory requirements increased by nearly 64 MB (KEMTLS) and 43 MB (PQTLS) when using hybrid alternatives, which is not significant in current server configurations.

6 Conclusions

Hybrid modes are one first step toward the PQC transition process. This work is the first to investigate hybrids using the KEMTLS and KEMTLS-PDK approaches. KEMTLS can achieve smaller size configurations for TLS, but providing hybrids for KEMTLS allows for a safer transition since it keeps classical algorithms. Compared to PQC algorithms, classical ones had more time for security analysis.

Overall, the Hybrid Penalties when adding classical algorithms in the KEMTLS design are minor, particularly when analyzing instantiations with lower security parameters. We confirm this result in simulated and geographically distant connections. Therefore, hybrids are suitable for KEMTLS (and the PDK variant) since they add confidence from classical algorithms, but the performance penalties are insignificant. Our evaluation includes algorithms from the NIST PQC standardization process. It is worth mentioning that the average timings of Kyber, Saber, and NTRU (in KEMTLS), Dilithium, and Falcon (in PQTLS) were close enough in a way that we could not select between the best algorithm configuration in the hybrid instantiations. This result was expected since all of them are based on lattices and finalists in the NIST's standardization process (Round 3).

We also compared distinct hybrid approaches under different metrics, namely hybrid modes for KEMTLS and PQTLS. At security level 3, we observed that every byte saved counts since hybrid KEMTLS performed much better than hybrid PQTLS, as the first had suitably fit in the TCP congestion window. Aiming at a different evaluation metric, we performed a load test in a hybrid HTTPS web server. As the number of client threads increased, we observed superior performance of hybrid PQTLS over hybrid KEMTLS, but both lost to the classical cryptography configuration. This load-test experiment allowed us to estimate the impacts on the server providing HTTP content when adopting hybrid PQC.

Experimenting with PQC in TLS is a cumbersome task because there is a wide range of choices: evaluation metrics, security parameters, algorithm selection, certificate chain configurations, mutual authentication, and caching certificates, among others. Although we did not evaluate PKI revocation scenarios, which would increase TLS handshake sizes, one could use short-lived certificates [26] in which the revocation impact is reduced. Investigating approaches to mitigate the revocation impact is an interesting line of work that would benefit the PQC adoption in TLS.

However, there are other scenarios for evaluating the PQC adoption in TLS. We left for future work investigating (hybrid) KEMTLS applied in scenarios with Internet-of-Things (IoT) and 5G networks, which might require energy consumption as an essential evaluation metric.

References

1. Archive's Web Almanac, H.: Median Page Weight over Time. Available at: https://almanac.httparchive.org/en/2021/page-weight#fig-3. Accessed on 21.03.2022. (2022)
2. Author(s), A.: Hybrid_kemtls_tests repository. https://mega.nz/file/II1njaoK#_c8_pAhzd1-OuOuysPbdt6PTDL2bhOxZl0gbY7s-08U (2022), [Online; accessed 17-Aug-2022]
3. Barker, E., Chen, L., Davis, R.: Recommendation for key-derivation methods in key-establishment schemes revision 2. NIST Special Publication **800**, 56C (2020). https://doi.org/https://doi.org/10.6028/NIST.SP.800-56Cr2
4. Barker, E., Chen, L., Keller, S., Roginsky, A., Vassilev, A., Davis, R.: Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. Tech. rep., National Institute of Standards and Technology (2017)
5. Barker, E., Chen, L., Roginsky, A., Vassilev, A., Davis, R., Simon, S.: Recommendation for pair-wise key-establishment schemes using integer factorization cryptography. Tech. rep., National Institute of Standards and Technology (2019)
6. Bellare, M., Lysyanskaya, A.: Symmetric and dual prfs from standard assumptions: A generic validation of an hmac assumption. Cryptology ePrint Archive, Report 2015/1198 (2015), https://ia.cr/2015/1198
7. Bernstein, D.J., Lange, T.: Post-quantum cryptography. Nature **549**(7671), 188–194 (Sep 2017). https://doi.org/10.1038/nature23461, https://doi.org/10.1038/nature23461
8. Beullens, W.: Breaking rainbow takes a weekend on a laptop. Cryptology ePrint Archive, Paper 2022/214 (2022), https://eprint.iacr.org/2022/214, https://eprint.iacr.org/2022/214
9. Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., Stebila, D.: Hybrid key encapsulation mechanisms and authenticated key exchange. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography. pp. 206–226. Springer International Publishing, Cham (2019)
10. Castryck, W., Decru, T.: An efficient key recovery attack on sidh (preliminary version). Cryptology ePrint Archive, Paper 2022/975 (2022), https://eprint.iacr.org/2022/975, https://eprint.iacr.org/2022/975
11. Celi, S., Faz-Hernández, A., Sullivan, N., Tamvada, G., Valenta, L., Wiggers, T., Westerbaan, B., Wood, C.A.: Implementing and measuring kemtls. In: Longa,

P., Ràfols, C. (eds.) Progress in Cryptology – LATINCRYPT 2021. pp. 88–107. Springer International Publishing, Cham (2021)
12. Clancy, T.C., McGwier, R.W., Chen, L.: Post-quantum cryptography and 5g security: Tutorial. In: Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks. p. 285. WiSec '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3317549.3324882, https://doi.org/10.1145/3317549.3324882
13. Cloudflare: Circl (cloudflare interoperable, reusable cryptographic library). Online (2021), https://github.com/cloudflare/circl
14. Crockett, E., Paquin, C., Stebila, D.: Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh. Cryptology ePrint Archive, Report 2019/858 (2019)
15. Fouque, P.A., Pointcheval, D., Zimmer, S.: Hmac is a randomness extractor and applications to tls. In: Proceedings of the 2008 ACM symposium on Information, computer and communications security (ASIACCS). p. 21–32. ASIACCS '08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1368310.1368317, https://doi.org/10.1145/1368310.1368317
16. Giacon, F., Heuer, F., Poettering, B.: Kem combiners. In: Abdalla, M., Dahab, R. (eds.) Public-Key Cryptography – PKC 2018. pp. 190–218. Springer International Publishing, Cham (2018)
17. Günther, F., Rastikian, S., Towa, P., Wiggers, T.: Kemtls with delayed forward identity protection in (almost) a single round trip. Cryptology ePrint Archive, Report 2021/725 (2021), https://ia.cr/2021/725
18. Hemminger, S.: Linux network emulator. Online (2011), https://www.linux.org/docs/man8/tc-netem.html
19. Krawczyk, H.: Cryptographic extraction and key derivation: The hkdf scheme. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010. pp. 631–648. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
20. Kurosawa, K., Desmedt, Y.: A new paradigm of hybrid encryption scheme. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. pp. 426–442. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
21. Moody, D.: Let's get ready to rumble- the nist pqc competition. https://csrc.nist.gov/CSRC/media/Presentations/Let-s-Get-Ready-to-Rumble-The-NIST-PQC-Competiti/images-media/PQCrypto-April2018_Moody.pdf (2018)
22. Mosca, M., Piani, M.: Quantum threat timeline report 2020. Available at: https://globalriskinstitute.org/publications/quantum-threat-timeline-report-2020/. Accessed on 20.07.2021. (2020)
23. NIST: Post-quantum cryptography (2016), https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. Acessed: 2021-06-26
24. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking post-quantum cryptography in tls. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography. pp. 72–91. Springer International Publishing, Cham (2020)
25. Paul, S., Kuzovkova, Y., Lahr, N., Niederhagen, R.: Mixed certificate chains for the transition to post-quantum authentication in tls 1.3. Cryptology ePrint Archive, Report 2021/1447 (2021), https://ia.cr/2021/1447
26. Payne, B.: PKI at scale using Short–lived certificates. USENIX Association, San Francisco, CA (Jan 2016)
27. Pirandola, S., Andersen, U.L., Banchi, L., Berta, M., Bunandar, D., Colbeck, R., Englund, D., Gehring, T., Lupo, C., Ottaviani, C., Pereira, J.L., Razavi, M.,

Shaari, J.S., Tomamichel, M., Usenko, V.C., Vallone, G., Villoresi, P., Wallden, P.: Advances in quantum cryptography. Adv. Opt. Photon. **12**(4), 1012–1236 (Dec 2020). https://doi.org/10.1364/AOP.361502, http://aop.osa.org/abstract.cfm?URI=aop-12-4-1012
28. Open Quantum Safe Project: Oqs-OpenSSL github repository. Available at: https://github.com/open-quantum-safe/openssl (2022), [Online; accessed 10-Mar-2022]
29. Project, O.Q.S.: Kem memory consumption. Available at: https://openquantumsafe.org/benchmarking/visualization/mem_kem.html (2022), [Online; accessed 17-Ago-2022]
30. Project, O.Q.S.: liboqs-go: Go bindings for liboqs. Available at: https://github.com/open-quantum-safe/liboqs-go (2022), [Online; accessed 25-Jan-2022]
31. Rescorla, E.: The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor (August 2018)
32. Santesson, S., Tschofenig, H.: Transport layer security (tls) cached information extension. RFC 7924, RFC Editor (July 2016)
33. Schwabe, P., Stebila, D., Wiggers, T.: Post-Quantum TLS Without Handshake Signatures, p. 1461–1480. Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3372297.3423350
34. Schwabe, P., Stebila, D., Wiggers, T.: More efficient post-quantum kemtls with pre-distributed public keys. In: Bertino, E., Shulman, H., Waidner, M. (eds.) Computer Security – ESORICS 2021. pp. 3–22. Springer International Publishing, Cham (2021)
35. Shamay, U.: Go-bench github repository. Available at: https://github.com/cmpxchg16/gobench. Accessed on 21.03.2022. (2022)
36. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. pp. 124–134. IEEE, IEEE, Santa Fe, NM, USA (1994)
37. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Assessing the overhead of post-quantum cryptography in tls 1.3 and ssh. In: Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies. pp. 149–156. Association for Computing Machinery, New York, NY, USA (2020)
38. Stebila, D., Fluhrer, S., Gueron, S.: Hybrid key exchange in TLS 1.3. http://tools.ietf.org/html/draft-ietf-tls-hybrid-design-05 (August 2022), internet-Draft
39. Stebila, D., Mosca, M.: Post-quantum key exchange for the internet and the open quantum safe project. In: Avanzi, R., Heys, H. (eds.) Selected Areas in Cryptography – SAC 2016. pp. 14–37. Springer International Publishing, Cham (2017)
40. Team, N.P.: Pqc standardization process: Announcing four candidates to be standardized, plus fourth round candidates. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4 (2022)
41. Weller, D., van der Gaag, R.: Incorporating post-quantum cryptography in a microservice environment. Tech. rep., Security and Network Engineering - University of Amsterdam (2020)
42. Westerbaan, B.: Sizing up post-quantum signatures. https://blog.cloudflare.com/sizing-up-post-quantum-signatures/ (2021)
43. Zhang, S., Wang, Y., Zhou, W.: Towards secure 5G networks: A Survey. Computer Networks **162**, 106871 (2019). https://doi.org/https://doi.org/10.1016/j.comnet.2019.106871, http://www.sciencedirect.com/science/article/pii/S138912861830817X

A Mutual Authentication

Figure 15 shows the Hybrid KEMTLS design in the mutual authentication case. The main changes are the `CertificateRequest` message sent by the server, which triggers the client to send its certificate containing two KEM public keys (from a PQC and classical algorithm). The server encapsulates under the client's keys and replies the `ServerKEMCiphertext` message. The new shared secrets K_c (PQC) and K_cc (classical) are used in the key derivation process.

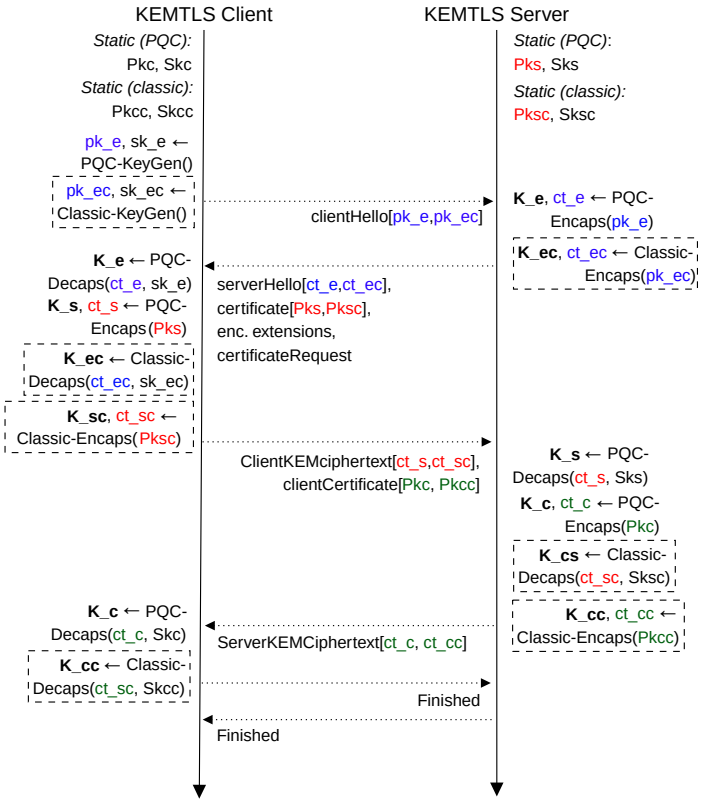

Fig. 15: Hybrid KEMTLS Handshake (Mutual authentication)

Since the `CertificateRequest` message is sent by the server, there is another round trip added to the handshake time. The extra round trip affects timings, specially if we consider that the additional payload (i.e., the client's certificate) could help to surpass the TCP congestion window. However, note that the time-to-send-application-data can be the same as server-only authentication (similar to RFC 8446 [31]). Therefore mutual authentication might not affect this metric.

B KEMTLS Additional Results

This Appendix is divided in two parts: geo-distant server and simulated network experiment (Sections B.1 and B.2).

B.1 Geographically Distant Server

Figures 16 to 18 shows the penalties when using hybrid KEMTLS with mutual authentication, under the geographically distant server experiment. The metric in use is the handshake time. Similarly to the server-only authentication results, the penalties are small using NIST Security level 1 parameters, and they increase as the security levels increase. The main difference is that mutual authentication imposes an additional round-trip time (RTT) and therefore the required amount of time to establish the connection (this already observed in other works, e.g. [11], but not with hybrids). It is worth mentioning that in this experiment we do not control the network parameters, so latency variations can happen. In contrast to security levels 3 and 5, we can see the overlap between boxplots (level one), thus the usage of such hybrids does not degrade protocol performance significantly.

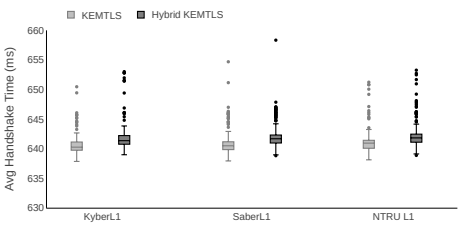

Fig. 16: Mutual Auth. Comparison (L1)

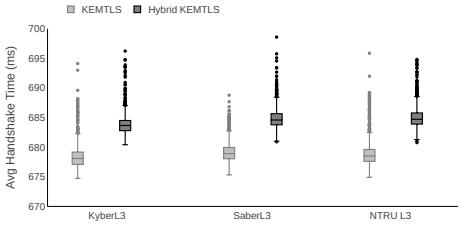

Fig. 17: Mutual Auth. Comparison (L3)

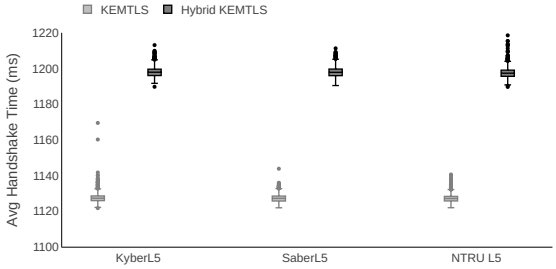

Fig. 18: Mutual Auth. Comparison (L5)

B.2 Simulated Environment

Table 7 presents KEMTLS results regarding the average handshake time metric, using the simulated latencies but with mutually authenticated connections.

The results corroborates to the findings about hybrid penalty in KEMTLS using server-only authentication. Increasing the latencies does not change the overall penalty. Recall that the hybrid penalty is the penalty over PQC-only instantiation. For example, in security level 1 instantiations, the penalties are small (near 1 ms) and their increase for level five ranges from 190 to 195 ms.

Table 7: Average handshake time (HS, in ms) for PQC-only and hybrid KEMTLS in mutually-authenticated connections.

Algorithm	Latency: 1ms			Latency: 5ms			Latency: 50ms			Latency: 150ms		
	HS Time (ms)	Penalty (ms)	St. Dev. (ms)	HS Time (ms)	Penalty (ms)	St. Dev. (ms)	HS Time (ms)	Penalty (ms)	St. Dev. (ms)	HS Time (ms)	Penalty (ms)	St. Dev. (ms)
KyberL1	9.5		0.7	33.8		0.7	304.8		0.6	905.0		0.9
KyberL1 H.	11.0	1.5	0.8	35.2	1.4	0.8	306.1	1.3	0.8	906.3	1.3	0.9
KyberL3	79.4		1.5	102.7		1.4	374.3		1.9	974.2		1.8
KyberL3 H.	89.6	10.2	1.3	114.8	12.1	1.3	384.0	9.7	1.5	984.6	10.4	1.3
KyberL5	128.7		1.7	153.2		2.5	422.9		2.3	1021.7		2.5
KyberL5 H.	319.6	190.9	3.3	344.0	190.9	3.1	615.5	192.6	3.3	1216.4	194.7	6.1
SaberL1	9.5		0.7	33.8		0.7	304.9		0.9	905.0		0.7
SaberL1 H.	11.0	1.5	0.8	35.2	1.4	0.8	306.2	1.3	0.8	906.3	1.3	0.9
SaberL3	79.0		1.4	103.3		1.6	374.0		1.8	973.6		1.5
SaberL3 H.	89.4	10.4	1.3	114.1	10.8	1.4	384.3	10.3	1.5	984.5	10.9	1.2
SaberL5	128.9		1.7	152.6		1.7	422.8		1.9	1023.3		1.8
SaberL5 H.	320.8	192.0	4.1	344.0	191.5	3.9	616.0	193.2	3.4	1214.0	190.7	3.4
NTRU L1	9.5		0.8	33.8		0.8	304.8		0.7	904.9		0.8
NTRU L1 H.	11.0	1.5	0.7	35.2	1.4	0.8	306.1	1.3	0.8	906.3	1.3	0.9
NTRU L3	79.5		1.5	105.4		6.6	373.9		1.8	974.1		1.7
NTRU L3 H.	90.4	10.9	1.3	114.1	8.6	1.8	384.3	10.4	1.4	985.0	10.9	1.3
NTRU L5	129.8		1.5	152.6		1.8	422.3		1.8	1025.2		1.7
NTRU L5 H.	325.0	195.2	26.3	345.0	192.4	2.9	614.5	192.2	4.2	1214.6	189.4	4.0

Table 8: KEMTLS Time-to-send-app-data (in ms) considering different packet loss probabilities and mutual authentication.

Algorithm	Packet Loss: 1%		Packet Loss: 2%		Packet Loss: 3%		Packet Loss: 5%	
	Median	95% percentile	Median	95% percentile	Median	95% percentile	Median	95% percentile
KyberL1	3.1	4.9	3.1	209.4	3.2	210.6	3.2	211.2
KyberL1 H.	4.5	10.2	4.5	209.8	4.5	211.6	4.6	212.3
KyberL3	73.2	80.6	72.9	279.0	73.1	280.1	73.4	282.8
KyberL3 H.	82.9	88.3	84.0	290.1	83.7	290.8	83.7	292.8
KyberL5	124.0	132.1	122.5	328.3	123.7	330.5	124.0	334.8
KyberL5 H.	317.0	511.9	318.0	522.8	316.5	523.3	317.6	527.2
SaberL1	3.2	8.6	3.2	209.5	3.2	210.5	3.3	211.0
SaberL1 H.	4.6	9.8	4.6	210.3	4.6	212.0	4.6	212.5
SaberL3	72.7	77.6	75.2	280.9	74.8	281.7	74.6	283.1
SaberL3 H.	85.2	91.9	85.4	291.4	85.0	292.3	85.0	294.6
SaberL5	122.7	127.5	124.7	329.6	124.3	331.8	124.7	332.6
SaberL5 H.	317.2	324.0	318.4	523.7	318.0	525.0	318.8	528.2
NTRU L1	3.2	8.7	3.2	208.9	3.2	210.3	3.3	211.4
NTRU L1 H.	4.6	10.0	4.6	210.2	4.6	211.7	4.6	212.7
NTRU L3	73.3	79.3	74.7	280.8	73.8	280.4	74.9	285.1
NTRU L3 H.	85.3	90.3	84.7	291.0	85.1	291.4	85.6	293.8
NTRU L5	123.1	128.7	124.1	330.4	125.6	331.4	125.4	333.4
NTRU L5 H.	317.3	324.9	318.1	522.2	317.3	525.5	319.0	527.8

On the other hand, about the overall performance, the number of round-trips of mutual authentication design is different, which adds one RTT to the network latency. Using the 1 ms simulated latency as example, in server-only authentication the link latency will be 2 ms (client side and server side). Since KEMTLS has two RTTs, the network latency will be 4 ms, and in the mutual authenti-

cation case it will be 6 ms (three RTTs). Analyzing KyberL1 H. performance, we obtained 11 ms in average (Table 7), which means 5 ms of computational cost and 6 ms of network latency (3 RTTs times 2 ms). Now looking at 150 ms simulated latency, KyberL1 H. gets an average handshake time of 906.3 ms, which means approximately 6 ms of computational cost and 900 ms of network latency (3 RTTs times 300 ms). Given a standard deviation near to 1 ms, we can observe very close computational cost and, consequently, the penalties are also similar.

Regarding packet loss, Table 8 shows the variations for KEMTLS in the time-to-send-app-data metric and considering mutual authentication. Again, the behavior is similar to server-only authentication (in terms of the hybrid penalties) except that the timings have greater latency. These are caused mainly by the increased handshake size.

Since mutually authentication requires client certificates, it not only increases the RTTs but it also increases the handshake sizes. Please refer to Table 3.2 for the precise numbers. Now, if the packet loss probabilities increase, the likelihood of a re-transmission also increases, triggering TCP transmission control mechanisms. Looking at security level 1 and using 5% loss probability, KyberL1 H., for example, has achieved 212 ms at the 95th percentile (Table 8), compared to the 209.4 ms percentile in server-only authentication (Table 5). On the other hand, looking at security level 5, KyberL5 H. has a 95% percentile of 527.2 ms (mutual authentication) versus 368.0 ms (server-only authentication). These results shows that the impact of size is significant when considering packet loss scenarios. It is worthy to note that all decimals in the tables are due to the average computations. We did not employed additional controls measurements for micro or nanosecond precision.