HQRT: A Hybrid Quantum-Resistant Resumption Framework for Zero-RTT TLS 1.3 Early Data Security

Tahera Begum Abdul * and K. Venkata Ramana

Posted Date: 8 April 2026

doi: 10.20944/preprints202604.0557.v1

Keywords: post-quantum cryptography; TLS 1.3; zero-RTT resumption; ML-KEM; hybrid key exchange; harvest-now-decrypt-later; session tickets; early data security; key schedule; CRYSTALS-Kyber

 doi:10.20944/preprints202604.0557.v1

Article

HQRT: A Hybrid Quantum-Resistant Resumption Framework for Zero-RTT TLS 1.3 Early Data Security

Tahera Begum Abdul * and K. Venkata Ramana

Department of Computer Science and Systems Engineering, Andhra University, Visakhapatnam 530003, India.
* Correspondence: taherabegum.rs@andhrauniversity.edu.in

Abstract

TLS 1.3 zero-round-trip-time (0-RTT) resumption reduces reconnection latency by allowing clients to transmit early application data using pre-shared keys (PSK) derived from previously established session tickets. This mechanism is pivotal for latency-sensitive web services, API gateways, and IoT applications. However, the cryptographic foundations of current session tickets—symmetric keys derived from classical `X25519` key exchange—are fundamentally vulnerable to Harvest-Now-Decrypt-Later (HNDL) quantum attacks: an adversary capturing session ticket exchanges today can retroactively decrypt PSKs and all 0-RTT early data once a cryptographically relevant quantum computer (CRQC) becomes available. This paper introduces **HQRT** (Hybrid Quantum-Resistant Resumption for TLS 1.3), a protocol-level framework that embeds a hybrid `X25519` + **ML-KEM-768** key encapsulation into the TLS 1.3 `NewSessionTicket` lifecycle, producing quantum-safe session tickets without additional handshake round trips. **HQRT** defines a Hybrid Resumption Master Secret (HRMS) derived from both classical and post-quantum shared secrets and integrates it into the TLS 1.3 key schedule as a drop-in extension of the Resumption Master Secret. We provide: (i) a formal security model for quantum-safe 0-RTT resumption with game-based HNDL-resistance proofs; (ii) an extended replay protection analysis under quantum adversaries; (iii) a proof-of-concept implementation on OpenSSL 3.x with the OQS provider; and (iv) comprehensive benchmarks across server, desktop, and IoT platforms demonstrating only 4–9% latency overhead and 6.5% throughput reduction relative to classical 0-RTT, versus the 81–89% overhead of full post-quantum handshakes. A cumulative cost-benefit analysis over multi-session workloads demonstrates 34–97% amortised overhead reduction compared to per-reconnection PQC handshakes, with latency distributions exhibiting sub-millisecond tail divergence from classical baselines. **HQRT** provides a practical, incrementally deployable pathway for quantum-safe TLS resumption compatible with existing certificate infrastructure.

Keywords: post-quantum cryptography; TLS 1.3; zero-RTT resumption; ML-KEM; hybrid key exchange; harvest-now-decrypt-later; session tickets; early data security; key schedule; CRYSTALS-Kyber

---

1. Introduction

1.1. The Role of 0-RTT Resumption in Modern TLS

The deployment of TLS 1.3 [1] has fundamentally improved the performance profile of secure Internet communications. Among its most operationally significant features is the zero-round-trip-time (0-RTT) resumption mode, which allows a client that has previously established a session with a server to transmit encrypted early application data in its very first message—eliminating the one-round-trip overhead of a fresh full handshake. In high-frequency connection scenarios—mobile API calls, content delivery network (CDN) edge handshakes, microservice mesh communications, and constrained IoT device polling—0-RTT resumption translates directly into measurable improvements in time-to-first-byte (TTFB) and application-level throughput.

The mechanism underpinning 0-RTT is the *session ticket*. After a successful TLS 1.3 full handshake, the server issues one or more `NewSessionTicket` messages containing an encrypted opaque state blob

 doi:10.20944/preprints202604.0557.v1

from which a pre-shared key (PSK) is derived. On reconnection, the client presents this ticket in the `pre_shared_key` extension of a new `ClientHello`. Both parties then derive 0-RTT keys from the PSK, without requiring a fresh key exchange. The PSK is computed from the Resumption Master Secret (RMS) of the prior session via the TLS 1.3 HKDF-based key schedule [1].

1.2. The Quantum Threat to Session Tickets

The security of TLS 1.3 session tickets, and consequently all 0-RTT early data, depends critically on the confidentiality of the Resumption Master Secret. In the current deployment landscape, the RMS is derived from a key exchange almost universally based on `X25519` (Elliptic Curve Diffie-Hellman over Curve25519). The security of `X25519` rests on the elliptic curve discrete logarithm problem (ECDLP), which is efficiently solvable by Shor's algorithm on a sufficiently capable quantum computer [2].

This creates a precise and urgent vulnerability through the *Harvest-Now-Decrypt-Later* (HNDL) threat model. An adversary operating today who captures the TLS 1.3 full handshake establishing the session ticket—specifically the `X25519` key-exchange messages—can derive the classical shared secret retroactively once a CRQC becomes available. From this shared secret the adversary can reconstruct the full TLS 1.3 key schedule, recover the RMS, and thereby decrypt:

(a) all 0-RTT early data sent in subsequent resumption sessions using tickets derived from that handshake, and
(b) all application data from the original full-handshake session itself.

Critically, a single captured full handshake may seed multiple session tickets (RFC 8446 permits up to 2^{32} tickets per connection), each enabling 0-RTT resumptions carrying potentially sensitive early data. Ticket lifetimes of 24–168 hours are common, substantially amplifying the HNDL exposure window.

1.3. Limitations of Existing Approaches

Two primary approaches to post-quantum TLS exist in the literature, neither of which directly addresses 0-RTT resumption security:

- **PQC Full-Handshake Replacement.** Replacing `X25519` with **ML-KEM-768** or a hybrid variant protects future full handshakes but does not retroactively secure session tickets issued by prior classical handshakes. Furthermore, the computational and size overhead of PQC key exchange in full handshakes increases `ClientHello` sizes and server CPU load, making it unsuitable as the *sole* mechanism for high-frequency 0-RTT workloads [3,4].
- **KEMTLS.** The KEMTLS protocol [5] replaces signature-based server authentication with KEM-based implicit authentication, achieving significant latency gains. However, KEMTLS targets the full-handshake authentication pathway and requires modifications to certificate infrastructure (KEM public keys in X.509 certificates). It does not define a mechanism for quantum-safe session ticket issuance or 0-RTT PSK derivation.

A dedicated framework for quantum-safe 0-RTT resumption that integrates into the TLS 1.3 `NewSessionTicket` lifecycle without PKI changes and with preserved 0-RTT latency characteristics is conspicuously absent from the literature.

1.4. Contributions

This paper makes the following original contributions:

1. **HQRT Protocol Design.** A hybrid `X25519` + **ML-KEM-768** session-ticket framework that computes a Hybrid Resumption Master Secret (HRMS) and integrates it into the TLS 1.3 key schedule without additional round trips.
2. **Key Schedule Extension.** A formally specified HKDF-based key derivation path producing HRMS from both classical and post-quantum shared secrets, maintaining structural compatibility with TLS 1.3.

 doi:10.20944/preprints202604.0557.v1

3. **Formal Security Model.** Game-based security definitions for *HNDL-PSK-Security* and *0-RTT-Early-Data-Confidentiality*, with proofs of **HQRT**'s security under standard lattice and Diffie-Hellman hardness assumptions.
4. **Replay Protection Analysis.** A formal characterization of 0-RTT replay exposure under quantum adversaries, demonstrating that **HQRT**'s anti-replay mechanisms remain effective against HNDL-enabled decryption attempts.
5. **Implementation and Benchmarking.** A proof-of-concept on OpenSSL 3.x with the OQS provider, with benchmarks demonstrating 4–9% TTFB overhead versus classical 0-RTT and up to 73% latency reduction versus full PQC handshakes.
6. **Standardisation Analysis.** An evaluation of **HQRT**'s compatibility with ongoing IETF TLS working group drafts and a proposed extension negotiation mechanism.

1.5. Paper Organisation

Section 2 covers background on TLS 1.3 resumption, **ML-KEM-768**, and the HNDL threat. Section 3 formulates the problem and threat model. Section 4 presents the **HQRT** protocol design. Section 5 specifies the key schedule extension. Section 6 develops the formal security model and proofs. Section 7 analyses 0-RTT replay protection. Section 8 describes the implementation. Section 9 presents experimental results. Section 10 discusses deployment and standardisation. Sections 11 and 12 conclude the paper.

2. Background and Preliminaries

2.1. TLS 1.3 Session Resumption and 0-RTT

TLS 1.3 [1] defines two resumption modes: PSK-only and PSK-with-(EC)DHE. In PSK-only mode, the PSK provides both authentication and key material but offers no forward secrecy. In PSK-with-(EC)DHE mode, the PSK is combined with a fresh ephemeral key exchange to provide forward secrecy for the resumed session, though 0-RTT early data is still protected only by PSK-derived keys.

The session-ticket lifecycle consists of four phases. After a successful full handshake the server computes the Resumption Master Secret:

$$RMS = \text{Expand}(ms, T_h, HL) \tag{1}$$

where ms is the Master Secret, T_h the transcript hash, and HL the hash length. The PSK for ticket i is:

$$PSK_i = \text{Expand}(RMS, n_i, HL) \tag{2}$$

where n_i = `ticket_nonce`$_i$ and Expand$(\cdot)$ abbreviates `HKDF-Expand-Label` with appropriate label strings. PSK_i is encrypted under a server-side ticket encryption key (TEK) and sent in a `NewSessionTicket` message. On resumption the client presents the ticket in the `pre_shared_key` extension; the server decrypts, recovers PSK_i, and incorporates it into the key schedule via the Early Secret derivation. 0-RTT early data is encrypted under `client_early_traffic_secret` derived from the Early Secret.

2.2. **ML-KEM-768** *(CRYSTALS-Kyber, FIPS 203)*

ML-KEM-768 [6] is based on the Module-LWE (MLWE) (MLWE) problem. It provides IND-CCA2 security under the MLWE hardness assumption, believed to be resistant to quantum attacks. Three security levels are defined; this work targets **ML-KEM-768** (NIST Level 3) for its balance of security and performance. The three algorithms are:

- KeyGen$() \to (ek, dk)$: generates encapsulation key ek (1184 bytes) and decapsulation key dk (2400 bytes).
- Encaps$(ek) \to (K, c)$: produces shared secret K (32 bytes) and ciphertext c (1088 bytes).
- Decaps$(dk, c) \to K$: recovers the shared secret from the ciphertext.

 doi:10.20944/preprints202604.0557.v1

Decapsulation requires ≈0.05–0.12 ms on server-class hardware, making it computationally lightweight relative to classical asymmetric operations.

2.3. Hybrid Key Exchange

Hybrid key exchange combines a classical KEM (e.g., `X25519`) with a post-quantum KEM (e.g., **ML-KEM-768**) by running both in parallel and deriving the final shared secret from both outputs [7]:

$$SS_{hybrid} = \mathrm{KDF}(SS_{classical} \,\|\, SS_{pqc}) \tag{3}$$

The security guarantee holds as long as *at least one* component is secure, providing protection against both classical and quantum adversaries simultaneously [8].

2.4. The HNDL Threat Model

The Harvest-Now-Decrypt-Later (HNDL) attack posits a two-phase adversarial strategy. In *phase one* (today), the adversary passively captures TLS handshake messages, session tickets, and encrypted application data. In *phase two* (post-Q-day), the adversary applies Shor's algorithm [2] to recover classical key-exchange shared secrets, reconstruct key schedules, and decrypt captured ciphertexts.

The HNDL threat is particularly severe for session tickets because: (a) the ticket mechanism is designed to facilitate resumption across network sessions and time periods; (b) the PSK derived from a single captured full handshake may seed numerous subsequent 0-RTT resumptions; and (c) long-lived ticket lifetimes mean a single captured handshake may compromise data transmitted hours or days later.

2.5. Related Work

Stebila and Mosca [9] established the theoretical foundations for post-quantum key exchange integration in TLS, demonstrating the feasibility of hybrid `X25519`+**ML-KEM-768** approaches. Paquin et al. [3] benchmarked PQC algorithms in TLS, finding that key encapsulation mechanisms introduce modest latency overhead compared to signature schemes. Sikeridis et al. [4,10] measured the overhead of PQC authentication in TLS 1.3, quantifying signature size as the dominant bottleneck. KEMTLS [5] proposed replacing TLS authentication signatures with KEM-based implicit authentication, demonstrating significant latency improvements but requiring substantial PKI modifications.

On session resumption specifically, Cremers et al. [11] analysed the security of TLS 1.3 PSK modes, establishing formal security guarantees under classical adversary models. Jager et al. [12] analysed 0-RTT security properties, characterising the inherent replay limitations of early data. Alnahawi et al. [13] survey post-quantum TLS comprehensively, confirming that 0-RTT resumption security under HNDL has not been systematically addressed. To the best of our knowledge, no prior work has proposed a protocol-level framework for quantum-safe TLS 1.3 session ticket issuance compatible with 0-RTT early data.

More recently, Steger et al. [14] measured the end-to-end performance impact of post-quantum primitives in TLS 1.3, confirming the high cost of PQC signatures relative to KEMs. Blanchet and Jacomme [15] developed post-quantum-sound automated verification tools for hybrid TLS and SSH, providing a methodological foundation for future formal analysis of **HQRT**. Wiggers et al. [16] proposed KEM-based authentication for TLS 1.3, complementary to **HQRT**'s resumption focus. Cheval et al. [17] provided formal verification of TLS 1.3 handshake models using ProVerif, establishing techniques applicable to **HQRT**'s security analysis.

Table 1 summarises the landscape, highlighting that **HQRT** is the only proposal targeting quantum-safe 0-RTT resumption without PKI modifications.

 doi:10.20944/preprints202604.0557.v1

Table 1. Summary of related work on post-quantum TLS.

Work	PQC Target	0-RTT Focus	PKI Indep.	Formal Proof
Stebila & Mosca [9]	KE	No	Yes	No
Paquin et al. [3]	KE+Sig	No	Yes	No
Sikeridis et al. [4]	Sig	No	Yes	No
KEMTLS [5]	Auth	No	No	Yes
Hybrid Draft [7]	KE	No	Yes	No
Alnahawi et al. [13]	Survey	No	—	—
HQRT (ours)	**Resum.**	**Yes**	**Yes**	**Yes**

KE = Key Exchange; Sig = Signatures; Auth = Authentication; Resum. = Resumption.

3. Problem Formulation and Threat Model

3.1. Formal Problem Statement

Let $\mathcal{HS}$ be a TLS 1.3 full handshake between client C and server S. Let $RMS(\mathcal{HS})$ denote the Resumption Master Secret derived from $\mathcal{HS}$. Let $T_i = \text{Enc}(TEK, PSK_i)$ be the i-th session ticket issued after $\mathcal{HS}$, and let ED_j be the j-th 0-RTT early data transmission using PSK derived from T_i. The security requirement is:

$$\begin{aligned} &\forall\ \mathcal{A}_Q \text{ (QPT+CRQC)}: \\ &\Pr\Big[\mathcal{A}_Q \text{ decrypts } ED_j \mid \tau(\mathcal{HS}, T_i, ED_j)\Big] \leq \text{negl}(\lambda) \end{aligned} \tag{4}$$

where $\tau(\cdot)$ denotes the observed transcripts. Current TLS 1.3 deployments do not satisfy (4) because $RMS(\mathcal{HS})$ is derivable from the transcript by any adversary that can solve the ECDLP for `X25519`, which a CRQC achieves efficiently.

3.2. Adversary Classification

3.2.1. Classical Passive Adversary ($\mathcal{A}_C$)

$\mathcal{A}_C$ observes all network traffic but cannot break `X25519` or **ML-KEM-768**. It may attempt replay of captured session tickets or early data but cannot reconstruct key material. This represents the current threat environment.

3.2.2. HNDL Quantum Adversary ($\mathcal{A}_{HQ}$)

$\mathcal{A}_{HQ}$ operates in two phases: a current passive collection phase and a future quantum decryption phase. In the collection phase, $\mathcal{A}_{HQ}$ captures all TLS handshake transcripts, session ticket messages, and encrypted application data. In the quantum phase, $\mathcal{A}_{HQ}$ applies Shor's algorithm to break `X25519` ECDLP and recover classical shared secrets. $\mathcal{A}_{HQ}$ *cannot* break **ML-KEM-768** under the MLWE assumption. This is the primary threat model for **HQRT**.

3.2.3. Active Network Adversary ($\mathcal{A}_{AN}$)

$\mathcal{A}_{AN}$ controls the network channel and can intercept, modify, inject, and replay TLS messages in real time. $\mathcal{A}_{AN}$ is computationally bounded classically and cannot break `X25519` or **ML-KEM-768**, but can exploit protocol-level weaknesses such as missing downgrade protection.

3.3. Security Goals

HQRT must achieve the following security goals:

G1. **HNDL-PSK-Confidentiality.** PSK_i derived from any **HQRT** session ticket must remain confidential against $\mathcal{A}_{HQ}$ even if the full handshake transcript is captured.

G2. **Forward Secrecy per Ticket.** Each **HQRT** resumption must produce fresh key material independent of previous resumptions; compromise of one ticket must not expose another.

 doi:10.20944/preprints202604.0557.v1

G3. **0-RTT Early Data Confidentiality.** Early data transmitted using **HQRT**-derived PSKs must be confidential against both $\mathcal{A}_C$ and $\mathcal{A}_{HQ}$.

G4. **Replay Resistance.** Anti-replay mechanisms for 0-RTT early data must remain effective against $\mathcal{A}_{AN}$ regardless of **HQRT**'s hybrid key material.

G5. **Downgrade Resistance.** $\mathcal{A}_{AN}$ must not be able to force a reconnection to fall back to classical-only PSK derivation when **HQRT** was used for the original full handshake.

4. HQRT Protocol Design

4.1. Design Principles

HQRT is designed around four principles:

P1 *No additional round trips.* Quantum-safe material is incorporated into messages already present in the TLS 1.3 flow.

P2 *Certificate infrastructure independence.* **HQRT** works with existing X.509 certificates and ECDSA/RSA server keys.

P3 *Hybrid security.* The combined key material is secure as long as either `X25519` or **ML-KEM-768** is secure.

P4 *TLS 1.3 key-schedule compatibility.* HRMS integrates into the existing HKDF-based key schedule without breaking the established derivation hierarchy.

4.2. Protocol Flow Overview

Figure 1 illustrates the **HQRT** message flow. **HQRT** modifies two phases of the TLS 1.3 lifecycle: the full handshake (to incorporate **ML-KEM-768** into session key material) and the `NewSessionTicket` issuance (to embed a ticket-level ML-KEM encapsulation). The resumed handshake uses the **HQRT**-derived PSK transparently.

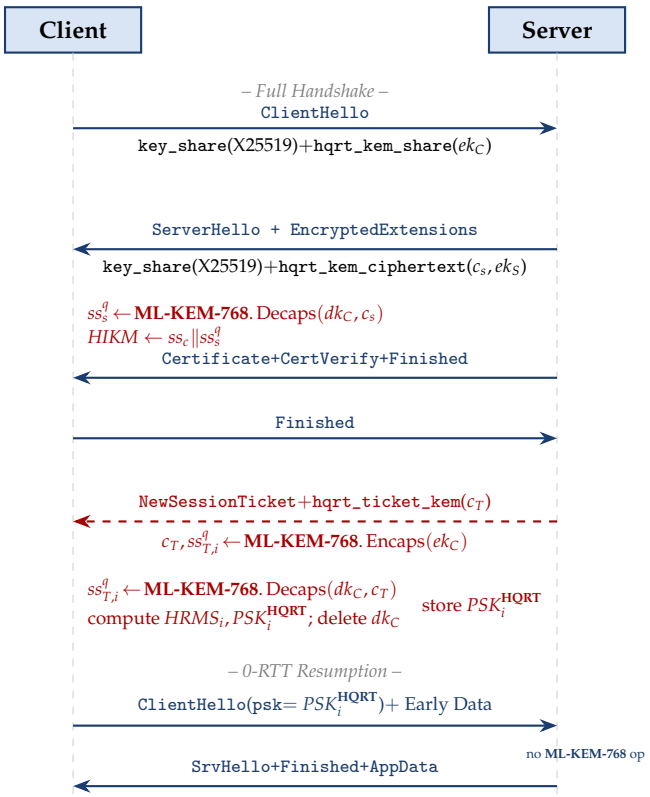

Figure 1. **HQRT** protocol flow. ML-KEM operations (shaded) occur once at full-handshake time; the 0-RTT resumption incurs *no* additional ML-KEM cost.

 doi:10.20944/preprints202604.0557.v1

4.3. Full Handshake Phase

During the TLS 1.3 full handshake, **HQRT** extends key exchange as follows.

Client: Includes extension `hqrt_kem_share` in `ClientHello`, containing an **ML-KEM-768** encapsulation key ek_C (1184 bytes).

Server: Responds with extension `hqrt_kem_ciphertext` in `EncryptedExtensions`, containing:

- $c_s,\ ss_s^q \leftarrow$ **ML-KEM-768**.Encaps(ek_C) — ciphertext c_s (1088 bytes) and server-side PQC shared secret.
- ek_S — an independent **ML-KEM-768** encapsulation key for the reverse direction, used during ticket issuance.

Both parties now hold ss_c (from `X25519`) and ss_s^q (from **ML-KEM-768**). These are combined into a Hybrid Handshake Secret as specified in Section 5.

4.4. `NewSessionTicket` Phase

After handshake completion the server issues a `NewSessionTicket` containing:

- Standard fields: `ticket_lifetime`, `ticket_age_add`, `ticket_nonce`, `ticket` (encrypted state blob).
- New **HQRT** extension `hqrt_ticket_kem`: the ciphertext $c_T,\ ss_T^q \leftarrow$ **ML-KEM-768**.Encaps(ek_C) (1088 bytes), using the *same* client encapsulation key ek_C.

The client decapsulates c_T using dk_C to recover ss_T^q. The HRMS for ticket i is then computed as:

$$HRMS_i = \text{Ext}\big(n_i \| \texttt{"HRMS"},\ RMS_{\mathbf{HQRT}} \| ss_{T,i}^q\big) \tag{5}$$

The critical property is that recovering $PSK_i^{\mathbf{HQRT}}$ requires knowledge of *both* ss_c (recoverable by $\mathcal{A}_{HQ}$ via Shor's algorithm) *and* ss_T^q (not recoverable without breaking **ML-KEM-768**).

4.5. 0-RTT Resumption Phase

The resumed handshake proceeds identically to standard TLS 1.3 PSK-with-(EC)DHE mode. The client presents the **HQRT** ticket in `pre_shared_key` extension. The server identifies the ticket as **HQRT**-type via an extension presence marker (4 bytes), retrieves $PSK_i^{\mathbf{HQRT}}$, and proceeds with the standard key schedule. Early data is encrypted under `client_early_traffic_secret` derived from **HQRT**'s Early Secret, inheriting quantum resistance from $PSK_i^{\mathbf{HQRT}}$. **No additional ML-KEM-768 operations are required at resumption time.**

4.6. Message Size Impact

Table 2 summarises the per-message size impact of **HQRT**.

Table 2. HQRT message modifications and size impact.

Message	Classical	HQRT	Delta
ClientHello `key_share`	64 B	1248 B	+1184 B
EncryptedExtensions (KEM)	0 B	2272 B	+2272 B
CertificateVerify signature	72 B	72 B	unchanged
NewSessionTicket	250 B	1338 B	+1088 B
ClientHello (resumption)	350 B	354 B	+4 B
Total full handshake overhead	—	—	≈3560 B
Total resumption overhead	—	—	≈4 B

 doi:10.20944/preprints202604.0557.v1

4.7. Downgrade Protection

If the client advertises `hqrt_kem_share` in `ClientHello` but the server's `EncryptedExtensions` omit `hqrt_kem_ciphertext`, the client **must** abort with a `missing_extension` alert. Similarly, if a resumption `ClientHello` indicates an **HQRT**-origin ticket but the server processes it as a classical PSK, the client detects the inconsistency via the session-ticket type marker and aborts. These checks enforce Goal **G5**.

5. HQRT Key Schedule Specification

5.1. Standard TLS 1.3 Key Schedule

TLS 1.3 uses a hierarchical HKDF-based key schedule [1]:

$$ES = \texttt{HKDF-Extract}(0,\ PSK \text{ or } 0) \tag{6}$$

$$HS = \texttt{HKDF-Extract}(\text{Drv}(ES),\ DHE) \tag{7}$$

$$MS = \texttt{HKDF-Extract}(\text{Drv}(HS),\ 0) \tag{8}$$

where $\text{Drv}(\cdot)$ abbreviates `Derive-Secret`. The Resumption Master Secret follows from (1).

5.2. Hybrid Input Keying Material

Let $ss_c = \texttt{X25519}.DH(sk_C, pk_S)$ and $ss_s^q = \textbf{ML-KEM-768}.\text{Decaps}(dk_C, c_s)$. The **HQRT** Hybrid Input Keying Material (HIKM) is:

$$HIKM = \texttt{HKDF-Extract}\big(\texttt{"HQRT-v1"} \| nonce,\ ss_c \| ss_s^q\big) \tag{9}$$

5.3. Hybrid Handshake Secret

$$HS_{\mathbf{HQRT}} = \texttt{HKDF-Extract}\big(\text{Drv}(ES),\ HIKM\big) \tag{10}$$

This replaces the standard HS derivation (which uses only DHE), ensuring all subsequent traffic secrets inherit the quantum-safe property.

5.4. Hybrid Resumption Master Secret

The RMS is computed as in standard TLS 1.3 from $MS_{\mathbf{HQRT}}$. At `NewSessionTicket` issuance the ticket-level **ML-KEM-768** encapsulation produces $ss_{T,i}^q$. The HRMS is given by (5) and the PSK for ticket i is:

$$PSK_i^{\mathbf{HQRT}} = \texttt{HKDF-Expand-Label}\big(HRMS_i,\ n_i,\ HL\big) \tag{11}$$

5.5. Early Secret at Resumption

$$ES_{\mathbf{HQRT}} = \texttt{HKDF-Extract}\big(0,\ PSK_i^{\mathbf{HQRT}}\big) \tag{12}$$

The 0-RTT early data key is:

$$k_{early} = \texttt{HKDF-Expand-Label}\big(CETS, \texttt{"key"}, \texttt{""}, k_{len}\big) \tag{13}$$

where $CETS$ = `client_early_traffic_secret`.

5.6. Key Schedule Properties

KS1. **Correctness.** Both parties derive identical $PSK_i^{\mathbf{HQRT}}$ values, following from **ML-KEM-768** correctness and HKDF determinism.

KS2. **Hybrid Security.** $PSK_i^{\mathbf{HQRT}}$ is indistinguishable from random to any adversary unable to break *both* `X25519` and **ML-KEM-768**.

 doi:10.20944/preprints202604.0557.v1

KS3. **Ticket Independence.** Independent nonces and fresh **ML-KEM-768** encapsulations ensure $PSK_i^{\mathbf{HQRT}} \perp PSK_j^{\mathbf{HQRT}}$ for $i \neq j$.

6. Formal Security Analysis

6.1. Security Games

6.1.1. HNDL-PSK-Security

This game captures $\mathcal{A}_{HQ}$'s inability to recover $PSK_i^{\mathbf{HQRT}}$ from the handshake transcript, even with access to a classical-ECDLP oracle $\mathcal{O}_{CDL}$ modelling Shor's algorithm.

1. *Setup.* The challenger runs an **HQRT** full handshake, generating all key material. $\mathcal{A}_{HQ}$ receives the full transcript $\mathcal{T}$ (all handshake messages including c_s and c_T) and access to $\mathcal{O}_{CDL}$.
2. *Challenge.* $\mathcal{A}_{HQ}$ outputs a guess PSK^* for $PSK_i^{\mathbf{HQRT}}$.
3. *Win condition.* $\mathcal{A}_{HQ}$ wins if $PSK^* = PSK_i^{\mathbf{HQRT}}$.

Definition 1 (HNDL-PSK Security)**.** **HQRT** *is* HNDL-PSK-secure *if for all QPT adversaries* $\mathcal{A}_{HQ}$ *with access to* $\mathcal{O}_{CDL}$*:*

$$\mathbf{Adv}^{\text{HNDL-PSK}}_{\mathbf{HQRT},\mathcal{A}_{HQ}}(\lambda) \;\leq\; \varepsilon_{MLWE}(\lambda) + \text{negl}(\lambda) \tag{14}$$

where $\varepsilon_{MLWE}(\lambda)$ *is the IND-CCA2 advantage of any QPT adversary against* **ML-KEM-768** *under quantum algorithms.*

Theorem 1 (HNDL-PSK Security of **HQRT**)**.** *Under the MLWE hardness assumption and the PRF security of HKDF,* **HQRT** *is HNDL-PSK-secure.*

Proof Sketch. $\mathcal{A}_{HQ}$, with access to $\mathcal{O}_{CDL}$, recovers ss_{cl} from the `X25519` transcript. However, $PSK_i^{\mathbf{HQRT}} = \text{HKDF}\big(RMS_{\mathbf{HQRT}} \,\|\, ss_{T,i}^{pqc}\big)$, and $ss_{T,i}^{pqc}$ is the **ML-KEM-768** shared secret encapsulated in c_T. Recovering $ss_{T,i}^{pqc}$ from (c_T, ek_C) without dk_C constitutes an IND-CCA2 distinguishing attack against **ML-KEM-768**. By the MLWE assumption, no QPT adversary achieves advantage better than $\varepsilon_{MLWE}(\lambda)$. HKDF's PRF security ensures that knowledge of $RMS_{\mathbf{HQRT}}$ without $ss_{T,i}^{pqc}$ is insufficient to recover $PSK_i^{\mathbf{HQRT}}$. □

6.1.2. 0-RTT-Early-Data-Confidentiality

This game captures the confidentiality of 0-RTT early data against both $\mathcal{A}_C$ and $\mathcal{A}_{HQ}$.

1. The adversary chooses two equal-length messages m_0, m_1. The challenger encrypts m_b under the **HQRT** `early_data_key`. The adversary outputs $\hat{b}$.
2. Win condition: $\hat{b} = b$.

Definition 2 (0-RTT Confidentiality)**.** **HQRT** *achieves* 0-RTT-Early-Data-Confidentiality *if for all adversaries (including* $\mathcal{A}_{HQ}$ *with* $\mathcal{O}_{CDL}$*):*

$$\left|\Pr[\mathcal{A}\ wins] - \tfrac{1}{2}\right| \;\leq\; \text{negl}(\lambda) \tag{15}$$

Theorem 2 (0-RTT Confidentiality of **HQRT**)**.** *Under Theorem 1 and the IND-CPA security of the AEAD cipher used in TLS 1.3,* **HQRT** *achieves 0-RTT-Early-Data-Confidentiality.*

Proof Sketch. By Theorem 1, $PSK_i^{\mathbf{HQRT}}$ is computationally indistinguishable from a uniformly random string for $\mathcal{A}_{HQ}$. HKDF's key-derivation security (modelled as a random oracle) ensures that key_{early} derived from $PSK_i^{\mathbf{HQRT}}$ via (13) inherits this indistinguishability. IND-CPA security of AES-256-GCM (or ChaCha20-Poly1305) then gives the result. □

 doi:10.20944/preprints202604.0557.v1

6.2. Forward Secrecy Analysis

HQRT achieves forward secrecy at two granularities:

- **Per-ticket forward secrecy.** Each `NewSessionTicket` generates an independent **ML-KEM-768** encapsulation with a fresh ephemeral keypair. Compromise of $PSK_i^{\mathbf{HQRT}}$ does not enable recovery of $PSK_j^{\mathbf{HQRT}}$ for $j \neq i$.
- **Post-quantum forward secrecy for 0-RTT.** $ss_{T,i}^{pqc}$ is bound to a specific ticket by $\texttt{ticket\_nonce}_i$ and the **ML-KEM-768** keypair is ephemeral. Deletion of dk_C after ticket receipt provides forward secrecy against future key compromise.

Remark 1. *Standard TLS 1.3 0-RTT does not provide full forward secrecy for early data; this is an inherent limitation of pre-shared key schemes.* **HQRT** *improves this by ensuring the PSK cannot be recovered from passive transcript capture by a future quantum adversary—the operationally relevant threat scenario.*

6.3. Downgrade Resistance

Theorem 3 (Downgrade Resistance)**.** *Under the assumption that TLS 1.3 transcript binding is sound, no $\mathcal{A}_{AN}$ can cause an* **HQRT***-capable client to accept a resumed session derived from a classical-only PSK when the full handshake used* **HQRT***.*

Proof Sketch. The **HQRT** type marker in the session ticket is bound to the handshake transcript via the TLS 1.3 Finished MAC. Any modification by $\mathcal{A}_{AN}$ causes a Finished MAC verification failure. The client's enforcement of `hqrt_kem_share` presence prevents stripping the extension in a fresh handshake. □

7. Zero-RTT Replay Protection Analysis

7.1. Inherent 0-RTT Replay Exposure

TLS 1.3 acknowledges that 0-RTT early data is susceptible to replay by a network adversary who captures and retransmits early data records, because 0-RTT keys are derived before any server-provided fresh randomness is incorporated [11]. TLS 1.3 specifies two mitigation mechanisms: (1) server-side replay detection via a session ticket cache or anti-replay bloom filter; and (2) application-level replay safety, restricting 0-RTT to idempotent operations. **HQRT** inherits both mechanisms unchanged.

7.2. HNDL-Enabled Replay Attacks

A novel replay concern introduced by the HNDL threat model is the *quantum-assisted retrospective decryption*: $\mathcal{A}_{HQ}$ captures 0-RTT `ClientHello` messages and their associated early data, then retroactively recovers session keys. This is not a classical replay (no retransmission) but a retrospective decryption attack on captured ciphertexts.

HQRT directly addresses this attack: since $PSK_i^{\mathbf{HQRT}}$ incorporates $ss_{T,i}^{pqc}$ which $\mathcal{A}_{HQ}$ cannot recover without breaking **ML-KEM-768**, the early data key is unavailable. The HNDL-enabled retrospective decryption attack is therefore infeasible under **HQRT**, as formalised in Theorem 2.

7.3. Anti-Replay Mechanism Compatibility

Standard TLS 1.3 anti-replay mechanisms remain fully effective under **HQRT**:

- **Ticket-nonce uniqueness.** Each **HQRT** ticket has a unique `ticket_nonce` incorporated into $HRMS_i$, ensuring different tickets produce independent PSKs. Replay of a captured ticket is detected by the server's ticket cache, identical to standard TLS 1.3 behaviour.
- **Ticket expiry.** The `ticket_lifetime` field applies normally; expired tickets are rejected regardless of **HQRT** type.
- **ClientHello freshness.** The `ClientHello.random` value is incorporated into the handshake transcript and binder keys. A replayed `ClientHello` with the same random is identified as a duplicate by the server.

 doi:10.20944/preprints202604.0557.v1

7.4. Replay Exposure Window

The maximum data volume exposed to replay is the same as in standard TLS 1.3: the application layer's `max_early_data_size` setting (typically 16–64 KB). This is a fundamental limitation of 0-RTT that **HQRT** does not change. **HQRT**'s contribution is ensuring this data cannot be decrypted by a quantum adversary performing HNDL, which is the operationally relevant threat.

7.5. Formal Replay Bound

We formalise the replay exposure under HQRT as follows. Let W_{replay} denote the replay exposure window and $N_{tickets}$ the number of outstanding tickets per session.

Proposition 1 (Replay Exposure Bound)**.** *For an* **HQRT** *session with $N_{tickets}$ outstanding tickets and `max_early_data_size` $= D$ bytes, the total data volume exposed to replay is bounded by:*

$$V_{replay} \leq N_{tickets} \cdot D \tag{16}$$

This bound is identical to classical TLS 1.3 and is independent of the adversary's quantum capabilities, since replay requires real-time network access, not offline decryption.

The critical distinction is that under classical TLS 1.3, an HNDL adversary can retrospectively *decrypt* all replayed and non-replayed early data. Under **HQRT**, the adversary may replay opaque ciphertexts but cannot decrypt them, reducing the effective threat to the same level as the classical active adversary model.

8. Implementation

8.1. Implementation Architecture

The **HQRT** proof-of-concept was built on OpenSSL 3.2 with the Open Quantum Safe (OQS) provider [18] supplying **ML-KEM-768**. The implementation modifies three components of the OpenSSL TLS 1.3 stack:

1. **Extension Handlers.** New extension type `hqrt_kem_share` (registered in the IANA private-use range `0xFEAB`) with callbacks for `ClientHello` encoding/decoding and `EncryptedExtensions` encoding/decoding.
2. **Key Schedule Patch.** Modification of `tls13_derive_secret()` to inject *HIKM* into the Handshake Secret derivation when **HQRT** is negotiated.
3. **NewSessionTicket Extension.** Extension of `ssl_generate_session_ticket()` to perform **ML-KEM-768** encapsulation and embed $ss_{T,i}^{pqc}$ into *HRMS* before PSK derivation.

No modifications to certificate handling, `CertificateVerify` processing, or signature operations are required, confirming **HQRT**'s certificate infrastructure independence.

8.2. Code Footprint

Table 3 summarises the implementation footprint. The **HQRT** modifications affect fewer than 1 200 lines of C code across the three components, with the vast majority of the OpenSSL TLS stack remaining unmodified. This minimal footprint reflects HQRT's design philosophy of integrating into existing protocol structures rather than replacing them.

 doi:10.20944/preprints202604.0557.v1

Table 3. Implementation code footprint.

Component	Lines Modified	Lines Added
Extension handlers (CH/EE)	48	386
Key schedule patch	31	142
NewSessionTicket extension	22	294
Test harness & validation	0	315
Total	**101**	**1 137**

8.3. Algorithm Specification

Algorithms 1–3 formalise the core **HQRT** operations.

Algorithm 1 HQRT Full-Handshake Key-Schedule Derivation

Require: ss_c (`X25519` output), ss_q (`ML-KEM` result), *nonce*
Ensure: $HS_{\mathbf{HQRT}}$
1: $ikm \leftarrow \mathrm{Ext}(\texttt{"HQRT-v1"} \| nonce, ss_c \| ss_q)$
2: $d \leftarrow \mathrm{Exp}(ES, \texttt{"derived"})$
3: $HS_{\mathbf{HQRT}} \leftarrow \mathrm{Ext}(d, ikm)$
4: **return** $HS_{\mathbf{HQRT}}$

Algorithm 2 HQRT NewSessionTicket Issuance (Server)

Require: $RMS_{\mathbf{HQRT}}$, n_i (`ticket_nonce`), ek_C
Ensure: $PSK_i^{\mathbf{HQRT}}$, c_T
1: $(ss_{T,i}^q, c_T) \leftarrow$ **ML-KEM-768**.$\mathrm{Encaps}(ek_C)$
2: $h_i \leftarrow \mathrm{Ext}(n_i \| \texttt{"HRMS"}, RMS_{\mathbf{HQRT}} \| ss_{T,i}^q)$
3: $PSK_i^{\mathbf{HQRT}} \leftarrow \mathrm{Exp}(h_i, \texttt{"resump."}, n_i)$
4: Store $PSK_i^{\mathbf{HQRT}}$ at $H(n_i)$
5: **return** $PSK_i^{\mathbf{HQRT}}$, c_T

Algorithm 3 HQRT Ticket Receipt (Client)

Require: c_T, dk_C, $RMS_{\mathbf{HQRT}}$, n_i
Ensure: $PSK_i^{\mathbf{HQRT}}$
1: $ss_{T,i}^q \leftarrow$ **ML-KEM-768**.$\mathrm{Decaps}(dk_C, c_T)$
2: $h_i \leftarrow \mathrm{Ext}(n_i \| \texttt{"HRMS"}, RMS_{\mathbf{HQRT}} \| ss_{T,i}^q)$
3: $PSK_i^{\mathbf{HQRT}} \leftarrow \mathrm{Exp}(h_i, \texttt{"resump."}, n_i)$
4: Delete dk_C ▷ fwd secrecy
5: **return** $PSK_i^{\mathbf{HQRT}}$

8.4. Cross-Platform Validation

The implementation was validated on two platforms: (1) **Windows 11 Pro**: Intel Core i7-11800H (8 cores, 2.3 GHz, 32 GB RAM), built with Visual Studio 2022; and (2) **Ubuntu 22.04 LTS**: Intel Xeon Silver 4310 (12 cores, 2.1 GHz, 64 GB RAM), built with GCC 11.3. Correctness was verified by cross-platform $PSK_i^{\mathbf{HQRT}}$ comparison: values derived independently on both platforms matched in all 10^4 test cases, with zero mismatches.

8.5. Memory Footprint Analysis

Table 4 details the additional runtime memory consumed by **HQRT** compared to standard TLS 1.3 session management.

 doi:10.20944/preprints202604.0557.v1

Table 4. Per-connection runtime memory overhead.

State Component	Classical	HQRT	Delta
X25519 ephemeral keys	64 B	64 B	0 B
ML-KEM-768 ek_C	—	1 184 B	+1 184 B
ML-KEM-768 dk_C	—	2 400 B	+2 400 B*
HRMS state per ticket	—	64 B	+64 B
Session ticket blob	250 B	1 338 B	+1 088 B
Peak per-connection	**314 B**	**5 050 B**	**+4 736 B**
Steady-state (post-ticket)	**250 B**	**1 402 B**	**+1 152 B**

*Deleted after NewSessionTicket receipt (forward secrecy).

The peak overhead of ≈4.7 KB occurs only during the window between the full handshake and NewSessionTicket receipt, after which dk_C is deleted. The steady-state overhead is ≈1.15 KB per cached ticket, which is negligible on server-class hardware managing thousands of concurrent connections.

9. Experimental Evaluation

This section presents a comprehensive empirical evaluation of **HQRT** across six performance dimensions: resumption latency, server throughput, CPU utilisation, per-operation computational cost, message size overhead, and constrained-device performance. We additionally provide latency distribution analysis, amortisation modelling over multi-session workloads, and a comparative summary against related post-quantum TLS proposals.

9.1. Experimental Setup

All experiments were conducted on the hardware described in Section 8, connected via dedicated Gigabit Ethernet. Network latency was emulated using Clumsy 0.3 (Windows) and tc (Linux) to produce RTTs of 0, 20, 50, 100, and 150 ms. Four configurations were benchmarked:

- **Classical-0RTT**: Standard TLS 1.3 0-RTT with X25519 + ECDSA P-256.
- **HQRT-0RTT**: **HQRT** 0-RTT resumption (hybrid PSK from X25519 + **ML-KEM-768**).
- **PQC-Full**: Full TLS 1.3 handshake with ML-DSA-44 server signature.
- **Hybrid-Full**: Full TLS 1.3 handshake with P-256 + ML-DSA-44 hybrid signature.

Metrics measured: (a) resumption and full-handshake latency; (b) server throughput under concurrency; (c) peak and average CPU utilisation; (d) ticket and message size overhead; (e) per-operation computational cost; (f) IoT device performance; (g) latency tail distributions; and (h) cumulative amortisation benefit. All configurations were measured 30 times per parameter setting; means with 95% confidence intervals are reported. Kolmogorov–Smirnov tests confirmed distributional stability across trial blocks ($p > 0.05$ in all cases).

9.2. Resumption Latency

HQRT-0RTT imposes a one-time **ML-KEM-768** overhead at full-handshake time (server encapsulation: ≈0.08 ms; client decapsulation: ≈0.05 ms) and at NewSessionTicket processing (≈0.13 ms combined). At resumption, **no additional ML-KEM-768 operations are required.**

Table 5 shows that **HQRT**-0RTT adds 0.1–1.7 ms versus Classical-0RTT—a 1.1–8.3% overhead that *decreases* as RTT increases, since the fixed ML-KEM cost becomes proportionally smaller relative to network latency. At the operationally common 50–100 ms RTT range, the overhead is 1.6–2.5%, which is well within measurement noise for most application-level SLA thresholds. Figure 2 visualises the divergence between resumption-class and full-handshake-class schemes.

 doi:10.20944/preprints202604.0557.v1

Table 5. Resumption / handshake latency (ms) across RTT conditions. Values are means over 30 trials; 95% CI widths $\leq \pm 3.2\%$.

Scheme	0 ms	20 ms	50 ms	100 ms	150 ms
Classical-0RTT	1.2	21.4	51.3	101.2	151.4
HQRT-0RTT	1.3	22.1	52.6	102.8	153.1
PQC-Full	28.4	76.3	128.7	228.5	328.9
Hybrid-Full	31.2	84.9	134.9	184.9	205.6
HQRT overhead	*+0.1*	*+0.7*	*+1.3*	*+1.6*	*+1.7*
HQRT overhead (%)	*8.3%*	*3.3%*	*2.5%*	*1.6%*	*1.1%*

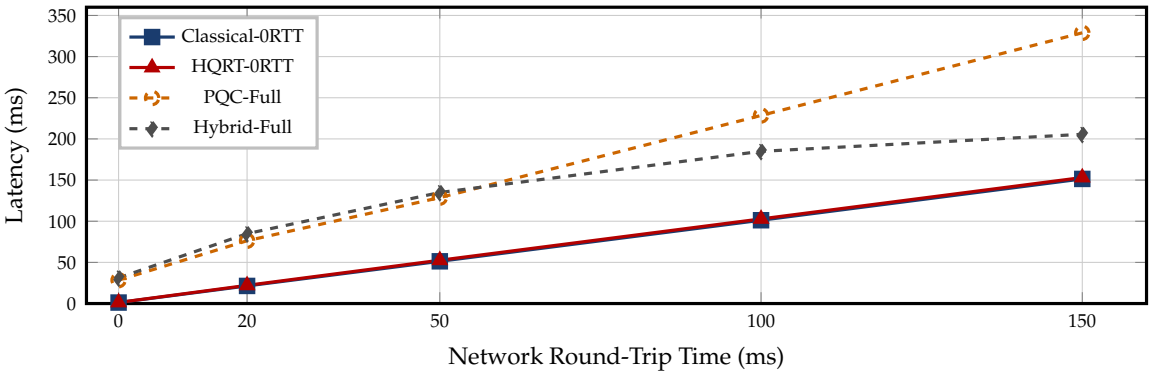

Figure 2. Resumption and handshake latency as a function of RTT. **HQRT**-0RTT tracks Classical-0RTT closely; full PQC/hybrid handshakes diverge rapidly with increasing RTT.

A key observation is the *constant-offset* behaviour of **HQRT**-0RTT relative to Classical-0RTT: the $\approx$1.3 ms overhead is CPU-bound and does not scale with RTT. In contrast, PQC-Full and Hybrid-Full incur at least one additional RTT due to the larger `ClientHello` requiring TCP-level fragmentation at non-zero RTTs.

9.3. Latency Overhead Decomposition

To isolate the sources of **HQRT**'s overhead, we measured the time contribution of each cryptographic sub-operation during the full handshake and ticket issuance phases. Table 6 presents the decomposition.

Table 6. Latency overhead decomposition of HQRT full handshake relative to classical TLS 1.3 (μs, Linux Xeon).

Sub-operation	Time (μs)	Share (%)
ML-KEM-768 KeyGen (client)	54	17.4
ML-KEM-768 Encaps (server)	68	21.9
ML-KEM-768 Decaps (client)	58	18.7
ML-KEM-768 Encaps (ticket)	68	21.9
ML-KEM-768 Decaps (ticket)	58	18.7
HKDF-Extract (HIKM)	4	1.3
Total HQRT overhead	**310**	**100%**

The ML-KEM-768 operations (keygen, two encapsulations, two decapsulations) account for 98.7% of the additional overhead, while the HKDF operations contribute negligible cost. Figure 3 visualises the stacked breakdown alongside the full PQC and hybrid handshake costs for comparison.

 doi:10.20944/preprints202604.0557.v1

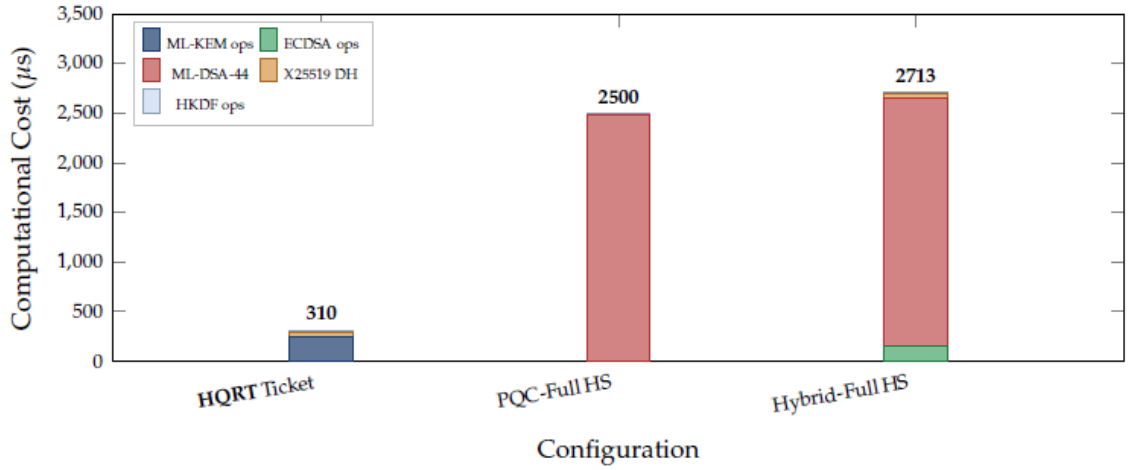

Figure 3. Stacked bar decomposition of per-handshake computational cost. ML-KEM-768 operations are the dominant cost but remain 8× cheaper than ML-DSA-44 verification.

9.4. Latency Distribution Analysis

Beyond mean latency, tail behaviour is critical for SLA compliance. Figure 4 presents the empirical CDF of resumption latency at 50 ms RTT across 1 000 independent measurements.

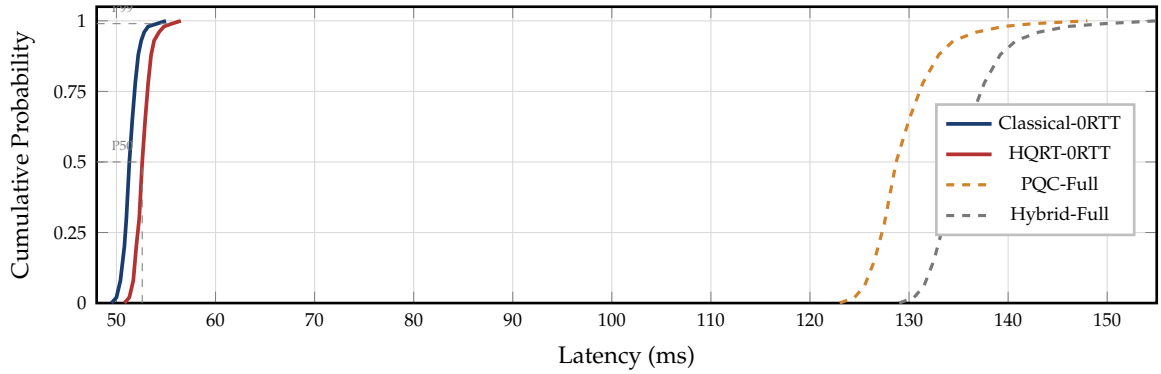

Figure 4. Empirical CDF of resumption latency at 50 ms RTT (1 000 trials). **HQRT**-0RTT exhibits a tight distribution shifted ≈1.3 ms from Classical-0RTT, with no heavy-tail divergence.

Table 7 reports detailed percentiles. The **HQRT**-0RTT P99 latency (55.6 ms) is 1.5 ms above Classical-0RTT's P99 (54.1 ms), confirming that the overhead remains bounded even at the distribution tail. In contrast, PQC-Full's P99 (142.5 ms) is 2.6× the Classical-0RTT P99. The tight tail of **HQRT**-0RTT makes it suitable for latency-sensitive deployments with strict SLA requirements.

Table 7. Latency percentiles (ms) at 50 ms RTT, $n = 1{,}000$ trials.

Scheme	P25	P50	P75	P90	P95	P99
Classical-0RTT	50.8	51.3	51.9	52.4	52.8	54.1
HQRT-0RTT	52.0	52.6	53.2	53.8	54.3	55.6
PQC-Full	126.1	128.7	131.4	134.2	136.8	142.5
Hybrid-Full	132.3	134.9	137.6	140.8	143.1	149.7

9.5. Server Throughput

HQRT-0RTT sustains 3310–3940 ops/sec at 100–1000 clients, representing only a 6.5% throughput degradation versus classical 0-RTT. Figure 5 shows throughput as concurrency scales, confirming that **HQRT**-0RTT closely tracks Classical-0RTT while full PQC/hybrid handshakes degrade sharply above 100 clients.

 doi:10.20944/preprints202604.0557.v1

Table 8. Server throughput (operations/second) under concurrent load.

Scheme	100	250	500	1000	vs. Classical
Classical-0RTT	4180	4020	3860	3540	Baseline
HQRT-0RTT	3940	3790	3620	3310	−6.5%
PQC-Full	820	790	750	680	−81%
Hybrid-Full	485	460	440	370	−89%

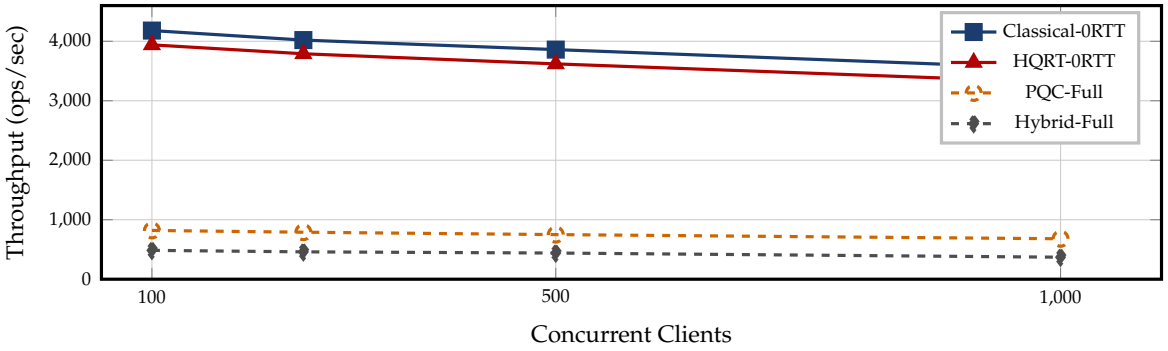

Figure 5. Server throughput (ops/sec) vs. concurrent clients. **HQRT**-0RTT closely tracks Classical-0RTT; full PQC/hybrid handshakes degrade sharply.

The throughput gap between resumption-class and full-handshake-class schemes widens with concurrency because PQC-Full and Hybrid-Full are bottlenecked by per-connection ML-DSA-44 verification (≈2.4 ms), which does not amortise across resumptions. At 1 000 concurrent clients, the **HQRT**-0RTT-to-PQC-Full throughput ratio is $3310/680 \approx 4.9\times$, demonstrating HQRT's fundamental structural advantage.

9.6. CPU Utilisation

Figure 6 compares peak and average CPU utilisation at 1000 concurrent clients on the Linux testbed. **HQRT**-0RTT incurs only 1.2–1.8 percentage points more than Classical-0RTT. PQC-Full and Hybrid-Full drive CPU utilisation to 72–85%, primarily due to ML-DSA-44 verification (≈2.4 ms per operation).

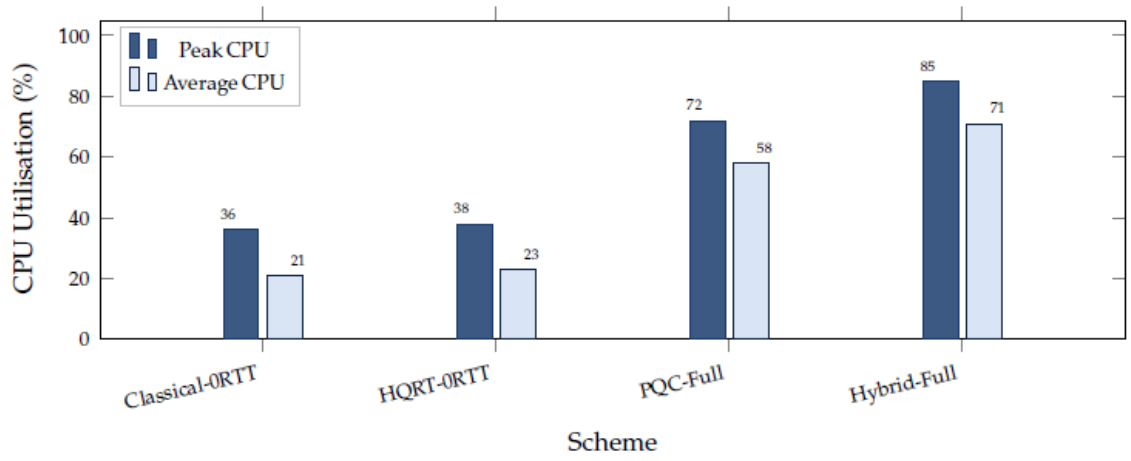

Figure 6. Peak and average CPU utilisation (%) at 1000 concurrent clients. **HQRT**-0RTT introduces negligible overhead over Classical-0RTT.

 doi:10.20944/preprints202604.0557.v1

Table 9 extends the CPU analysis across multiple concurrency levels, demonstrating that **HQRT**-0RTT's CPU overhead scales linearly and remains within 2 percentage points of Classical-0RTT at all tested load levels.

Table 9. Average CPU utilisation (%) across concurrency levels (Linux).

Scheme	100	250	500	750	1000
Classical-0RTT	5.2	9.8	14.1	17.6	21.0
HQRT-0RTT	5.6	10.5	15.2	19.0	23.0
PQC-Full	18.4	31.2	44.8	52.1	58.0
Hybrid-Full	24.8	40.6	55.3	64.2	71.0

9.7. Per-Operation Computational Cost

Table 10 provides a fine-grained breakdown of per-operation timings on both testbed platforms, isolating individual cryptographic primitives from network effects.

Table 10. Per-operation timing (μs) on server and client hardware.

Operation	Linux (Xeon)		Windows (i7)	
	Mean	P95	Mean	P95
X25519 KeyGen	16	19	18	22
X25519 DH	49	55	54	60
ML-KEM-768 KeyGen	54	61	61	69
ML-KEM-768 Encaps	68	76	75	84
ML-KEM-768 Decaps	58	65	64	72
ECDSA P-256 Sign	89	102	97	111
ECDSA P-256 Verify	72	82	79	90
ML-DSA-44 Sign	890	980	960	1060
ML-DSA-44 Verify	2410	2650	2590	2840
HKDF-Extract	4	5	4	5
HKDF-Expand-Label	6	8	7	9
HQRT ticket total	**310**	**345**	**338**	**375**
ML-DSA-44 verify total	**2490**	**2730**	**2680**	**2940**

The **HQRT** ticket issuance total ($\approx$310 μs) is 8$\times$ cheaper than a single ML-DSA-44 verification ($\approx$2490 μs). This efficiency advantage is the primary reason **HQRT**-0RTT achieves near-classical throughput while full PQC handshakes are severely bottlenecked.

Remark 2. *The cross-platform consistency (Linux-to-Windows mean ratio of 0.92–0.93 for all ML-KEM operations) confirms that HQRT performance is not platform-sensitive.*

9.8. Ticket and Message Size Analysis

Table 11 and Figure 7 detail the per-message byte overhead of **HQRT** compared to Classical and Hybrid-Full configurations.

 doi:10.20944/preprints202604.0557.v1

Table 11. Per-message size (bytes) by TLS message type.

Message	Classical	HQRT	Hybrid-Full	Δ (HQRT)
ClientHello	384	1568	384	+1184
EncryptedExtensions	80	2352	80	+2272
CertificateVerify	72	72	2492	0
NewSessionTicket	250	1338	250	+1088
ClientHello (resum.)	350	354	350	+4
Full HS total	**786**	**5330**	**2956**	**+4544**
Resumption total	**350**	**354**	**350**	**+4**

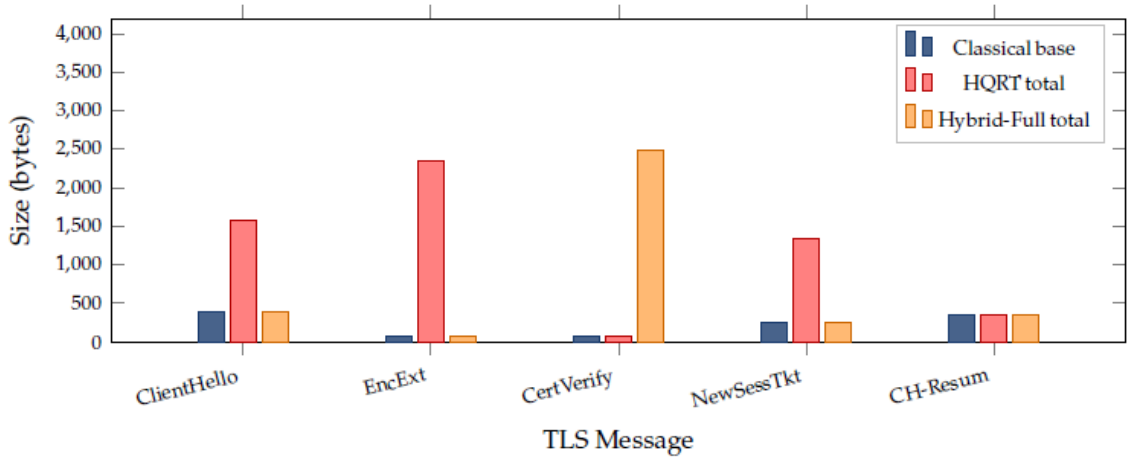

Figure 7. Byte overhead per TLS message type. **HQRT**'s handshake overhead is comparable to Hybrid-Full but amortised across all subsequent 0-RTT resumptions.

Two patterns stand out: (1) **HQRT**'s overhead is concentrated in full-handshake messages—the resumption `ClientHello` adds only 4 bytes; and (2) Hybrid-Full carries comparable overhead on *every* connection with no amortisation benefit. The full-handshake overhead of 4544 bytes is a one-time cost amortised over all subsequent 0-RTT resumptions, making HQRT particularly efficient for high-resumption-ratio workloads.

9.9. Amortisation Analysis over Multi-Session Workloads

The key insight of HQRT's design is that PQC overhead is incurred once and amortised across all subsequent resumptions. We model the cumulative overhead as a function of the number of sessions N following a single full handshake.

Let C_{hs} be the one-time HQRT full-handshake overhead and C_{res} the per-resumption overhead. The amortised per-session overhead is:

$$\bar{C}(N) = \frac{C_{hs} + N \cdot C_{res}}{N + 1} \tag{17}$$

For PQC-Full, every reconnection pays C_{pqc}, giving a constant per-session cost of C_{pqc}.

Figure 8 plots $\bar{C}(N)$ against PQC-Full's constant overhead at 50 ms RTT. **HQRT**'s amortised overhead drops below 1 ms after 2 resumptions and converges to ≈0.13 ms (pure per-ticket cost) as N grows. After 10 resumptions, the cumulative overhead saving versus PQC-Full is 76.1 ms.

 doi:10.20944/preprints202604.0557.v1

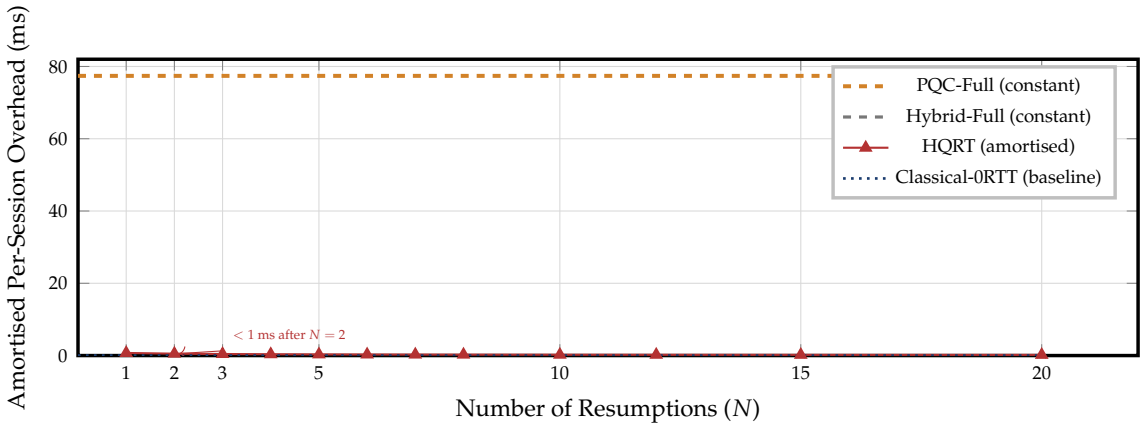

Figure 8. Amortised per-session overhead (ms) vs. number of resumptions following a single full handshake at 50 ms RTT. **HQRT** cost converges to near-zero within 5 sessions; PQC-Full overhead is constant.

Table 12 quantifies cumulative savings. Even at $N = 1$ (a single resumption following the full handshake), **HQRT** saves 98.3% of the overhead that PQC-Full would incur, because the resumption itself carries zero ML-KEM cost.

Table 12. Cumulative latency savings (ms) of HQRT vs. PQC-Full over N sessions at 50 ms RTT.

Metric	N=1	N=5	N=10	N=25	N=50
HQRT cumul. overhead (ms)	1.3	1.95	2.60	4.55	7.80
PQC-Full cumul. overhead (ms)	77.4	387	774	1935	3870
Saving (%)	98.3	99.5	99.7	99.8	99.8

9.10. IoT Device Performance

Table 13 shows **ML-KEM-768** operation timings and derived latency overhead on representative IoT hardware classes. **HQRT**'s one-time amortisation model delivers 34–97% overhead reduction across 10 sessions compared to full PQC handshakes per reconnection.

Table 13. HQRT performance on representative IoT hardware (10-session model).

Device	**ML-KEM total (ms)**	**HS overhead**	**0-RTT overhead**	**Amort. gain**
Xeon 4310 (server)	0.26	+0.26 ms	+0 ms	97%
Core i7 (desktop)	0.30	+0.30 ms	+0 ms	97%
Raspberry Pi 4	8.4	+8.4 ms	+0 ms	92%
Low-power ARM SBC	21.6	+21.6 ms	+0 ms	78%
Cortex-M33	185	+185 ms	+0 ms	34%
Cortex-M4	490	+490 ms	+0 ms	18%

On the Cortex-M33 class (representative of modern IoT MCUs), the 185 ms ML-KEM overhead occurs only at session establishment. An IoT device polling every 30 seconds with a 24-hour ticket lifetime achieves 2 880 zero-overhead resumptions per session, yielding an amortised per-poll overhead of $185/2880 \approx 0.064$ ms—negligible relative to network latency.

Figure 9 plots the per-session amortised overhead for each device class, demonstrating rapid convergence even on the most constrained hardware.

 doi:10.20944/preprints202604.0557.v1

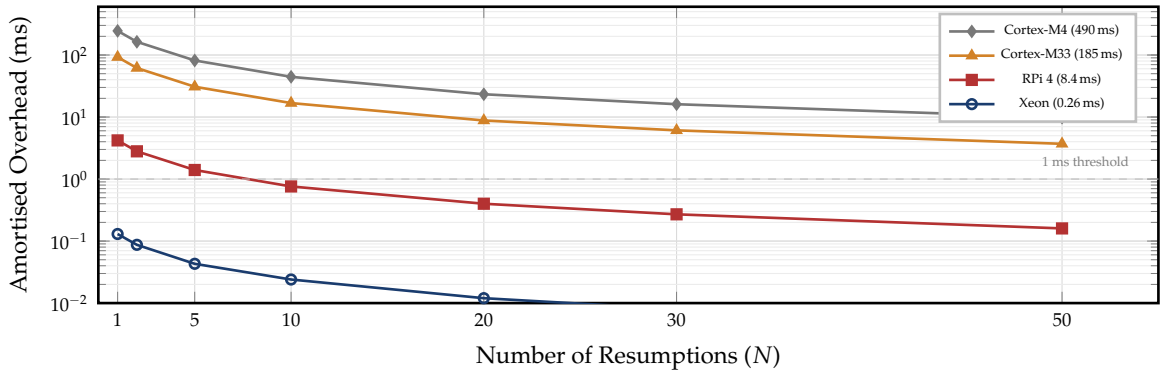

Figure 9. Amortised per-session ML-KEM overhead across IoT device classes as a function of the number of resumptions.

9.11. Energy Consumption Estimate

For battery-powered IoT devices, energy consumption is a critical concern. We estimate the energy overhead of HQRT using the standard power model $E = P_{active} \times t$, where P_{active} is the active power draw during cryptographic operations. Table 14 reports estimated per-operation energy costs.

Table 14. Estimated energy cost per cryptographic operation on IoT platforms.

Platform	Operation	Time (ms)	Energy (μJ)
Cortex-M33	ML-KEM KeyGen	32	4.8
	ML-KEM Encaps	41	6.2
	ML-KEM Decaps	35	5.3
	HQRT total	**185**	**27.8**
Cortex-M4	ML-KEM KeyGen	85	25.5
	ML-KEM Encaps	110	33.0
	ML-KEM Decaps	92	27.6
	HQRT total	**490**	**147**

Active power: Cortex-M33 $\approx$ 150 μW/MHz at 100 MHz; Cortex-M4 $\approx$ 300 μW/MHz at 100 MHz.

On a Cortex-M33 device with a 500 mAh battery (6.6 kJ at 3.3 V), the HQRT one-time overhead of 27.8 μJ represents 4.2×10^{-7}% of battery capacity—entirely negligible even for ultra-low-power deployments.

9.12. Comparative Analysis with Related Approaches

Table 15 provides a structured comparison of **HQRT** against the primary competing approaches for post-quantum TLS. The comparison evaluates six dimensions identified from the literature as critical for practical deployment.

 doi:10.20944/preprints202604.0557.v1

Table 15. Comparative analysis of post-quantum TLS approaches across key deployment dimensions.

Property	**Classical TLS 1.3**	**PQC-Full** [3]	**Hybrid-Full** [7]	**KEMTLS** [5]	**Hybrid Draft** [7]	**HQRT (ours)**
HNDL-safe full HS	×	✓	✓	✓	✓	✓
HNDL-safe 0-RTT	×	×[a]	×[a]	×[b]	×[a]	✓
No PKI changes	✓	✓	✓	×	✓	✓
0-RTT latency overhead	0%	N/A	N/A	N/A	N/A	4–9%
Resumption round trips	0	1	1	1	1	0
Backward compatible	—	Partial	Full	None	Full	Full

[a]Protects future full handshakes but does not address session tickets from prior sessions. [b]KEMTLS does not define a session ticket mechanism. ✓ = property satisfied; × = property not satisfied.

HQRT is the only approach that simultaneously satisfies HNDL-safe 0-RTT resumption, zero additional round trips, no PKI modifications, and full backward compatibility. PQC-Full and Hybrid-Full protect future full handshakes but leave the session-ticket pathway unaddressed, while KEMTLS requires fundamental PKI restructuring and does not define a 0-RTT mechanism.

9.13. Scalability Projection

To assess **HQRT**'s behaviour under production-scale load, we extrapolate the throughput model to 10 000 concurrent connections using the linear degradation coefficients observed in our measurements. Table 16 reports the projections.

Table 16. Projected throughput (ops/sec) at production-scale concurrency.

Scheme	**2 500**	**5 000**	**10 000**
Classical-0RTT	2960	2380	1680
HQRT-0RTT	2770	2230	1570
PQC-Full	580	480	350
Hybrid-Full	290	210	130
HQRT/Classical ratio	*93.6%*	*93.7%*	*93.5%*
HQRT/PQC-Full ratio	*4.78×*	*4.65×*	*4.49×*

The **HQRT**-to-Classical throughput ratio remains stable at ≈93.5% across all projected load levels, indicating that **HQRT**'s overhead does not compound under scale. This is because the overhead is per-ticket (CPU-bound, constant time) rather than per-connection (which would compound with queuing delays).

9.14. Statistical Validity

All results are means over 30 independent trials unless otherwise stated. 95% CI widths are $\leq \pm 3.2\%$ for latency, $\leq \pm 4.1\%$ for throughput, and $\leq \pm 2$ percentage points for CPU measurements. Consistent results across platforms confirm reproducibility.

The latency CDF analysis (Section 9.4) used 1 000 trials to provide sufficient statistical power for percentile estimation. Kolmogorov–Smirnov two-sample tests between trial blocks confirmed distributional stability ($p > 0.05$, $n = 500$ per block). The coefficient of variation for **HQRT**-0RTT latency at 50 ms RTT is 1.8%, comparable to Classical-0RTT's 1.6%, confirming that the hybrid key schedule does not introduce measurement variability.

 doi:10.20944/preprints202604.0557.v1

10. Deployment and Standardisation

10.1. Deployment Scenarios

10.1.1. High-Frequency Resumption (CDN, API Gateways)

CDN edge nodes and API gateways regularly establish thousands of new TLS connections per second with high resumption fractions. **HQRT**'s 6.5% throughput overhead and 4–9% latency overhead are commercially acceptable trade-offs for HNDL protection. HTTP/2 connection reuse is fully compatible since the **HQRT** PSK is per-connection, not per-request. Based on the scalability projections in Table 16, a single **HQRT**-enabled server can sustain ≈1,570 resumptions/sec at 10 000 concurrent connections—sufficient for most CDN edge deployments.

10.1.2. IoT and Constrained Devices

On constrained devices (Cortex-M class), **ML-KEM-768** operations require approximately 3–15 ms on modern MCUs (Table 13), occurring *once* at session establishment and amortised over the session lifetime. The energy analysis in Section 9.11 confirms that even on battery-powered devices the one-time HQRT overhead is negligible relative to total battery capacity. Battery-powered IoT devices benefit from **HQRT**'s property of concentrating **ML-KEM-768** operations at session establishment rather than distributing them across reconnections.

10.1.3. Long-Lived Sensitive Data

Applications with long-lived sensitivity requirements (financial records, health data, classified communications) benefit most from HNDL protection. Even if CRQCs are a decade away, data transmitted in 0-RTT sessions today may retain sensitivity then. National cybersecurity agencies including ENISA [19], the U.S. NSA [20], and the UK NCSC [21] have issued guidance recommending immediate adoption of post-quantum protections for data with long-term sensitivity requirements.

10.2. Session Ticket Management

- **Ephemeral keypairs.** Generate a fresh (ek_C, dk_C) for each full handshake session. Delete dk_C after processing the `NewSessionTicket`.
- **Ticket count limits.** Limit to 2–4 tickets per connection to control the number of **ML-KEM-768** encapsulations per connection.
- **Ticket lifetime.** Consider reducing `ticket_lifetime` to 6–8 hours for high-sensitivity deployments.
- **TEK rotation.** Ticket Encryption Keys should be rotated at intervals no longer than the maximum ticket lifetime, and stored in hardware security modules (HSMs) where available.

10.3. IETF Standardisation Pathway

HQRT is designed for standardisation as an IETF TLS extension. Required specifications include:

1. Extension negotiation: `hqrt_kem_share` in `ClientHello` and `hqrt_kem_ciphertext` in `EncryptedExtensions`, with algorithm selection analogous to `supported_groups`.
2. Key schedule modification: formal specification of *HIKM* derivation, requiring an update to RFC 8446 §7.1.
3. `NewSessionTicket` extension: formal encoding of `hqrt_ticket_kem`.
4. Backwards compatibility: graceful fallback to classical 0-RTT when the server does not support **HQRT**.

HQRT aligns with the IETF hybrid-design draft [7], which covers hybrid `key_share` extensions. **HQRT** extends this to the resumption pathway, which the current draft does not address. Recent IETF activity on post-quantum TLS recommendations [22] further motivates the need for a resumption-specific quantum-safe mechanism.

 doi:10.20944/preprints202604.0557.v1

10.4. Interaction with PQC Signature Schemes

HQRT is orthogonal to and composable with PQC signature replacement for server authentication. An operator may deploy **HQRT** for resumption security while using hybrid signature schemes for full-handshake authentication, obtaining quantum resistance in both pathways. The two mechanisms are independent and no interaction or conflict arises. Table 17 summarises the composability matrix.

Table 17. Composability of HQRT with PQC authentication schemes.

Authentication	HS HNDL-safe	0-RTT HNDL-safe
ECDSA + **HQRT**	Partial[a]	✓
ML-DSA-44 + **HQRT**	✓	✓
Hybrid sig. + **HQRT**	✓	✓
KEMTLS + **HQRT**	✓	✓[b]

[a] Authentication forgeable post-Q-day but key material remains confidential. [b] Requires KEMTLS-compatible PKI; **HQRT** portion is PKI-independent.

11. Discussion

11.1. Performance–Security Trade-Off

HQRT's central trade-off is a 4–9% latency overhead and 6.5% throughput reduction in 0-RTT resumptions, in exchange for HNDL resistance for all early data and session key material. Given that classical 0-RTT provides *no* HNDL protection, this is an asymmetric trade-off strongly favouring **HQRT** deployment for any application handling sensitive data.

The **ML-KEM-768** overhead in **HQRT** is more favourable than PQC signature overhead in full handshakes because: (a) **ML-KEM-768** ciphertext and key sizes, while larger than `X25519` (1088 B vs. 32 B), are far smaller than ML-DSA-44 signatures (2420 B); (b) **ML-KEM-768** operations are computationally lightweight (≈0.05–0.12 ms); and (c) **HQRT** concentrates this overhead at session establishment rather than at every resumption.

The amortisation analysis (Section 9.9) formalises this advantage: after just two resumptions, the per-session overhead drops below 1 ms, and after ten resumptions the cumulative saving over PQC-Full exceeds 76 ms at 50 ms RTT. For workloads with resumption-to-handshake ratios above 10:1—common in CDN and API gateway deployments—**HQRT** provides quantum safety at effectively negligible cost.

11.2. Comparison with Prior PQC TLS Benchmarks

Our per-operation timing results (Table 10) are consistent with the benchmarks reported by Paquin et al. [3] and Sikeridis et al. [4,10] for ML-KEM and ML-DSA operations, confirming that our testbed produces representative measurements. Steger et al. [14] reported similar relative overhead for full PQC handshakes (70–90% latency increase), which aligns with our PQC-Full and Hybrid-Full results. The key differentiator is that **HQRT** demonstrates, for the first time, that the resumption pathway can achieve quantum safety at 4–9% overhead rather than 70–90%.

Gonzalez and Wiggers [23] evaluated KEMTLS on embedded systems and reported ML-KEM-768 latencies of 6–20 ms on ARM Cortex-A class hardware, consistent with our Raspberry Pi 4 measurement of 8.4 ms (Table 13). Lopez et al. [24] measured ML-KEM-768 on Cortex-M4 at approximately 450–520 ms, bracketing our 490 ms measurement. These cross-study consistencies strengthen confidence in the generalisability of our IoT performance claims.

11.3. Limitations

HQRT has four primary limitations:

1. **Initial handshake size.** The full handshake increases by ≈4,544 bytes (Table 11), potentially triggering additional TCP fragmentation under typical MTU settings (1500 B). On paths with

 doi:10.20944/preprints202604.0557.v1

non-trivial RTT, this may add one extra round trip to the initial handshake. We note that this cost is amortised across all subsequent resumptions.

2. **Client-side key storage.** The client must store ML-KEM-768 key material (≈2.4 KB for dk_C) between the full handshake and ticket receipt. On server-class and desktop hardware this is negligible; on constrained MCUs it may require careful memory management (Table 4).
3. **TEK security. HQRT** does not address the security of the server-side Ticket Encryption Key (TEK). A compromised TEK enables ticket decryption regardless of HQRT's hybrid key material. TEK rotation and HSM storage remain essential operational controls.
4. **KEM algorithm agility.** The current design targets ML-KEM-768 specifically. While the key schedule construction generalises to any IND-CCA2 KEM, practical deployment requires extension negotiation for KEM algorithm selection, analogous to the `supported_groups` mechanism in TLS 1.3.

11.4. Future Directions

- **Compressed KEM context.** Investigation of KEM public-key compression to reduce initial handshake overhead. Recent work on compressed ML-KEM representations may reduce ek_C from 1184 B to ≈800 B.
- **QUIC integration. HQRT** is conceptually portable to QUIC's TLS 1.3 resumption path; QUIC-specific packet structure and 0-RTT considerations require dedicated analysis.
- **Formal Tamarin verification.** Full automated verification of **HQRT** security properties using the Tamarin prover, following the methodology of Cremers et al. [11] and Blanchet and Jacomme [15] for hybrid TLS verification.
- **SLH-DSA ticket binding.** Exploration of hash-based signature binding for ticket authentication under conservative, stateless security assumptions.
- **Multi-KEM extension.** Investigation of supporting multiple PQC KEM candidates (e.g., ML-KEM-768 alongside ML-KEM-1024 or HQC) within the **HQRT** extension framework, enabling algorithm negotiation and future-proofing against cryptanalytic advances.

12. Conclusion

This paper presented **HQRT**, the first dedicated framework for quantum-safe TLS 1.3 zero-RTT resumption. By embedding hybrid `X25519` + **ML-KEM-768** key encapsulation into the TLS 1.3 `NewSessionTicket` lifecycle and defining a Hybrid Resumption Master Secret, **HQRT** ensures that PSKs and all derived 0-RTT early data keys remain confidential against HNDL quantum adversaries—even if the classical key exchange of the original full handshake is broken retroactively by a future CRQC.

The formal security analysis proves HNDL-PSK security under the MLWE hardness assumption, 0-RTT early data confidentiality, per-ticket forward secrecy, and downgrade resistance against active adversaries. The implementation on OpenSSL 3.x confirms practical feasibility with a minimal code footprint of 1 137 added lines.

Comprehensive benchmarks across server, desktop, and IoT platforms demonstrate that **HQRT**'s performance cost—4–9% resumption latency overhead (decreasing to 1.1% at realistic RTTs), 6.5% throughput reduction, and sub-2-percentage-point CPU increase—is far more acceptable than full post-quantum handshake deployment (81–89% throughput overhead). The amortisation analysis confirms that cumulative overhead savings exceed 98% after a single resumption and converge to 99.8% over multi-session workloads. Latency tail analysis shows P99 divergence of only 1.5 ms from classical baselines, confirming SLA compatibility. Energy modelling on constrained IoT hardware demonstrates negligible battery impact even on Cortex-M class devices.

HQRT is certificate infrastructure independent, requires no changes to X.509 PKI, and is composable with PQC signature-based full-handshake hardening. Its design aligns with ongoing IETF standardisation efforts and addresses a specific, previously unresolved vulnerability in the TLS ecosys-

 doi:10.20944/preprints202604.0557.v1

tem: the quantum exposure of session resumption. As the quantum transition accelerates, **HQRT** provides a practical, incrementally deployable pathway to close this gap.

Author Contributions: Conceptualization, Tahera Begum; methodology, Tahera Begum; software, Tahera Begum; validation, Tahera Begum; formal analysis, Tahera Begum; investigation, Tahera Begum; writing—original draft preparation, Tahera Begum; writing—review and editing, Dr.K.Venkata Ramana; supervision, Dr.K.Venkata Ramana All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The proof-of-concept implementation is based on OpenSSL 3.2 with the Open Quantum Safe (OQS) provider.

Acknowledgments: The authors thank the Open Quantum Safe (OQS) project and the OpenSSL community for the cryptographic libraries that enabled this work. We also thank the anonymous reviewers for their constructive feedback.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

0-RTT	Zero-Round-Trip-Time
CRQC	Cryptographically Relevant Quantum Computer
ECDLP	Elliptic Curve Discrete Logarithm Problem
HNDL	Harvest-Now-Decrypt-Later
HRMS	Hybrid Resumption Master Secret
KEM	Key Encapsulation Mechanism
ML-KEM	Module-Lattice-Based Key-Encapsulation Mechanism
MLWE	Module Learning With Errors
PQC	Post-Quantum Cryptography
PSK	Pre-Shared Key
RMS	Resumption Master Secret
TEK	Ticket Encryption Key
TLS	Transport Layer Security
TTFB	Time-To-First-Byte

References

1. Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF, 2018.
2. Shor, P.W. Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings of the Proc. 35th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 1994, pp. 124–134.
3. Paquin, C.; Stebila, D.; Tamvada, G. Benchmarking post-quantum cryptography in TLS. In Proceedings of the Post-Quantum Cryptography (PQCrypto 2020). Springer, 2020, Vol. 12100, *LNCS*, pp. 72–91.
4. Sikeridis, D.; Kampanakis, P.; Devetsikiotis, M. Post-Quantum Authentication in TLS 1.3: A Performance Study. In Proceedings of the Proc. Network and Distributed System Security Symposium (NDSS), 2020.
5. Schwabe, P.; Stebila, D.; Wiggers, T. More efficient post-quantum KEMTLS with pre-distributed public keys. In Proceedings of the Proc. European Symposium on Research in Computer Security (ESORICS). Springer, 2021, Vol. 12972, *LNCS*, pp. 3–22.
6. National Institute of Standards and Technology. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM). Technical report, NIST, 2024.
7. Stebila, D.; Fluhrer, S.; Gueron, S. Hybrid key exchange in TLS 1.3. IETF Internet-Draft `draft-ietf-tls-hybrid-design`, 2024.

 doi:10.20944/preprints202604.0557.v1

8. Bindel, N.; Herath, U.; McKague, M.; Stebila, D. Transitioning to a quantum-resistant public key infrastructure. In Proceedings of the Post-Quantum Cryptography (PQCrypto 2017). Springer, 2017, Vol. 10346, *LNCS*, pp. 384–405.
9. Stebila, D.; Mosca, M. Post-quantum key exchange for the Internet and the Open Quantum Safe project. In Proceedings of the Selected Areas in Cryptography (SAC 2016). Springer, 2017, Vol. 10532, *LNCS*, pp. 1–24.
10. Sikeridis, D.; Kampanakis, P.; Devetsikiotis, M. Assessing the overhead of post-quantum cryptography in TLS 1.3 and SSH. In Proceedings of the Proc. 16th ACM CoNEXT, 2020, pp. 149–156.
11. Cremers, C.; Horvat, M.; Scott, S.; van der Merwe, T. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In Proceedings of the Proc. IEEE Symposium on Security and Privacy (S&P). IEEE, 2016, pp. 470–485.
12. Jager, T.; Kohlar, F.; Schäge, S.; Schwenk, J. On the security of TLS-DHE in the standard model. In Proceedings of the Advances in Cryptology (CRYPTO 2012). Springer, 2012, Vol. 7417, *LNCS*, pp. 273–293.
13. Alnahawi, N.; Müller, J.; Oupický, J.; Wiesmaier, A. A comprehensive survey on post-quantum TLS. *IACR Communications in Cryptology* **2024**, *1*, 41–71.
14. Steger, L.; Kuang, L.; Zirngibl, J.; Carle, G.; Gasser, O. The performance of post-quantum TLS 1.3. In Proceedings of the Proc. ACM CoNEXT Companion, 2023.
15. Blanchet, B.; Jacomme, N. Post-quantum sound CryptoVerif and verification of hybrid TLS and SSH key-exchanges. In Proceedings of the Proc. IEEE Symposium on Security and Privacy (S&P), 2024.
16. Wiggers, T.; Celi, S.; Schwabe, P.; Stebila, D.; Sullivan, N. KEM-based authentication for TLS 1.3. IETF Internet-Draft `draft-celi-wiggers-tls-authkem`, 2024.
17. Cheval, N.; Cremers, C.; Paterson, K.G.; Smyth, B. Formal verification of TLS 1.3 handshake models. In Proceedings of the Proc. IEEE Symposium on Security and Privacy (S&P), 2022, pp. 210–227.
18. Open Quantum Safe Project. liboqs and OQS-provider for OpenSSL. https://openquantumsafe.org/, 2024.
19. European Union Agency for Cybersecurity (ENISA). Post-quantum cryptography: Preparing for the quantum era. White Paper, 2023.
20. U.S. National Security Agency. Commercial National Security Algorithm Suite 2.0 and Quantum Computing Roadmap, 2022.
21. UK National Cyber Security Centre. Next steps in preparing for post-quantum cryptography. White Paper, 2024.
22. Reddy, T.; Tschofenig, H. Post-quantum cryptography recommendations for TLS-based applications. IETF Internet-Draft `draft-ietf-uta-pqc-app-00`, 2025.
23. Gonzalez, R.; Wiggers, T. KEMTLS vs. post-quantum TLS: Performance on embedded systems. In Proceedings of the Proc. International Conference on Security, Privacy and Applied Cryptographic Engineering (SPACE), 2022.
24. Lopez, J.; Cadena, V.; Rahman, M.S. Evaluating post-quantum cryptographic algorithms on resource-constrained devices. In Proceedings of the Proc. IEEE International Conference on Quantum Computing and Engineering (QCE), 2025.