network

MDPI

Article

A Practical Implementation of Post-Quantum Cryptography for Secure Wireless Communication †

Babatunde Ojetunde [1,*,‡], Takuya Kurihara [1,‡], Kazuto Yano [1,‡], Toshikazu Sakano [1] and Hiroyuki Yokoyama [2]

[1] Wave Engineering Laboratories, Advanced Telecommunications Research Institute International, 2-2-2 Hikaridai, Seika, Soraku, Kyoto 619-0288, Japan; tkurihara@atr.jp (T.K.); kzyano@atr.jp (K.Y.); t.sakano@atr.jp (T.S.)

[2] Adaptive Communications Research Laboratories, Advanced Telecommunications Research Institute International, 2-2-2 Hikaridai, Seika, Soraku, Kyoto 619-0288, Japan; hr-yokoyama@atr.jp

* Correspondence: ojetunde@atr.jp; Tel.: +81-774-95-1595

† This material was presented in part at the IEEE CCNC 2025 conference.

‡ These authors contributed equally to this work.

Abstract: Recent advances in quantum computing have prompted urgent consideration of the migration of classical cryptographic systems to post-quantum alternatives. However, it is impossible to fully understand the impact that migrating to current Post-Quantum Cryptography (PQC) algorithms will have on various applications without the actual implementation of quantum-resistant cryptography. On the other hand, PQC algorithms come with complexity and long processing times, which may impact the quality of service (QoS) of many applications. Therefore, PQC-based protocols with practical implementations across various applications are essential. This paper introduces a new framework for PQC standalone and PQC–AES (Advanced Encryption Standard) hybrid public-key encryption (PKE) protocols. Building on prior results, we focus on securing applications such as file transfer, video streaming, and chat-based communication using enhanced PQC-based protocols. The extended PQC-based protocols use a sequence number-based mechanism to effectively counter replay and man-in-the-middle attacks and mitigate standard cybersecurity attack vectors. Experimental evaluations examined encryption/decryption speeds, throughput, and processing overhead for the standalone PQC and the PQC–AES hybrid schemes, benchmarking them against traditional AES-256 in an existing client–server environment. The results demonstrate that the new approaches achieve a significant balance between security and system performance compared to conventional deployments. Furthermore, a comprehensive security analysis confirms the robustness and effectiveness of the proposed PQC-based protocols across diverse attack scenarios. Notably, the PQC–AES hybrid protocol demonstrates greater efficiency for applications handling larger data volumes (e.g., 10–100 KB) with reduced latency, underscoring the practical necessity of carefully balancing security and operational efficiency in the post-quantum migration process.

Keywords: post-quantum cryptography; post-quantum key exchange; public-key encryption; IoT/5G/6G security; application; secure communication; digital signature

Academic Editor: Luis Alonso

Received: 1 May 2025
Revised: 5 June 2025
Accepted: 6 June 2025
Published: 10 June 2025

Citation: Ojetunde, B.; Kurihara, T.; Yano, K.; Sakano, T.; Yokoyama, H. A Practical Implementation of Post-Quantum Cryptography for Secure Wireless Communication. *Network* **2025**, *5*, 20. https://doi.org/10.3390/network5020020

1. Introduction

The development of quantum computing is advancing rapidly, transforming theoretical concepts into reality and raising concerns about the security of classical cryptographic systems. Among recent breakthroughs, Equal1 has launched Bell-1: The First Quantum System Purpose-Built for the HPC Era [1], complementing advancements by companies

such as IBM and Microsoft. Recent studies, such as Esmailiyan et al.'s work on CMOS position-based charge qubits [2] and Staszewski et al.'s development of cryogenic controllers for quantum dots [3], provide critical insights into the foundational technologies driving this revolution. It is anticipated that fully developed quantum computers will render classical cryptography insecure, potentially compromising all data encrypted using classical methods, including currently encrypted data. Therefore, addressing the imminent risks posed by traditional cryptographic algorithms has become increasingly urgent.

To address this issue and provide quantum-resistant cryptographic algorithms, the National Institute of Standards and Technology (NIST) introduced PQC algorithms [4–6], which will ensure that applications and network communications are secure against quantum attacks. However, it is not clear what impact or overhead will be introduced by migrating applications to be PQC-compliant, as PQC algorithms come with large ciphertext and key sizes. It is important to evaluate the practical implementation of PQC algorithms in real-world applications. Therefore, there is a need to investigate the performance trade-offs associated with the practical use of PQC algorithms and to address the unique requirements of emerging technologies such as 5G/6G and IoT. Furthermore, it remains uncertain how to ensure and meet their stringent quality-of-service (QoS) requirements.

NIST has selected four PQC algorithms [7] and released the standards for the selected PQC algorithms, which include one key encapsulation mechanism (KEM), CRYSTALS-Kyber (renamed to Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) in the standards) [8], and three digital signature algorithms: CRYSTALS-Dilithium (Module-Lattice-Based Digital Signature Algorithm (ML-DSA)) [9]; FALCON (Fast-Fourier Transform over NTRU-Lattice-Based Digital Signature Algorithm (FN-DSA)), whose draft will be released in FIP206; and SPHINCS+ (Stateless Hash-Based Digital Signature Algorithm (SLH-DSA)) [10]. In addition, as a backup to the ML-KEM algorithm, NIST has selected HQC [11] for the next PQC KEM algorithm to be standardized among the four PQC algorithms in Round 4: BIKE, Classic McEliece, HQC, and SIKE. The draft standard for HQC is expected to be released later.

Additionally, while the NIST standard [8] provides detailed specifications for implementing the ML-KEM algorithm, including key generation, key encapsulation, and decapsulation, it does not finalize specific methods for securing application data. Instead, it recommends the application of established symmetric-key cryptographic techniques, as outlined in other NIST standards, for encrypting and decrypting data to protect against unauthorized access. Therefore, to fully understand the implications of integrating PQC algorithms into various applications, particularly those in IoT, 5G, and 6G systems with stringent QoS requirements, a dedicated PQC-based protocol for securing data transmission is essential.

This paper introduces new and improved PQC-based protocols that extend the PQC standalone and PQC–AES hybrid PKE protocols to secure the communication of application data transmitted between two parties and mitigate common cybersecurity attack vectors. Specifically, we introduce a sequence-number-based mechanism to the PQC-based protocols to prevent replay attacks and man-in-the-middle (MITM) attacks, providing a practical application of PQC algorithms in real-world scenarios. This offers a tangible example of how these quantum-resistant cryptographic methods can be applied to secure communication and their impact on applications.

In the PQC standalone mode, a secure communication channel is established between the sender and receiver through a PQC key encapsulation algorithm. The exchanged public key is used for encryption, while each party uses its secret key for decryption (asymmetric-key cryptography). On the other hand, in the hybrid mode, a two-way shared secret key between the client and server is used to generate a symmetric key (symmetric-key

cryptography) as the encryption and decryption keys using a key derivation function (KDF). The encryption and decryption keys are then used by the client and server to encrypt and decrypt their files.

As stated in our prior work [12], the client generates a pair of PQC public and private keys, and the server generates its own keys in a similar manner. Using the PQC KEM, the public key is exchanged, establishing secure communication. Subsequently, the client encrypts its file using the encryption key, based on the protocol being used (PQC standalone or PQC–AES hybrid). Upon receiving the encrypted file, the server decrypts it using the decryption key to access the content. This process ensures end-to-end secure communication between the two parties (i.e., the client and the server). Furthermore, we devise various use-case scenarios for implementing the proposed PQC-based protocols in real-world applications. To integrate these protocols with pre-existing applications, they are designed to mitigate replay and MITM attacks, enhancing connection security. This ensures secure and seamless end-to-end communication between the two parties using the process defined by the proposed protocols. This improvement demonstrates the feasibility of these protocols for real-world secure communications over wireless networks while minimizing their impact on application use cases and network performance.

The goal of our proposed protocols is to enable the practical adoption of the PQC algorithm in real-world applications, balancing its impact on use cases and network performance. Our approach could also benefit 5G and 6G development by guiding the transition to PQC, particularly as there is growing consideration for shifting 5G security to a PKI-based trust model. The proposed protocols adopt the standardized algorithms by NIST, as in [8–10], and are designed to facilitate a smooth transition of applications from classical cryptography to quantum resistance utilizing both standalone PQC and PQC–AES hybrid protocols, thereby aligning with NIST's PQC migration guidelines as stated in SP 800-208 [13].

This paper significantly extends our prior work [12], in which we proposed the first PQC-based and PQC–AES hybrid PKE protocols for wireless file transfer and conducted initial performance benchmarking. In this paper, we expand on that work by introducing a new framework that includes a sequence-number-based mechanism to defend against replay and MITM attacks, along with an expanded security evaluation against common cybersecurity threats. Moreover, we extend the protocols to support a wider range of applications, including chat-based communication, video streaming, and live streaming. We also evaluate system performance under more diverse and realistic conditions and analyze the impact of these protocols on QoS and operational efficiency. This detailed analysis, along with the additional protocol advancements, provides critical insights beyond what was reported in [12] and offers practical guidance for deploying PQC in next-generation communication systems.

The rest of this paper is organized as follows. Section 2 reviews related works on PKE and PQC. Section 3 describes the proposed PQC-based protocols, while Section 4 details the use cases and the implementation of the protocol in applications. In Section 5, we discuss the security analysis of the proposed PQC standalone and PQC–AES hybrid PKE protocols. Section 6 presents a performance evaluation of the proposed method, and Section 7 concludes this paper.

2. Related Works

There has been extensive work, as well as several ongoing works, on the implementation of the PQC algorithm and its variants. In this section, we review some of the existing works and explain the major difference between our paper and such research.

Numerous studies have explored the operation [14,15] and implementation of the PQC KEM algorithm across different hardware platforms. Additionally, most current implementations of the PQC algorithm focus on its integration with Secure Sockets Layer (SSL), Transport Layer Security (TLS) [16–18], and Secure Shell (SSH). Furthermore, several works on PKE have been conducted, and there is ongoing work on the implementation of PQC algorithms and their variants.

In [19], Singh et al. proposed a new multivariate public-key cryptosystem based on permutation p-polynomials over finite fields, focusing on the need for secure and efficient public-key systems that can withstand increasing computing power and quantum computing. Marco et al. [20] proposed a generic transformation that achieves post-quantum security for classical signature schemes by hiding the public key with a one-time use of the key pair. Meanwhile, da Silva Lima et al. [21] evaluated the efficiency of the Kyber KEM algorithm in a mobile application, specifically analyzing its performance on the x86 and ARM architectures.

In addition, surveys and reviews highlighting the issues and constraints of migrating applications to be PQC-compliant have been carried out. Giron [22] discussed the challenges and research efforts required for migrating applications to PQC due to the vulnerability of current public-key cryptography schemes to quantum computers. Liu et al. [23] surveyed the performance and optimization of PQC algorithms for Internet of Things (IoT) systems. They highlighted the computational cost challenges of PQC algorithms for resource-constrained IoT devices and reviewed recent proposals for optimization. Similarly, Asif [24] presented a comprehensive review of PQC for the IoT, covering both theoretical and practical aspects. Li et al. [25] reviewed PQC algorithms, focusing on the Kyber algorithm, and discussed the challenges and opportunities of post-quantum security. Balamurugan et al. [26] discussed the need for PQC in light of the threat quantum computing poses to classical encryption schemes.

Fakhruldeen et al. [27] provided a comprehensive introduction to the emerging field of quantum-resistant cryptography, emphasizing its critical importance in safeguarding wireless networks against quantum computing threats. In their paper, they highlighted the vulnerabilities of current cryptographic standards to quantum attacks and explored solutions such as PQC and Quantum Key Distribution (QKD), alongside practical migration strategies like hybrid approaches. They further underscored the role of global standardization efforts, particularly by NIST, in advancing PQC algorithms while addressing challenges in integrating these techniques into existing infrastructures.

Despite these varied approaches, the practical application of the PQC algorithm remains confined to the aforementioned protocols and has not expanded to other commonly used applications. Moreover, directly applying PQC algorithms to various applications and systems may lead to performance issues, as many of these have stringent QoS requirements. Therefore, to fully assess the impact of direct PQC algorithm usage on application scenarios within a wireless network, a protocol that allows a PQC algorithm or hybrid mode to be used for securing application data is essential.

In addition, unlike in most implementations of PQC algorithms, the design and implementation of our proposed protocols extend beyond the integration of PQC into the traditional TLS/SSH to direct application-layer security and real-world scenarios. The use of a sequence-number-based mechanism to counter replay and MITM attacks provides an additional layer of security in addition to providing robust PQC algorithm protection. Unlike previous studies that focus mainly on algorithmic benchmarks or protocol integration at the transport layer, our proposed protocols demonstrate how existing applications can be practically migrated to achieve quantum resistance at the application layer.

3. PQC-Based PKE Protocols for Using the PQC Algorithm in Applications

According to the protocols described in [12], NIST recommends using the strongest algorithm parameters to ensure the effective use of PQC algorithms. The main reason for this is to avoid the cost and resources required to upgrade cryptographic implementations to a higher security level, as it is possible that upgrading may require modifications to the application/software where it will be needed. However, using stronger security parameters may have an adverse effect on application performance. Therefore, it is important to understand the impact of using PQC algorithms in various applications before carefully designing an optimization method or selecting a PQC algorithm for data encryption and decryption in various applications. To determine the impact of using PQC algorithms on applications and various networks, a protocol to encrypt and decrypt data transferred over the network is necessary. Therefore, we propose PQC standalone and PQC–AES hybrid PKE protocols that assume safe key exchange between the client and the server using the PQC KEM algorithm.

3.1. PQC Standalone PKE Protocol

The KEM is often used in conjunction with a PKE scheme to securely exchange encryption keys, which can then be used to encrypt data using the PKE scheme [6]. Therefore, in this paper, we select PKE as the cryptographic scheme for using PQC algorithms in various applications.

In the improved PQC standalone PKE protocol shown in Algorithm 1, both the client (C) and the server (S) generate two pairs of keys using post-quantum cryptographic algorithms. These include a public/private (secret) key pair for the KEM (pkk_c and skk_c for the client, and pkk_s and skk_s for the server), and a public/private (secret) key pair for the signature (pks_c and sks_c for the client, and pks_s and sks_s for the server). The client begins by signing its public KEM key (pkk_c) with its private signature key (sks_c) to create a digital certificate ($cert[pkk_c]$). This certificate, along with the client's public signature key (pks_c), is sent to the server to establish authenticity (Algorithm 1, lines 3–7).

Upon receiving the certificate and public signature key from the client, the server verifies the authenticity of the certificate using the client's public signature key (pks_c). If the verification is successful, it confirms that the client's public KEM key (pkk_c) has not been tampered with. The server then performs encapsulation on the client's public KEM key (pkk_c) to generate a shared secret (ss_c) and a ciphertext (ct_c). The server also signs its public KEM key (pkk_s) with its private signature key (sks_s) to create its digital certificate ($cert[pkk_s]$). The server sends this certificate, along with its public signature key (pks_s) and the ciphertext (ct_c), back to the client (Algorithm 1, lines 8–18).

The client receives the server's certificate, public signature key, and ciphertext. It verifies the authenticity of the server's certificate using the server's public signature key (pks_s). If this verification succeeds, it confirms that the server's public KEM key (pkk_s) is legitimate. The client then decapsulates the received ciphertext (ct_c) using its private KEM key (skk_c) to derive a shared secret (ss'_c). Next, the client performs encapsulation on the server's public KEM key (pkk_s) to generate another shared secret (ss_s) and a ciphertext (ct_s), which it sends back to the server (Algorithm 1, lines 19–30).

Algorithm 1 PQC standalone PKE protocol

Input: Data to be transmitted ($data_c$ from Client, $data_s$ from Server)
Output: Securely exchanged data
Client (C):
Generate PQC KEM key pair: (pkk_c, skk_c)
Generate PQC Signature key pair: (pks_c, sks_c)
$cert[pkk_c] \leftarrow \text{Sign}(sks_c, pkk_c)$
Send $(cert[pkk_c], pks_c, pkk_c)$ to Server (S)
Server (S):
Generate PQC KEM key pair: (pkk_s, skk_s)
Generate PQC Signature key pair: (pks_s, sks_s)
$\text{valid}_c \leftarrow \text{Verify}(pks_c, cert[pkk_c], pkk_c)$
if $\text{valid}_c = \text{TRUE}$ **then**
    $(ct_c, ss_c) \leftarrow \text{Encapsulate}(pkk_c)$
    $cert[pkk_s] \leftarrow \text{Sign}(sks_s, pkk_s)$
    Send $(cert[pkk_s], pks_s, pkk_s, ct_c)$ to Client (C)
else
    Abort
end if
Client (C):
$\text{valid}_s \leftarrow \text{Verify}(pks_s, cert[pkk_s], pkk_s)$
if $\text{valid}_s = \text{TRUE}$ **then**
    $ss'_c \leftarrow \text{Decapsulate}(skk_c, ct_c)$
    $(ct_s, ss_s) \leftarrow \text{Encapsulate}(pkk_s)$
    Send ct_s to Server (S)
    $seq_c \leftarrow \text{GenerateSequenceNumber}()$
    $mac_c \leftarrow \text{HMAC-SHA256}(ss'_c || ss_s, \text{“confirmation”} || seq_c)$
    Send mac_c to Server (S)
else
    Abort
end if
Server (S):
$ss'_s \leftarrow \text{Decapsulate}(skk_s, ct_s)$
$seq_s \leftarrow \text{GenerateSequenceNumber}()$
$mac_s \leftarrow \text{HMAC-SHA256}(ss_c || ss'_s, \text{“confirmation”} || seq_s)$
if $mac_c = \text{HMAC-SHA256}(ss_c || ss'_s, \text{“confirmation”} || seq_c)$ **then**
    Send mac_s to Client(C)
else
    Abort
end if
Client(C):
if $mac_s = \text{HMAC-SHA256}(ss'_c || ss_s, \text{“confirmation”} || seq_s)$ **then**
    Secure communication is established.
else
    Abort
end if
Client-to-Server Data Transfer:
$ct_{data_c} \leftarrow \text{PQC-Encrypt}(pkk_s, data_c || seq_c)$
Send ct_{data_c} to Server (S)
Server (S):
$data'_c || seq'_c \leftarrow \text{PQC-Decrypt}(skk_s, ct_{data_c})$
if seq'_c is valid (not replayed) **then**
    Process $data'_c$
else
    Reject // Replay detected
end if

Algorithm 1 *Cont.*

Server-to-Client Data Transfer:
$ct_{data_s} \leftarrow$ PQC-Encrypt$(pkk_c, data_s||seq_s)$
Send ct_{data_s} to Client (C)
Client (C):
$data'_s||seq'_s \leftarrow$ PQC-Decrypt(skk_c, ct_{data_s})
if seq'_s is valid **then**
    Process $data'_s$
else
    Reject
end if

To ensure that both parties have derived identical shared secrets and to prevent MITM attacks, a robust key confirmation process is employed. After deriving their respective shared secrets, both parties compute message authentication codes (MACs) using "hash-based message authentication code using the secure hash algorithm 256-bit (HMAC-SHA256)" over a combination of their shared secrets, a predefined "confirmation" string, and unique sequence numbers generated independently by each party. The MACs are exchanged between the client and server. After receiving a MAC, each party verifies that it matches its locally computed MAC. If both MACs match, it confirms that no MITM attack has occurred and that both parties share identical secrets (i.e., as shown for the client and the server in Algorithm 1, lines 31–45).

Once mutual confirmation is achieved, secure communication is established. The proposed protocols utilize core PQC primitives (KEMs and digital signatures) for key establishment and authentication, which is similar to the PQC-TLS/PQC-SSH handshake without the additional message negotiation steps (i.e., ClientHELLO/ServerHELLO) required in PQC-TLS. For data transfer from the client to the server, the client encrypts the data ($data_c$) along with its sequence number (seq_c) using the server's public KEM key (pkk_s). The resulting ciphertext (ct_{data_c}) is sent to the server (Algorithm 1, lines 46–48). The server receives and decrypts the ciphertext using its private KEM key (skk_s), retrieves both the data and sequence number, and checks whether the sequence number is valid (i.e., not replayed or out of order). If valid, it processes the data; otherwise, it rejects the data to prevent replay attacks (Algorithm 1, lines 49–55).

A similar process is followed for data transfer from the server back to the client. The server encrypts its data ($data_s$) along with its sequence number (seq_s) using the client's public KEM key (pkk_c) to produce a ciphertext ct_{data_s}, which is sent to the client. Upon receiving this ciphertext, the client decrypts it using its private KEM key (skk_c), retrieves both the data and the sequence number (seq'_s), and checks whether the sequence number is valid. If the sequence number is valid, the client processes the data; otherwise, it rejects the data to prevent replay attacks (Algorithm 1, lines 56–65).

The proposed protocol introduces various enhancements compared to traditional PQC-based PKE protocols. First, we incorporate explicit certificate validation during the initial handshake steps to ensure that the exchanged KEM public keys are authentic and untampered. Second, we strengthen the replay protection by including unique sequence numbers in data encryption. Finally, by integrating robust HMAC-based key confirmation mechanisms, our proposed protocol significantly reduces vulnerabilities to MITM attacks while ensuring that shared secrets are securely established between legitimate parties.

3.2. PQC–AES Hybrid PKE Protocol

To enhance security and performance, especially against evolving threats, we introduce a PQC–AES hybrid PKE protocol. This approach combines the quantum resistance of PQC with the efficiency and established security of AES.

In this PQC–AES hybrid PKE protocol, as shown in Algorithm 2, we build upon the key exchange process already outlined for the PQC standalone PKE protocol. Following a successful and authenticated key exchange between the client and the server using the steps defined in Algorithm 1 (lines 1–24), both parties transition to utilizing AES encryption for subsequent data transfer. This approach leverages the best attributes of both cryptographic approaches: the quantum-resistant security offered by PQC algorithms during the initial key establishment and the speed and efficiency of AES for the bulk encryption of application data.

After the successful establishment and confirmation of shared secrets ss_c and ss_s, both the client and the server independently derive AES encryption and MAC keys. A KDF is applied to the concatenation of these shared secrets, along with a unique salt and information string, using the HMAC-based extract-and-expand hash-based message authentication code key derivation function (HKDF) with SHA-256. This process yields two essential keys: K_1, designated for AES encryption, and K_2, designated for HMAC-based message authentication (Algorithm 2, lines 5–6).

For data transfer from the client to the server, the client begins by generating a unique initialization vector, iv_c, and a sequence number, seq_c, to provide replay protection. Subsequently, the data, $data_c$, is encrypted using AES with the derived key, K_1, and the initialization vector, iv_c, and incorporating the sequence number, seq_c, to form the ciphertext, ct_{data_c}. Following this, the client computes a MAC over the ciphertext, ct_{data_c}, and the sequence number, seq_c, using HMAC-SHA-256 with the authentication key, K_2, resulting in mac_{data_c}. The client then sends the initialization vector, iv_c; the ciphertext, ct_{data_c}; and the MAC, mac_{data_c} to the server (Algorithm 2, lines 7–12).

Upon receiving this data, the server begins by verifying the integrity and authenticity of the data. It computes its own MAC over the received ciphertext, ct_{data_c}, and the sequence number, seq_c, using the same HMAC-SHA-256 algorithm and the authentication key K_2, generating a server-side MAC. The server then compares the received client-side MAC, mac_{data_c}, with its locally computed MAC. If these MACs match, the server proceeds with decryption, ensuring that the received data has not been tampered with during transmission and that it originates from the authenticated client. The server decrypts the ciphertext, ct_{data_c}, using AES with the key, K_1, and the initialization vector, iv_c, and after decryption, it also verifies the sequence number against previously received sequence numbers to detect any replay attacks. If the sequence number is valid and the decryption is successful, the server processes the received data, $data'_c$; otherwise, the server rejects the data to prevent potential security breaches (Algorithm 2, lines 13–23).

Similarly, the procedure is used for secure data transfer from the server to the client. The server also generates a unique iv_S and uses K_1 to encrypt the data while applying HMAC to ensure authentication of the server by the client. The client also performs decryption in a similar way to ensure that the data is from the intended server (Algorithm 2, lines 24–40).

By combining the strengths of both PQC and AES, the protocol provides resilience against both classical and quantum attacks. Even if the PQC algorithm used for key exchange were compromised, AES encryption would still provide a layer of security. Furthermore, the hybrid approach can improve overall performance since AES encryption and decryption operations are generally faster and more efficient than PQC standalone

encryption. Moreover, since AES-256 is a widely adopted and well-analyzed symmetric encryption algorithm, it has been demonstrated to be secure.

In the implementation of the proposed hybrid protocol and conventional AES-256 used in the evaluation, we adopt AES-GCM as the symmetric encryption scheme due to its strong security properties as an authenticated encryption algorithm [28] and its efficient performance. In comparison, AES-CBC requires separate MAC handling, making it more prone to implementation issues. Therefore, AES-GCM is prioritized over other types of symmetric encryption schemes.

Algorithm 2 PQC–AES hybrid PKE protocol

Input: Data to be transmitted ($data_c$ from Client, $data_s$ from Server)
Output: Securely exchanged data
Client (C) and Server (S):
Perform steps 1–24 of Algorithm 1 to establish shared secrets ss_c and ss_s (and confirm them).
Key Derivation (Both C and S):
$(K_1, K_2) \leftarrow \text{HKDF}(ss_c||ss_s, \text{salt}, \text{“AES keys”})$
Client-to-Server Data Transfer:
$iv_c \leftarrow \text{GenerateInitializationVector}()$
$seq_c \leftarrow \text{GenerateSequenceNumber}()$
$ct_{data_c} \leftarrow \text{AES-Encrypt}(K_1, iv_c, data_c, \text{mode}, seq_c)$
$mac_{data_c} \leftarrow \text{HMAC-SHA256}(K_2, ct_{data_c}||seq_c)$
Send $(iv_c, ct_{data_c}, mac_{data_c})$ to Server (S)
Server (S):
if $mac_{data_c} = \text{HMAC-SHA256}(K_2, ct_{data_c}||seq_c)$ **then**
  $data'_c \leftarrow \text{AES-Decrypt}(K_1, iv_c, ct_{data_c}, \text{mode}, seq_c)$
  if seq_c is valid (not a replay) AND Decryption successful **then**
    Process $data'_c$
  else
    Reject
  end if
else
  Abort
end if
Server-to-Client Data Transfer:
$iv_s \leftarrow \text{GenerateInitializationVector}()$
$seq_s \leftarrow \text{GenerateSequenceNumber}()$
$ct_{data_s} \leftarrow \text{AES-Encrypt}(K_1, iv_s, data_s, \text{mode}, seq_s)$
$mac_{data_s} \leftarrow \text{HMAC-SHA256}(K_2, ct_{data_s}||seq_s)$
Send $(iv_s, ct_{data_s}, mac_{data_s})$ to Client (C)
Client (C):
if $mac_{data_s} = \text{HMAC-SHA256}(K_2, ct_{data_s}||seq_s)$ **then**
  $data'_s \leftarrow \text{AES-Decrypt}(K_1, iv_s, ct_{data_s}, \text{mode}, seq_s)$
  if seq_s is valid AND Decryption Successful **then**
    Process $data'_s$
  else
    Reject
  end if
else
  Abort
end if

4. Practical Implementation of PQC-Based PKE Protocols

Since there is no function for data encryption and decryption in current PQC algorithm implementations, we enhance the PQC algorithm implementation by adding encryption and decryption functions. This enables data encryption and decryption in both PQC stan-

dalone and PQC–AES hybrid protocols. Both protocols utilize the PQC key encapsulation mechanism (KEM) for secure key exchange between the client and server.

4.1. Post-Quantum Cryptography (PQC) Uses in Applications

To demonstrate the impact of the characteristics of PQC algorithms, PQC-based protocols need to be utilized in real-world applications. However, current implementations of PQC algorithms using PKE are limited to encrypting messages of a fixed size. For instance, CRYSTALS-Kyber, which is based on an IND-CPA-secure PKE scheme, and BIKE, which utilizes PKE from Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) codes, can only encrypt messages of a fixed length of 32 bytes. HQC, also based on IND-CPA-secure properties, can accept messages of a fixed length of 16 bytes before encoding.

Conversely, when a hybrid approach is employed, the KDF generates symmetric keys of a consistent length for all PQC algorithms. While AES cipher modes typically use key sizes of 128, 192, or 256 bits (16, 24, or 32 bytes), it does not support key sizes larger than 256 bits. Therefore, to use the PQC standalone PKE directly for the encryption and decryption of data that exceeds these fixed sizes without the hybrid implementation, it is necessary to segment the message and reassemble the decrypted segments to reconstruct the original data. The process for this splitting is shown in Figure 1. In this instance, first, the input data is split into blocks of an acceptable size, as shown in Figure 1a. Each block is then encrypted sequentially using the receiver's public key. The encrypted blocks are sent to the intended party. As shown in Figure 1b, on the receiver side, after the data is received, the encrypted data is processed as a stream. Each block is decrypted using the receiver's private key and then aggregated to reconstruct the original data.

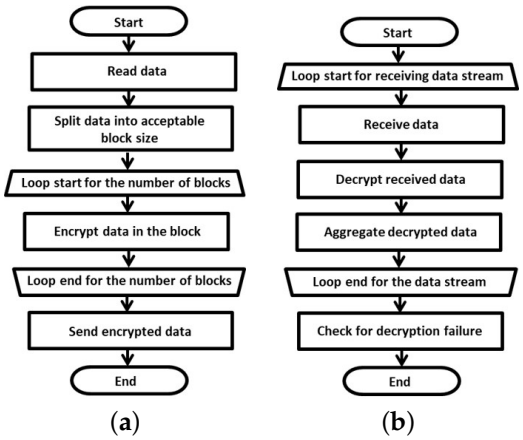

Figure 1. Encryption and decryption processes of PQC protocols. (**a**) Encryption process. (**b**) Decryption process.

In our proposed PQC standalone protocol, after establishing a secure communication (i.e., session key) using the standardized PQC KEMs and digital signatures (e.g., CRYSTALS-Kyber, BIKE, and HQC), each segmented message is encrypted within that session using the established encryption and decryption keys. This approach aligns with standard cryptographic practice, where a session key securely negotiated using an IND-CCA2 secure KEM is reused for multiple encryptions within the same session. The security of each encrypted segment is ensured by the PQC algorithm and the sequence number mechanism introduced to counter replay attacks. Moreover, the proposed PQC standalone protocol leverages periodic re-keying when a session is too long. For this, a new set of PQC KEM and signature algorithm keys is generated. The re-keying process is currently not implemented in our evaluation and will be considered as part of future work.

4.1.1. Real-World Scenarios for Proposed PQC-Based Protocols

Consider a real-world scenario, shown in Figure 2, involving two devices representing the client and server, each equipped with the necessary software, including PQC and AES libraries. For this application scenario, we consider four use cases: (i) file transfer, (ii) chat, (iii) video streaming, and (iv) live streaming. Assuming that the client is Alice while the server is Bob, for Alice to send confidential data to Bob using any of these use cases within a PQC-based system to ensure the data remains secure against potential future quantum computer attacks, the following process is used:

1. Bob generates a pair of keys using a PQC algorithm: a public key (pkk_b) and a private key (skk_b).
2. Bob shares his public key (pkk_b) with Alice while keeping the private key (skk_b) secret.
3. Alice prepares the data for transmission and splits it into multiple blocks if it exceeds the maximum byte size allowed by the PQC algorithm, as shown in Figure 1a.
4. Alice encrypts each block sequentially using Bob's public key (pkk_b). The encrypted blocks are combined into a single encrypted file.
5. Alice sends the encrypted file through a secure network.
6. The access point routes the encrypted file to Bob without decrypting it.
7. Bob receives the encrypted file and uses his private key (skk_b) to decrypt each block.
8. He reassembles the decrypted blocks to reconstruct the original document, as shown in Figure 1b.

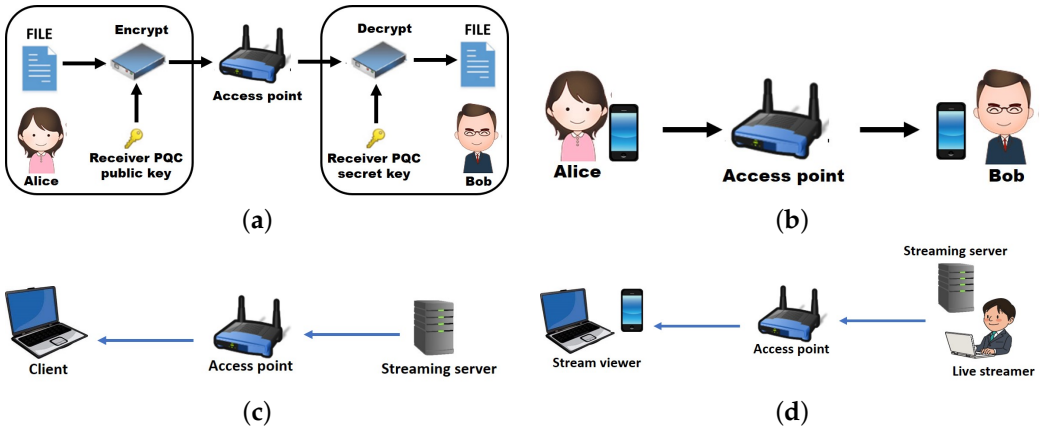

Figure 2. Use cases for PQC algorithm: (**a**) File transfer, (**b**) chat, (**c**) video streaming, (**d**) live streaming.

Using PQC-based protocols, Alice and Bob can ensure that their communication remains secure against threats posed by quantum computing. The workflow of the proposed PQC-based protocols described above can be applied to various real-world scenarios. For file transfers, the protocol ensures the secure transmission and reconstruction of files divided into encrypted blocks. In chat applications, messages can be encrypted as individual small blocks to ensure confidentiality in real-time conversations. For video streaming, secure pre-encryption of video segments guarantees that the data remains protected during transmission. Lastly, live streaming benefits from the protocol by encrypting and transmitting content in near-real time, with a focus on maintaining low latency to preserve streaming quality. These use cases demonstrate the versatility of the PQC-based protocols in addressing diverse application requirements while securing data against quantum threats.

5. Security Analysis of the Proposed Protocols

This section presents a detailed security analysis of the proposed PQC standalone and PQC–AES hybrid PKE protocols. We evaluate the resilience of these protocols against various attack vectors, considering both classical and post-quantum threats.

5.1. Security Analysis of the PQC Standalone PKE Protocol

The PQC standalone PKE protocol relies entirely on the security of the chosen PQC algorithms for both key exchange (KEM) and data encryption. We analyze its strengths and weaknesses, detailing the assumptions underlying its security.

5.1.1. Key Exchange (PQC KEM)

The key exchange phase uses a post-quantum secure KEM, such as CRYSTALS-Kyber, standardized by NIST.

Security Assumption

The security of the key exchange depends on the computational infeasibility of solving the underlying hard mathematical problem (e.g., Module-LWE problem) for quantum adversaries. This is based on cryptanalysis conducted during the NIST PQC standardization process.

Formal Statement

The confidentiality of the shared secret rests on the IND-CCA2 security of the chosen KEM in the quantum random-oracle model. If an adversary solves the Module-LWE problem or breaks the IND-CCA2 security of the KEM, the confidentiality of the shared secret is compromised.

5.1.2. Key Confirmation

Key confirmation prevents MITM attacks, ensuring both parties derive the same shared secret.

Mechanism

After decapsulation, parties derive a shared secret using a KDF. To confirm the shared secret, each party computes and exchanges HMACs (using HMAC-SHA-256) over a predefined data structure including nonces or session identifiers. The MACs are verified to ensure that no MITM attacker has replaced either party's public key.

Security Assumption

The security relies on the following:

1. The collision resistance of SHA-256.
2. The pseudorandomness and unforgeability properties of HMAC.

Formal Statement

If SHA-256 is collision-resistant and HMAC is unforgeable under chosen-message attacks, an adversary cannot impersonate either party or inject their own public key into the key exchange process.

5.1.3. Data Encryption and Decryption (PQC)

The PQC standalone PKE protocol directly encrypts data using a public-key encryption scheme derived from the chosen KEM.

Security Assumption

The confidentiality of encrypted data depends on the post-quantum security of the public-key encryption scheme derived from the KEM, including resistance to chosen-ciphertext attacks (IND-CCA2 security).

Formal Statement

The confidentiality of transmitted data is guaranteed if the following conditions hold:

1. The underlying KEM is IND-CCA2 secure.
2. The public-key encryption scheme derived from it inherits this IND-CCA2 security property.

If an adversary breaks these assumptions, they can recover plaintext data from ciphertexts.

5.1.4. Replay Protection

Replay attacks involve an attacker capturing and retransmitting a valid encrypted message to disrupt communication or impersonate a legitimate sender.

Mechanism

To mitigate replay attacks, the protocol implements the following mechanisms:

1. Sequence numbers are incorporated into each encrypted message.
2. The receiver tracks these sequence numbers to ensure they are unique and monotonically increasing.
3. Messages with duplicate or out-of-order sequence numbers are rejected.

Security Assumption

This approach assumes that sequence numbers are generated uniquely and managed correctly by both parties during communication.

Formal Statement

If sequence numbers are generated uniquely and monotonically increasing, replayed messages will be detected and rejected, preventing replay attacks.

5.1.5. Overall Security

The overall security of this protocol is determined by its reliance on post-quantum cryptographic primitives for both key exchange and data encryption.

Advantages

1. Post-Quantum Security: Provides resistance against quantum adversaries due to reliance on PQC algorithms.
2. Simplified Design: Avoids additional complexity introduced by hybrid schemes.

Limitations

1. Single Point of Failure: The protocol's security entirely depends on the strength of the PQC algorithms used (both KEM and public-key encryption). If these algorithms are compromised, both key exchange and data confidentiality are at risk.
2. Performance Overhead: PQC standalone encryption/decryption operations are computationally intensive compared to symmetric cryptography like AES-256.

Formal Statement

The overall security of this protocol is bounded by the security assumptions of its underlying PQC algorithms. If an adversary breaks the post-quantum hardness assumptions

(e.g., Module-LWE problem), they can compromise both key exchange and data confidentiality. Unlike hybrid approaches, breaking one component compromises overall security.

5.2. Security Analysis of the PQC–AES Hybrid PKE Protocol

The hybrid protocol leverages the strengths of both post-quantum key exchange (using a KEM like CRYSTALS-Kyber) and a well-established symmetric cipher (AES-256). This defense-in-depth approach enhances overall security.

5.2.1. Security of the Key Exchange (PQC KEM)

The key exchange phase is identical to that described for the PQC standalone PKE protocol (Section 5.1.1). The same security assumptions and formal statements apply.

5.2.2. Key Confirmation

Key confirmation is performed as described for the PQC standalone PKE protocol (Section 5.1.2) to prevent MITM attacks.

5.2.3. Security of Data Encryption (AES-256)

Data encryption uses AES-256, which is well-established and understood.

Security Assumption

AES-256 is considered secure against classical attacks. The hybrid approach aims for defense-in-depth; even if the PQC KEM is broken, the attacker still faces the challenge of breaking AES-256.

Formal Statement

The data encryption phase utilizes AES-256 with a key size of 256 bits. AES-256 is widely considered secure against classical computational attacks, providing 256 bits of security. We assume that the AES-256 implementation is free from side-channel vulnerabilities.

5.2.4. Overall Security Argument

The hybrid protocol combines the strengths of both PQC and AES-256.

Formal Statement

The overall security of the hybrid protocol is bounded by the security of its weakest component. By combining a post-quantum KEM with AES-256, the protocol achieves a defense-in-depth strategy. An attacker would need to break both the PQC KEM and AES-256 to compromise the confidentiality of the data. This provides a higher level of security against both classical and potential future quantum attacks. Since PQC encryption/decryption algorithms are often significantly slower than AES-256, the hybrid approach leverages the best aspects of both: post-quantum key exchange and fast symmetric encryption.

5.3. Robustness Against Common Attack Vectors

To demonstrate the robustness of our proposed protocols, we consider their resilience against several standard cybersecurity attack vectors.

5.3.1. Eavesdropping

Eavesdropping is the primary threat addressed by encryption. Both protocols provide protection against eavesdropping, although with different security assumptions.

PQC Standalone PKE

The PQC standalone PKE protocol protects against eavesdropping by encrypting data using the receiver's public key, which only the receiver, possessing the corresponding private key, can decrypt. The security relies on the post-quantum security of the underlying PQC algorithm.

PQC–AES Hybrid PKE

The hybrid protocol provides two layers of protection against eavesdropping. First, the AES-256 key is exchanged securely using the PQC KEM. Second, the data itself is encrypted with AES-256. An eavesdropper would need to break both the PQC KEM and AES-256 to recover the plaintext.

5.3.2. Man-in-the-Middle (MITM) Attacks

Both protocols include a key confirmation step using HMAC-SHA-256 to prevent MITM attacks, ensuring that both parties have derived the same shared secret and that an attacker has not successfully impersonated either party during the key exchange.

5.3.3. Replay Attacks

Replay attacks are mitigated differently depending on the AES mode used.

PQC–AES Hybrid PKE (with AES-GCM)

When using AES-GCM, the GCM mode inherently provides protection against replay attacks. The authentication tag incorporates the initialization vector (IV) and ciphertext, so replaying a message with a different IV (which should be unique for each message) will result in authentication failure.

PQC–AES Hybrid PKE (with AES-CBC or Other Modes)

To mitigate replay attacks when not using GCM, we incorporate sequence numbers or timestamps within the HMAC calculation. The receiver tracks these sequence numbers or /timestamps and rejects any messages that are out of order or outside an acceptable time window.

5.3.4. Chosen-Ciphertext Attacks (CCAs)

This is relevant to the PQC KEM used in both protocols. NIST's selections are generally CCA-secure. The chosen PQC KEM (e.g., CRYSTALS-Kyber) is designed to be IND-CCA2 secure, meaning it is resistant to chosen-ciphertext attacks. This ensures that an attacker cannot gain information about the plaintext even if they can choose ciphertexts to be decrypted.

5.3.5. Side-Channel Attacks

Side-channel attacks exploit physical information leaked during computation (power consumption, timing, etc.). While our current evaluation does not directly address side-channel attacks, we acknowledge their importance, particularly for implementations on embedded devices. Mitigation strategies against side-channel attacks such as constant-time implementations and masking techniques are crucial for real-world deployments and are a subject of ongoing research in the PQC community. We recommend using side-channel-resistant implementations of the chosen PQC algorithms.

5.3.6. Known-Plaintext Attacks

If an attacker somehow knows some of the plaintext and tries to use this to break the encryption, the AES-256 and PQC KEMs are designed to be resistant to known-plaintext attacks.

6. Evaluation

In this section, we conduct a performance evaluation of the proposed protocols through experiments with PQC algorithms. First, we enhance the PQC algorithm implementation by adding encryption and decryption functions, enabling data encryption and decryption using both PQC standalone algorithms and the PQC–AES hybrid protocols. Second, we implement our proposed protocols (i.e., PQC standalone and PQC–AES hybrid PKE protocols). Furthermore, we measure the time required to generate PQC KEM keys and PQC signature keys, as well as the time required to perform encapsulation, decapsulation, and file encryption and decryption, among other operations. We use our extended version (which can now handle data encryption and decryption) of the PQC algorithm API implementation provided by Open Quantum Safe (Liboqs) [29], which is a collection of all the PQC algorithms selected by NIST as one standard library.

6.1. Experimental Settings

In our implementation, we used the four use cases described in Section 4.1.1 (i.e., file transfer, chat, video streaming, and live streaming) and evaluated our protocols based on these cases. The experiment was conducted on two separate machines: one with an Intel® Core™ i7-10510U CPU, Mouse laptop (sourced: Osaka, Japan) running at a base frequency of 1.80 GHz × 8 and 15.5 GiB of memory, running Ubuntu 20.04 LTS desktop, which served as the server; and another with an Intel® Core™ i7-10510U CPU, Mouse laptop (sourced: Osaka, Japan) running at a base frequency of 1.80 GHz × 8 and 15.5 GiB of memory, running Ubuntu 20.04 LTS desktop, which was used as the client system. Table 1 presents the PC settings and the configurations of the systems used.

Table 1. Configuration of client–server system.

Parameter	Value	
	Server	**Client**
PC	Mouse Laptop	Mouse Laptop
OS	Ubuntu 20.04 desktop	Ubuntu 20.04 desktop
Network/Protocol	IEEE 802.11n	IEEE 802.11n
Frequency band	2.4 GHz	2.4 GHz
Maximum data rate	144 Mbps	144 Mbps
Sniffer	Wireshark	Wireshark

Furthermore, the standardized algorithms selected by NIST were considered. These include the CRYSTALS-Kyber KEM algorithms—ML-KEM-512, ML-KEM-768, and ML-KEM-1024 (referred to as K512, K768, and K1024 hereafter)—as well as alternative KEM algorithms: BIKE (Round 4)—BIKE-L1, BIKE-L3, and BIKE-L5—and HQC–HQC-128, HQC-192, and HQC-256. For the signature algorithms, FALCON (Round 3)variants—FN-DSA-512 and FN-DSA-1024 (referred to as F512 and F1024 hereafter)—were also evaluated. To evaluate the proposed protocols and the impact of PQC algorithms on applications, we used client–server file transfer communication between two different systems using Python 3.8.10 socket programming with Transmission Control Protocol/User Datagram Protocol (TCP/UDP). The client encrypts a file and sends it to the server. TCP traffic generated from the socket programming was sent from the PC2 machine, acting as the client, and received by the PC1 machine, acting as the server. The traffic data was transmitted using IEEE 802.11n wireless LAN in the 2.4 GHz band. The traffic between the client and server was captured using Wireshark [30] from the server system. The experimental setup is shown in Figure 3.

Additionally, we implemented conventional cryptography (i.e., AES-256) to compare and highlight the impact of the PQC algorithm on file transfer. The goal of our evaluation was to facilitate the practical use of the PQC algorithm in real applications and to determine the impact of using the PQC algorithm on the application use case and network. We considered two main evaluation scenarios: (i) the performance of the PQC algorithm to determine how fast it takes to generate PQC algorithm keys, complete the KEM process (encapsulation and decapsulation) (i.e., non-pre-distributed key agreement process), apply digital signatures (signing and verification), and perform encryption and decryption; and (ii) the impact of the PQC algorithm on file transfer applications to determine which PQC algorithm is more suitable for real-time applications without extensive overhead. For this, we measured the processing time for encrypted data transfer, throughput, and round-trip time (RTT).

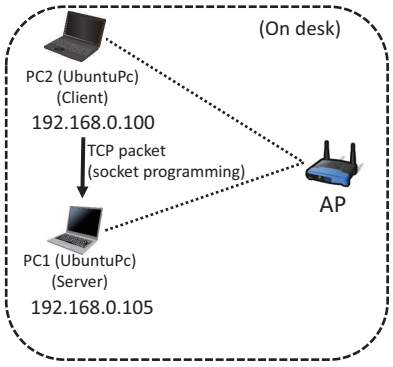

Figure 3. Experimental settings.

6.2. Results and Discussion

6.2.1. Performance Results of PQC Algorithm Key Generation

In order to evaluate the performance implications of integrating PQC algorithms, we measured the average key generation time (Avg. Key Gen.), key encapsulation time (Avg. Encap.), and key decapsulation time (Avg. Decap.) for the PQC KEM algorithms, as well as the signature key generation time (Sig. Gen. Time), signing time, and verification time for the PQC signature algorithms across the client and server systems. Tables 2 and 3 show the average time it took to generate the PQC algorithm keys (PQC KEM and signature keys), encapsulate and decapsulate the KEM keys, and sign and verify PQC signatures on the client and server systems.

For the KEM algorithms, Table 2 shows that for BIKE KEM at all security levels, BIKE-L1 achieved the fastest average key generation time on the client system of 0.43 ms, while BIKE-L5 required 1.95 ms, reflecting a progressive increase in computational demand with higher security levels. Conversely, the server-side key generation times for BIKE-L1, BIKE-L3, and BIKE-L5 were 1.23 ms, 2.63 ms, and 5.05 ms, respectively, demonstrating a more pronounced rise compared to the client-side metrics. The encapsulation and decapsulation times also increased with security levels, particularly for the BIKE and HQC algorithms.

Furthermore, as shown in the table, BIKE-L5 decapsulation required 14.42 ms on the client system and 15.94 ms on the server system, while HQC-256 decapsulation required 49.77 ms (client) and 53.40 ms (server). However, the CRYSTALS-Kyber algorithms diverged from this trend: K768 achieved the fastest client-side key generation time of 0.19 ms, surpassing both K512 (0.92 ms) and K1024 (1.00 ms). On the server side, the CRYSTALS-Kyber results showed that K1024 was the most efficient for key generation (0.49 ms), encapsulation (0.09 ms), and decapsulation (0.11 ms), with K512 and K768 achieving marginally higher times.

Table 2. Average times to generate PQC algorithm KEM keys.

PQC KEM Algorithm	Client Avg. Key Gen. (ms)	Client Avg. Encap. (ms)	Client Avg. Decap. (ms)	Server Avg. Key Gen. (ms)	Server Avg. Encap. (ms)	Server Avg. Decap. (ms)
BIKE-L1	0.43	0.12	1.64	1.23	0.12	1.52
BIKE-L3	0.80	0.21	5.68	2.63	0.19	5.79
BIKE-L5	1.95	0.36	14.42	5.05	0.33	15.94
HQC-128	3.45	5.76	10.23	5.60	4.03	8.44
HQC-192	4.88	12.83	28.11	9.70	11.99	32.20
HQC-256	8.73	17.96	49.77	14.32	1.81	53.40
K512	0.92	0.10	0.17	1.83	0.10	0.11
K768	0.19	0.09	0.16	0.37	0.10	0.10
K1024	1.00	0.14	0.18	0.49	0.09	0.11

Table 3. Average times to generate PQC algorithm signature keys.

PQC Signature	Client Avg. Sig. Key Gen. (ms)	Client Avg. Signing (ms)	Client Avg. Ver. (ms)	Server Avg. Sig. Key Gen. (ms)	Server Avg. Signing (ms)	Server Avg. Ver. (ms)
F512	7.01	0.33	0.28	9.68	0.37	0.82
F1024	19.00	1.43	0.31	24.51	0.67	0.79

Furthermore, Table 3 shows the performance of the PQC signature algorithms. According to the results, F512 required 7.01 ms for key generation on the client system, significantly faster than F1024's 19.00 ms. A similar trend was observed on the server system, with F512 and F1024 requiring 9.68 ms and 24.51 ms, respectively. The signing times for F512 were 0.33 ms (client) and 0.37 ms (server), whereas F1024 took 1.43 ms (client) and 0.67 ms (server), illustrating a reversal in efficiency, favoring the server for higher security levels. The verification times remained consistent for F512 across the systems (0.28 ms client vs. 0.82 ms server) but diverged slightly for F1024 (0.31 ms client vs. 0.79 ms server). These results underscore the trade-offs between security levels and computational overhead, particularly for server-side operations.

6.2.2. Performance Results of PQC Algorithm Encryption and Decryption

In the hybrid protocol, AES-256 is used for data encryption/decryption. It was expected that the times for these operations should remain consistently low, meaning they should not vary significantly with the chosen PQC KEM, as the underlying symmetric cipher remains the same. To focus on the computational differences introduced by the PQC algorithms themselves, the subsequent figures present the encryption and decryption times specifically for the PQC-only protocol. Therefore, we evaluated the data encryption and decryption times for the PQC KEM algorithms. The encryption process was executed on the client system, while decryption was performed on the server system.

Figure 4 shows the average encryption times for the BIKE, HQC, and CRYSTALS-Kyber algorithms when processing files of sizes 1 KB, 10 KB, and 100 KB. For BIKE-L1, as shown in Figure 4a, the encryption times remained stable across file sizes, with 0.20 ms for 1 KB, 0.199 ms for 10 KB, and 0.148 ms for 100 KB. However, BIKE-L3 exhibited higher encryption times for larger files, increasing from 0.47 ms (1 KB) to 0.63 ms (10 KB) before declining to 0.46 ms (100 KB), while BIKE-L5 ranged from 0.58 ms (1 KB) to 0.93 ms (10 KB) and 0.82 ms (100 KB).

Conversely, the HQC encryption times scaled significantly with both security levels and file sizes, as shown in Figure 4b. HQC-128 averaged 6.13 ms (1 KB), 5.96 ms (10 KB), and 5.63 ms (100 KB), whereas HQC-256 averaged 32.82 ms (1 KB), 37.04 ms (10 KB), and 37.54 ms (100 KB). The CRYSTALS-Kyber algorithms, as shown in Figure 4c, demonstrated modest variations, with K768 achieving the fastest 1 KB encryption time of 0.10 ms, outperforming K512 (0.15 ms) and K1024 (0.13 ms). For 100 KB files, the CRYSTALS-Kyber encryption times remained low compared to those of the other algorithms, ranging from 0.10 ms (K768) to 0.17 ms (K1024).

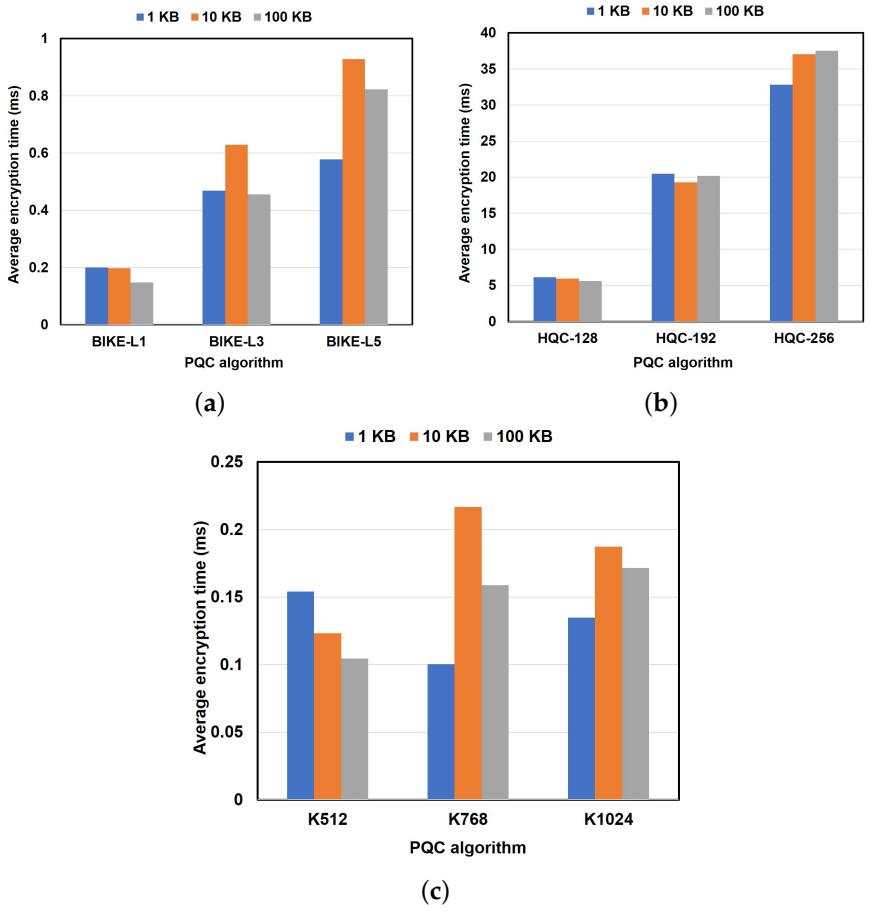

Figure 4. Average encryption times: (**a**) BIKE. (**b**) HQC. (**c**) CRYSTALS-Kyber.

Figure 5 shows the average decryption times on the server system. The BIKE decryption times increased sharply as the security level increased, ranging from an average decryption time of 2.17 ms (1 KB) to 23.85 ms (100 KB). BIKE-L1 achieved the lowest decryption time across all file sizes, with 2.17 ms (1 KB), 2.94 ms (10 KB), and 2.32 ms for 100 KB. BIKE-L5 showed a slight increase in decryption time as the file size increased, with average decryption times of 23.17 ms (1 KB), 27.70 ms (10 KB), and 27.85 ms (100 KB), as shown in Figure 5a. Figure 5b shows that HQC followed a similar pattern to BIKE, ranging from an average decryption time of 2.82 ms to 26.18 ms for 100 KB. HQC-128 required an average decryption time of 2.82 ms for 100 KB files, whereas HQC-192 required 9.38 ms, which increased sharply to 26.18 ms for HQC-256 when decrypting the same file size.

In contrast, Figure 5c shows that the CRYSTALS-Kyber algorithms maintained consistently low decryption times for the same file size (i.e., 100 KB), with K512 requiring an average decryption time of 0.12 ms and K1024 requiring 0.16 ms. Notably, K768's decryption time decreased slightly to 0.11 ms compared to K512 and K1024, deviating from the upward trend observed in BIKE and HQC. Smaller file sizes (1 KB) yielded minimal average

decryption times for all algorithms, with K512 and K768 requiring 0.09 ms. These results reaffirm the computational efficiency of lattice-based algorithms like CRYSTALS-Kyber for both encryption and decryption, particularly when handling larger file sizes, compared to code-based (BIKE) and hash-based (HQC) alternatives.

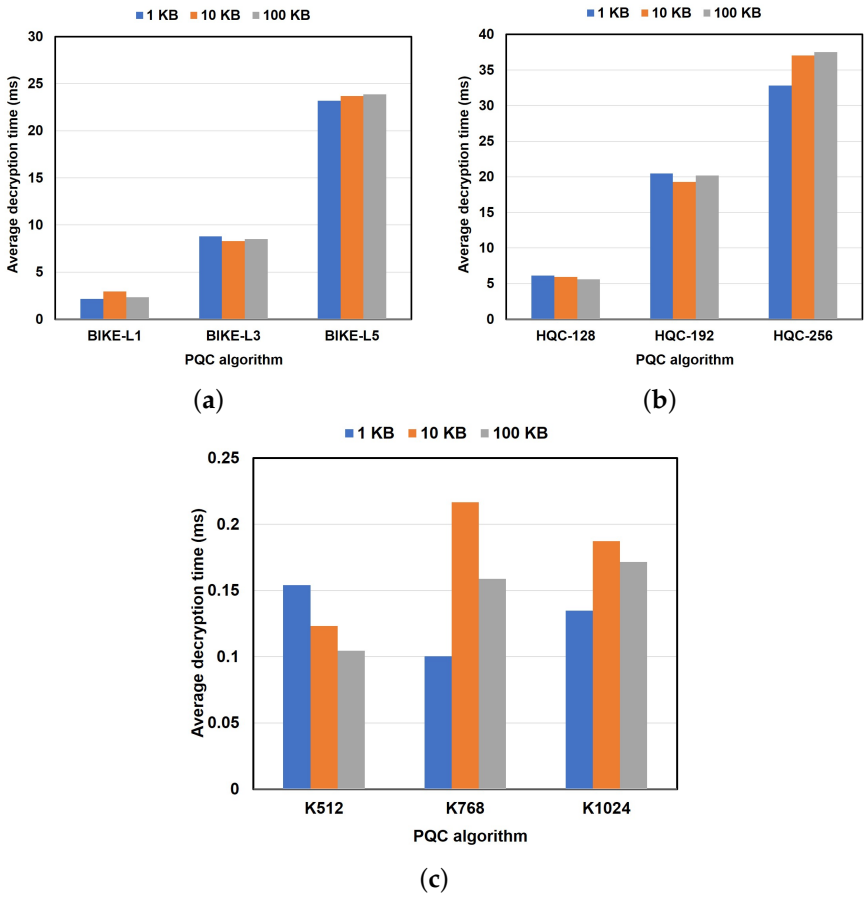

Figure 5. Average decryption times: (**a**) BIKE, (**b**) HQC, (**c**) CRYSTALS-Kyber.

6.2.3. Impact of PQC Algorithms on Performance in the File Transfer Use Case

To demonstrate the impact of PQC algorithms on various applications, we evaluated the proposed protocols using a file transfer application. We measured the overall processing time, throughput, and RTT as metrics. Since PQC algorithms can be more computationally intensive than traditional algorithms, the requirements to secure the data may adversely affect the performance of the applications and the network's data-handling capabilities. Therefore, file transfer was selected as the use case to test the application of the proposed protocols.

In our evaluation, we paired F512 and F1024 with each PQC KEM algorithm (the KEM algorithm paired with the F512 and F1024 signature algorithms is denoted as BIKE-L1-F512, BIKE-L3-F512, BIKE-L5-F512, BIKE-L1-F1024, BIKE-L3-F1024, BIKE-L5-F1024, HQC-128-F512, HQC-192-F512, HQC-256-F512, HQC-128-F1024, HQC-192-F1024, HQC-256-F1024, K512-F512, K768-F512, K1024-F512, K512-F1024, K768-F1024, and K1024-F1024) to perform the key exchange process for each algorithm in our proposed protocols. We measured the processing time, throughput, and RTT efficiency of the proposed protocols using various paired KEM and signature PQC algorithms during data transfer across a wireless LAN environment and compared the metrics achieved by different PQC algorithms for data sizes of 1 KB, 10 KB, and 100 KB. The results of the data transfer between the client system and the server, including a key agreement (i.e., the PQC public keys of both parties are exchanged securely) and the actual file transfer, are shown in Figures 6–8.

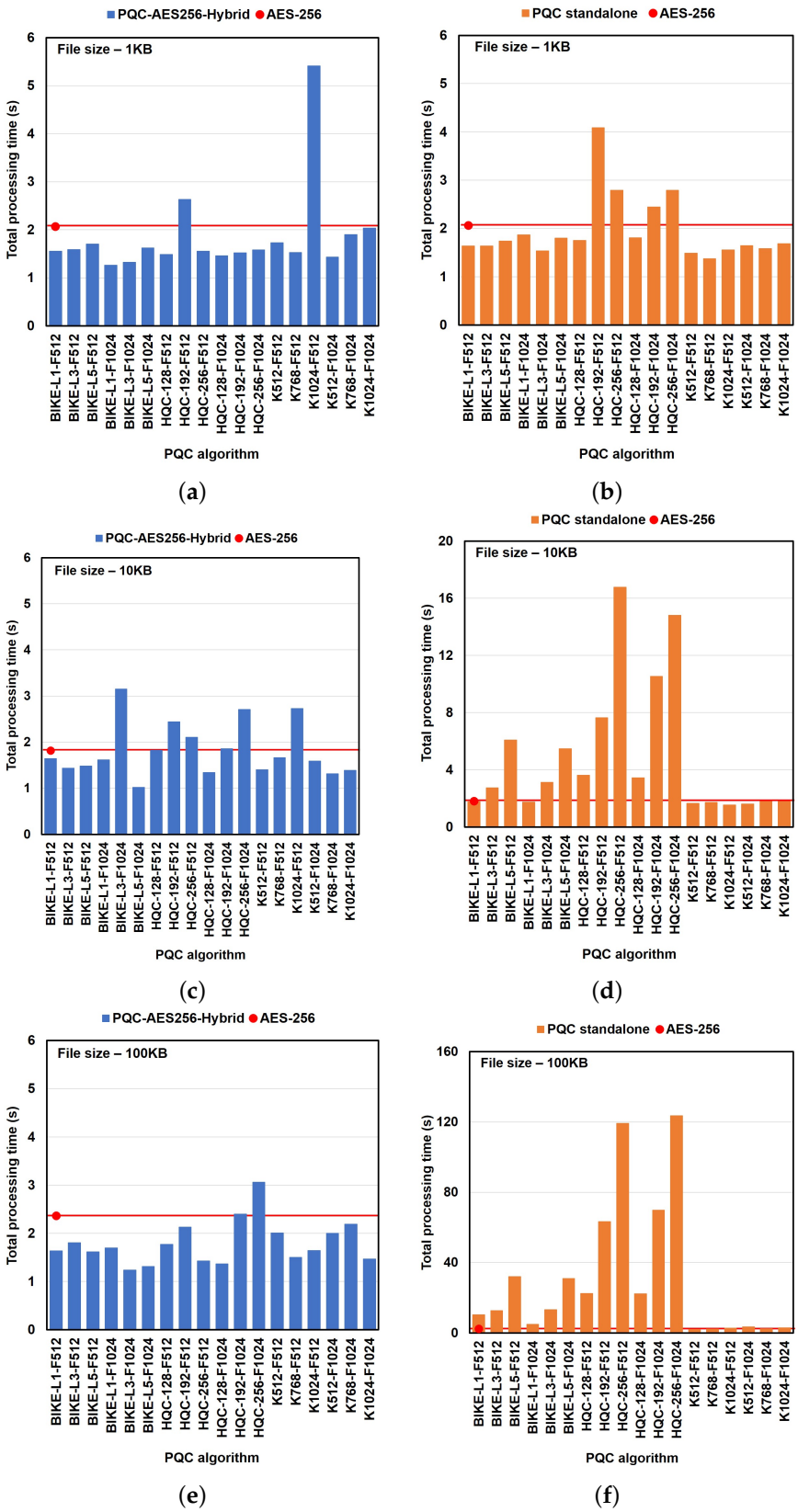

Figure 6. File transfer total processing times: (**a**) PQC–AES-256 for 1 KB. (**b**) PQC standalone for 1 KB. (**c**) PQC–AES-256 for 10 KB. (**d**) PQC standalone for 10 KB. (**e**) PQC–AES-256 for 100 KB. (**f**) PQC standalone for 100 KB.

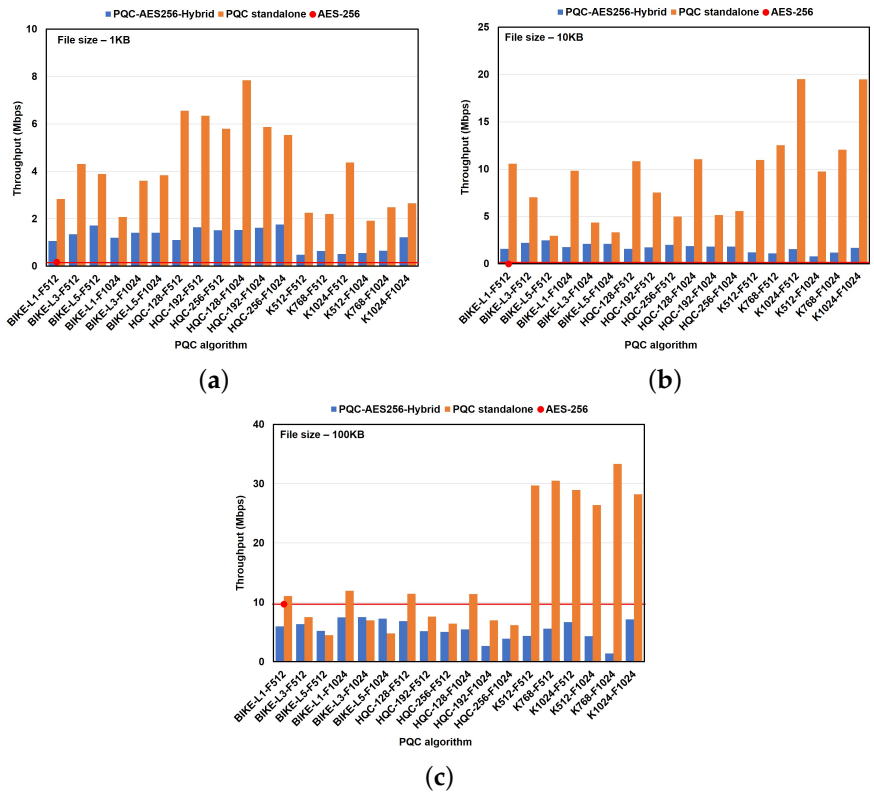

Figure 7. Throughputs of the encrypted files: (**a**) 1 KB file. (**b**) 10 KB file. (**c**) 100 KB file.

Processing Time

The overall processing times on the server system, segmented into 1 KB, 10 KB, and 100 KB file sizes, highlight the performance disparities between PQC–AES hybrid implementations, PQC standalone methods, and conventional AES-256. For 1 KB files, as shown in Figure 6a, the hybrid BIKE-L1-F512 protocol recorded an average processing time of 1.56 s, marginally faster than its PQC standalone counterpart (1.64 s), while AES-256 achieved 2.06 s, demonstrating comparable efficiency to hybrid approaches. Conversely, according to the results in Figure 6b, HQC-256-F512 in PQC standalone mode required 2.79 s, exceeding AES-256 by 35%, whereas its hybrid variant reduced this to 1.56 s. For CRYSTALS-Kyber, K512-F512 exhibited inverse behavior, with hybrid processing at 1.73 s outperformed by its PQC standalone counterpart (1.48 s), suggesting algorithm-specific optimization variances.

Figure 6c shows the processing times for 10 KB files. Hybrid methods generally retained advantages: BIKE-L5-F512 hybrid processing required 1.48 s, significantly lower than the time of its PQC standalone counterpart of 6.08 s, while AES-256 required 1.82 s. HQC-256-F512's PQC standalone time, as shown in Figure 6d, surged to 16.78 s, 9.2 times slower than that of AES-256, whereas its hybrid implementation reduced this to 2.11 s. In comparison, K1024-F512 hybrid processing (2.73 s) was slightly higher than the PQC standalone variant (1.55 s), although both remained within 1.5–2.7 s, aligning closer with the efficiency of AES-256.

Similarly, the 100 KB file processing times showed the scalability challenges of PQC standalone methods. As shown in Figure 6f, BIKE-L5-F512 PQC standalone processing reached 32.02 s, 13.5 times slower than AES-256's 2.37 s, while its hybrid variant, as shown in Figure 6e, achieved 1.62 s, outperforming AES-256 by 34%. HQC-256-F512's PQC standalone time escalated to 119.18 s, 50.3 times higher than AES-256, whereas hybrid modes like HQC-128-F512 (1.77 s) and K768-F512 (2.20 s) demonstrated competitive or superior performance relative to AES-256. These results emphasize hybrid protocols'

potential to mitigate PQC overheads, particularly for larger data volumes, while reaffirming AES-256's fixed efficiency for mid-sized operations.

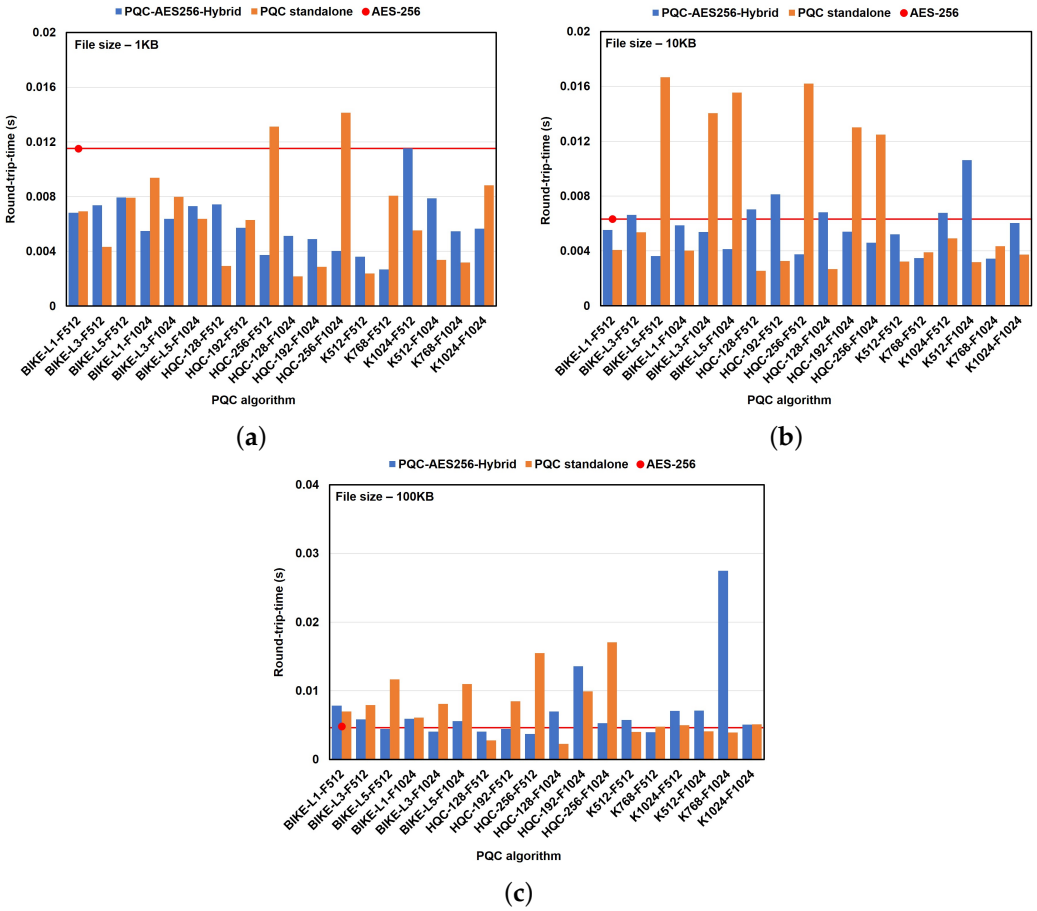

Figure 8. Round-trip times of the encrypted files: (**a**) RTT for 1 KB file. (**b**) RTT for 10 KB file. (**c**) RTT for 100 KB file.

Throughput

Furthermore, the server-side throughput results shown in Figure 7, categorized by 1 KB, 10 KB, and 100 KB file sizes, illustrate the interplay between ciphertext dimensions and quantum-resistant algorithm performance relative to AES-256. Figure 7a shows results for the 1 KB data, confirming that AES-256 achieved a throughput of 0.170 Mbps, while the PQC standalone methods with larger ciphertexts, such as HQC-128-F1024, reached 7.841 Mbps, a 46.1-fold increase. This variation arose not from superior efficiency but from expanded ciphertext sizes increasing the throughput metrics. It was observed in the hybrid implementations that the CRYSTALS-Kyber algorithm paired with the FALCON digital signature achieved throughput results comparable to AES-256. The results show total throughput values of K512-F512 (0.476 Mbps), K768-F512 (0.517 Mbps), K512-F1024 (0.554 Mbps), and K768-F1024 (0.647 Mbps).

Additionally, Figure 7b shows that at 10 KB, AES-256 throughput dropped to 0.0035 Mbps, whereas PQC standalone total throughput increased. K1024-F512 achieved a total throughput of 19.516 Mbps, reflecting the compounding effect of ciphertext expansion on throughput. BIKE-L1-F512 followed this trend at 10.578 Mbps, although its larger ciphertexts inherently amplified throughput values despite slower absolute processing speeds. For the hybrid modes, HQC-256-F512 showed a total throughput of 2.005 Mbps, 565 times higher than that of AES-256. This confirms the higher computational latency

of the PQC standalone algorithm compared to the conventional cryptographic algorithm, underscoring the metric's limitations in isolation.

For 100 KB files, as shown in Figure 7c, the AES-256 total throughput further increased to 9.725 Mbps, while PQC standalone K768-F1024 also increased to 33.320 Mbps, 3.4 times higher than AES-256. This disparity arose from the file size and ciphertext increase, as PQC algorithms produce larger ciphertexts than AES-256. BIKE-L1-F1024 achieved 11.968 Mbps in PQC standalone mode, while its hybrid variant (7.459 Mbps) lagged, emphasizing the trade-offs between post-quantum security and protocol complexity. Notably, BIKE-L5-F512's PQC standalone mode achieved a throughput (4.476 Mbps) lower than AES-256 by a factor of 1.9, despite its larger ciphertexts, highlighting vulnerabilities in the scalability of code-based algorithms. These results underscore that while PQC algorithms may exhibit higher throughput due to ciphertext expansion, this metric alone does not capture their operational efficiency or suitability for latency-sensitive applications. The results demonstrate that although larger ciphertext sizes in PQC algorithms result in higher throughput compared to AES-256, this does not directly imply greater efficiency.

Round-Trip Time

The RTT measurements for 1 KB, 10 KB, and 100 KB file sizes shown in Figure 8 reflect the operational latency introduced by ciphertext expansion and protocol overhead in PQC algorithms, despite their higher throughput metrics. Figure 8a shows the results for 1 KB files. AES-256 recorded an RTT of 0.0115 s, while multiple PQC standalone protocol algorithms achieved slightly lower latencies in comparison. HQC-128-F1024 achieved 0.0022 s, 5.2 times faster than AES-256, and K512-F512 achieved 0.0024 s. In the hybrid implementations, however, the algorithm pairs incurred slightly lower latencies than AES-256, while K1024-F512 (0.0115 s) achieved similar latency, underscoring the handshake overhead inherent to hybrid protocols.

At 10 KB, as shown in Figure 8b, AES-256's RTT decreased to 0.0063 s, while the PQC standalone protocol algorithms maintained similar trends as with the 1 KB file size, except for a few algorithm pairs that showed slightly higher latencies. The PQC standalone protocols achieved higher RTTs for BIKE-L5-F512 (0.017 s), BIKE-L3-F1024 (0.014 s), BIKE-L5-F1024, HQC-256-F512 (0.016 s), HQC-192-F1024 (0.013 s), and HQC-256-F1024 (0.012 s) compared to the same algorithm pairs in hybrid mode and AES-256. Conversely, it was observed that the remaining algorithm pairs in hybrid mode showed the opposite trend, with slightly higher RTTs (HQC-128-F512 (0.007 s), HQC-192-F512 (0.008 s), HQC-128-F1024 (0.007 s), and K512-F1024 (0.011 s)) than those of PQC standalone mode. This showcases the suitability of migrating applications to PQC algorithms while highlighting the processing bottlenecks associated with certain PQC algorithms.

Lastly, Figure 8c shows that for 100 KB files, AES-256 achieved 0.0048 s, outperforming many PQC standalone methods. BIKE-L5-F512 required 0.0117 s, 2.4 times slower, and HQC-256-F512 reached 0.015 s, 3.1 times slower. Exceptions included HQC-128-F512 (0.0028 s) and K768-F1024 (0.0039 s), which were 1.7 times and 1.2 times faster than AES-256, respectively. Notably, in hybrid mode, K768-F1024 incurred a latency of 0.0275 s for 100 KB files, 5.7 times slower than AES-256, highlighting the compounded overhead of hybrid key exchanges and ciphertext processing. These results emphasize that while PQC algorithms may exhibit competitive throughput due to larger ciphertexts, their RTT performance is more directly tied to computational complexity and protocol design, exposing trade-offs between post-quantum security and real-time application suitability.

6.2.4. Impact of PQC Algorithms on Performance in the Chat-Based Use Case

To further evaluate the performance of our proposed protocols, we implemented the chat-based use case and measured the total processing time that users spent engaged in

an active chat session. The total processing time includes the KEM process, the time taken to input the chat message, and the encryption and decryption of the messages in a chat session. The text input time varied slightly for each algorithm, as it would in a real chat, and may account for slight variations in the total processing time (expected to be insignificant). In addition, we measured the RTT, which is the time taken from when a user sends a message to when a response is received (i.e., request and response pairs). The RTT is one of the most crucial metrics for measuring the responsiveness between chat partners. For this evaluation, we created custom messages, which were used across all PQC algorithms within our protocols. We used UTF-8 encoding, in which each ASCII character (including spaces and punctuation) is 1 byte. Table 4 shows the text and size of each message used for client–server communication in the chat use case.

Table 4. Examples of text messages.

User 1		User 2	
Text Message	**Size (Bytes)**	**Text Message**	**Size (Bytes)**
Hello	5	Hi, how are you?	16
I'm good, thanks. And you?	26	You are welcome. I'm fine too.	30
Are you ready for the demo?	27	Yes, all set. Let's get started!	32
Great! Here's the first part of the code. What do you think?	60	Looks interesting. I have a few questions about the results.	60
Sure, I can explain anything about it.	38	Thanks! I want to know how you handled the data preprocessing step.	67
I used the standard normalization method and added custom	57	That makes sense. Let's move on to the next section.	52

Processing Time

It was observed that the proposed protocols with various PQC algorithms are more suitable for the chat-based use case in comparison to the AES-256 conventional method. Figure 9 shows the performance of the protocols in maintaining end-to-end encryption similar to those in modern chat applications. Figure 9a shows the results when PQC KEM algorithms were paired with the F512 signature algorithm in hybrid mode. A lower processing time between 61 s and 73 s was achieved by all algorithm pairs than the conventional AES-256 algorithm, which recorded a total processing time of 80 s. For the BIKE algorithms, BIKE-L5-F512 achieved the lowest time of approximately 61 s, while for the HQC algorithms, HQC-256-F512 achieved the lowest time (63 s), and for the CRYSTALS-Kyber algorithms, K1024-F512 achieved the lowest time (64 s).

As shown in Figure 9b, when the signature algorithm F1024 was paired with each KEM algorithm, the total processing time ranged from 64 s to 69 s, with most of the algorithms achieving processing times similar to when the KEM algorithm was paired with F512. BIKE-L3-F1024 (64 s) and K1024-F1024 (64 s) achieved the lowest processing times compared to the other algorithms. However, slightly higher processing times were observed for the PQC standalone protocol. According to the results in Figure 9c, the CRYSTALS-Kyber algorithms achieved higher processing times for K512-F512 (77 s), K768-F512 (96 s), and K768-F512 (74 s).

A similar trend was observed when F1024 was used, as shown in Figure 9d, with a lower processing time for K768-F1024 (73 s), while the time increased for K512-F1024 (85 s). The HQC algorithms maintained almost consistent processing times compared to the hybrid protocol, with slight variations of less than 5 s. Conversely, BIKE experienced higher processing times in the PQC standalone protocol than in the hybrid protocol but lower processing times than the conventional AES-256 algorithm, as shown in Figure 9a,c. With the exception of K768-F512 and K512-F1024, which achieved higher processing times

than AES-256 in the PQC standalone protocol, all algorithms in this mode experienced lower overheads and are expected to function effectively without adversely impacting the QoS of a chat-based application.

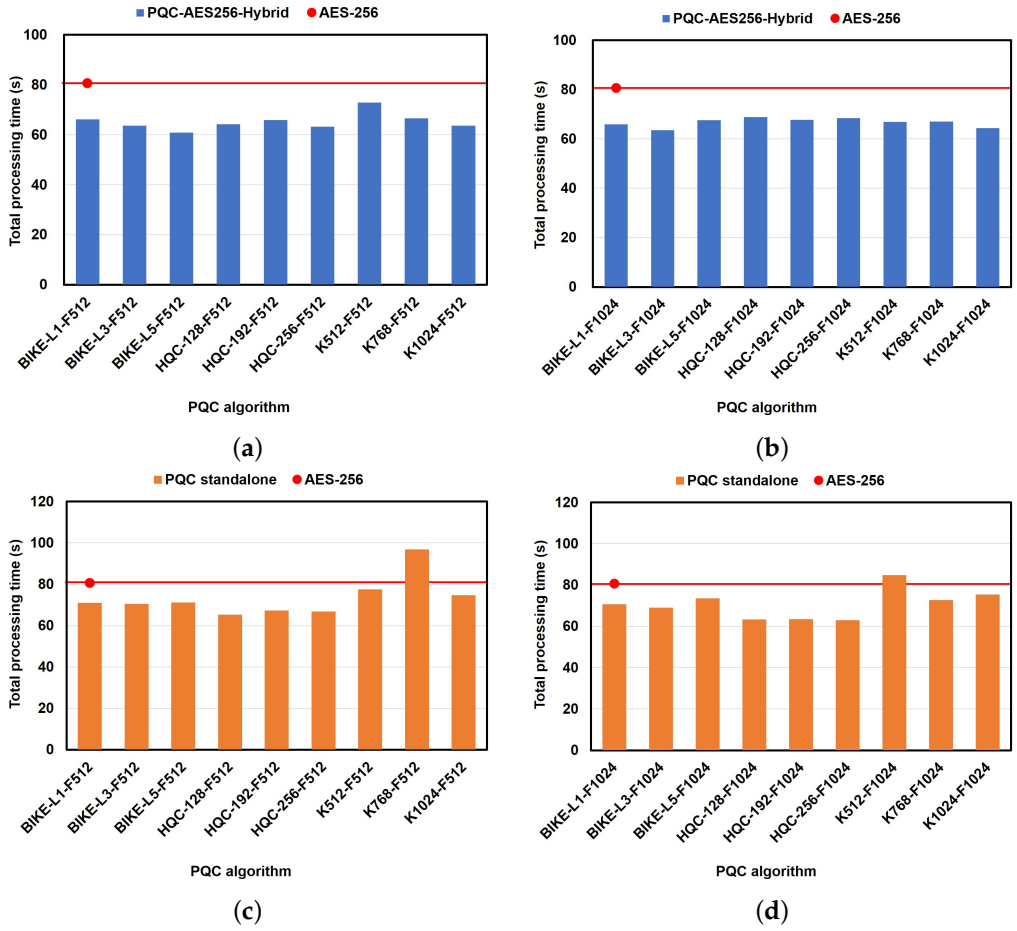

Figure 9. Chat total processing times: (**a**) PQC–AES-256 (with F512). (**b**) PQC–AES-256 (with F1024). (**c**) PQC standalone (with F512). (**d**) PQC standalone (with F1024).

Round-Trip Time

The end-to-end delay likely to be introduced by the proposed protocols was measured using the average RTT. It was expected that a delay for small, bursty messages, including delays due to encryption, network transit, and decryption, might occur. Therefore, the RTT, which is important in interactive chat, was considered. Figure 10 shows the average RTTs of the end-to-end delay of the proposed protocols. It was observed that the proposed protocols achieved better performance than conventional AES-256, which had an average RTT of 2.44 ms. According to the results shown in Figure 10a,b for the hybrid protocol, an average RTT of less than 1.5 ms was observed for all algorithms, except for K512-F512 (1.66 ms), K768-F512 (1.51 ms), and K512-F1024 (1.52 ms).

In the PQC standalone protocol, a lower delay was observed for most of the PQC algorithms, with the exception of CRYSTALS-Kyber, compared to the hybrid protocol. Figure 10c confirms that the average RTT decreased with the increase in the KEM security level for most of the PQC algorithms, ranging from 0.4 ms to 1.7 ms, except for K768-F512, which had a higher average RTT of 2.10 ms, slightly closer to that of AES-256 (2.44 ms). As shown in Figure 10d, a similarly lower average RTT trend, decreasing as the security level increased, was observed. However, there was a slight variation in the pattern for the BIKE algorithms compared to the hybrid protocol. It was confirmed that BIKE-L5-F1024

increased slightly to 0.99 ms. It was further confirmed that HQC-256-F1024 achieved the lowest average RTT of 0.28 ms.

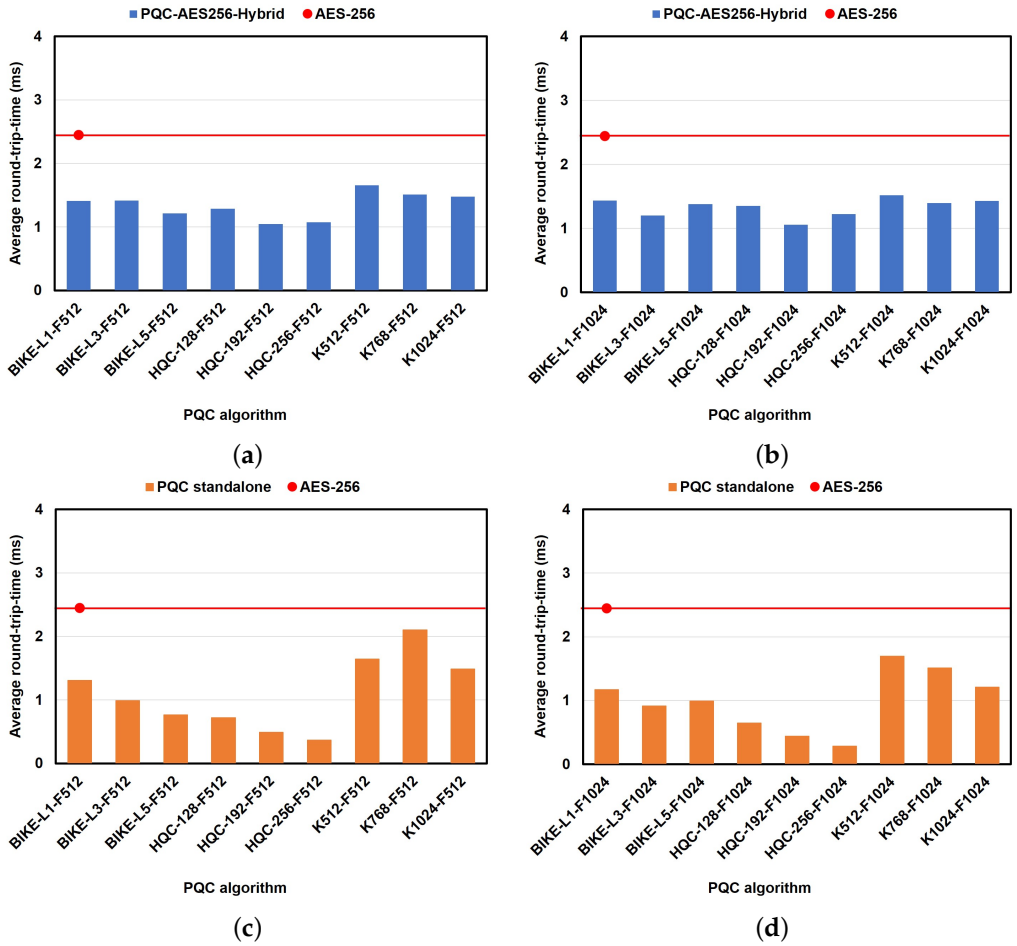

Figure 10. Chat average round-trip times: (**a**) PQC–AES-256 (with F512). (**b**) PQC–AES-256 (with F1024). (**c**) PQC standalone (with F512). (**d**) PQC standalone (with F1024).

6.2.5. Impact of the PQC Algorithm on Performance in the Video-Streaming Use Case

To demonstrate the impact of PQC algorithms on real-time applications such as video streaming, we evaluated the proposed protocols using a video-streaming application, as described in Section 4.1.1. We measured the overall throughput and retransmission rate as metrics. However, PQC algorithms are not suitable for applications such as video and live streaming in their current form without substantial optimization due to the large ciphertext size, which results in continuous freezes of video frames caused by the frequent retransmission of frames. Therefore, to evaluate the performance of our proposed protocol, we utilized the hybrid protocol and compared its performance with that of conventional AES-256.

In our evaluation, we streamed a short pre-recorded "MP4" video with the following properties: a total size of 17 MiB, a resolution of 1920 × 1080, 30 frames per second, a bitrate of 2.863 Mbps, and a duration of 47 s 249 ms. The server acquired each frame from the video file and encoded it before encrypting it. The encrypted frames were transmitted sequentially as rapidly as possible without limiting the rate. On the other hand, the client (stream viewer) decrypted and decoded each received encrypted frame. These frames were rendered using an OpenCV window for viewing the stream. The display loop included a standard 1 ms delay (i.e., cv2.waitkey(1)), which is necessary for OpenCV to process GUI events and refresh the displayed frame. This delay also allows for user interaction

(e.g., capturing user input, such as 'q' for quitting the stream). The delay is an attribute of the OpenCV GUI mechanism used for the experiment and is not part of the processing overhead of the proposed protocols.

Throughput

We measured the total throughput, which reflects how many encrypted video frames can be successfully transmitted. Since the user experience in video streaming is important, we measured the metrics from the client side. Figure 11 shows the total throughput achieved for streaming an encrypted video over the network. An almost constant total throughput was observed across all PQC algorithms. The results shown in Figure 11a show an approximate total throughput of 22 Mbps for K512-F512 and K768-F512 compared to the other algorithms and conventional AES-256. The other algorithms used in the hybrid protocol achieved a slightly lower approximate total throughput (20.1 Mbps) than conventional AES-256 (20.8 Mbps).

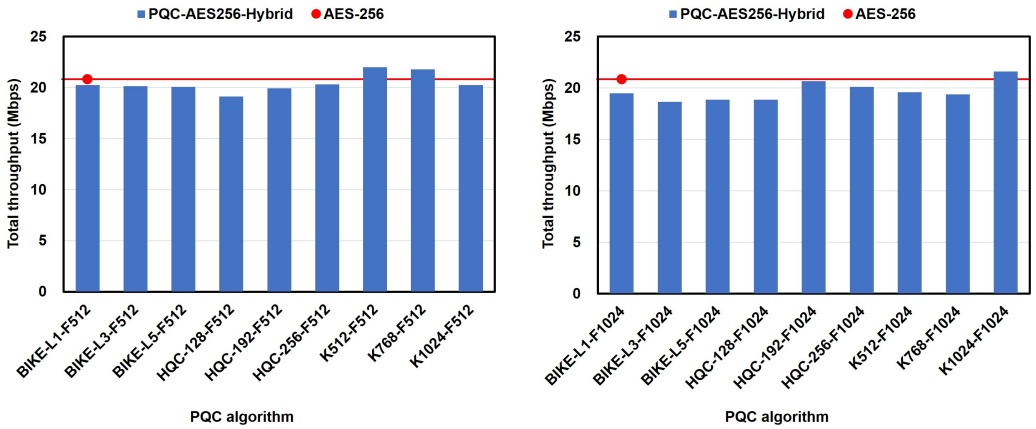

Figure 11. Video streaming total throughputs: (**a**) PQC–AES-256 (with F512). (**b**) PQC–AES-256 (with F1024).

Furthermore, as shown in Figure 11b, a similar trend was observed for the KEM algorithms paired with F1024, with almost all the algorithms achieving a slightly lower total throughput compared to the performance of AES-256, except for K1024-F1024, which achieved the highest total throughput of 21.6 Mbps, an 0.8 Mbps increase compared to conventional AES-256. The pairs BIKE-L3-F1024, BIKE-L5-F1024, HQC-128-F1024, and K768-F1024 achieved the lowest total throughput of approximately (19 Mbps) compared to all the other algorithms.

Retransmission Rate

Additionally, we measured the TCP retransmission rate. A higher retransmission rate indicates an increase in packet loss, often due to network congestion, which can result in freezes or buffering in the video stream. The results shown in Figure 12a confirm lower retransmission rates ranging from 0.05% to 0.20% for most algorithms in the hybrid protocol, except for BIKE-L1-F512, HQC-192-F512, K768-F512, and K1024-F512, which showed retransmission rates above 0.20%. BIKE-L5-F512 achieved the lowest retransmission rate, with 0.05% of the total encrypted frames streamed. Similarly, the pairs BIKE-L3-F512 (0.16%), HQC-128-F512 (0.16%), HQC-256-F512 (0.20%), and K512-F512 (0.15%) achieved lower retransmission rates than AES-256. Conversely, BIKE-L1-F512 (0.22%), HQC-192-F512 (0.25%), K768-F512 (0.27%), and K1024-F512 (0.26%) achieved higher retransmission rates than AES-256.

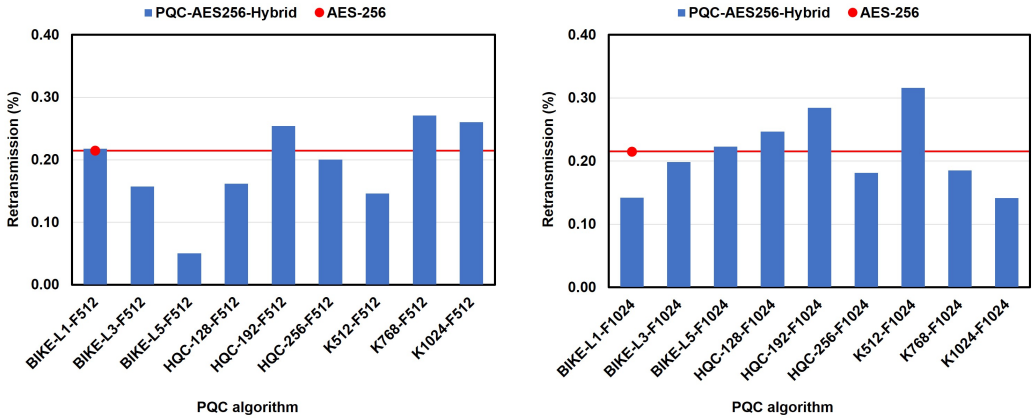

Figure 12. Video streaming frame retransmission times: (**a**) PQC–AES-256 (with F512). (**b**) PQC–AES-256 (with F1024).

As shown in Figure 12b, in comparison to the algorithms paired with F512, the KEM algorithms paired with F1024 exhibited an opposite trend, with the exception of HQC-192. According to the results, HQC-192-F1024 achieved a retransmission rate of 0.28%, which was 0.03% higher than that of HQC-192-F512. Similarly, a downward trend was recorded for CRYSTALS-Kyber as the security level increased, which differs from the retransmission rates observed when the KEM algorithm was paired with F512. K512-F1024 achieved the highest retransmission rates of 0.32%, which was higher than that of K512-F512 by 0.17% while K768-F1024 and K1024-F1024 achieved lower retransmission rates of 0.09% and 0.12% than K768-F512 and K1024-F512, respectively. It was further confirmed that BIKE-L5-F1024 (0.22%), HQC-128-F1024 (0.25%), and K512-F1024 (0.32%) all achieved higher retransmission rates than conventional AES-256. This suggests that further optimization may be necessary in order to use these algorithms effectively in real-time applications.

6.2.6. Impact of the PQC Algorithm on Performance in the Live-Streaming Use Case

Similar to video streaming, we evaluated the impact of the proposed protocols using a live-streaming session. We measured the total throughput and retransmission rate, which are essential metrics in real-time applications. The live-streaming use case involved capturing frames from a webcam in real time. For this, a live video was captured from the system's default webcam using OpenCV's video interface. A constant live-streaming duration of 30 s was used across all PQC algorithms. This was done to keep the size of the captured file smaller, as it increases exponentially with the number of frames and stream duration. After capturing each frame, the same process used for encoding and encryption, as described in the video-streaming use case, was utilized in the live-streaming use case.

Throughput

Figure 13 shows the total throughputs observed from the live-streaming session. As shown in Figure 13a, an almost constant total throughput was achieved with most of the PQC algorithms across all security levels compared to AES-256 (9.7 Mbps). BIKE-L3-F512 achieved the lowest throughput of 8.6 Mbps, while K512-F512 achieved a total throughput of 9.2 Mbps, slightly less than AES-256. The other algorithms recorded slightly higher total throughputs. Similarly, as shown in Figure 13b, total throughputs comparable to that of AES-256 were observed across all PQC algorithms, with BIKE-L1-F1024 (10.1 Mbps), HQC-128-F1024 (10.0 Mbps), and HQC-192-F1024 (10.3 Mbps) achieving total throughputs slightly higher than AES-256.

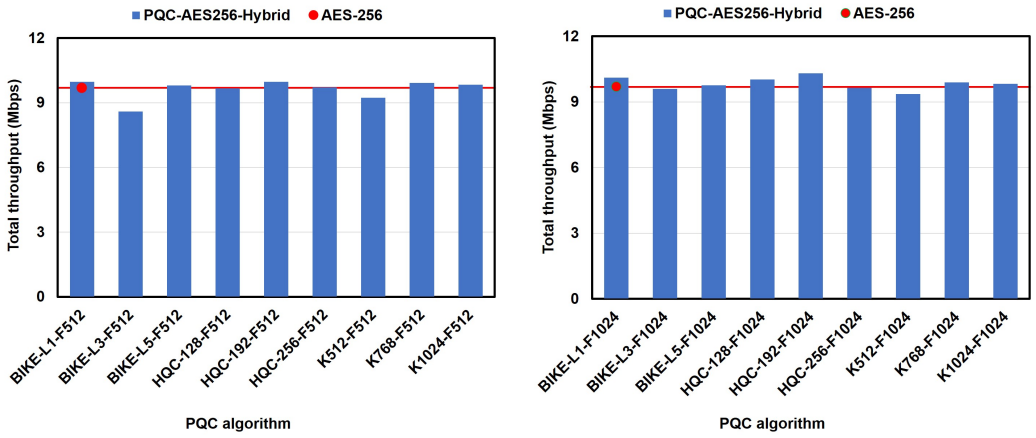

Figure 13. Live streaming total throughputs: (**a**) PQC–AES-256 (with F512). (**b**) PQC–AES-256 (with F1024).

Retransmission Rate

Finally, we measured the retransmission rates of the live stream. According to the results shown in Figure 14a, most of the PQC algorithms in the hybrid protocols are suitable for the live-streaming use case and comparable to AES-256 performance. A lower retransmission rate of less than 0.1% was observed in most algorithms, except for HQC-256-F512 (0.16%) and K512-F512 (0.67%), compared to AES-256 (0.72%), while higher retransmission rates were observed for BIKE-L3-F512 (2.28%) and HQC-128-F512 (1.46%).

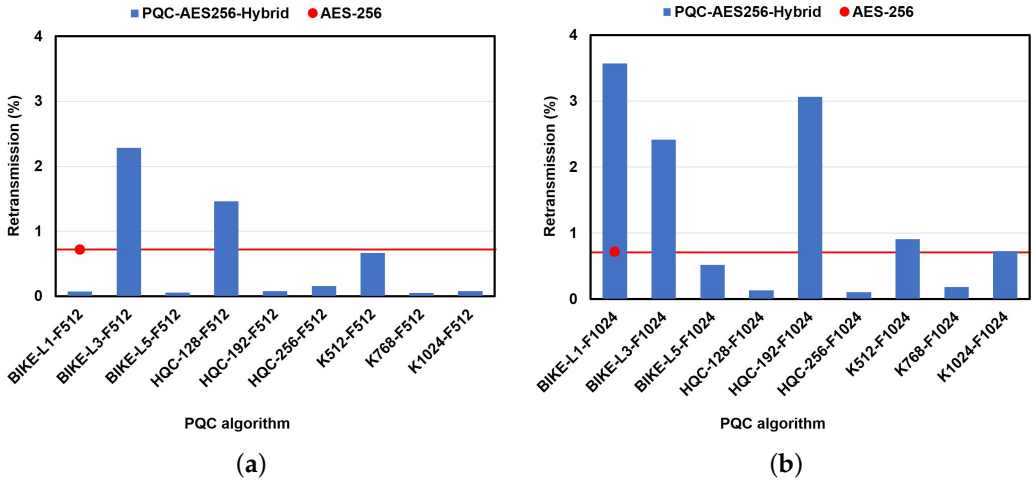

Figure 14. Live streaming frame retransmission rates: (**a**) PQC–AES-256 (with F512). (**b**) PQC–AES-256 (with F1024).

Additionally, for KEM algorithms paired with F1024, Figure 14b shows the retransmission rates. BIKE-L1-F1024 (3.57%), BIKE-L3-F1024 (2.42%), and HQC-192-F1024 (3.06%) had higher retransmission rates compared to the other algorithms and AES-256. The lowest retransmission rate was observed for HQC-256-F1024 (0.1%). This suggests that BIKE-L1-F1024, BIKE-L3-F1024, and HQC-192-F1024 may not be suitable for real-time live streaming without freezing and buffering compared to the other algorithms.

6.3. Limitations and Future Directions

In the current implementation of our proposed protocols, use cases in the area of distributed systems and network scenarios with congestion or high mobility are not currently considered. While the proposed protocols are designed for adoption in various use

cases, extensive adaptation to mitigate challenges common in these areas requires further study to ensure correct implementation and migration of applications associated with such scenarios to the quantum-secure protocol. To further improve our proposed protocols, as part of our future efforts, we will consider expansion to various systems such as distributed architectures, network scenarios with high overhead impacts, and mobility.

Furthermore, while our current implementation ensures MAC synchronization and consistency between the client and the server, formal modeling remains an open issue. In future work, we will consider the use of formal verification tools to analyze the robustness of MAC synchronization in scenarios where delays, packet loss, or concurrent events may occur. Additionally, we will expand the proposed protocols to include additional network environments and platforms, including energy consumption analysis on ARM and other IoT-specific hardware. Moreover, the current implementation of the proposed protocols focuses on the application layer and does not directly address side-channel attacks at the lower levels. A detailed side-channel analysis using tools such as ChipWhisper and constant-time execution testing in embedded scenarios is necessary and will be considered as part of the expansion of the proposed protocols' implementation.

7. Conclusions

As part of the advancement toward the post-quantum era, this paper proposed PQC standalone and PQC–AES hybrid protocols to evaluate their practical implementation within real-world secure communication systems. Specifically, the secure exchange of keys is achieved using PQC algorithms (KEM) in either PQC standalone or hybrid protocol modes. In PQC standalone mode, the KEM (public) keys exchanged between the client and server are used for encryption, while each party uses their secret keys for decryption, following the principles of asymmetric-key cryptography. Conversely, PQC–AES hybrid mode uses two-way shared secret keys derived from PQC KEM algorithms to generate encryption and decryption keys using a key derivation function (KDF), thereby reducing the computational overhead for larger data sizes. The proposed PQC-based protocols mitigate replay, MITM attacks, and other conventional cybersecurity attack vectors.

A performance evaluation of the proposed protocols was conducted through experiments using various real-world use cases (file transfer, chat-based communication, video streaming, and live streaming) in a client–server setup. For file transfer, the results confirmed that the hybrid protocols, which leverage the strengths of PQC algorithms, achieved performance relatively closer to that of AES-256 for larger payloads. For 10 KB files, the K512-F1024 hybrid mode achieved 0.8 Mbps compared to AES-256's 0.0035 Mbps, without the significant latency bottlenecks observed in PQC standalone mode, such as 19.5 Mbps for K1024-F512 and K1024-F1024. Additionally, it was confirmed that the hybrid protocol achieved better performance in terms of processing time compared to the PQC standalone protocol and AES-256. The processing times for the hybrid protocol were consistently lower across all algorithms, except for HQC192-F512 and K1024-F512, which recorded 2.6 s and 5.4 s for 1 KB files, compared to AES-256's 2.06 s. For 100 KB files, HQC-192-F1024 and HQC-256-F1024 experienced higher processing times of 2.4 s and 3.1 s compared to AES-256's 2.37 s.

Our findings confirm that PQC standalone algorithms remain effective for lightweight applications with smaller data sizes (e.g., file transfer with smaller size data and chat-based communication), while PQC–AES hybrid protocols provide a scalable and efficient solution for larger payloads (i.e., all use cases), ensuring quantum-resistant security with manageable latency and computational complexity. Using PQC algorithms directly in most applications will adversely impact performance, while the hybrid approach is more suitable

for most applications. This reaffirms the necessity of designing and migrating applications to quantum-safe cryptographic protocols tailored to specific application requirements.

In the advancement toward quantum-resistant security, a critical focus should be on addressing stringent QoS requirements in emerging technologies like 5G/6G and IoT. These sectors necessitate a balance between robust security and real-time performance, requiring a hybrid approach that leverages the strengths of both PQC and traditional cryptographic algorithms. Our evaluation results provide valuable insights into the practicality and performance of standalone PQC protocols and can be extended to facilitate secure data transmission in 5G and IoT environments, ensuring a balance between quantum-resistant security and operational efficiency that upholds the stringent QoS requirements inherent in modern wireless networks.

In future work, we will evaluate the complexity and overhead of the proposed protocols. We will extend the implementation of the PQC standalone PKE to include other PQC algorithms such as McEliece, Dilithium, and $SPHINCS^{+}$, among others. In addition, we will explore methods for utilizing the PQC standalone PKE in various applications like video streaming and messaging. Furthermore, we will perform an extensive evaluation using various network environments, including 5G/6G.

Author Contributions: Conceptualization, B.O. and K.Y.; methodology, B.O.; software, B.O.; validation, B.O. and K.Y.; formal analysis, B.O.; investigation, B.O., K.Y.; resources, T.K., K.Y., T.S. and H.Y.; data curation, B.O.; writing—original draft preparation, B.O.; writing—review and editing, B.O., T.K., K.Y., T.S. and H.Y.; visualization, B.O.; supervision, K.Y., T.S. and H.Y.; project administration, K.Y., T.S. and H.Y.; funding acquisition, K.Y., T.S. and H.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research and development work was supported by "Research and development on new-generation encryption technology for secure wireless communication services" under "Research and development for expansion of radio wave resources" (JPJ000254) of the Ministry of Internal Affairs and Communications.

Data Availability Statement: The original contributions presented in this study are included in the article.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

PQC	Post-Quantum Cryptography
KEM	Key Encapsulation Mechanism
PKE	Public-Key Encryption
AES	Advanced Encryption Standard
NIST	National Institute of Standards and Technology
SSL	Secure Sockets Layer
TLS	Transport Layer Security
SSH	Secure Shell
MITM	Man-in-the-Middle
MAC	Message Authentication Code
KDF	Key Derivation Function
HKDF	Hash-Based Message Authentication Code Key Derivation Function
QC-MDPC	Quasi-Cyclic Moderate Density Parity Check
ACK	Acknowledgment

References

1. Equal1. Bell-1: The First Quantum System Purpose-Built for the HPC Era. Available online: https://www.equal1.com/post/equal1-launches-bell-1-the-first-quantum-system-purpose-built-for-the-hpc-era (accessed on 16 April 2025).
2. Esmailiyan, A.; Wang, H.; Asker, M.; Koskin, E.; Leipol, D.; Bashir, I.; Xu, K.; Koziol, A.; Blokhina, E.; Staszewski, R.B. A Fully Integrated DAC for CMOS Position-Based Charge Qubits with Single-Electron Detector Loopback Testing. *IEEE Solid-State Circuits Lett.* **2020**, *3*, 354–357. [CrossRef]
3. Staszewski, R.B.; Esmailiyan, A.; Wang, H.; Koskin, E.; Giounanlis, P.; Wu, X.; Koziol, A.; Sokolov, A.; Bashir, I.; Asker, M.; et al. Cryogenic Controller for Electrostatically Controlled Quantum Dots in 22-nm Quantum SoC. *IEEE Open J. Solid-State Circuits Soc.* **2022**, *2*, 103–121. [CrossRef]
4. Barker, W.; Polk, W.; Souppaya, M. *Getting Ready for Post-Quantum Cryptography: Exploring Challenges Associated with Adopting and Using Post-Quantum Cryptographic Algorithms*; NIST Cybersecurity White Paper; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2021. [CrossRef]
5. Chen, L.; Jordan, S.; Liu, Y.-K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D. *Report on Post-Quantum Cryptography*; NIST Internal Report 8105; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2016. [CrossRef]
6. Alagic, G.; Apon, D.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Miller, C.; Moody, D.; Peralta, R.; et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*; NIST Internal Report 8413-upd1; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2022. [CrossRef]
7. *Special Publication 800-227*; Recommendations for Key-Encapsulation Mechanisms. NIST: Gaithersburg, MD, USA, 2024.
8. *FIPS 203*; Module-Lattice-based Key-Encapsulation Mechanism. Federal Information Processing Standards Publication. NIST: Gaithersburg, MD, USA, 2024. [CrossRef]
9. *FIPS 204*; Module-Lattice-Based Digital Signature Standard. Federal Information Processing Standards Publication. NIST: Gaithersburg, MD, USA, 2024. [CrossRef]
10. *FIPS 205*; Stateless Hash-Based Digital Signature Standard. Federal Information Processing Standards Publication. NIST: Gaithersburg, MD, USA, 2024. [CrossRef]
11. Alagic, G.; Bros, M.; Ciadoux, P.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Liu, Y.; Miller, C.; et al. *Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process*; NIST Internal Report (NIST IR 8545); National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2025. [CrossRef]
12. Ojetunde, B.; Kurihara, T.; Yano, K.; Sakano, T.; Yokoyama, H. Performance Evaluation of Post-Quantum Cryptography Algorithms for Secure Communication in Wireless Networks. In Proceedings of the IEEE Consumer Communications & Networking Conference (CCNC2025), Las Vegas, NV, USA, 10–13 January 2025.
13. Cooper, D.A.; Apon, D.C.; Dang, Q.H.; Davidson, M.S.; Dworkin, M.J.; Miller, C.A. *Recommendation for Stateful Hash-Based Signature Schemes*; NIST Special Publication 800-208; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2020. [CrossRef]
14. Ghosh, S.; Upadhyay, S.; Saki, A.A. A Primer on Security of Quantum Computing. *arXiv* **2023**, arXiv:2305.02505v1.
15. Perepechaenko, M.; Kuang, R. Quantum Encryption and Decryption in IBMQ Systems using Quantum Permutation Pad. *J. Commun.* **2022**, *17*, 972–978. [CrossRef]
16. Schwabe, P.; Stebila, D.; Wiggers, T. Post-Quantum TLS Without Handshake Signatures. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, 9–13 November 2020; Association for Computing Machinery: New York, NY, USA; pp. 1461–1480. [CrossRef]
17. Schwabe, P.; Stebila, D.; Wiggers, T. More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys. In *Computer Security—ESORICS 2021*; Bertino, E., Shulman, H., Waidner, M., Eds.; Lecture Notes in Computer Science 12972; Springer: Cham, Switzerland, 2021. [CrossRef]
18. Gunther, F.; Rastikian, S.; Towa, P.; Wiggers, T. KEMTLS with Delayed Forward Identity Protection in (Almost) a Single Round Trip. In *Applied Cryptography and Network Security, ACNS 2022*; Ateniese, G., Venturi, D., Eds.; Lecture Notes in Computer Science 13269; Springer: Cham, Switzerland, 2022. [CrossRef]
19. Singh, R.P.; Sarma, B.K.; Saikia, A. Public Key Cryptography Using Permutation P-Polynomials over Finite Fields. Cryptology ePrint Archive, Paper 2009/208. 2009. Available online: https://eprint.iacr.org/2009/208.pdf (accessed on 4 April 2025).
20. Marco, L.; Talayhan, A.; Vaudenay, S. Making Classical (Threshold) Signatures Post-Quantum for Single Use on a Public Ledger. Cryptology ePrint Archive, Paper 2023/420. 2023. Available online: https://eprint.iacr.org/2023/420 (accessed on 4 April 2025).
21. da Silva Lima, P.; Ribeiro, L.A.; Queiroz, R.J.; Quintino, J.P.; Silva, F.Q.; Santos, A.L.; Roberto, J. Evaluating Kyber Post-Quantum KEM in a Mobile Application. 2021. Available online: https://csrc.nist.gov/CSRC/media/Events/third-pqc-standardization-conference/documents/accepted-papers/ribeiro-evaluating-kyber-pqc2021.pdf (accessed on 30 May 2025).
22. Giron, A.A. Migrating Applications to Post-Quantum Cryptography: Beyond Algorithm Replacement. Cryptology ePrint Archive, Paper 2023/709. 2023. Available online: https://eprint.iacr.org/2023/709 (accessed on 4 April 2025).

23. Liu, T.; Ramachandran, G.; Jurdak, R. Post-Quantum Cryptography for Internet of Things: A Survey on Performance and Optimization. *arXiv* **2024**, arXiv:2401.17538.
24. Asif, R. Post-Quantum Cryptosystems for Internet-of-Things: A Survey on Lattice-Based Algorithms. *IoT* **2021**, *2*, 71–91. [CrossRef]
25. Li, S.; Yuxiang, C.; Lin, C.; Jing, L.; Chanchan, K.; Kuanching, L.; Wei, L.; Naixue, X. Post-Quantum Security: Opportunities and Challenges. *Sensors* **2023**, *23*, 8744. [CrossRef] [PubMed]
26. Balamurugan, C.; Singh, K.; Ganesan, G.; Rajarajan, M. Post-Quantum and Code-Based Cryptography—Some Prospective Research Directions. *Cryptography* **2021**, *5*, 38. [CrossRef]
27. Fakhruldeen, H.F.; Al-Kaabi, R.A.; Jabbar, F.I.; Al-Kharsan, I.H.; Shoja, S.J. Post-quantum Techniques in Wireless Network Security: An Overview. *Malays. J. Fundam. Appl. Sci.* **2023**, *19*, 337–344. [CrossRef]
28. Lawo, D.C.; Abu Bakar, R.; Cano Aguilera, A.; Cugini, F.; Imaña, J.L.; Tafur Monroy, I.; Vegas Olmos, J.J. Wireless and Fiber-Based Post Quantum-Cryptography-Secured IPsec Tunnel. *Future Internet* **2024**, *16*, 300. [CrossRef]
29. Liboqs. Available online: https://openquantumsafe.org/liboqs/ (accessed on 4 April 2025).
30. Sharpe, R.; Warnicke, E.; Lamping, U. Wireshark User's Guide. Version 4.5.0. Available online: https://www.wireshark.org/docs/wsug_html_chunked/ (accessed on 30 May 2025).