

Article

Not peer-reviewed version

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

[Babatunde Ojetunde](#) *, [Takuya Kurihara](#) , [Kazuto Yano](#) , [Toshikazu Sakano](#) , [Hiroyuki Yokoyama](#)

Posted Date: 13 May 2025

doi: [10.20944/preprints202505.0990.v1](https://doi.org/10.20944/preprints202505.0990.v1)

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



Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

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

Babatunde Ojetunde *, Takuya Kurihara, Kazuto Yano , Toshikazu Sakano and Hiroyuki Yokoyama [†]

Wave Engineering Laboratories, Advanced Telecommunications Research Institute International, 2-2-2 Hikaridai, Seika, Soraku, Kyoto, 619-0228, Japan; ojetunde@atr.jp (B.O); tkurihara@atr.jp (T.K); kzyano@atr.jp (K.Y); t.sakano@atr.jp (T.S)

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

[†] Adaptive Communications Research Laboratories, Advanced Telecommunications Research Institute International, 2-2-2 Hikaridai, Seika, Soraku, Kyoto, 619-0228, Japan; hr-yokoyama@atr.jp (H.Y)

‡ These authors contributed equally to this work.

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

Abstract: Recent advances in quantum computing have prompted urgent consideration for 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 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 implementation 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 these standalone PQC and 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, the 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.

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 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 to 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 migrating applications to be PQC compliant will introduce, 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 emphasize their stringent quality-of-service (QoS) requirements.

NIST has selected four PQC algorithms [7], and also 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, NIST selected HQC [11] for the next PQC KEM algorithm to be standardized, out of 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 protocols 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), 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 impacts on applications.

In the PQC standalone mode, secure communication channel is established between 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 a secure and seamless end-to-end communication between the two parties by 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 effects 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 increasing consideration of shifting 5G to a PKI-based trust model.

This paper significantly extends our prior work [12], where 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 work on PKE and PQC. Section 3 describes the proposed PQC-based protocols, while Section 4 details the use cases and 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 the performance evaluation of the proposed method, and Section 7 concludes the paper.

2. Related Work

There have been an extensive work and several ongoing work on the implementation of 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 [13,14] and implementation of the PQC KEM algorithm across different hardware platforms. Additionally, most current implementations of the PQC algorithm have focused on its integration with Secure Sockets Layer (SSL), Transport Layer Security (TLS) [15–17], 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 [18], Singh *et al.* propose a new multivariate public key cryptosystem based on permutation p-polynomials over finite fields, which focuses on the need for secure and efficient public key systems that can withstand increasing computing power and quantum computing. Marco *et al.* [19] propose 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.* [20] evaluated the efficiency of the Kyber KEM algorithm in a mobile application, specifically analyzing its performance on x86 and ARM architectures.

In addition to this, surveys and reviews that highlighted the issues and constraints of migrating applications to be PQC compliant have been carried out. Giron [21] discussed the challenges and research efforts required for migrating applications to PQC due to the vulnerability of current public-key cryptography schemes against quantum computers. Liu *et al.* [22] 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 [23] presented a comprehensive review of PQC for IoT, covering both theoretical and practical aspects. Li *et al.* [24] reviewed PQC algorithms, focusing on the Kyber algorithm, and discussed the challenges and opportunities of post-quantum security.

Balamurugan *et al.* [25] discussed the need for PQC in light of the threat quantum computing poses to classical encryption schemes.

Fakhruldeen *et al.* [26] provides 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 highlights the vulnerabilities of current cryptographic standards to quantum attacks and explores solutions such as PQC and Quantum Key Distribution (QKD), alongside practical migration strategies like hybrid approaches. They further underscores 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.

3. PQC-Based PKE Protocols for Using PQC Algorithm in Application

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 know 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 algorithm KEM.

3.1. PQC Standalone PKE Protocol

The KEM is often used in conjunction with a PKE scheme for securely exchanging encryption keys, which can then be used to encrypt data using the PKE scheme [28]. Therefore, in this paper, we selected PKE as the choice of cryptographic scheme for using PQC algorithms in various applications.

In the improved PQC standalone PKE protocol as 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 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 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

- 1: **Input:** Data to be transmitted ($data_c$ from Client, $data_s$ from Server)
- 2: **Output:** Securely exchanged data
- 3: **Client (C):**
- 4: Generate PQC KEM key pair: (pkk_c, skk_c)
- 5: Generate PQC Signature key pair: (pks_c, sks_c)
- 6: $cert[pkk_c] \leftarrow \text{Sign}(skc_s, pkk_c)$
- 7: Send $(cert[pkk_c], pks_c, pkk_c)$ to Server (S)
- 8: **Server (S):**
- 9: Generate PQC KEM key pair: (pkk_s, skk_s)
- 10: Generate PQC Signature key pair: (pks_s, sks_s)
- 11: $valid_c \leftarrow \text{Verify}(pks_c, cert[pkk_c], pkk_c)$
- 12: **if** $valid_c = \text{TRUE}$ **then**
- 13: $(ct_c, ss_c) \leftarrow \text{Encapsulate}(pkk_c)$
- 14: $cert[pkk_s] \leftarrow \text{Sign}(skc_s, pkk_s)$
- 15: Send $(cert[pkk_s], pks_s, pkk_s, ct_c)$ to Client (C)
- 16: **else**
- 17: **Abort**
- 18: **end if**
- 19: **Client (C):**
- 20: $valid_s \leftarrow \text{Verify}(pks_s, cert[pkk_s], pkk_s)$
- 21: **if** $valid_s = \text{TRUE}$ **then**
- 22: $ss'_c \leftarrow \text{Decapsulate}(skk_c, ct_c)$
- 23: $(ct_s, ss_s) \leftarrow \text{Encapsulate}(pkk_s)$
- 24: Send ct_s to Server (S)
- 25: $seq_c \leftarrow \text{GenerateSequenceNumber}()$
- 26: $mac_c \leftarrow \text{HMAC-SHA256}(ss'_c || ss_s, \text{"confirmation"} || seq_c)$
- 27: Send mac_c to Server (S)
- 28: **else**
- 29: **Abort**
- 30: **end if**
- 31: **Server (S):**
- 32: $ss'_s \leftarrow \text{Decapsulate}(skk_s, ct_s)$
- 33: $seq_s \leftarrow \text{GenerateSequenceNumber}()$
- 34: $mac_s \leftarrow \text{HMAC-SHA256}(ss_c || ss'_s, \text{"confirmation"} || seq_s)$
- 35: **if** $mac_c = \text{HMAC-SHA256}(ss_c || ss'_s, \text{"confirmation"} || seq_c)$ **then**
- 36: Send mac_s to Client(C)
- 37: **else**
- 38: **Abort**
- 39: **end if**
- 40: **Client(C):**
- 41: **if** $mac_s = \text{HMAC-SHA256}(ss'_s || ss_s, \text{"confirmation"} || seq_s)$ **then**
- 42: Secure communication is established.
- 43: **else**
- 44: **Abort**
- 45: **end if**
- 46: **Client-to-Server Data Transfer:**
- 47: $ct_{data_c} \leftarrow \text{PQC-Encrypt}(pkk_s, data_c || seq_c)$
- 48: Send ct_{data_c} to Server (S)
- 49: **Server (S):**
- 50: $data'_c || seq'_c \leftarrow \text{PQC-Decrypt}(skk_s, ct_{data_c})$
- 51: **if** seq'_c is valid (not replayed) **then**

```

52: Process  $data'_c$ 
53: else
54:   Reject // Replay detected
55: end if
56: Server-to-Client Data Transfer:
57:  $ct_{data_s} \leftarrow \text{PQC-Encrypt}(pkk_c, data_s || seq_s)$ 
58: Send  $ct_{data_s}$  to Client (C)
59: Client (C):
60:  $data'_s || seq'_s \leftarrow \text{PQC-Decrypt}(skk_c, ct_{data_s})$ 
61: if  $seq'_s$  is valid then
62:   Process  $data'_s$ 
63: else
64:   Reject
65: 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. 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 if 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 if 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 the traditional PQC-based PKE protocols. Firstly, we incorporate explicit certificate validation during initial handshake steps to ensure that exchanged KEM public keys are authentic and untampered. Secondly, 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 then 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.

Algorithm 2 PQC-AES hybrid PKE Protocol

```

1: Input: Data to be transmitted ( $data_c$  from Client,  $data_s$  from Server)
2: Output: Securely exchanged data
3: Client (C) and Server (S):
4: Perform steps 1-24 of Algorithm 1 to establish shared secrets  $ss_c$  and  $ss_s$  (and confirm them).
5: Key Derivation (Both C and S):
6:  $(K_1, K_2) \leftarrow \text{HKDF}(ss_c || ss_s, \text{salt}, \text{"AES keys"})$ 
7: Client-to-Server Data Transfer:
8:  $iv_c \leftarrow \text{GenerateInitializationVector}()$ 
9:  $seq_c \leftarrow \text{GenerateSequenceNumber}()$ 
10:  $ct_{data_c} \leftarrow \text{AES-Encrypt}(K_1, iv_c, data_c, \text{mode}, seq_c)$ 
11:  $mac_{data_c} \leftarrow \text{HMAC-SHA256}(K_2, ct_{data_c} || seq_c)$ 
12: Send  $(iv_c, ct_{data_c}, mac_{data_c})$  to Server (S)
13: Server (S):
14: if  $mac_{data_c} = \text{HMAC-SHA256}(K_2, ct_{data_c} || seq_c)$  then
15:    $data'_c \leftarrow \text{AES-Decrypt}(K_1, iv_c, ct_{data_c}, \text{mode}, seq_c)$ 
16:   if  $seq_c$  is valid (not a replay) AND Decryption successful then
17:     Process  $data'_c$ 
18:   else
19:     Reject
20:   end if
21: else
22:   Abort
23: end if
24: Server-to-Client Data Transfer:
25:  $iv_s \leftarrow \text{GenerateInitializationVector}()$ 
26:  $seq_s \leftarrow \text{GenerateSequenceNumber}()$ 
27:  $ct_{data_s} \leftarrow \text{AES-Encrypt}(K_1, iv_s, data_s, \text{mode}, seq_s)$ 
28:  $mac_{data_s} \leftarrow \text{HMAC-SHA256}(K_2, ct_{data_s} || seq_s)$ 
29: Send  $(iv_s, ct_{data_s}, mac_{data_s})$  to Client (C)
30: Client (C):
31: if  $mac_{data_s} = \text{HMAC-SHA256}(K_2, ct_{data_s} || seq_s)$  then
32:    $data'_s \leftarrow \text{AES-Decrypt}(K_1, iv_s, ct_{data_s}, \text{mode}, seq_s)$ 
33:   if  $seq_s$  is valid AND Decryption Successful then
34:     Process  $data'_s$ 
35:   else
36:     Reject
37:   end if
38: else
39:   Abort
40: end if

```

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 , purposed 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 , 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 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 to be 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.

4. Practical Implementation of PQC-Based PKE Protocols

Since there is no function for data encryption and decryption in the current PQC algorithm implementation, we enhance the PQC algorithm implementation by adding encryption and decryption functions. This enables data encryption and decryption in both PQC standalone and the 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 PQC algorithm characteristics, 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, CRYSTAL-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 mode typically uses 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 read-in 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.

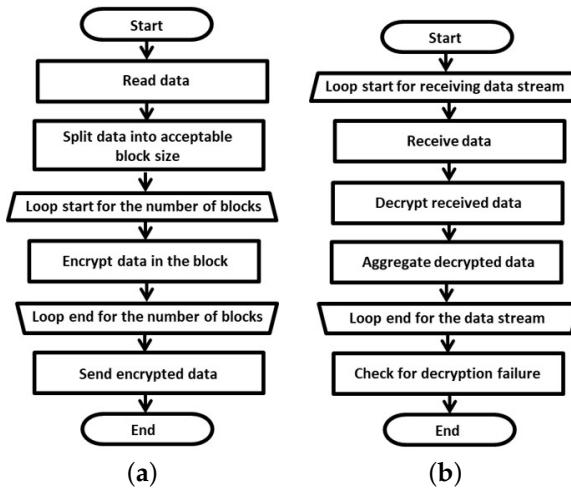


Figure 1. Encryption and decryption process of PQC protocols. (a) Encryption process. (b) Decryption process.

4.1.1. Real-World Scenario Use Cases for Proposed PQC-Based Protocols

Consider a real-world scenario of using our PQC-based protocols in real applications as shown in Figure 2: two devices representing the client and server, with the necessary software installed, including PQC and AES libraries. For the application scenario, we consider four use case scenarios: (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 the use case scenarios and a PQC-based system to ensure the data remains secure against potential future quantum computer attacks, the following process is used:

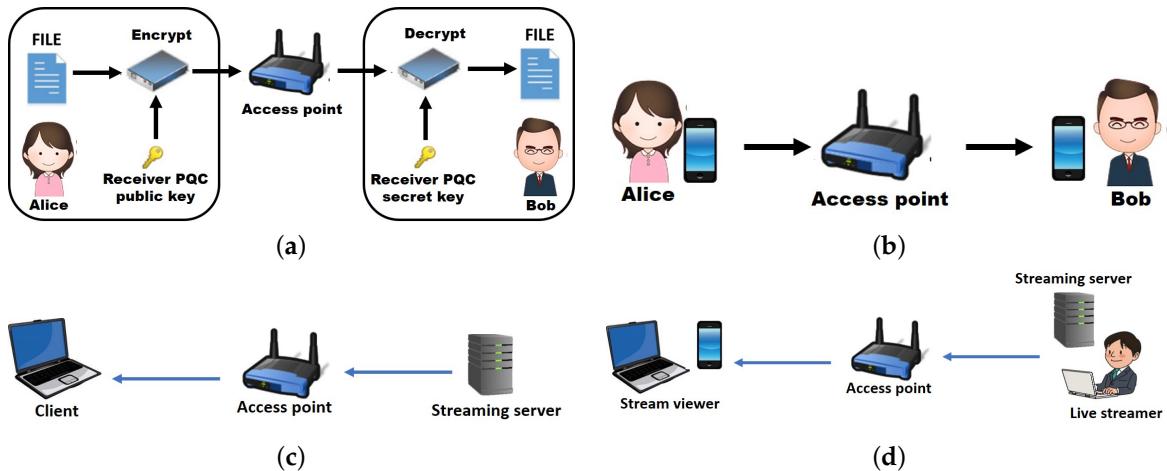


Figure 2. Use case scenario application for using PQC algorithm. (a) File transfer, (b) chat, (c) video streaming, (d) live streaming.

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 aggregated 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 being able to decrypt 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.

Using PQC-based protocols, Alice and Bob can ensure that their communication remains secure against threats posed by quantum computing. The proposed PQC-based protocols workflow 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 scenarios benefit from the protocol by encrypting and transmitting content in near real-time, with a focus on maintaining low latency to preserve the 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 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 pre-defined data structure including nonces or session identifiers. The MACs are verified to ensure that no MITM attacker has replaced either public key.

Security Assumption

The security relies on:

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:

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:

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 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 are applied.

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, we achieve 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 assurance 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 an 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/timestamps and rejects any messages that are out of order or outside an acceptable time window.

5.3.4. Chosen-Ciphertext Attacks (CCA)

This is relevant to the PQC KEM used in both protocols. NIST's selections are generally CCA-secure. The chosen PQC KEM (e.g., CRYSTAL-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, 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, PQC signature keys, perform encapsulation, decapsulation, file encryption, and decryption, among other operations. We use our extended version (i.e., our version 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 provided directly by NIST as one standard library.

6.1. Experiment Settings

In our implementation, we describe four use cases in Section 4.1.1. However, we use the file transfer application scenario to evaluate our protocols. The experiment is conducted on two separate machines: one with an Intel® Core™ i7-10510U CPU running at a base frequency of 1.80 GHz × 8 and 15.5 GiB of memory, running Ubuntu 20.04 LTS desktop, which serves as the server; and another with an Intel® Core™ i7-10510U CPU running at a base frequency of 1.80 GHz × 8 and 15.5 GiB of memory, running Ubuntu 20.04 LTS desktop, which is used as the client system. Table 1 presents the PC settings and the configuration 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 by NIST were considered, including KEM algorithms CRYSTAL-KYBER – 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) - 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 socket programming with Transport Control Protocol/User Datagram protocol (TCP/UDP protocols). 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, serving 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 experiment setup is shown in Figure 3.

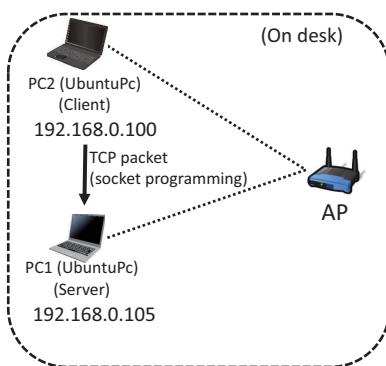


Figure 3. Experiment settings.

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 is to facilitate the practical use of the PQC algorithm in real applications and to determine the impact of using the PQC algorithm in the application use case and network. We consider two main evaluation scenarios: (i) the PQC algorithm performance—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 will be more suitable for real-time applications without extensive overhead. For this, we measure the processing time for encrypted data transfer, throughput, and round-trip time (RTT).

6.2. Results and Discussion

6.2.1. PQC Algorithm Key Generation Performance Results

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 PQC KEM algorithms, as well as the signature key generation time (Sig. Gen. Time), signing time, and verification time for PQC signature algorithms across client and server systems. Tables 2 and 3 show the average time it takes to generate PQC algorithm keys (PQC KEM and signature keys), encapsulate and decapsulate the KEM key, 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 at 0.43 ms while BIKE-L5 required 1.95 ms, reflecting a progressive increase in computational demand with higher security levels. Conversely, 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 client-side metrics. Encapsulation and decapsulation times also increased with security levels, particularly for 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 reached 49.77 ms (client) and 53.40 ms (server). Additionally, CRYSTAL-KYBER algorithms diverged from this trend: K768 achieved the fastest client-side key generation at 0.19 ms, surpassing both K512 (0.92 ms) and K1024 (1.00 ms). On the Server-side, CRYSTAL-KYBER results showed K1024 as 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 time to generate PQC algorithm KEM keys.

PQC KEM Algorithm	Client			Server		
	Ave. Key Gen. (ms)	Ave. Encap. (ms)	Ave. Decap. (ms)	Ave. Key Gen. (ms)	Ave. Encap. (ms)	Ave. 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

Furthermore, Table 3 shows the performance of 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. 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. Verification times remained consistent for F512 across systems (0.28 ms client vs. 0.82 ms server) but diverged slightly for F1024 (0.31 ms client against 0.79 ms server). These results underscore the trade-offs between security levels and computational overhead, particularly for server-side operations.

Table 3. Average time to generate PQC algorithm signature key.

PQC Signature	Client			Server		
	Ave. Sig. Key Gen. (ms)	Ave. Signing (ms)	Ave. Ver. (ms)	Ave. Sig. Key Gen. (ms)	Ave. Signing (ms)	Ave. 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

6.2.2. PQC Algorithm Encryption and Decryption Performance Results

In the hybrid protocol, AES-256 was used for data encryption/decryption. It is expected that the time for these operations should remain consistently low, meaning it will 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 total encryption times for BIKE, HQC, and CRYSTAL-KYBER algorithms when processing files of sizes 1 KB, 10 KB, and 100 KB. For BIKE-L1, as shown in Figure 4a, encryption times remained stable across file sizes, with 0.20 s for 1 KB, 0.198 s for 10 KB, and 0.148 s for 100 KB. However, BIKE-L3 exhibited higher encryption times for larger files, increasing from 0.47 s (1 KB) to 0.63 s (10 KB) before declining to 0.46 s (100 KB), while BIKE-L5 ranged from 0.58 s (1 KB) to 0.93 s (10 KB) and 0.82 s (100 KB).

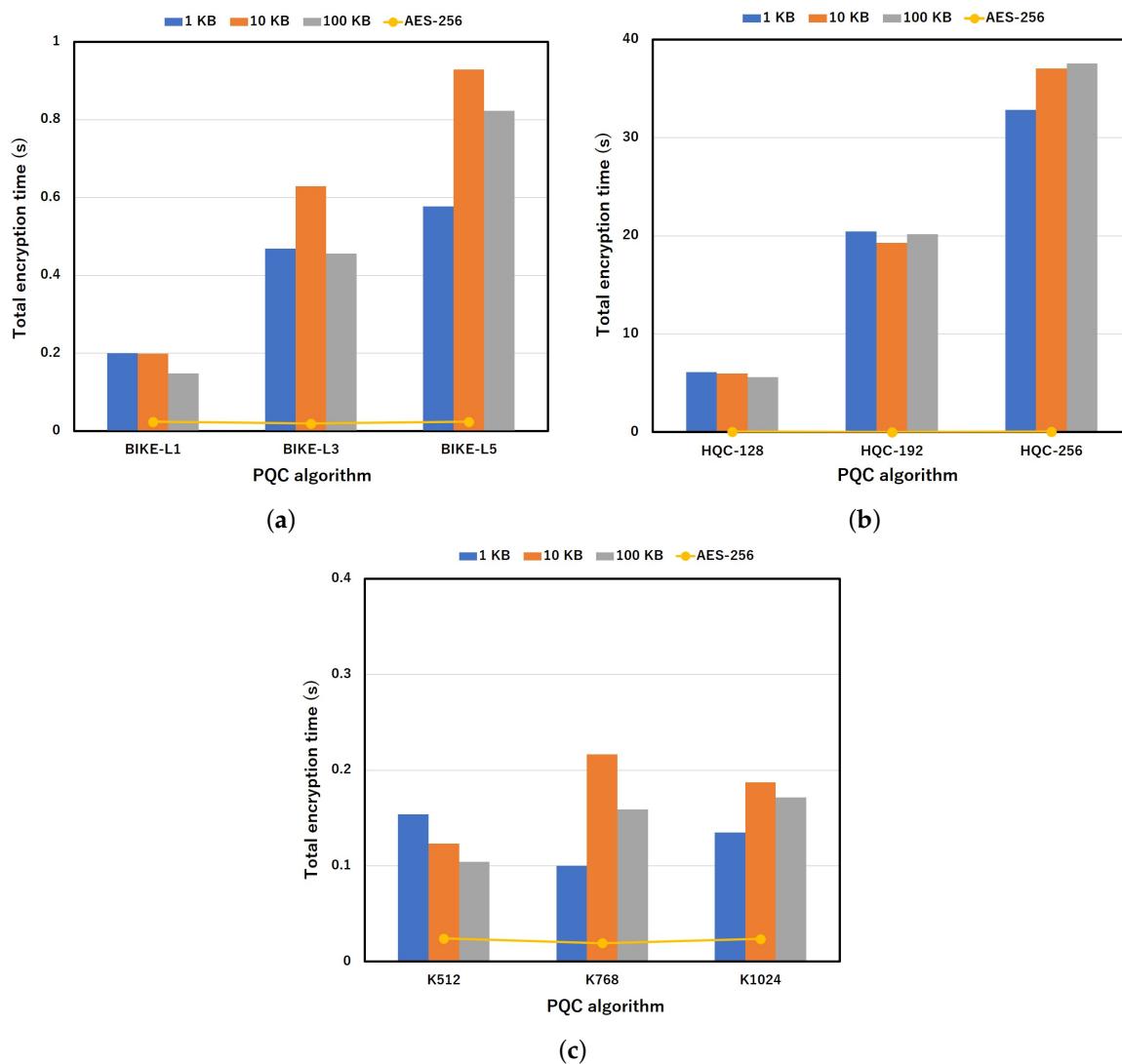


Figure 4. Total encryption time. (a) BIKE. (b) HQC. (c) CRYSTAL-KYBER.

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

Figure 5 shows the decryption times on the server system. BIKE-L1 decryption times increased sharply with file size, rising from 0.04 s (1 KB) to 2.89 s (100 KB), while BIKE-L5 shows the same trend from 0.29 s (1 KB) to 27.60 s (100 KB) as shown in Figure 5a. Figure 5b shows that HQC followed a similar pattern: HQC-128 decryption required 13.52 s for 100 KB files, whereas HQC-256 reached 71.31 s for the same file size.

In contrast, Figure 5c shows that the results of CRYSTAL-KYBER algorithms maintained consistently low decryption times, with K512 requiring 0.17 s and K1024 requiring 0.24 s for 100 KB files. Notably, K768's decryption time decreased slightly from 0.17 s (10 KB) to 0.17 s (100 KB), deviating from the upward trend observed in BIKE and HQC. Smaller file sizes (1 – 10 KB) exhibited minimal decryption times for all algorithms, with K512 and K768 achieving 0.03 s performance. These results reaffirm the computational efficiency of lattice-based algorithms like CRYSTAL-KYBER for both encryption and decryption, particularly when handling larger file sizes, compared to code-based (BIKE) and hash-based (HQC) alternatives.

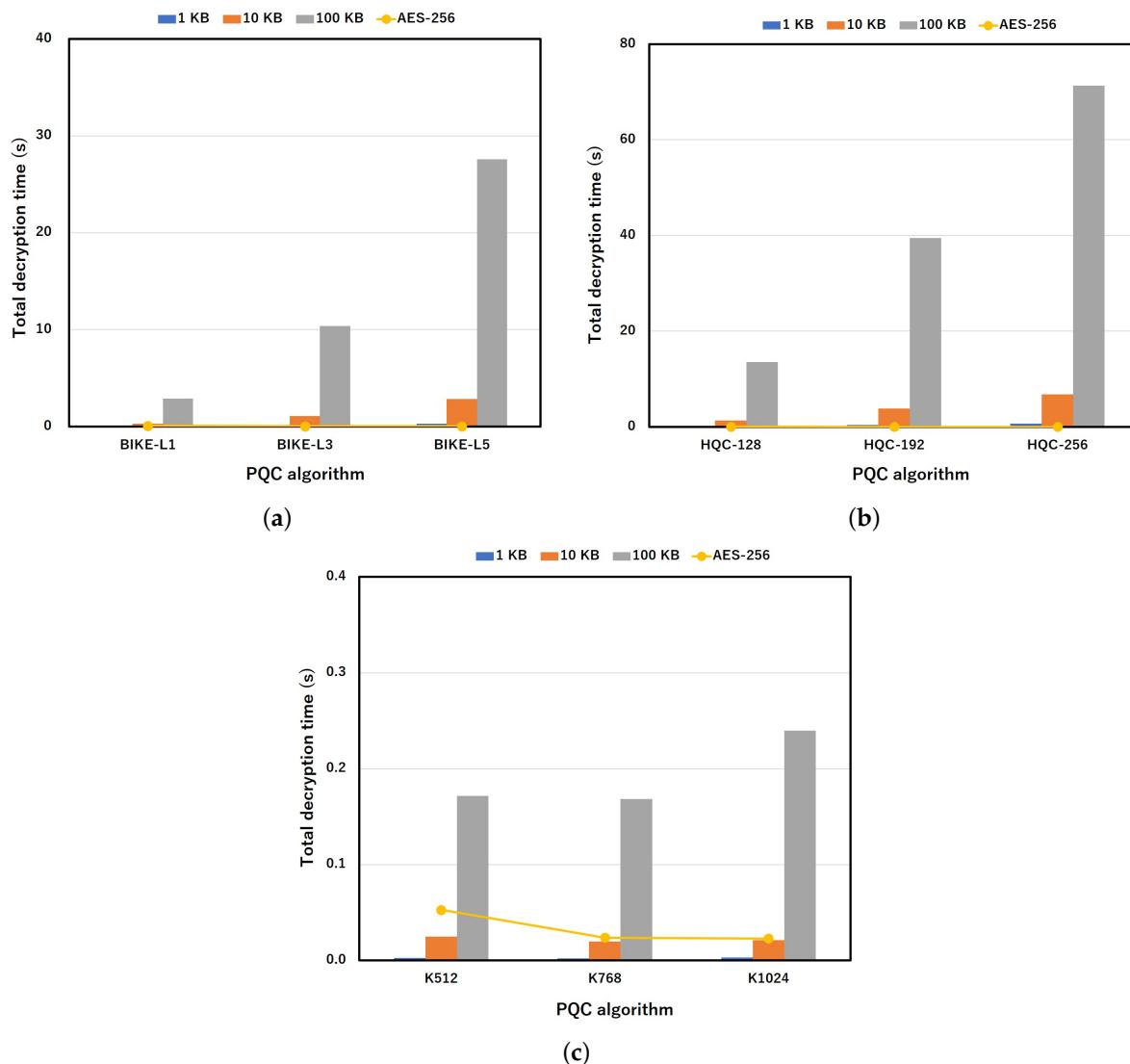


Figure 5. Total decryption time. (a) BIKE, (b) HQC, (c) CRYSTAL-KYBER.

The results confirm the computational overhead of PQC algorithms compared to conventional AES-256, as shown in Figures 4 and 5. In comparison, for encryption, AES-256 shows consistently lower times across all file sizes, with 0.024 s for 1 KB, 0.019 s for 10 KB, and 0.024 s for 100 KB, significantly outperforming the PQC algorithms. In the PQC standalone algorithm, the encryption time increased as the security level increases, with the exception of CRYSTAL-KYBER. The CRYSTAL-KYBER algorithm at all security levels has the lowest encryption time for all file sizes, as shown in Figure 4c. The fastest PQC algorithm in terms of encryption time is K768, which requires 0.10 s for 1 KB, exceeding AES-256's time by approximately 4.1 times.

Similarly, Figure 5 shows that the decryption times for AES-256 remained stable across the file sizes, with 0.053 s (1 KB), 0.024 s (10 KB), and 0.023 s (100 KB), which is in contrast with BIKE-L5's 27.60 s and HQC-256's 71.31 s for 100 KB files, as shown in Figure 5a,b. CRYSTAL-KYBER algorithms demonstrated closer alignment with AES-256, as shown in Figure 5c, with K768 decryption times of 0.17 s for 100 KB files, yet still exceeding AES-256 by 7.3 times. These findings underscore the persistent efficiency gap between conventional and post-quantum algorithms, emphasizing the trade-offs between quantum-resistant security and computational performance.

6.2.3. PQC Algorithm Impact on File Transfer Performance Results

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 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 measured 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 (this includes 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.

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 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, it was confirmed that 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 CRYSTAL-KYBER, K512-F512 exhibited inverse behavior, with hybrid processing at 1.73 s outperformed by PQC standalone (1.48 s), suggesting algorithm-specific optimization variances.

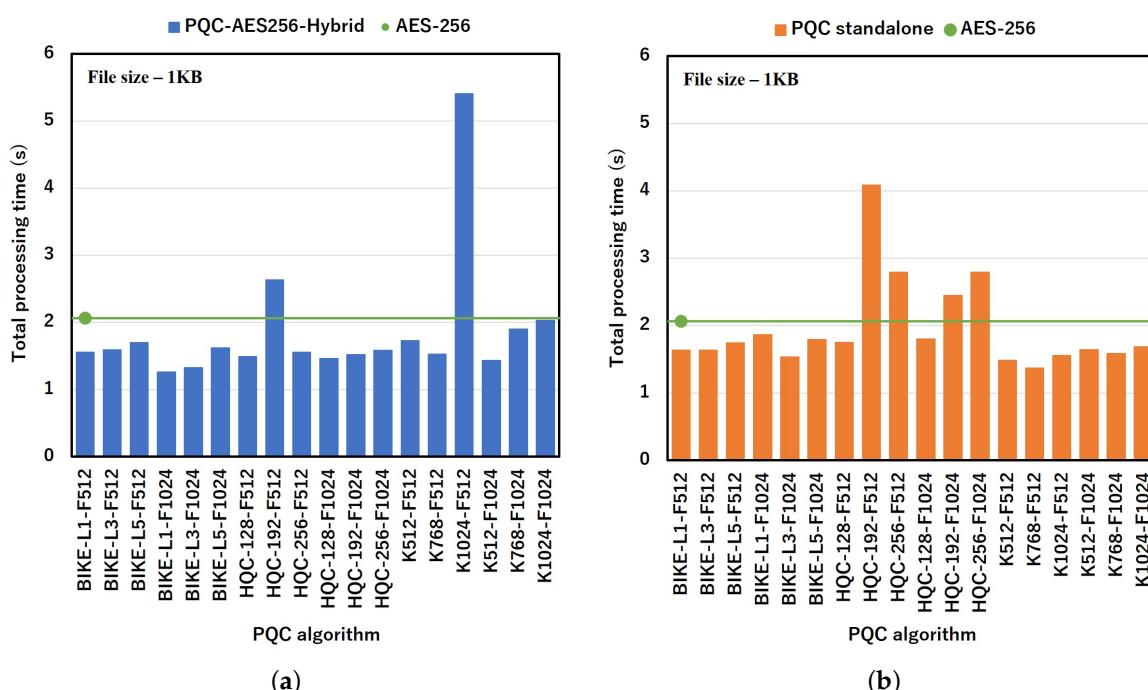


Figure 6. Cont.

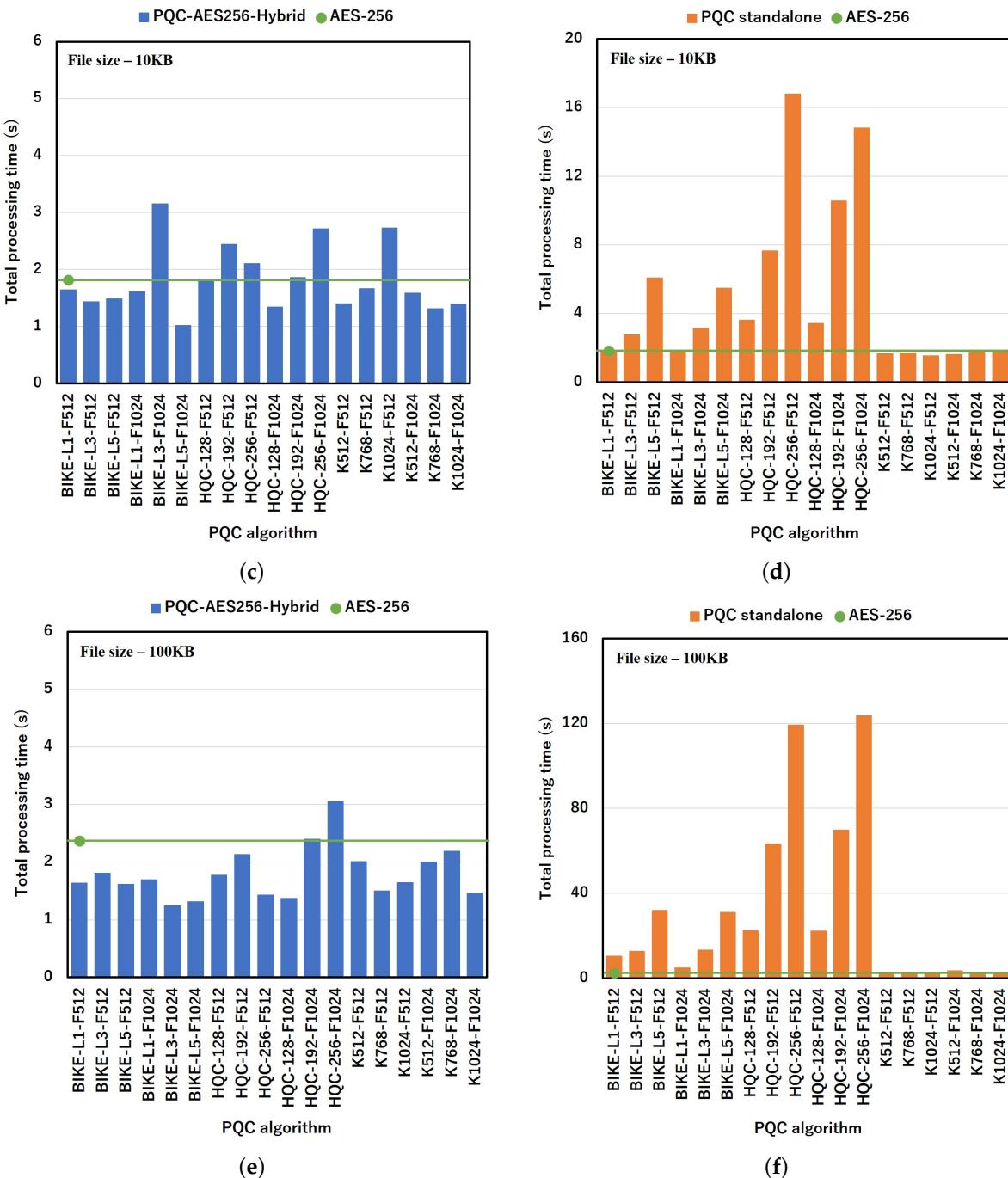


Figure 6. File transfer total processing time. (a) PQC-AES256 1 KB. (b) PQC standalone 1 KB. (c) PQC-AES256 10 KB. (d) PQC standalone 10 KB. (e) PQC-AES256 100 KB. (f) PQC standalone 100 KB.

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

Similarly, the 100 KB file processing time shows 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.

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 the 1 KB data; it was confirmed that AES-256 achieved a throughput of 0.170 Mbps, while PQC standalone methods with larger ciphertexts, such as HQC-128-F1024, reached 7.841 Mbps, a 46.1-fold increase. This variation arises not from superior efficiency but from expanded ciphertext sizes increasing throughput metrics. It was observed in the hybrid implementations that the CRYSTAL-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).

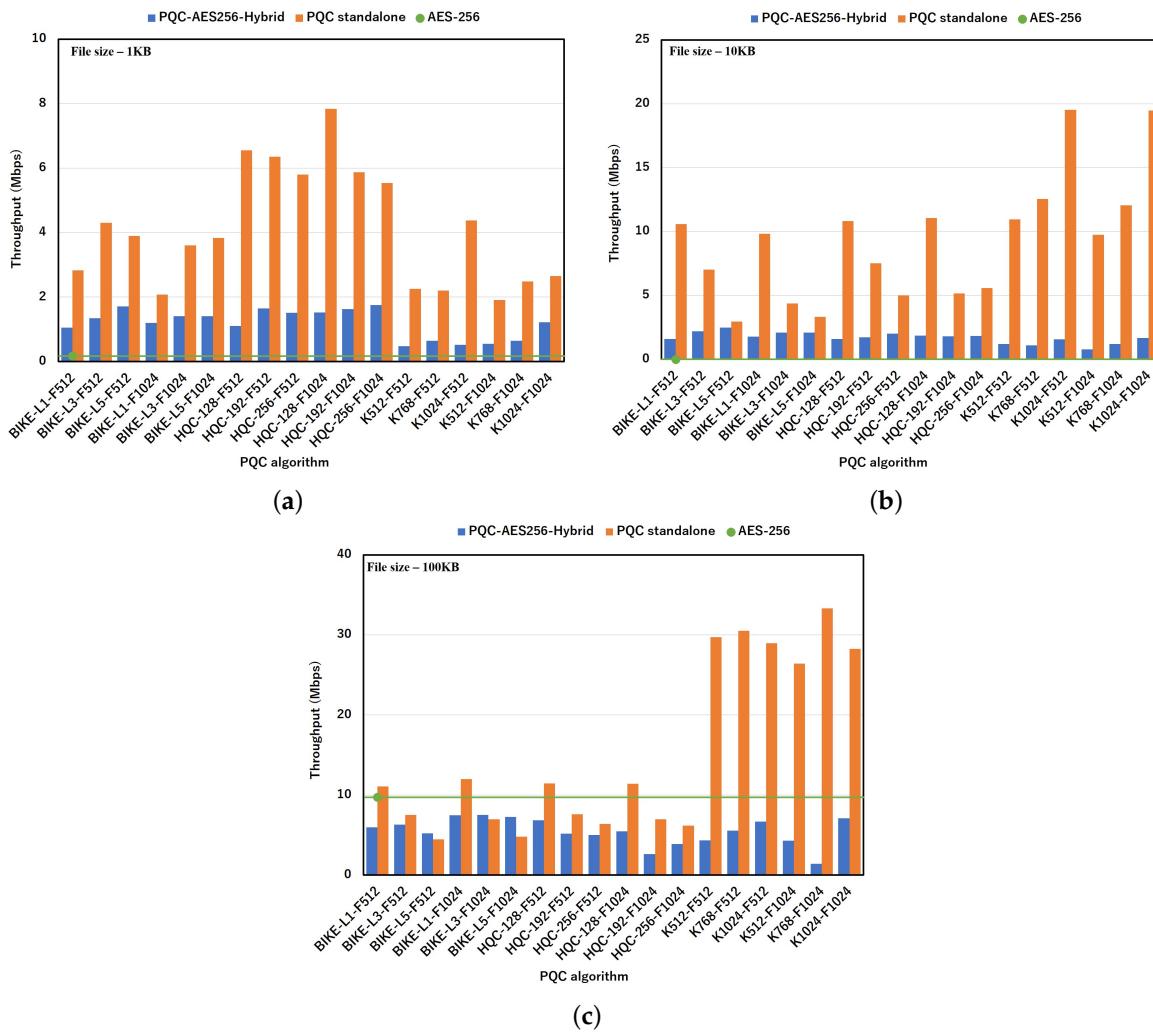


Figure 7. Throughput of the encrypted file. (a) 1KB file. (b) 10KB file. (c) 100KB file.

Additionally, Figure 7b shows that at 10KB, 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, though its larger ciphertexts inherently amplify 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 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 arises 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 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 latency 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 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 the 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 the 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.

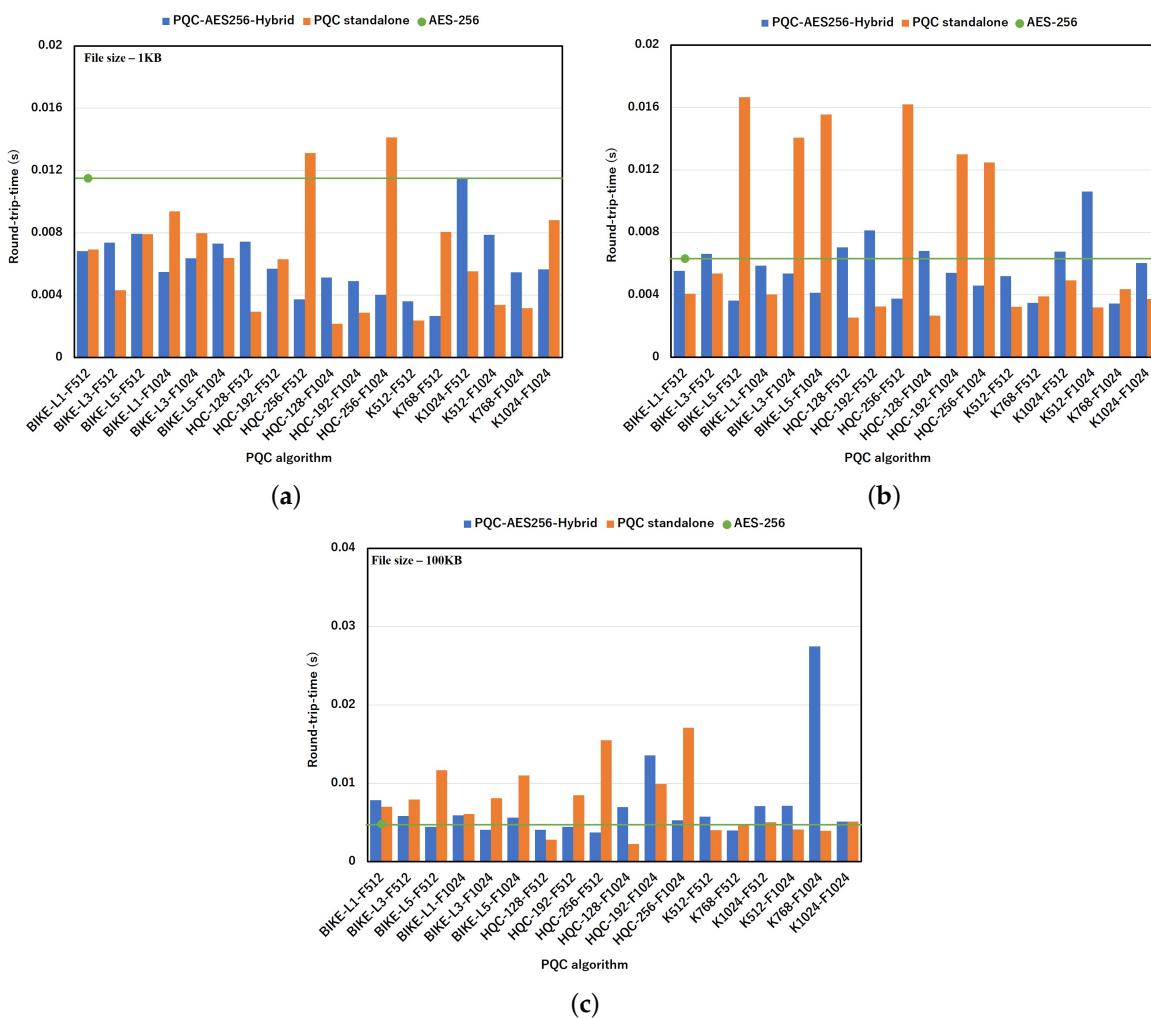


Figure 8. Round-trip-time of the encrypted file. (a) RTT 1KB file. (b) RTT 10KB file. (c) RTT 100KB file.

7. Conclusions

This paper proposed PQC standalone and PQC-AES hybrid protocols as part of the advancement towards the post-quantum era 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 the 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, the 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.

The performance evaluation of the proposed protocols was conducted through experiments using a file transfer communication scenario in a client-server setup. The results confirm that the hybrid protocols, which leverage the strengths of PQC algorithms, achieved performance relatively closer to 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 standalone PQC modes, 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 hybrid protocols 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 confirmed that standalone PQC algorithms remain effective for lightweight applications with smaller data sizes, while PQC-AES hybrid protocols provide a scalable and efficient solution for larger payloads, ensuring quantum-resistant security with manageable latency and computational complexity. Using PQC algorithms directly in most applications will adversely impact performance, while the hybrid mode 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 towards quantum-resistant security, a critical focus should be directed towards 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 PQC standalone 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.

As part of our 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 to utilize 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” in “Research and development for expansion of radio wave resources” (JPJ000254) of the Ministry of Internal Affairs and Communications.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: Not applicable

Acknowledgments: Not applicable

Conflicts of Interest: The authors declare no conflict 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.

Sample Availability: Not applicable

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 Codes
KDF	Key Derivation Function
HKDF	Hash-based Message Authentication code Key Derivation Function
QC-MDPC	Quasi-Cyclic Moderate Density Parity-Check
ACK	Acknowledgement

References

1. Equal1. Bell-1: The First Quantum System Purpose-Built for the HPC Era. [Online]. Available: <https://www.equal1.com/post/equal1-launches-bell-1-the-first-quantum-system-purpose-built-for-the-hpc-era>. [Accessed: April 16, 2025].
2. Esmailyan, 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. In IEEE Solid-State Circuits Letters, 3, pp. 354–357, 2020. <https://doi:10.1109/LSSC.2020.3018707>.
3. Staszewski R.B.; Esmailyan, A.; Wang, H.; Koskin, E.; Giounanlis, P.; Wu, X.; Koziol, A.; Sokolov, A.; Bashir, I.; Asker, M.; Leipol, D.; Nikandish, R.; Siriburanon, T.; Blokhina, E. Cryogenic Controller for Electrostatically Controlled Quantum Dots in 22-nm Quantum SoC. In IEEE Open Journal of the Solid-State Circuits Society, 2, pp. 103–121, 2022. <https://doi:10.1109/OJSSCS.2022.3213528>.
4. Barker, W.; Polk, W.; Souppaya, M. Getting Ready for Post-Quantum Cryptography: Exploring Challenges Associated with Adopting and Using Post-Quantum Cryptographic Algorithms. (National Institute of Standards and Technology (NIST),) NIST Cybersecurity White Paper, April 28, 2021. <https://doi.org/10.6028/NIST.CSWP.04282021>.
5. Chen, L.; Jordan, S.; Liu, Y-K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D. Report on Post-Quantum Cryptography. (National Institute of Standards and Technology (NIST)), NIST Internal Report 8105, April 2016. <https://doi.org/10.6028/NIST.IR.8105>.
6. Alagic, G.; Apon, D.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Miller, C.; Moody, D.; Peralta, R.; Perlner, R.; Robinson, A.; Smith-Tone, D.; Liu, Y-K. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. (National Institute of Standards and Technology (NIST)), NIST Internal Report 8413-upd1, Sept. 2022. <https://doi.org/10.6028/NIST.IR.8413-upd1>.
7. NIST. Special Publication 800-227: Recommendations for key-encapsulation mechanisms, 2024.
8. NIST. Module-Lattice-based Key-Encapsulation Mechanism. Federal Information Processing Standards Publication, FIPS 203. August 13, 2024. <https://doi.org/10.6028/NIST.FIPS.203.ipd>.
9. NIST. Module-Lattice-Based Digital Signature Standard. Federal Information Processing Standards Publication, FIPS 204. August 13, 2024. <https://doi.org/10.6028/NIST.FIPS.204>.
10. NIST. Stateless Hash-Based Digital Signature Standard. Federal Information Processing Standards Publication, FIPS 205. August 13, 2024. <https://doi.org/10.6028/NIST.FIPS.205>.
11. Alagic, G.; Bros, M.; Ciadoux, P.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Liu, Y.; Miller, C.; Moody, D.; Peralta, R.; Perlner, R.; Robinson, A.; Silberg, H.; Smith-Tone, D.; Waller, N. Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process. (National Institute of Standards and Technology (NIST),) NIST Internal Report (NIST IR 8545), March 2025. <https://doi.org/10.6028/NIST.IR.8545>.
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 IEEE Consumer Communications & Networking Conference (CCNC2025), Las Vegas, USA, Jan. 2025.
13. Ghosh, S.; Upadhyay, S.; Saki, A.A. A Primer on Security of Quantum Computing. arXiv 2023, arXiv:2305.02505v1.
14. Perepechaenko, M.; Kuang, R. Quantum Encryption and Decryption in IBMQ Systems using Quantum Permutation Pad. Journal of Communications 17, 12, pp. 972–978, Dec. 2022. <https://doi.org/10.12720/jcm.17.12.972-978>.

15. 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, Association for Computing Machinery, pp. 1461–1480, Oct. 2020, New York, USA. <https://doi.org/10.1145/3372297.3423350>.
16. Schwabe, P.; Stebila, D.; Wiggers, T. More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys. In: Bertino, E., Shulman, H., Waidner, M. (eds) Computer Security – ESORICS 2021. ESORICS 2021. Lecture Notes in Computer Science, 12972, Sept. 30 2021, Springer, Cham. https://doi.org/10.1007/978-3-030-88418-5_1.
17. Gunther, F.; Rastikian, S.; Towa, P.; Wiggers, T. KEMTLS with Delayed Forward Identity Protection in (Almost) a Single Round Trip. In: Ateniese, G., Venturi, D. (eds) Applied Cryptography and Network Security. ACNS 2022. Lecture Notes in Computer Science, 13269. June 18 2022, Springer, Cham. https://doi.org/10.1007/978-3-031-09234-3_13.
18. 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. <https://eprint.iacr.org/2009/208.pdf>.
19. 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. <https://eprint.iacr.org/2023/420>.
20. 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. <https://api.semanticscholar.org/CorpusID:237235623>.
21. Giron, A.A. Migrating Applications to Post-Quantum Cryptography: Beyond Algorithm Replacement. Cryptology ePrint Archive, Paper 2023/709, 2023. <https://eprint.iacr.org/2023/709>.
22. Liu, T.; Ramachandran, G.; Jurdak, R. Post-Quantum Cryptography for Internet of Things: A Survey on Performance and Optimization. arXiv. <https://doi.org/10.48550/arXiv.2401.17538>.
23. Asif, R. Post-Quantum Cryptosystems for Internet-of-Things: A Survey on Lattice-Based Algorithms. Algorithms. IoT 2021, 2, pp. 71–91. <https://doi.org/10.3390/iot2010005>.
24. Li, S.; Yuxiang, C.; Lin, C.; Jing, L.; Chanchan, K.; Kuanching, L.; Wei, L.; Naixue, X. Post-Quantum Security: Opportunities and Challenges. Sensors 23, 21: 8744, 2023. <https://doi.org/10.3390/s23218744>.
25. Balamurugan, C.; Singh, K.; Ganesan, G.; Rajarajan, M. Post-Quantum and Code-Based Cryptography—Some Prospective Research Directions. Cryptography, 4: 38, 2021. <https://doi.org/10.3390/cryptography5040038>.
26. 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. Malaysian Journal of Fundamental and Applied Sciences, Vol. 19 2023 pp. 337–344. <https://doi.org/10.11113/mjfas.v19n3.2905>.
27. 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. <https://doi.org/10.3390/fi16080300>.
28. <https://www.nist.gov/news-events/news/2022/07/pqc-standardization-process-announcing-four-candidates-be-standardized-plus>.
29. Liboqs, <https://openquantumsafe.org/liboqs/>.
30. Sharpe, R.; Warnicke, E.; Lampert, U. Wireshark User's Guide. Version 4.1.0. https://www.wireshark.org/docs/wsug_html_chunked/.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.