ExpressPQDelivery : Toward Efficient and Immediately Deployable Post-Quantum Key Delivery for Web-of-Things

Anonymous Author(s)

ABSTRACT

Post-quantum cryptography (PQC) aims to develop quantum-safe algorithms against attacks by a quantum computer. As quantum-safe algorithms require much larger keys in their operation compared to the current RSA/ECC practice, the networking latency significantly increases when executing the protocols with sending such large keys. This problem gets more challenging in the era of Web-of-Things (WoTs) with low-memory devices. To tackle the problem, we propose ExpressPQDelivery, which is, to the best of our knowledge, the first immediately deployable protocol to efficiently transport large keys. It leverages the DNS infrastructure, as DNS is close to clients, guaranteeing express key delivery with a short round-trip time (RTT). We split a large PQ key along with a server's signature and feed them into several DNS records. To show the feasibility of ExpressPQDelivery, we instantiate it with TLS 1.3 [40] and demonstrate that it reduces 27% of network latency between a server and a client on average compared to the standard TLS 1.3. We deploy ExpressPQDelivery on a low-capability board with 256 KB RAM, showing a significant high gain (34%).

CCS CONCEPTS

• **Networks** → *Cross-layer protocols*;

KEYWORDS

large key delivery, post-quantum cryptography, DNS

ACM Reference Format:

Anonymous Author(s). 2024. ExpressPQDelivery : Toward Efficient and Immediately Deployable Post-Quantum Key Delivery for Web-of-Things. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 14 pages.

https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recent advances in quantum computing pose a significant threat to the security of asymmetric cryptographic algorithms, such as RSA [23] and Diffie-Hellman (DH) [33], which are integral to Transport Layer Security (TLS) [40] and Public Key Infrastructure (PKI) [11]. This is because quantum computers can solve underlying hard problems, such as discrete logarithm problems and factoring problems, upon which the security of these algorithms relies, by using Shor's algorithm [48]. In response, the academic and industrial sectors have been actively studying quantum-safe algorithms to mitigate the aforementioned issues. NIST[1] has spearheaded efforts to standardize such algorithms through its PQC project. In 2022, NIST selected the three signature schemes DILITHIUM [16], FALCON [38], and SPHINCS+ [7] as well as the Key Encapsulation Mechanism (KEM) KYBER [9]. Since then, many governments and organizations have also announced their roadmaps for migrating to PQC [24, 57, 64].

Networking delay due to PQC. PQC requires much larger keys in operation compared to the current RSA/ECC practices; thus, migrating to PQC is challenging. As the size of PQ keys typically exceeds the maximum transmission unit (MTU), it requires multiple packets to deliver the PQ keys. There are three factors, leading to significant networking delay due to PQC, that academic and industrial communities have been identified. They are 1) the initial congestion window size [50], 2) the bandwidth [26], and 3) the packet loss rate [26, 36]. Note that these factors are more impactful when considering WoTs with low-capability devices. Due to the resource-constraints in WoT, the TCP receive buffer size is an important factor in addition to the initial congestion window size.

Our solution. We propose ExpressPQDelivery, a one-size-fits-all solution to the three networking issues above. It is an immediately deployable protocol designed to efficiently deliver large PQ keys in a subsequent protocol-independent fashion. ExpressPQDelivery uses the DNS infrastructure to deliver keys since DNS resolvers are usually near clients; thus, this protocol guarantees express delivery with a short RTT. To this end, we design an E-Box, which contains a large PQ key signed by a server, and is split into several TXT/TLSA DNS records with their unique names. One who intends to have a PQ key can get the key by sending query messages with the names of the records and assembling the received records.

Instantiation. We implement ExpressPQDelivery by applying it to TLS 1.3 [40] as TLS is the most widely deployed protocol on the Internet [30]. We show a dramatic decrease, 27% in the execution of TLS 1.3, with and without E-Box. We employ our procedure on Nucleo-F439ZI with 256 KB RAM, showing 34% of a dramatic gain.

Contributions. We make the following contributions:

- We design ExpressPQDelivery, an immediately deployable protocol that aims to deliver large PQ keys efficiently in a subsequent protocol-independent way (see §4 and §5).
- We implement ExpressPQDelivery while applying it to TLS 1.3[2] (see §6).
- We conduct a comprehensive experiment considering receive window size, bandwidth, loss rate, and network latency between the server and client, and show a dramatic decrease (27%) in the execution of TLS 1.3 (see §7).

[1]The U.S. National Institute of Standards and Technology

[2]We publicly release the source code at https://github.com/ExpressPQDelivery

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Total size of messages used in TLS

Algorithm	Size	Algorithm	Size
ECDH+ECDSA(P-256)	224	DH+RSA (2048)	1,280
KYBER512+DILITHIUM2	7,720	KYBER512+FALCON512	3,797
KYBER768+DILITHIUM3	10,810	KYBER1024+FALCON1024	7,489
KYBER1024+DILITHIUM5	14,918		

2 BACKGROUND

2.1 Post-Quantum Cryptography

PQC has been developed in response to potential threats from quantum computing against currently used public key cryptographic algorithms, such as RSA and ECC. NIST standardizes four algorithms – KYBER [9], FALCON [38], DILITHIUM [16], and SPHINCS+ [7]. As SPHINCS+ relies on large keys and consumes lots of energy not relevant for low-capability devices [5, 37, 47], we only focus on FALCON and DILITHIUM in this paper. The sizes of a public key, a private key, a ciphertext, and a signature of PQC algorithms are much longer than those of classical public key algorithms (see Table 6). For instance, the public key size of KYBER is 25 to 49 times larger than that of ECDH. Due to these large sizes, when PQC is integrated into TLS 1.3, it requires 17 to 369 times larger messages than classical public key cryptographic algorithms (see Table 1).

2.2 Web-of-Things

Web-of-Things (WoT) [60] is Internet-of-Things (IoT) that connects to the web architecture to facilitate interoperability, fragmentation, and usability. It relies on web protocols such as REST, HTTP, and URIs to communicate with each other. The primary use cases are smart cities and smart grids. These use cases rely on low-capability devices such as sensors. One such device is the M4 board series, which is widely used as cellular modules and smart meters [17, 56]. Those devices, used in previous PQC for IoT work [4, 28, 44, 54, 55], are typically equipped with small RAM ranging from 16 KB to 640 KB (196 KB on average). Note that the receive buffer size is also small in these devices. For instance, those boards use FreeRTOS, where the receive buffer size is set to 2 KB by default. Furthermore, these devices use diverse connectivity technologies like LoRa or LTE-M. These technologies work on low bandwidth. For example, LoRa operates at bandwidths between 0.3 Kbps and 50 Kbps [63], and LTE-M can support up to 1 Mbps [8].

2.3 Domain Name System

The domain name system (DNS) [34] is a large and distributed infrastructure that maintains information about domain names. A server uploads its associated attributes, such as its IP addresses or certificates, using various types of resource records (RRs) to DNS. Examples of RRs are as follows:

- **A:** It contains an IP address.
- **TXT:** It provides arbitrary base64-encoded text.
- **TLSA:** It stores a DER-formatted certificate or public key [21].

With an RR type and the domain name, a client can query specific information about the domain, and DNS responds with the corresponding records. For instance, when a client queries the name to DNS with an RR `A`, DNS responds with one or more IP addresses for the name. There is a *DNS resolver* (or a resolver), which is an interface between DNS and a client. A client asks a resolver to retrieve records on their behalf within DNS. At least 10M resolvers distributed worldwide are close to clients [1]. For the rapid retrieval service, resolvers typically cache retrieved records for a specified period. The DNS protocol originally relies on UDP for its transport. Considering UDP is unreliable, a DNS response should be delivered in one packet. As the maximum transmission unit (MTU) is usually 1.5 KB, the length of a DNS response is limited to 1.2 KB due to other headers. When a DNS response exceeds 1.2 KB, a resolver falls back to TCP for the underlying transport with a client according to its fallback mechanism. We call this mechanism *TCP fallback* [13, 32].

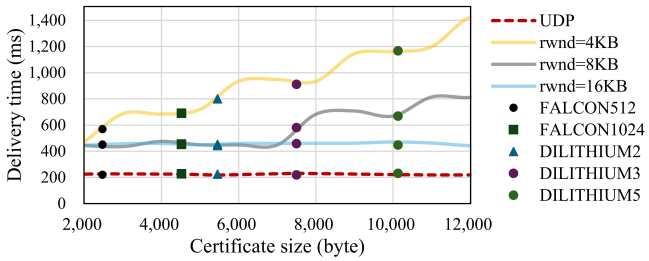

Figure 1: PQC certificate delivery time.

3 LARGE PQ KEY DELIVERY PROBLEM

This section reviews the previous benchmarking work that has identified networking issues due to the large PQ key and defines the issues in the context of WoT.

Previous works on networking issues. From a networking perspective, delivering a large PQ key is not trivial because a key cannot be delivered in a single packet, as in RSA or ECC. Considering the MTU size is 1.5 KB, the number of packets required for delivery is at least two packets for all signature algorithms except FALCON512. To analyze the impact of a large PQ key in networking, there have been PQ benchmarking studies in the general network, considering the following three aspects:

- **Initial congestion window size (`cwnd`):** It defines the number of packets that a sender can transmit at first in TCP. Christian *et al.* [20] show additional round-trips when the key size exceeds the size of the congestion window, resulting in a significant delay. Panos *et al.* [26] show that the networking latency decreases when increasing `cwnd` .
- **Bandwidth:** It specifies the number of bits that a device can receive per second. Panos *et al.* [26] demonstrate the elapsed time of the TLS handshake highly increases in LTE-M [8].
- **Packet loss rate:** It refers to the percentage of packets that fail to reach the destination due to events such as network congestion. The packet loss results in a significant delay due to retransmissions. Several studies [26, 36] report that the elapsed time is more sensitive to the packet loss rate if the TLS handshake is executed with the larger PQ key.

Consideration of WoT. The aforementioned problems become much more challenging in the era of WoT with low-capability devices. First, we find that the network delay time due to the small TCP receive buffer (`rwnd`) of the low-capability device is greater than the delay time due to the `cwnd` . Our experiment measuring

Table 2: Average ping time test.

Location	Top 1m Server	DNS	
		Google (8.8.8.8)	Cloudflare (1.1.1.1)
North America	50.26 ms	1.23 ms	0.95 ms
Western Europe	51.61 ms	0.83 ms	0.64 ms
East Asia	103.42 ms	38.14 ms	10.10 ms
Oceania	108.63 ms	1.72 ms	1.68 ms

the key delivery time varying the receive buffer size shows the delivery time increases as a step graph with regard to the number of bytes sent by a server in TCP (see Figure 1). The increment in TCP is mainly due to the TCP window size and the ACK mechanism. We detail the experiment to analyze the networking delay due to transport layer services in Appendix D. Second, there are many low-bandwidth connectivity technologies (e.g., LoRa, LTE-M) used in WoT, which have bandwidths below 1 Mbps (see §2.2) compared with the conventional network. Finally, WoT devices tend to be susceptible to the packet loss rate as mobile and wireless networking is prevalent [12].

4 OVERVIEW

This section presents the overview of ExpressPQDelivery.

4.1 Design Goals and Threat Model

Design goals. To address the large PQ key delivery problem, we consider the following goals:

- **(G1) Efficiency:** The solution should not incur high networking latency. It should not introduce additional round-trips for delivering PQ keys.
- **(G2) Universality:** The solution should be independent of specific protocols; thus, it can be applied to diverse protocols.
- **(G3) Immediate deployability:** The solution should be compatible with the current practice without requiring any change in the existing infrastructure.

Threat model. We assume the Dolev-Yao attack model [15] in which an attacker can view, record, insert, block, and modify all messages on public network channels. Therefore, an attacker can perform DNS poisoning attacks to insert arbitrary messages in DNS caches [61]. However, an attacker is computationally bounded; thus, the attacker cannot break cryptographic assumptions. For instance, the attacker cannot decrypt encrypted messages without corresponding keys. Furthermore, we do not consider attacks that disrupt the DNS service, such as denial of service attacks [3]; that is, we assume DNS always properly works.

4.2 Solution Overview

We design ExpressPQDelivery that leverages the DNS infrastructure, achieving the aforementioned goals (see Figure 2). When using DNS, we encounter two challenges that are detailed below.

Leverage of the DNS infrastructure. To achieve design goals, we utilize the properties of the DNS infrastructure:

- **(P1) Distributed worldwide:** As DNS resolvers that cache the DNS records are distributed globally closer to a client than a server, the DNS resolvers can deliver PQ keys efficiently. We measure the round-trip times to the top 1 million sites listed in Cisco Umbrella [58] and the popular public DNS resolvers – Cloudflare DNS and Google DNS – from four continents (i.e., North America, Western Europe, East Asia, and Oceania) and show that the round-trip times to DNS are about 3 to 60 times faster than the average round-trip times to the top 1 million servers (see Table 2).
- **(P2) Independence:** The DNS mechanism is independently used; thus, delivering PQ keys through DNS guarantees universality. In general, a client uses the DNS mechanism to look up IP addresses or other information about the domain before executing the specific protocols to establish connections with a server.
- **(P3) Flexible RRs:** DNS provides the records for keys (TLSA) and texts (TXT); thus, we can simply use such records to immediately deploy our solution without requiring any modifications to the existing infrastructure. Furthermore, we can negotiate the use of the solution according to the values in such RRs and introduce the fallback mechanism to guarantee compatibility with the current practice.

Challenges. To benefit from DNS in delivering E-Box , we should address the following two challenges:

- **(C1) TCP fallback:** A PQ key generally exceeds the size limit of a single DNS packet, which is 1,232 bytes. Therefore, when we deliver a PQ key via DNS record, the DNS mechanism falls back to TCP instead of UDP as its transport according to its TCP fallback mechanism [13, 32]. The network latency increases significantly in TCP as the PQ key size grows. Furthermore, 11% of DNS resolvers do not support TCP [35]. Therefore, it is necessary to fragment the PQ key within the size limit to avoid using TCP and to support our solution. However, this fragmentation is challenging since only one record of the same type can exist under a single domain name.
- **(C2) Unreliable connection:** The mechanism to guarantee reliable key delivery should be considered if we want to deliver the PQ key over UDP, as UDP is based on the best-effort delivery, not providing reliability. Note that the solution becomes unavailable even if a single UDP packet is lost. Therefore, we should consider ways to ensure the reliability of the key delivery. In other words, it is necessary for clients to deploy a mechanism to re-deliver a fragmented key when it turns out to be lost.

Our solution. ExpressPQDelivery introduces E-Box to facilitate the delivery of a server's PQ key to clients using existing RRs. E-Box not only contains a server's PQ key but also includes attributes for operational purposes such as the version and protocol identifier. E-Box can also contain additional public keys required to execute the subsequent protocol. E-Box is split into multiple existing DNS records to guarantee immediate deployability, ensuring that each does not exceed the limit size of a DNS packet. We call each record the *E-Box fragment* that is assigned its own name under the domain. ExpressPQDelivery transmits E-Box through DNS, addressing two aforementioned challenges. To address (C1), ExpressPQDelivery provides the (S1) *optimized fragmentation* and the (S2) *query-based delivery*. To address (C2), ExpressPQDelivery provides the (S3) *re-delivery mechanism*, described as follows:

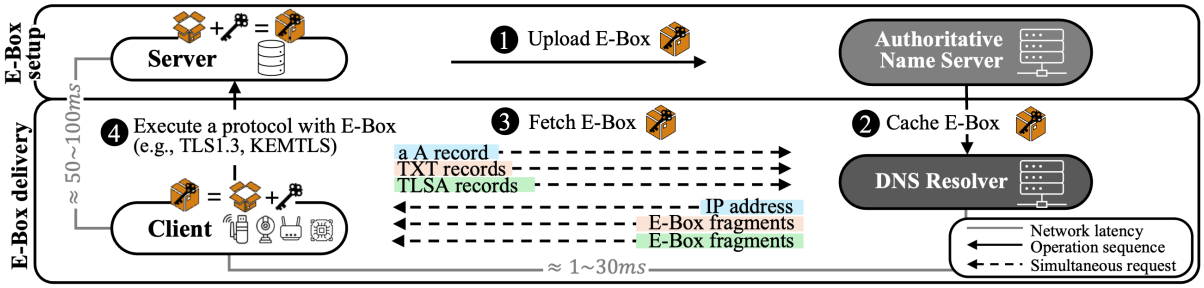

Figure 2: ExpressPQDelivery procedure.

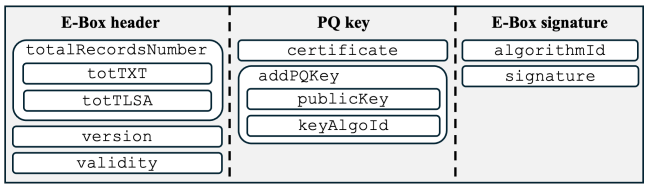

Figure 3: High-level design structure of E-Box

- **(S1) Optimized fragmentation:** We use TLSA and TXT records to encode E-Box. As TLSA that is for a certificate is more memory-efficient than TXT, we use TLSA for a PQ key and TXT for others.
- **(S2) Query-based delivery:** A client finally fetches E-Box by querying E-Box fragments with their names, assembling them, and verifying the signature.
- **(S3) Re-delivery mechanism:** It enables a client to re-query for the packets that a client does not receive for a timeout.

The procedure of ExpressPQDelivery, which consists of the E-Box *setup* and *delivery* phases, is depicted in Figure 2.

Phase 1. E-Box setup. A server creates an E-Box and uploads it in advance before communications between a client and a server.

- **Upload E-Box (❶):** A server packages its large PQ key(s) required for the subsequent protocol it supports and uploads an E-Box to the DNS infrastructure.

Phase 2. E-Box delivery. Delivering an E-Box to a client follows the procedure below:

- **Cache E-Box (❷):** A DNS resolver caches a server's E-Box when it is requested by a client.
- **Fetch E-Box (❸):** A client simultaneously queries an IP address and an E-Box with `A`, `TXT`, and `TLSA` records.
- **Execute a protocol (❹):** A client executes a subsequent protocol with a server using E-Box. Note that a server does not send its large PQ key during the protocol execution.

5 DETAIL OF EXPRESSPQDELIVERY

This section provides the details of ExpressPQDelivery. We present attributes of E-Box, followed by mechanisms to deliver an E-Box.

5.1 E-Box

A server generates an E-Box with its cryptographic information. It consists of three main fields as follows (see Figure 3):

E-Box header. This field contains information about the E-Box to facilitate the operation of ExpressPQDelivery as follows:

- `version`: the version of ExpressPQDelivery.
- `validity`: the validity period of an E-Box .
- `totalRecordsNumber` : the number of TLSA/TXT fragments.

PQ key. This field is the main field of E-Box used to contain one or multiple PQ keys. It has two subfields:

- `certificate`: a certificate of a PQ key used to sign an E-Box.
- `addPQKey` (optional): an additional PQ key with its algorithm.

Note that a PQ key to verify a signature must be encoded in E-Box in the form of a certificate, which specifies the subject name of a key holder. Otherwise, there is no guarantee that a signature is generated by a server.

E-Box signature. This field contains the server's signature over E-Box with the following two fields:

- `algorithmId`: a signature algorithm.
- `signature`: a signature

One can verify the signature using a public key from the certificate in the PQ key field, which validates the integrity and the generator (i.e., a server) of the E-Box.

5.2 Optimized Fragmentation

When dividing E-Box into multiple records, a server strategically utilizes TXT/TLSA records because a TXT record can contain any value in the form of the text, i.e., providing flexibility, and a TLSA record can have a PQ key in a compact way, i.e., optimizing key encoding. Therefore, we use a TLSA record to encode the values of the `certificate` and the `publicKey` , while the values in the other fields are encoded as a TXT record.

5.3 Delivery Mechanism

The E-Box fragments are delivered through the DNS mechanism. Below, we describe 1) how we assign names for E-Box fragments and 2) how we request E-Box fragments.

Naming convention for E-Box fragments. We assign a name to each fragment so that a client can get E-Box by requesting the fragments using their names and finally assembling them. Since a

single record of the same type can only exist under one name, we establish a naming convention to identify each fragment:

`ebox-{fragmentNumber}.{domainName}`, where the fragment number begins at 0. For instance, when a server `example.com` has a DILITHIUM3 PQ key, the size of E-Box is 11,755 bytes encoded as 5 TXT records and 6 TLSA records (see Table 3). Therefore, a server should prepare for 6 names (`ebox-0`, ... , `ebox-5`) to assign a name to each record. Then, a server assigns `ebox-0` to the first TXT record and the first TLSA record, `ebox-1` to the second TXT record and the second TLSA record, and similarly to other records one by one. Note that there is no TXT record with regard to the name `ebox-5` because the E-Box contains only 5 TXT records.

Query-based delivery. With the naming convention to indicate a specific E-Box fragment, a client should be able to fetch an E-Box through the DNS query-response mechanism. However, a client is unable to determine the signature algorithm a server uses to generate the E-Box or the number of fragments it has in advance. Therefore, our delivery mechanism consists of two round-trips to DNS. In detail, a client queries a TXT record of `ebox-0` (e.g., `ebox-0.example.com`) from which A client gets the total number of TLSA/TXT records. Then, the client simultaneously sends `totTXT` number of TXT queries and `totTLSA` number of TLSA queries to receive all the E-Box fragments. Finally, a client assembles the received fragments into the original E-Box. Recall that the RTT between a client and a DNS resolver is typically much shorter than the RTT between a client and a server; therefore, 2RTT between a client and a DNS resolver would be marginal even compared to RTT between a client and a server.

Backward compatibility. ExpressPQDelivery is backwards compatible. That is, a client has no problem with a server that does not support ExpressPQDelivery. A client receives `NXDOMAIN` instead of E-Box fragments when a client sends query messages. Then, a client can execute the standard protocol with a server. Therefore, ExpressPQDelivery does not cause side effects to existing servers; thus, ExpressPQDelivery is immediately deployable.

5.4 Re-delivery Mechanism

To guarantee reliability in delivering a PQ key, we devise the re-delivery mechanism. We define *re-delivery timeout* as the time interval to send the repeated DNS query message. We set the timeout to be double the round-trip time for the first received E-Box fragment. When the timeout occurs, it triggers a client to send query messages that correspond to unreceived E-Box fragments (see algorithm 1).

Algorithm 1: Re-delivery mechanism

```
Input: domainName, fragmentNumber, totTXT or totTLSA , timeout
Output: E-Box
set timeout = 2 × (duration for the first response)
int i = fragmentNumber
if i < totTXT then
    query TXT of ebox-[i].domainName
    waitFor(timeout )
    if TXT lookup failed then
        query again for 2 times
if i < totTLSA then
    query TLSA of ebox-[i].domainName
    waitFor(timeout)
    if TLSA lookup failed then
        query again for 2 times
It functions for all non-responsive queries.
```

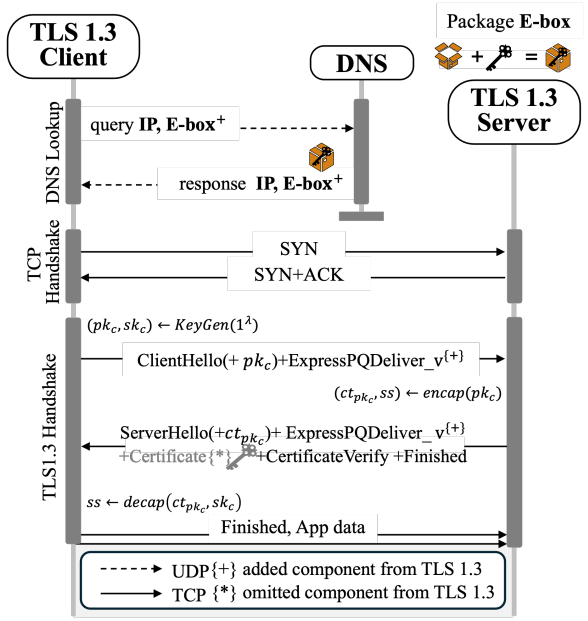

Figure 4: ExpressPQDelivery-applied PQC-TLS 1.3.

6 APPLICATION

TLS is the *de facto* standard security protocol in current practice [30] and has been widely studied across various environments [37, 50, 51, 54] for PQC. Thus, this section presents how ExpressPQDelivery can be integrated with the PQC version of the TLS 1.3 protocol (PQC-TLS 1.3) [10, 54, 55].

ExpressPQDelivery-applied PQC-TLS 1.3. ExpressPQDelivery is integrated with PQC-TLS 1.3 in two parts: the 1) E-Box setup and the 2) E-Box delivery, as depicted in Figure 4.

1) E-Box setup: As a PQC-TLS 1.3 server uses its digital signature algorithm (DSA) certificate, the server stores its DSA certificate in the PQ key field of the E-Box.

2) E-Box delivery: Two phases are different compared to PQC-TLS 1.3. One is the DNS lookup phase, and the other is `ServerHello`:

- **DNS lookup:** A client fetches the first E-Box fragment and simultaneously queries `TLSA`, `TXT`, and `A` records after finding the numbers of records from the first fragment. If the client receives `NXDOMAIN` or fails to verify the signature in E-Box, it falls back to the PQC-TLS 1.3 protocol.

- `ClientHello`: A client generates an ephemeral key encapsulation mechanism (KEM) key pair, i.e., (pk_c, sk_c) and sends the `ClientHello` message with pk_c and other parameters.
- **`ServerHello`:** When a server sends `ServerHello` i.e., ct_{pk_c}, the server sends its heavy PQ certificate along with the `ServerHello`, `CertificateVerify`, and `Finished`. When using ExpressPQDelivery, the server does not need to send the server's certificate because it has already been delivered to the client via the PQ key field of an E-Box.
- `Finished`: The client verifies the `CertificateVerify` using the server's public key from the certificate, then decapsulates shared secret ss from the ct_{pk_c} with its private key sk_c. From the ss, the client derives the session key, then verifies the finished message.

Universality. As discussed, ExpressPQDelivery is designed to provide universality. ExpressPQDelivery can be applied to all the subsequent protocols that require large PQ keys. We describe how ExpressPQDelivery can be integrated with DTLS 1.3 [42] and KEMTLS [46] in Appendix E.

7 EVALUATION

7.1 E-Box Analysis

We use five signature algorithms in experiments – DILITHIUM (2, 3, and 5) and FALCON (512 and 1024). Each E-Box contains a DSA certificate (`cert`) in its PQ key field with a signature in the E-Box signature field. In practice, a server sends a certificate chain that includes a leaf `cert`, one or two intermediate `certs` (ICAs) [43], and optionally a root `cert` [14]. Therefore, we calculate E-Box sizes based on four different cases per signature algorithm: delivering 1) only one single leaf `cert` (a `cert`), 2) a chain of a leaf `cert` with 1 ICA `cert` (1 ICA), and 3) a chain of a leaf `cert` with 2 ICA `certs` (2 ICA). So, there are 15 cases in total.

Comparison in delivery size. We compare the size of E-Box and the total number of queries required to fetch the E-Box when using only `TXT` records and when using the optimized fragmentation (see Table 3). The size difference between the E-Box and PQ key varies by algorithm and scenario, ranging from 1.4 KB (a FALCON512 `cert`) to 13.2 KB (2 ICA DILITHIUM5 `certs`). We calculate the *inflation rate* as:

$$\frac{(Size\ of\ EBox - Size\ of\ a\ PQkey)}{Size\ of\ a\ PQkey}$$

We find that the inflation rate is 78% on average in 15 cases. These overheads are necessary as due to additional information in E-Box.

Effect of the optimized fragmentation. To measure the effect, we define the *reduction rate* as

$$\frac{(\#\ of\ TXT{-}only\ records) - (\#\ of\ optimized\ records)}{\#\ of\ TXT{-}only\ records}$$

and calculate reduction rates for all the scenarios. The reduction rates range from 8.3% to 29.7% except in one case – a `cert` based on FALCON 1024. The highest reduction rate, 29.7%, is at the scenario of a certificate chain with one leaf certificate and two ICA certificates based on DILITHIUM5. The reason for the case where reduction rate is 0% is attributed to the size of PQ key being close to the multiple of the maximum size of the DNS records. In this case, the reduction amounts fall within the DNS record's size, preventing a decrease in record numbers. Note that other than these particular scenarios, we find that the optimized fragmentation highly contributes to the size reduction.

Table 3: Size of E-Box on the PQC digital signature algorithms

DSA	Scenario	TXT-only	Optimized fragmentation		
		totTXT	Total (optimize rate)	totTXT	totTLSA
DILITHIUM2	a cert	8	7 (12.5% ↓)	3	4
	1 ICA	13	11 (15.4% ↓)		8
	2 ICAs	18	14 (22.2% ↓)		11
DILITHIUM3	a cert	12	11 (8.30% ↓)	5	6
	1 ICA	18	16 (11.1% ↓)		11
	2 ICAs	25	21 (16.0% ↓)		16
DILITHIUM5	a cert	16	13 (18.8% ↓)	6	7
	1 ICA	27	19 (29.6% ↓)		13
	2 ICAs	37	26 (29.7% ↓)		20
FALCON512	a cert	4	3 (25.0% ↓)	1	2
	1 ICA	6	5 (16.7% ↓)		4
	2 ICAs	8	6 (25.0% ↓)		5
FALCON1024	a cert	6	6 (0.00% ↓)	2	4
	1 ICA	10	9 (10.0% ↓)		7
	2 ICAs	14	12 (12.5% ↓)		10

We note that the effect of optimized fragmentation is more significant as the key size increases. For instance, the decreased number of records is generally 3 or less for E-Box with a `cert` (i.e., case 1), but in other cases, the difference increases up to 11 queries. This shows that optimized fragmentation can help efficiently deliver higher-security level PQ keys.

7.2 Experimental Setting

To conduct a comprehensive evaluation of ExpressPQDelivery, we consider three factors: 1) the receive window size (`rwnd`), 2) the bandwidth, and 3) the packet loss rate. For each factor, we vary the RTTs between a client and a server by an intra-country scenario ($\approx$ 40ms), an inter-country scenario ($\approx$ 77ms), and an inter-region scenario ($\approx$ 180ms). We set the RTT between a client and a DNS resolver to be $<$ 10ms. For a server and a DNS resolver, we use an AWS Ubuntu 22.04 instance with Intel Xeon E5 CPU (2.30GHz) and 1 GB RAM. For a client, we consider two types of devices. To evaluate the factor of `rwnd`, we implement an ExpressPQDelivery client on a Nucleo-F439ZI [52] board with ARM Cortex-M4 and 256 KB RAM. We also implement a client on an AWS instance for the factors of the bandwidth and the packet loss rate. We set up an authoritative name server and a DNS resolver using `bind9`. We implement a prototype of the ExpressPQDelivery library by extending `WolfSSL-4.7.0` [55] for a Nucleo-F439ZI board and `OpenSSL-3.3.0` in other environments, leveraging `oqs-provider-0.5.4` [45] for all the PQC algorithms.

We measure the time required to execute the PQC-TLS handshake on a client considering the following four periods:

- **Period 1:** the elapsed time required for a DNS lookup time – from querying all the records to completely getting an E-Box .
- **Period 2:** the elapsed time right after the end of Period 1 until the time right after sending `ClientHello`.
- **Period 3:** the elapsed time right after the end of Period 2 until the time right after receiving a server's `CertificateVerify` message. In PQC-TLS, a client receives a certificate at this period.

- **Period 4:** the elapsed time right after the end of Period 3 until the time when the PQC-TLS handshake is finished.

E-Box gain. We focus on Period 1 and Period 3, where the effect of E-Box is applied, and define the *E-Box gain* as:

$$(1 - \frac{Period1_{EBox}+Period3_{EBox}}{Period1+Period3}) \times 100\%,$$

where Period 1 and Period 3 are the times for PQC-TLS without E-Box, while $Period1_{EBox}$ and $Period3_{EBox}$ indicate the times related to the ExpressPQDelivery.

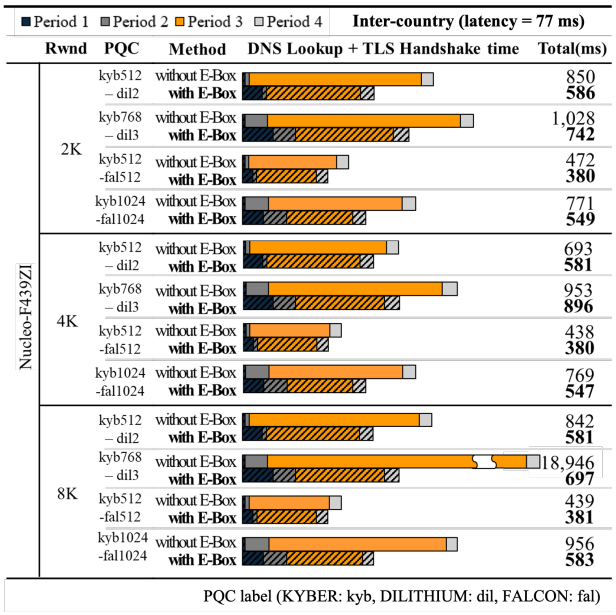

Figure 5: Evaluation results at NUCLEO-F439ZI.

7.3 Experiment Results

We describe only the results for the inter-country scenario. Other results, showing the similar tendency, are provided in Appendix F.

Effect on `rwnd`. Our experiment results on the board show that the handshake time with E-Box significantly decreases compared to the handshake time without E-Box (see Figure 5). With ExpressPQDelivery, the handshake time for all the scenarios decreases by 27% on average. ExpressPQDelivery reduces 25.3%, 19.7%, and 44.5% of the handshake time on the board with 2 KB, 4 KB, and 8 KB of `rwnd`, respectively. In detail, it takes 13 ms on average for a client to query an `A` record. An ExpressPQDelivery client queries a different number of records for each algorithm (kyb512-dil2, kyb768-dil3, kyb512-fal512, kyb1024-fal1024) in Period 1 for 88 ms, 136 ms, 47 ms, and 92 ms, respectively. In Period 3, the ExpressPQDelivery client is from 25.85% (`rwnd`=4 KB, kyb512-fal512) to 97.90% (`rwnd`=8 KB, kyb768-dil3) faster than the PQC-TLS client. Both clients show similar tendency in Periods 2 and 4.

We calculate the E-Box gains for algorithms (see Figure 6). When `rwnd` is increased (2 KB→4 KB), the handshake time is reduced as a server can send more bytes at the same time. However, for all algorithms except kyb512-fal512, the E-Box gains increase as `rwnd` increases from 4 KB to 8 KB. We find that many packets are dropped because the processing speed is slower than the PQ key

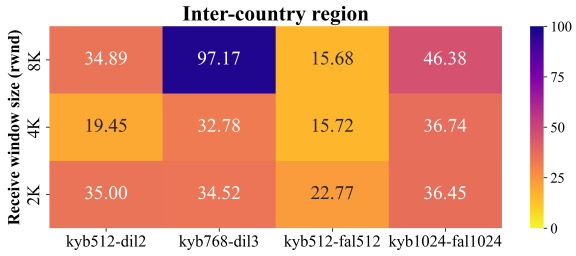

Figure 6: Gains from applying ExpressPQDelivery on PQC-TLS.

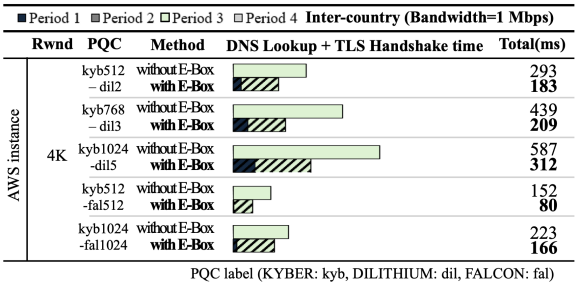

Figure 7: Evaluation results for 1 Mbps bandwidth.

transmission speed, resulting in many packet retransmissions. This leads to a significant elapsed time in Period 3. When ExpressPQDelivery is applied, the PQ key is delivered from the nearby DNS through the E-Box, reducing the number of retransmitted packets in $Period3_{EBox}$. On the other hand, in kyb512-fal512, the size of the PQ key is only 3.7 KB (see Table 1), which is smaller than 4 KB. Therefore, increasing the `rwnd` from 4 KB to 8 KB still results in a similar gain as when it was 4 KB. The average gain we calculated for all scenarios is 34%.

Effect on low-bandwidth. We evaluate the effect of ExpressPQDelivery on the low-bandwidth. We select a bandwidth of 1 Mbps between the client and server, considering LTE-M. We set the `rwnd` to 4 KB. Our results show that ExpressPQDelivery effectively reduces the time for all PQC algorithms in establishing handshake connections (see Figure 7). The E-Box gains are 37% (kyb512-dil2), 52% (kyb768-dil3), 47% (kyb1024-dil5), 48% (kyb512-fal512), and 26% (kyb1024-fal1024), with an average of 42%.

Effect on packet loss rate. We set the loss rate to 5%, considering the unstable network conditions. We fix the `rwnd` size at 4 KB. The bandwidth between the client and server in our environment was ≈ 200 Mbps. Our experimental results show that ExpressPQDelivery is effective even in unstable network environments (see Figure 8). The E-Box gains are 51% (kyb512-dil2), 57% (kyb768-dil3), 60% (kyb1024-dil5), 36% (kyb512-fal512), and 26% (kyb1024-fal1024), with an average of 46%.

7.4 Overheads on DNS

We measure the overhead on DNS in terms of its disk storage. During the execution of ExpressPQDelivery, DNS resolvers should send an `A` record, `TXT`, and `TLSA` records. To fully benefit from ExpressPQDelivery, those records should be cached in DNS resolvers, which require additional storage. We expect how much is required to guarantee 90% of a cache hit. When a DNS resolver caches the E-Box of a single domain, the required extra storage

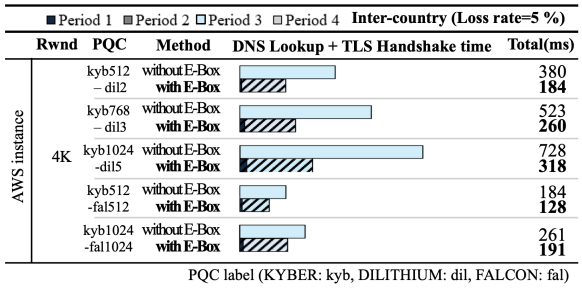

Figure 8: Evaluation results for 5% loss rate.

Table 4: E-Box size according to the number of domains.

# of domains		1	10K	100K
E-Box size	a cert	2.2 KB – 11.7 KB	21.4 MB – 114.2 MB	0.21 GB – 1.12GB (avg: 0.63 GB)
	cert chain	4.5 KB – 22.5KB	43.9 MB – 219.7 MB	0.43 GB – 2.15GB (avg: 1.22 GB)

is between 2.2 KB (FALCON512) and 11.7 KB (DILITHIUM5) (see Table 4). Considering the average length of a certificate chain is 2.5 [43], 4.5 KB (FALCON512) and 22.5 KB (DILITHIUM5) are additionally needed per domain. As it is commonly accepted that the popularity of websites follows the power-law distribution [25, 62], 90/10 law can be applied; thus, 90% of cache hit on Top-1M websites would be guaranteed with 100K domains, which requires around 1.22 GB. As the price of the disk storage and memory keeps falling, now solid-state drives are below $0.04 per GB, hard disk drives are below $0.02 per GB, and DRAM memory is below $1.30 per GB); it would not be a significant burden to cover the increment of the cache size. To increase the cache hit rate, we can also use prefetch techniques [53].

7.5 Security Analysis

DNS record manipulation attack. An attacker can modify the total number of TLSA and TXT records included in the E-Box fragment. For instance, in the case of a server using a single DILITHIUM2 certificate, the total numbers of TXT and TLSA records are 3 and 4, respectively. By manipulating these numbers, an attacker can force the client to send additional queries to the DNS, which may disrupt the DNS service. Although the impact is limited to a denial-of-service, which is an out-of-scope attack, we can set the upper bound of the number of packets to be simultaneously sent to mitigate such an attack. For instance, We can limit the total number of records to 20, the maximum number of records per RR in Table 3.

DNS poisoning attack. An attacker may tamper with and forge the E-Box that DNS has cached, causing a client to receive an incorrect E-Box. However, the E-Box contains the server's certificate and signature over the E-Box. The certificate guarantees that the server is the domain owner, and the signature ensures the integrity of the E-Box. If the client detects tampering with the E-Box by verifying the signature using the public key from the certificate, it falls back to the original protocol.

E-Box confidentiality. The data transmitted through the E-Box consists only of the server's public information (i.e., certificate and public key), which does not require confidentiality. Previous studies, such as ZTLS [31] and DANE [21], have already explored delivering a certificate and a public key by leveraging DNS. Note that DANE is particularly common in email systems [29].

8 RELATED WORK

Solution to the large PQ key delivery problem. We have discussed the analysis of the problem in §3. There have been two types of approaches to address the problem.

- **Approach 1) reducing the number of certificates in a chain:** Sikeridis *et al.* [49] propose an *intermediate certificate suppression* method where a client indicates a list of ICA certificates that a client already has, and a server removes ICA certificates specified from a chain and sends the remaining certificates. Kampanakis *et al.* [27] provide efficient caching mechanisms to make the ICA suppression methods feasible and boost their performance.
- **Approach 2) using a shorter algorithm:** Schwabe *et al.* [46] propose KEMTLS that substitutes KEMs for handshake signatures. The rationale behind this replacement is that the size of KEM is much shorter than that of digital signatures; thus, it contributes to addressing the problem by reducing the key size by an algorithm.
- **Our approach:** ExpressPQDelivery is the fourth approach to address the problem by reducing the physical distance for the key delivery by using DNS, also considering resource-constrained WoTs. The approaches are not mutually excluded; rather, they are independent and complementary.

Enhancing features using DNS. There have been techniques [21, 31, 41] that leverage DNS to introduce new features. TLS Encrypted Client Hello (ECH) [41] aims to protect the privacy of a client by encrypting the client's first TLS message containing a destination name. DANE [21] is designed to handle a public key from a domain owner through DNS. ZTLS [31] aims to reduce one round-trip of the TLS handshake by uploading a server's first handshake message on DNS. Note that no approach is presented to address the large PQ key delivery problem using DNS. ExpressPQDelivery contributes to extending the areas of this branch of work to PQC.

Splitting large material. TurboTLS [2] initiates TLS during the TCP handshake by exchanging the Hello messages split into several UDP packets. Goertzen *et al.* [18] propose a new RR, called RRFRAG, which contains split PQC signatures used in DNSSEC. This work, however, only focuses on addressing the problem in the context of DNSSEC by introducing a completely new RR without considering deployability. Twardokus *et al.* [59] sends the large certificates on multiple DSRC (Dedicated short-range communications) packets in V2V communications.

9 CONCLUSION

We propose ExpressPQDelivery, an immediately deployable, efficient, and universal protocol to deliver large PQ keys, leveraging DNS. We design E-Box that contains large PQ keys in multiple TXT/TLSA records. We show that the protocol reduces the latency by an average of 27% on ExpressPQDelivery-applied PQC-TLS 1.3. In the future, we plan to optimize the key delivery from the client side, considering the scenario of mutual authentication.

REFERENCES

[1] Joe Abley. 2012. Ten Million DNS Resolvers on the Internet. https://www.icann.org/en/blogs/details/ten-million-dns-resolvers-on-the-internet-22-3-2012-en. (2012).

[2] Carlos Aguilar-Melchor, Thomas Bailleux, Jason Goertzen, David Joseph, and Douglas Stebila. 2023. TurboTLS: TLS connection establishment with 1 less round trip. *arXiv preprint arXiv:2302.05311* (2023).

[3] Marios Anagnostopoulos, Georgios Kambourakis, Panagiotis Kopanos, Georgios Louloudakis, and Stefanos Gritzalis. 2013. DNS amplification attack revisited. *Computers & Security* 39 (2013), 475–485.

[4] Mila Anastasova, Reza Azarderakhsh, and Mehran Mozaffari Kermani. 2021. Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4. *IEEE Transactions on Circuits and Systems I: Regular Papers* 68, 10 (2021), 4129–4141.

[5] Tanushree Banerjee and M Anwar Hasan. 2018. Energy consumption of candidate algorithms for NIST PQC standards. *University of Waterloo Centre for Applied Cryptographic Research, Tech. Rep* (2018).

[6] Richard Barnes, Subodh Iyengar, Nick Sullivan, and Eric Rescorla. 2023. RFC 9345: Delegated Credentials for TLS and DTLS. (2023).

[7] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. 2019. The SPHINCS+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2129–2146.

[8] Suresh R Borkar. 2020. Long-term evolution for machines (LTE-M). In *LPWAN technologies for IoT and M2M applications*. Elsevier, 145–166.

[9] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 353–367.

[10] Deirdre Connolly. 2024. *ML-KEM Post-Quantum Key Agreement for TLS 1.3.* Internet-Draft draft-connolly-tls-mlkem-key-agreement-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-connolly-tls-mlkem-key-agreement/01/ Work in Progress.

[11] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and William Polk. 2008. *Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile*. Technical Report.

[12] Samir Dawaliby, Abbas Bradai, and Yannis Pousset. 2016. In depth performance evaluation of LTE-M for M2M communications. In *2016 IEEE 12th international conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 1–8.

[13] John Dickinson, Sara Dickinson, Ray Bellis, Allison Mankin, and Duane Wessels. 2016. RFC 7766: DNS transport over TCP-implementation requirements. (2016).

[14] Tim Dierks and Eric Rescorla. 2008. RFC 5246: The transport layer security (TLS) protocol version 1.2. (2008).

[15] Danny Dolev and Andrew C. Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.

[16] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 238–268.

[17] Würth Elektronik. 2024. Adrastea-I. (2024). https://www.we-online.com/en/components/products/ADRASTEA-I Retrieved 14-10-2024.

[18] Jason Goertzen and Douglas Stebila. 2023. Post-quantum signatures in DNSSEC via request-based fragmentation. In *International Conference on Post-Quantum Cryptography*. Springer, 535–564.

[19] Tatsuya Hagiwara, Hiroshi Majima, Takahiro Matsuda, and Miki Yamamoto. 2001. Impact of round trip delay self-similarity on TCP performance. In *Proceedings Tenth International Conference on Computer Communications and Networks (Cat. No. 01EX495)*. IEEE, 166–171.

[20] Johanna Henrich, Andreas Heinemann, Alex Wiesmaier, and Nicolai Schmitt. 2023. Performance Impact of PQC KEMs on TLS 1.3 Under Varying Network Characteristics. In *International Conference on Information Security*. Springer, 267–287.

[21] Paul Hoffman and Jakob Schlyter. 2012. *The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA*. Technical Report.

[22] Jana Iyengar and Martin Thomson. 2021. RFC 9000: QUIC: A UDP-based multiplexed and secure transport. *Omtermet Emgomeeromg Task Force* (2021).

[23] Jakob Jonsson and Burt Kaliski. 2003. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447. (Feb. 2003). https://doi.org/10.17487/RFC3447

[24] JOSEPH R. BIDEN JR. 2022. National Security Memorandum on Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems. https://www.whitehouse.gov/briefing-room/statements-releases/2022/05/04/national-security-memorandum-on-promoting-united-states-leadership-in-quantum-computing-while-mitigating-risks-to-vulnerable-cryptographic-systems/. (2022).

[25] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. 2001. DNS performance and the effectiveness of caching. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. 153–167.

[26] Panos Kampanakis and Will Childs-Klein. 2024. The impact of data-heavy, post-quantum TLS 1.3 on the Time-To-Last-Byte of real-world connections. *Cryptology ePrint Archive* (2024).

[27] Panos Kampanakis and Michael Kallitsis. 2022. Faster post-quantum TLS handshakes without intermediate CA certificates. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*. Springer, 337–355.

[28] Matthias J Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. 2019. pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. (2019).

[29] Hyeonmin Lee, Md Ishtiaq Ashiq, Moritz Müller, Roland van Rijswijk-Deij, Taejoong Chung, et al. 2022. Under the hood of {DANE} mismanagement in {SMTP}. In *31st USENIX Security Symposium (USENIX Security 22)*. 1–16.

[30] Hyunwoo Lee, Doowon Kim, and Yonghwi Kwon. 2021. TLS 1.3 in practice: How TLS 1.3 contributes to the internet. In *Proceedings of the Web Conference 2021*. 70–79.

[31] Sangwon Lim, Hyeonmin Lee, Hyunsoo Kim, Hyunwoo Lee, and Taekyoung Kwon. 2023. ZTLS: A DNS-based Approach to Zero Round Trip Delay in TLS handshake. In *Proceedings of the ACM Web Conference 2023*. 2360–2370.

[32] Jiarun Mao, Michael Rabinovich, and Kyle Schomp. 2022. Assessing Support for DNS-over-TCP in the Wild. In *International Conference on Passive and Active Network Measurement*. Springer, 487–517.

[33] Ueli M Maurer and Stefan Wolf. 2000. The diffie–hellman protocol. *Designs, Codes and Cryptography* 19, 2 (2000), 147–171.

[34] Paul V Mockapetris. 1987. RFC1034: Domain names-concepts and facilities. (1987).

[35] Moritz Müller, Jins de Jong, Maran van Heesch, Benno Overeinder, and Roland van Rijswijk-Deij. 2020. Retrofitting post-quantum cryptography in internet protocols: a case study of DNSSEC. *SIGCOMM Comput. Commun. Rev.* 50, 4 (oct 2020), 49–57. https://doi.org/10.1145/3431832.3431838

[36] Christian Paquin, Douglas Stebila, and Goutam Tamvada. 2020. Benchmarking post-quantum cryptography in TLS. In *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*. Springer, 72–91.

[37] Sebastian Paul, Yulia Kuzovkova, Norman Lahr, and Ruben Niederhagen. 2022. Mixed certificate chains for the transition to post-quantum authentication in TLS 1.3. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 727–740.

[38] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2022. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. https://csrc.nist.gov/projects/post-quantum-cryptography/selected-algorithms-2022. (2022).

[39] CCITT Recommendation. 1988. Specification of abstract syntax notation one (ASN. 1). *Specification of Abstract Syntax Notation One (ASN. 1)* (1988).

[40] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. (Aug. 2018). https://doi.org/10.17487/RFC8446

[41] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. 2022. *TLS Encrypted Client Hello*. Internet-Draft draft-ietf-tls-esni-14. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-14 Work in Progress.

[42] E Rescorla, H Tschofenig, and N Modadugu. 2022. RFC 9147: The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. (2022).

[43] Marcus Ringström and Anton Olivestam. 2022. Analysis of Longitudinal Changes of Certificate Chains. (2022).

[44] Markku-Juhani O Saarinen. 2020. Mobile energy requirements of the upcoming NIST post-quantum cryptography standards. In *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, 23–30.

[45] Open Quantum Safe. 2024. oqs-provider. (2024). https://github.com/open-quantum-safe/oqs-provider Retrieved 8-10-2024.

[46] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2020. Post-quantum TLS without handshake signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1461–1480.

[47] Kyung-Ah Shim. 2023. On the suitability of post-quantum signature schemes for internet of things. *IEEE Internet of Things Journal* (2023).

[48] Peter W Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 124–134.

[49] Dimitrios Sikeridis, Sean Huntley, David Ott, and Michael Devetsikiotis. 2022. Intermediate certificate suppression in post-quantum TLS: an approximate membership querying approach. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*. 35–42.

[50] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. Assessing the overhead of post-quantum cryptography in TLS 1.3 and SSH. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 149–156.

[51] Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. 2023. The performance of

post-quantum tls 1.3. In *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies*. 19–27.
[52] STMicroelectronics. 2024. STM32 Nucleo-144 development board with STM32F439ZI MCU, supports Arduino, ST Zio and morpho connectivity. (2024). https://www.st.com/en/evaluation-tools/nucleo-f439zi.html#overview July, 11st, 2024.
[53] Srikanth Sundaresan, Nazanin Magharei, Nick Feamster, and Renata Teixeira. 2012. Accelerating Last-Mile Web Performance with Popularity-Based Prefetching. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '12)*. Association for Computing Machinery, New York, NY, USA, 303–304. https://doi.org/10.1145/2342356.2342421
[54] George Tasopoulos, Charis Dimopoulos, Apostolos P Fournaris, Raymond K Zhao, Amin Sakzad, and Ron Steinfeld. 2023. Energy consumption evaluation of post-quantum TLS 1.3 for resource-constrained embedded devices. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*. 366–374.
[55] George Tasopoulos, Jinhui Li, Apostolos P Fournaris, Raymond K Zhao, Amin Sakzad, and Ron Steinfeld. 2022. Performance evaluation of post-quantum TLS 1.3 on resource-constrained embedded systems. In *International Conference on Information Security Practice and Experience*. Springer, 432–451.
[56] Chengdu Ebyte Electronic Technology. 2024. E77-900M22S. (2024). https://www.cdebyte.com/products/E77-900M22S/2 Retrieved 14-10-2024.
[57] Joao Diogo Duarte Thomas Attenma, Matthieu Lequesne Vincert Dnning, Marc Stevens Ward van der Schoot, and AIVD Cryptologists & Security Advisors. 2023. The PQC Migration Handbook. (2023).
[58] Top 1 million 2016. Umbrella Popularity List. https://s3-us-west-1.amazonaws.com/umbrella-static/index.html. (2016).
[59] Geoff Twardokus, Nina Bindel, Hanif Rahbari, and Sarah McCarthy. 2024. When Cryptography Needs a Hand: Practical Post-Quantum Authentication for V2V Communications.. In *NDSS*.
[60] W3C. 2024. Web of Things (WoT) Architecture 1.1. (2024). https://w3c.github.io/wot-architecture/ Retrieved 14-10-2024.
[61] Zheng Wang. 2014. POSTER: on the capability of DNS cache poisoning attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 1523–1525.
[62] Zheng Wang et al. 2013. Analysis of DNS cache effects on query distribution. *The Scientific World Journal* 2013 (2013).
[63] LoRa Alliance Technical Committee Regional Parameters Workgroup. 2018. LoRaWAN™ 1.0.3 Regional Parameters. (2018). https://lora-alliance.org/wp-content/uploads/2020/11/lorawan_regional_parameters_v1.0.3reva_0.pdf Retrieved 14-10-2024.
[64] The ETSI Quantum-Safe Cryptography working group. 2020. Migration strategies and recommendations to Quantum Safe schemes. (2020).

A E-BOX STRUCTURE DETAIL

This section describes the detailed format of E-Box . It follows the formatting conventions outlined in the 1988 abstract syntax notation one (ASN.1) [39] (also used in [11]).

FIELD	SUB-FIELD

E-box Header	totalRecordsNumber::= SEQUENCE{
	totTXT: Integer,
	totTLSA: Integer}
	version::= INTEGER{
	ExpressPQDelivery_v1(0),
	ExpressPQDelivery_v2(1),
	...}
	validity::= SEQUENCE{
	notBefore: utcTime,
	notAfter: utcTime}

PQ key	certificate:SEQUENCE{
	Cert. size: Integer,
	Cert. value: Bit String}
	addPQKey[OPTIONAL]:SEQUENCE{
	keyAlgoId::=INTEGER{
	Kyber512(0),
	Kyber768(1),
	...},
	Pub. size: Integer,
	Pub. value: Bit String}

E-box Signature	algorithmId::= INTEGER{
	DILITHIUM2(0),
	DILITHIUM3(1),
	DILITHIUM5(2),
	Falcon512(3),
	Falcon1024(4),
	Sphincs+128s(5),
	...}
	Signature::= SEQUENCE{
	Sign. size: Integer,
	Sign. value: Bit String}

Figure 9: E-box Structure

B IOT BOARDS

We summarize the hardware specification of IoT boards used in previous work.

Table 5: Specifications of various boards used in previous work. The columns detail the board name, the type of ARM Cortex processor used, RAM size, Flash memory size, and Ethernet support.

Board	Processor	RAM	Flash memory	Ethernet
Nucleo-F439ZI [54, 55]	M4	256KB	2MB	O
STM32F4-discovery [28]	M4	192KB	1MB	X
Nucleo-L4R5ZI [28]	M4	640KB	2MB	O
Nucleo-L476RG [28]	M4	128KB	1MB	X
Nucleo-F411RE [4, 44]	M4	128KB	512KB	O

C PQC KEY SIZES

Note that the public key size of KYBER is 25 to 49 times larger than that of ECDH, and that of DILITHIUM5 is 81 times larger that of ECDSA. The signature sizes of PQC algorithms are from 10.4 times to 779 times larger than that of ECDSA.

Table 6: Sizes of keys, ciphertexts, and signatures

KEM	NIST Level	Size (byte)		
		public key	*private key*	*ciphertext*
ECDH-P256	0	32	32	32
DH-2048	0	256	256	256
KYBER512	1	800	1,632	768
KYBER768	3	1,184	2,400	1,088
KYBER1024	5	1,568	3,168	1,568
DSA	**NIST Level**	**Size (byte)**		
		public key	*private key*	*signature*
ECDSA-P256	0	32	32	64
RSA-2048	0	259	256	256
DILITHIUM2	2	1,312	2,528	2,420
DILITHIUM3	3	1,952	4,000	3,293
DILITHIUM5	5	2,592	4,864	4,595
FALCON512	1	897	1,281	666
FALCON1024	5	1,793	2,305	1,280

D NETWORK DELAY DUE TO TRANSPORT LAYER SERVICES

The problem is that transport layer services would affect the delivery time required to send multiple packets. In detail, the time depends on many aspects, including a congestion window size (cwnd) related to the congestion control mechanism (e.g., the slow

start) or a receive window size (rwnd) with regard to the flow control mechanism. Therefore, we evaluate how transport protocols (i.e., TCP and UDP) affect the delivery time of PQC certificates.

Experiments on the delivery time. We conduct two experiments to see how TCP and UDP affect the delivery time of PQC certificates. First, we measure delivery time over TCP and UDP, respectively, to see the effect of reliable data transfer services. Second, we evaluate delivery time over TCP varying the size of rwnd considering low-memory devices. We set up a networking testbed with simple client/server applications to perform the experiments. The RTT between a server and a client is 200 ms, and both sides are connected to the Internet through Ethernet, of which MTU is 1.5 KB. Note that both a server and a client adopt TCP CUBIC that relies on the slow start algorithm for their congestion control mechanism. In our experimental setting, we fix the initial size of cwnd to 1.46 KB, which is ten times MSS. Also, we generate self-signed PQ certificates based on DILITHIUM, FALCON, and SPHINCS+ by `OpenSSL-3.3.0`, `liboqs-0.10.0`, and `oqs-provider-0.5.4`. The server responds with the PQ certificate according to the request sent by the client. All the results averaged over 30 trials are demonstrated in Figure 1.

TCP vs. UDP. Our first experiment demonstrates that the delivery time increases as a step graph with regard to the number of bytes sent by a server in TCP, while the time is stable in UDP (see Figure 1). The increment in TCP is mainly due to the TCP window size and the ACK mechanism. In detail, since the server can only transmit data as much as the TCP window size, the server cannot send the certificate in one transmission when the certificate size exceeds the TCP window size. The server must wait for the client's ACK message before sending the remaining part of the certificate. Therefore, the difference between steps in the step graph is around the round-trip time (200 ms in our setting). This latency is not marginal because the round-trip time is typically the dominant factor in the networking delay [19]. For instance, the delivery time of the SPHINCS+128f (23 KB) certificate was over 400 ms even if the rwnd was large (e.g., 128 KB) because the size of the initial cwnd was 14.6 KB, and thus two round-trips were required to send 23 KB. On the other hand, since UDP does not support such a mechanism, a UDP server sends all the fragments of the certificate in parallel; therefore, the delivery time is similar regardless of the certificate size.

Diverse receive window size over TCP. When we conduct an experiment varying the size of rwnd in TCP, we find that the networking overhead would be significant if a client could provide a small rwnd (e.g., 4 KB). For instance, upon comparing the delivery times for DILITHIUM5 and SPHINCS+128f certificates, we observe that the delivery time difference was 1052.87 ms and 231.21 ms when the rwnd size was 4 KB and 16 KB, respectively. It is mainly because the smaller rwnd incurs more round trips to deliver the same certificate. Therefore, resource-constrained IoT devices that use smaller rwnd or congested devices that use minimal rwnd would show high delivery times. Furthermore, it would not be suitable for battery-powered wireless devices in terms of power consumption as they cannot convert to sleep states to save their batteries while waiting much longer for data, compared to the current practice. On the other hand, we also observe that even if the size of rwnd is larger than the certificate size, the delivery time increases because the length of the certificates mostly exceeds the initial size of cwnd .

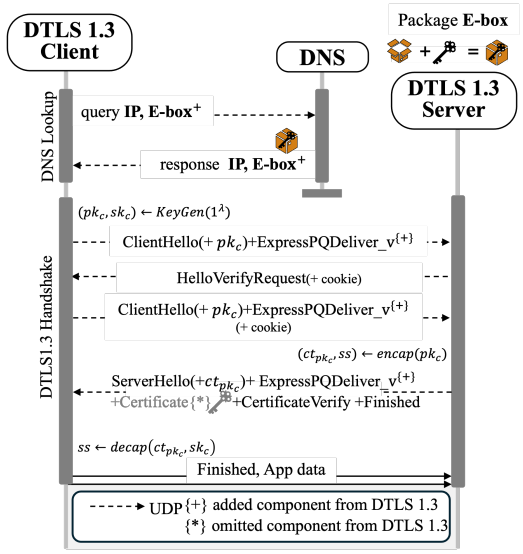

Figure 10: Overview of PQC-DTLS using ExpressPQDelivery

E UNIVERSALITY

This section provides the detailed designs of ExpressPQDelivery applications to DTLS 1.3 [42], QUIC [22], and KEMTLS [46].

E.1 Applying ExpressPQDelivery to PQC-DTLS

DTLS (Datagram Transport Layer Security) is a protocol that operates over UDP based on TLS 1.3. We first briefly explain PQC-DTLS and then describe PQC-DTLS using ExpressPQDelivery.

Overview of DTLS 1.3. When a client intends to connect to a server, the client first obtains an IP address through the DNS resolution with an A record. After a DNS lookup, the client immediately sends a ClientHello message to initiate the handshake. In DTLS, because it is UDP-based, messages can be lost, so the client sets a timer and retransmits the ClientHello message if there is no response (i.e., HelloVerifyRequest) from the server. Similar to TLS 1.3, the ClientHello message includes the ephemeral KEM public key pk_c and other parameters. When the server receives a ClientHello, the server sends a HelloVerifyRequest containing a stateless cookie as an extension to prevent DoS (denial-of-service) attacks. After the client receives the HelloVerifyRequest, it must send a new ClientHello with the cookie added as an extension. When the server receives a new ClientHello, it responds with the ct_{pk_c} in ServerHello, followed by Certificate, CertificateVerify and Finished. The client verifies the Certific-ateVerify using server's public key. The client derives the session key from the ct_{pk_c}, then verifies the Finished message.

ExpressPQDelivery-approach for DTLS 1.3. We describe how

ExpressPQDelivery is integrated with PQC-DTLS 1.3 in the E-Box setup and the E-Box delivery. Similar to PQC-TLS 1.3, during the E-Box setup process, a server packages an E-Box and uploads it to the authoritative name server. Next, two phases of E-Box delivery operation process are different compared to PQC-DTLS. One is the DNS lookup phase, and the other is the `ServerHello` message: In the DNS lookup phase, the client fetches the first E-Box fragment and simultaneously queries `TLSA`, `TXT`, and `A` records after finding the numbers of records from the first fragment. Once the client receives an E-Box and IP address, it first verifies the signature in an E-Box by using a PQ key in the E-Box to authenticate a server. If the client receives `NXDOMAIN` or fails to verify the signature, it falls back to the PQC-DTLS 1.3 protocol. Then, the client initiates a handshake by sending the `ClientHello` message to the server. For the `ServerHello` message phase, when the server sends `ServerHello`, a PQC- DTLS 1.3 server sends its heavy PQ certificate along with the `ServerHello`, but when using ExpressPQDelivery, the server does not need to send the server's certificate because it has already been delivered to the client via the PQ key field of an E-Box .

E.2 Applying ExpressPQDelivery to KEMTLS

KEMTLS is a TLS protocol proposed in 2020 by Schwabe *et al.* [46] that uses only KEM instead of signatures for server authentication. We first briefly explain the KEMTLS and then describe the KEMTLS using ExpressPQDelivery .

Overview of KEMTLS. Similar to TLS 1.3, the client first sends `ClientHello` with its ephemeral KEM public keys pk_c and the ExpressPQDelivery version extension type after a TCP connection. The server checks the extension type to determine whether it falls back to normal KEMTLS and then computes the shared secret ss_c and the encapsulation ct_{pk_c} against the pk_c. At this time, the ephemeral session keys K and K' are derived from ss_c using HKDF. In `ServerHello` , the server responds to ct_{pk_c} and $AEAD_K(Certificate[pk_s])$. The latter is the server's encrypted KEM certificate using the key K.

In the second client-to-server flight, the client decapsulates the ct_{pk_c} to obtain ss_c and derives session keys K and K'. Using these session keys, the client first decrypts the $AEAD_K(Certificate[pk_s])$ using the K, then calculates the ct_{pk_s} and shared secret ss_s against the pk_s. Next, the client sends the $AEAD_{K'}(ct_{pk_s})$, which is ct_{pk_s} encrypted with the K'. During the same flight, the client derives the authenticated session key ss from the combined ss_c and ss_s using HKDF and sends the `Finished` message.

On the server side, the server first decrypts the $AEAD_{K'}(ct_{pk_s})$ with k' then gets the shared secret ss_s as it decapsulates the cp_{pk_s}. Same as the client, the server combines the ss_c and the ss_s, derives the session key ss using HKDF, and then verifies the `Finished` message. In the last flight server-to-client, the server also sends a `Finished` message. Finally, the client verifies the `Finished` message and confirms the integrity of the handshake.

ExpressPQDelivery-approach for KEMTLS.

To implement ExpressPQDelivery, the E-Box must provide authentication that it was created by the domain owner. However, since a KEMTLS server uses a KEM certificate as its PQ key, it cannot provide authentication. As previously described, TLS 1.3 provides server authentication through a `CertificateVerify` (a signature over the handshake) and a `certificate` that contains the digital signature algorithm (DSA) public key and signature. In contrast, KEMTLS provides server authentication by performing KEM twice without a signature and using a KEM certificate instead of a DSA certificate. Thus, for applying ExpressPQDelivery, a server signature is needed for E-Box authentication.

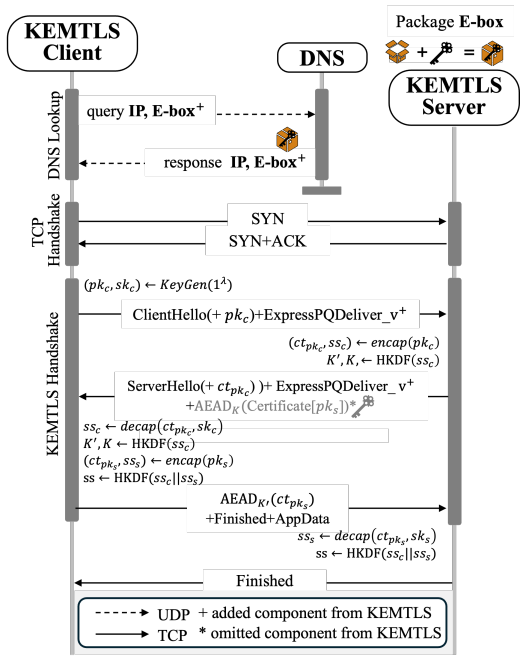

Figure 11: Overview of KEMTLS using ExpressPQDelivery

We describe the application of ExpressPQDelivery to KEMTLS into two parts: the E-Box setup and the E-Box delivery operation. During the E-Box setup process, the KEMTLS server stores its DSA `certificate` and KEM `publicKey` in the PQ key field of the E-Box and uploads the E-Box to its authoritative name server. Next, in the E-Box delivery operation process, it differs from the standard KEMTLS at two phases: 1. the DNS lookup phase; and 2. the `ServerHello` message phase. In the DNS lookup phase, the client selects a key delivery mode and queries TLSA, TXT, and A records. Upon receiving the E-Box and IP address, the client verifies the E-Box and checks the `version` field to determine if a fallback to the normal protocol is necessary when the server does not support ExpressPQDelivery . For the `ServerHello` message phase, a normal KEMTLS server sends its KEM certificate along with the `ServerHello`, whereas when using ExpressPQDelivery, the server does not send its KEM certificate.

F EVALUATION RESULTS FOR ALL SCENARIOS

Our experiment results at NUCLEO-F439ZI board for all scenarios (i.e., intra-country, inter-country, and inter-region) show that

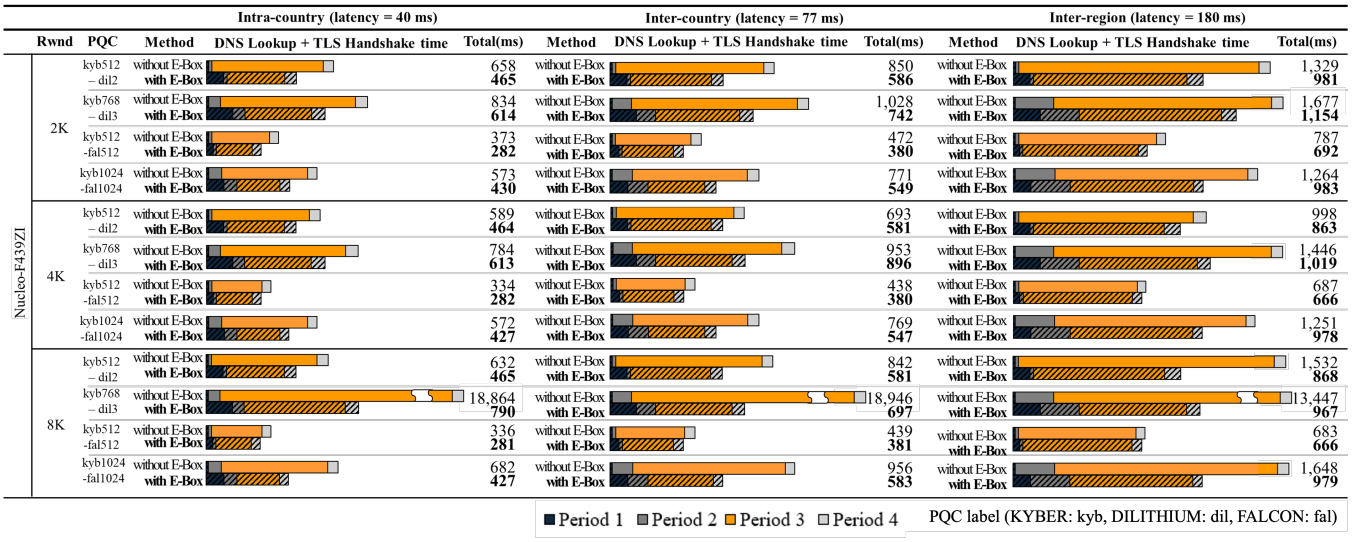

Figure 12: Evaluation results for all scenarios at NUCLEO-F439ZI.

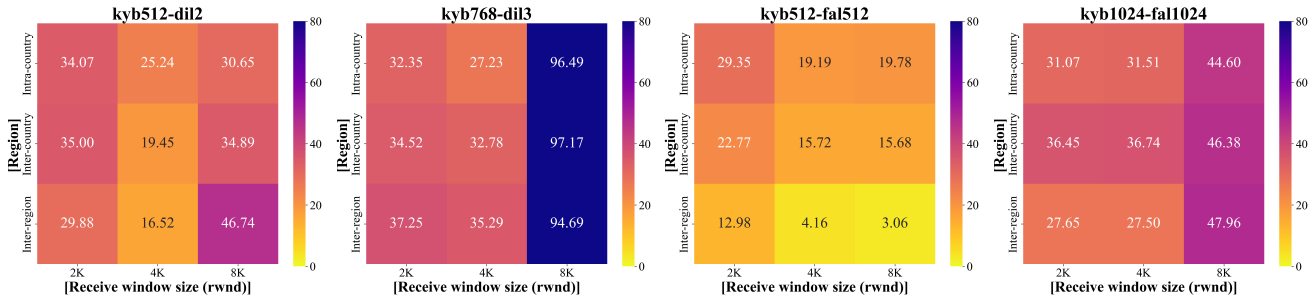

Figure 13: Gains from applying ExpressPQDelivery on PQC-TLS 1.3. The x-axis lists the window size and the y-axis indicates the server location: intra-country, inter-country, inter-region.

ExpressPQDelivery reduces the PQC-TLS handshake times regardless of the network latency between the server and client (see Figure 12). The E-Box gain we calculated for all scenarios ranges from 4.6% to 97%, with an average of 34%.

G DISCUSSION

In this section, we explain how ExpressPQDelivery securely delivers the PQ key through DNS, considering the threat model described in Section §4.1. Then we discuss the overheads incurred on DNS by using ExpressPQDelivery .

G.1 Networking Impact on DNS

Time-to-live (TTL) and networking within DNS. The number of packets exchanged between a resolver and an authoritative name server increases due to ExpressPQDelivery. It depends on the time-to-live (TTL) value on DNS records.

First, it is fine to set the TTL value to the value until the expiration time of the certificate if an E-Box only contains a certificate. Then, no field needs to be periodically updated. Considering that a server uses a long-term key for a signature, it would reduce the number of networking a lot within the DNS infrastructure. The TTL value should not exceed the difference between a certificate's current and expiration times for the operational purpose. In this case, the networking within DNS to get DNS records from an authoritative name server happens only when the records are evicted or the key is expired.

Second, we suggest setting the TTL value to be less than seven days if there is at least one public key encoded in an E-Box. The reason for the seven days is that it is used in practice as the validity period for a delegated credential [6], which has a similar structure, i.e., a public key guaranteed by a signature, to an E-Box.

Networking between a resolver and a client. Due to the increased number of records, DNS resolvers should exchange many more packets between DNS resolvers and a client. Note that these packets are to be originally exchanged between a client and a server to deliver a PQ key. With ExpressPQDelivery, the amount of bytes is moved from the connection between a client and a server to the connection between a client and a resolver. Because of this movement, the additional bytes are the bytes of the E-Box header and one signature. Note that many techniques that are proposed

or deployed leveraging DNS, such as `EncryptedClientHello` [41] or DANE [21] introduce similar types of overheads to DNS. ExpressPQDelivery does not introduce any new type.

G.2 Congestion due to multiple UDP packets

As ExpressPQDelivery delivers an E-Box over UDP, there is no mechanism to control congestion; thus, ExpressPQDelivery may contribute to increasing the amount of network congestion. We note two things regarding this issue. First, as the size of an E-Box is less than 200 KB per domain and it is used once per session, we believe routers can be resilient to the increments. Note that the throughput of routers keeps increasing to cover such increments. Second, as a client receives a PQ key not from a server but from a resolver, the number of hops is reduced. Our simple experiment shows that it is 17 hops on average toward servers and 8 hops to a Google resolver; therefore, packets would be disseminated less. If a local resolver is used, the possibility of congestion because of UDP packets would be less.