Replacing a DH with a KEM in protocols: how and why

Ronald Landheer-Cieslak

September 16, 2023

Abstract

The advent of quantum computing presents new challenges for secure OT protocols. The current lack of a secure post-quantum replacement of the Diffie-Hellman key exchange (and its ECDH counterpart) means that novel protocols such as DNP3-SAv6 may have to integrate alternative approaches such as Key Encapsulation Mechanisms (KEMs). This paper presents a means to simulate a KEM using only DH constructs to allow for experimentation as such alternative approaches are developed, reviews existing analyses for related protocols and analyzes how the lessons learned from those analyses apply to novel OT protocols like DNP3-SAv6 and IEC 62351-5, and presents recommendations for changes to DNP3-SAv6 ahead of that protocol's publication in the upcoming DNP3 standard to accomodate necessary changes to be made when more thorough cryptanalysis on post-quantum cryptographic standards is available.

1 Introduction

With the advent of quantum computers, quantum supremacy having been shown[1], quantum practicality[2] only being a decade or so away, and cryptanalytically-relevant quantum computers (CRQC) will perhaps follow shortly behind [3][4][5] , NIST is in its fourth round of looking for post-quantum cryptography algorithms[6].

Novel algorithms like SWOOSH[7] notwithstanding, no Diffie-Hellman-type algorithms have so far been chosen as standardization candidates[8] but three out of the four announced algorithms [9][10][11][12] are Key Encapsulation Mechanisms rather than Key Exchange Mechanisms.

This raises the question of the future of key exchange mechanisms like the Diffie-Hellman key exchange and its ECC cousin, Elliptic Curve Diffie-Hellman, both of which are known to be vulnerable to Shor's algorithm[13], but both of which are also at the heart of secure IT protocols such as TLS 1.2[14], TLS 1.3[15], and SSHv2[16], as well as novel secure OT protocols such as DNP3-SAv6[17] and IEC 62351-5[18][19][20][21][22], the latter two, both OT-oriented, lack any significant work (that I know of) on how the ECDH they use may be replaced with a KEM if the need arises.

This paper aims to fill that gap by contributing three things: first, I present an analysis on how DH and KEM differ, and how a KEM can be notationally expressed in terms of a DH. The purpose of this is to allow framing the question in terms of the DH construct, which is generally more familiar to members of the OT-cyber community. It also allows experimenting with tools we have at hand without needing to modify existing libraries such as OpenSSL. Second, I present lessons from experiments with TLS found in the existing literature, as applicable to the DNP3-SAv6 and IEC 62351-5 protocols. Finally, I present proposed changes to DNP3-SAv6, as presented to the DNP CSTF at the time of this writing, with their background reasoning.

The paper is structured as follows: in section 2, I will explain what a KEM is in terms of the use case it addresses and the minimum API it exposes. Section 3 I will explain how a KEM is different from a DH by highlighting the similarities and differences between the use cases, and the APIs the two schemes expose. In section 4, I look at the KEM use case through a DH lens, and show how KEMs have been implemented using classical algorithms, including RSA for comparison. This section also contains working examples, in Python for readability, of KEM

implementations. In section 5 I look at the impact of using a KEM where a DH would formerly have been used. In this section, I use DNP3-SAv6 as an example protocol in which the impact of using KEM can be analysed in a fairly straight-forward manner. Section 6 concludes the paper.

2 What is a KEM?

A KEM is used, as the name implies, to encapsulate a (symmetric) key in order to securely send it to someone else. As it's an asymmetric cryptography algorithm, you need the other person's public key to do this but, unlike key exchange algorithms, the other person does not need your public key to receive the symmetric key and, unlike digital signature algorithms and key exchange algorithms, the recipient cannot authenticate the received key with only a singge use of this mechanism.

This is actually one of the oldest applications of RSA besides digital signatures: because RSA allows things encrypted with a public key to be decrypted with the corresponding private key as well as allowing something encrypted with a private key to be decrypted with the corresponding public key, it can be used both for key encapsulation and signatures.

There are three use-cases for KEM, all three discussed below. The first is a simple key encapsulation, where Alice sends a message to Bob, ascertaining only Bob can receive it, but Bob cannot authenticate the message as being from Alice, and without providing forward secrecy. The second is called "Unilateral Authenticated Key Exchange" or UAKE, where Alice sends a message to Bob, ascertaining only Bob can receive it, but Bob can still not authenticate the message as being from Alice; this time providing forward secrecy but requiring messages to be exchanged in both directions for the key exchange to occur. The third, called "Authenticated Key Exchange" or AKE, allows Alice and Bob to mutually authenticate and also provides forward secrecy, also requiring messages to be exchanged in both directions. This latter use-case is closest to the DH use-case, but unlike DH does not allow for an off-line mutually authenticated key exchange.

2.1 KEM API

The KEM API consists of three functions:

- keypair generation: $G \rightarrow \langle sk, pk \rangle$ generates a key pair $\langle sk, pk \rangle$ where sk is the private key and pk is the public key. This function takes no parameters.
- encrypting: $E \rightarrow pk \rightarrow \langle s, ct \rangle$ takes the public key pk and generates a random shared secret s and the corresponding ciphertext ct. Due to the way the API is specified (which is essentially the smallest common denominator for these types of algorithms) the user has no control over the value of the shared secret. We will exploit this fact in our simulation.
- decrypting: $D \rightarrow sk \rightarrow ct \rightarrow s$ takes the private key sk and the ciphertext ct and produces the shared secret, which is identical to the one generated by E.

In the proof of concept below, G is called `kem_gen`, E is called `kem_enc` and D is called `kem_dec`.

2.2 KEM use cases

The typical use case for a key encapsulation is the same as for a key exchange: Alice wants to send a message to Bob and doesn't want Eve to be able to eavesdrop. Assuming Alice at least has Bob's public key, Alice has three options:

1. She can perform a straight key exchange (KE), which allows her to send a key to Bob knowing only Bob can decrypt it, but does not provide any forward secrecy. She does not have to receive any messages from Bob to do this (she already has his public key), and Bob does not need her public key.
2. She can perform a "Unilaterally Authenticated Key Exchange" (UAKE), which allows her to send a message to Bob knowing only Bob can decrypt it, and provides limited forward

secrecy. She does need to send a message to Bob and receive a response for this, in addition to the public key she already has. Bob does not need her public key.

3. She can perform an "Authenticated Key Exchange" (AKE). This allows her to send a message to Bob knowing only Bob decrypt it, but additionally allows Bob to authenticate the message as being from Alice. Alice and Bob need to exchange messages for this, similar to the UAKE case, but this does provide both mutual authentication and limited forward secrecy.

KEM does not have an option that provides mutual authentication without limited forward secrecy, like DH does. It is possible to construct a mutually-authenticated key exchange with perfect forward secrecy using only KEM, but that requires a ciphertext to be sent along with the encrypted message. This fourth construct is left as an exercise to the reader.

The three constructs above are each described in Bos et al. [9], and described below.

2.2.1 KEM Key Exchange (KE): unilateral authentication, no forward secrecy, off-line

Alice can send an unauthenticated message to Bob, without requiring any on-line message exchanges with Bob, but still making sure only Bob can read the message. To do this:

1. Bob generates a key pair (private key and public key) and securely provisions his public key to Alice (i.e. she can authenticate it). More formally, Bob performs $\langle sk, pk \rangle \leftarrow G$, then signs pk and sends it to Alice.
2. Alice uses Bob's public key to generate a shared secret and a ciphertext $(ct, s \leftarrow E\ pk)$. She then encrypts her message with that derived key using an AEAD.
3. Bob receives the message, decrypts the ciphertext using his private key and thus obtains the same shared secret as Alice, and uses that to the decrypt the message.

Eve will only have seen the public key and ciphertext, and thus has no feasible way to recover the shared secret or the message.

Note that this approach does not allow Bob to authenticate Alice's message. For that, she either has to use the AKE approach described below, of use a DSA to sign the message and the ciphertext.

Also note that, because no ephemeral keys are used here, if Eve ever obtains Bob's private key she can decrypt every message Alice ever sent to Bob.

2.2.2 KEM Unilaterally Authenticated Key Exchange (UAKE): unilateral authentication, limited forward secrecy, on-line

Alice can send an unauthenticated message to Bob, making sure that only Bob can read it, and making sure that even if Eve obtains Alice's (ephemeral) private key, she won't be able to decrypt every message Alice ever sent to Bob (provided she doesn't use the same ephemeral key pair every time she sends a message to Bob). Eve will still be able to do that if she obtains Bob's private key. To do this:

1. Bob generates a key pair (private key and public key) and securely provisions his public key to Alice (i.e. she can authenticate it). More formally, Bob performs $\langle sk_B, pk_B \rangle \leftarrow G$, then signs pk_B and sends it to Alice.
2. Alice generates an *ephemeral* key pair $\langle sk_A, pk_A \rangle \leftarrow G$ and a ciphertext and shared secret $\langle s_1, ct_1 \rangle \leftarrow E\ pk_B$ and sends $\langle pk_A, ct_1 \rangle$ to Bob.
3. Bob receives $\langle pk_A, ct_1 \rangle$, decapsulates the ciphertext to get the shared secret $s'_1 \leftarrow D\ sk_B\ ct_1$ and generates a ciphertext and shared secret from Alice's ephemeral public key: $\langle s_2, ct_2 \rangle \leftarrow E\ pk_A$. He then sends back the new ciphertext ct_2. Bob can now use a pre-agreed combination function H to combine s'_1 and s_2 into a shared secret $k \leftarrow H\ s'_1\ s_2$.
4. Alice receives ct_2 and decapsulates it using her ephemeral private key $s'_2 \leftarrow D\ sk_A\ ct_2$. Alice can now use the same pre-agreed combination function H to combine s_1 and s'_2 into the same shared secret $k' \leftarrow H\ s_1\ s'_2$, after which $k = k'$. She can then use the new key k to encrypt her message.

Note that this means Alice and Bob have to exchange two messages (one in each direction) to set up these keys, and only Bob is authenticated at this point. If Alice wants to authenticate herself to Bob, she needs to either use the AKE mechanism explained below, or use a DSA algorithm.

This method does use an ephemeral key, but only from Alice: if Eve were to obtain Bob's private key, she would still be able to decrypt every message ever sent to Bob, provided she kept the transcripts (which contain the public keys used by Alice).

2.2.3 KEM Authenticated Key Exchange (AKE): mutual authentication, limited forward secrecy, on-line

Alice can send an authenticated message to Bob, making sure only Bob can read it, and making sure Eve would have to know both Alice and Bob's private keys to decrypt messages exchanged between the two. To do this:

1. Bob generates a key pair (private key and public key) and securely provisions his public key to Alice (i.e. she can authenticate it). More formally, Bob performs $\langle sk_B, pk_B \rangle \leftarrow G$, then signs pk_B and sends it to Alice.
2. Alice generates a key pair (private key and public key) and securely provisions her public key to Bob (i.e. he can authenticate it). More formally, Alice performs $\langle sk_A, pk_A \rangle \leftarrow G$, then signs pk_A and sends it to Bob. Note that she *could* send her key to Bob at the same time she sends the next message (below).
3. Alice generates an *ephemeral* key pair $\langle sk_{A*}, pk_{A*} \rangle \leftarrow G$ and a ciphertext and shared secret $\langle s_1, ct_1 \rangle \leftarrow E\ pk_B$ and sends $\langle pk_{A*}, ct_1 \rangle$ to Bob.
4. Bob receives $\langle pk_{A*}, ct_1 \rangle$, decapsulates the ciphertext to get the shared secret $s'_1 \leftarrow D\ sk_B\ ct_1$, encapsulates a first secret using the just-received ephemeral public key $\langle s_2, ct_2 \rangle \leftarrow E\ pk_{A*}$ and a second secret using Alice's static key $\langle s_3, ct_3 \rangle \leftarrow E\ pk_A$. He now has three shared secrets, s'_1, s_2, and s_3 which he can combine using a pre-agreed function H into a single shared secret $k \leftarrow H\ s'_1\ s_2\ s_3$. He sends $\langle c_2, c_3 \rangle$ to Alice.
5. Alice receives $\langle c_2, c_3 \rangle$ and decapsulates them using her ephemeral and static private keys, resp.: $s'_2 \leftarrow D\ sk_{A*}\ ct_2$, and $s'_3 \leftarrow D\ sk_{A*}\ ct_3$. She now has the same three shared secrets, s_1, s'_2, and s'_3 which she can combine using a pre-agreed function H int a single shared secret $k' \leftarrow H\ s_1\ s'_2\ s'_3$. Now, $k = k'$, so Alice can encrypt her message using this key.

With this approach, Alice and Bob have to exchange two messages, one in each direction, but they get mutual authentication and limited forward secrecy.

Like UAKE, this method does use an ephemeral key, but only from Alice: if Eve were to obtain Bob's private key, she would still be able to decrypt every message ever sent to Bob, provided she kept the transcripts (which contain the public keys used by Alice).

3 How is a KEM different from DH?

The use case between a KEM and a DH is very similar: in both cases, Alice and Bob want to establish a symmetric key to use in communications. In the case of a DH, this happens by Alice and Bob exchanging their respective public keys, and each using their respective private keys and each others' public keys.

While that use case is similar, there are significant differences: KEM does not provide *offline* mutually authenticated key exchange where DH does, which means KEM requires the use of a separate digital signature algorithm if it to be used in an offline mode for mutual authentication. To use only a KEM algorithm for a mutually authenticated key exchange, at least two messages need to be exchanged and the algorithm is used three times on either side (Alice encapsulates once and decapsulates twice; Bob encapsulates twice and decapsulates once).

Similarly to KEM, DH does not allow its user any influence over the contents of the symmetric key. Also, because with the use of the same keys the resulting shared secret is always the

same, it is recommended to always include a salting step (generally in the form of a publicly-communicated salt and an HKDF function). This is not required in the case of a KEM.

Neither DH not KEM provide perfect forward secrecy using static keys.

3.1 The DH API

The DH API consists of two functions:

- keypair generation: $G \rightarrow \langle sk, pk \rangle$ generates a keypair $\langle sk, pk \rangle$ where sk is the private key and pk is the public key, and
- shared secret generation: $X \rightarrow sk \rightarrow fpk \rightarrow s$ where sk is the local private key and fpk is the foreign (remote) public key, and s is the generated shared secret.

Because there are only two functions, and the exchange function X is symmetrical (in that $X\ sk_a\ pk_b = X\ sk_b\ pk_a$), generation of the shared secret can be done without any on-line communication, provided the foreign public key fpk is known, and if the public key pk is trusted by the recipient of a message, no further signatures are necessary. This means that if Alice wants to send a message to Bob, and she wants to use a unique symmetric key to encrypt it (say using an AEAD), she only has to send the ciphertext of her message and the salt used to generate the symmetric key $\langle m', salt \rangle$ to Bob. With KEM, the equivalent would be to generate a a shared secret and corresponding ciphertext, and send both the ciphertext and the salt along with the message to Bob, signing the entire message with a separate private key so Bob can verify the message, sending $\langle m, ct, signature \rangle$.

4 Simulating a KEM

As mentioned above, there are currently no selected candidates for standardization for post-quantum cryptography in the DH family. We therefore need to look at two things:

1. How can we build a KEM algorithm today, using pre-quantum tools?
2. How can we use KEM in our existing protocols where we were using DH before?

For the first question, two possible answers come to mind:

1. use RSA: RSA supports both digital signatures by encrypting a hash value with the RSA private key (where verification decrypts the value with the RSA public key) and key encapsulation by encrypting the key value with the RSA public key and decrypting it with the corresponding RSA private key.
2. use DH: converting a DH algorithm into a KEM is a straight-forward exercise in software engineering, as shown below.

For the second question, the answer will depend on the protocol.

4.1 Using RSA as a KEM

The Python code below is an example of implementing the three KEM functions, `kem_gen`, `kem_enc` and `kem_dec` using RSA. This is by far the most common way of implementing a KEM using classical algorithms, and is included as a reference example. THe implementation presented here uses standard modules from the "Python Cryptographic Authority", PyCA, and is, in principle, secure.

[4]:
```
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
import os
```

The size of the RSA key chosen here is arbitrarily set at 2048 bits, which is generally secure but will, of course, be breakable by a quantum computer when those become available. With "harvest now, decrypt later" schemes potentially viable today (see for example [23], [24], and [25] for more on this topic) anything that would still be of interest in Q years, where Q is the

number of years until it is feasible with a quantum computer for *your particular adversary* to break your key and steal your data, should no longer use these key sizes.

[5]:
```
KEY_SIZE=2048
```

The three KEM API functions are the most straightforward available versions of these functions with RSA, in which `kem_enc` generates a random 256-bit symmetric keyand encrypts it, returning both the random shared key and the ciphertext.

[6]:
```
def kem_gen():
    sk = rsa.generate_private_key(public_exponent=65537, key_size=KEY_SIZE)
    pk = sk.public_key()
    return sk, pk

def kem_enc(pk):
    s = os.urandom(32)
    ct = pk.encrypt(
        s,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return s, ct

def kem_dec(sk, ct):
    return sk.decrypt(
        ct,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
```

Using this version of the functions is then simple a question of Bob generating his keypair, and Alice using Bob's public key to encrypt a randomly-generated shared secret.

[7]:
```
# Bob:
sk, pk = kem_gen()
# send pk to Alice

# Alice:
s_b, ct = kem_enc(pk)
# send ct to Bob

# Bob:
s_a = kem_dec(sk, ct)
```

Obviously, to show that both Alice and Bob posess the same shared secret, we can compare the two (the Python output block below will show `True` if this is the case).

[8]:
```
s_a == s_b
```

[8]: `True`

As shown, the RSA version requires some padding but can easily be used to implement the KEM API. Over-the-wire communications are not represented in this example.

4.2 The KEM use case using DH

Let's take another look at our use case: Alice wants to send a message to Bob without Eve being able to eavesdrop. In a typical DH case, this would mean:

1. Alice and Bob each generate key pairs and exchange their public keys. Like in KEM, provisioning the keys is done securely such that the other party can authenticate the public key but unlike KEM, public keys are exchanged in both directions.
2. Alice uses her own private key and Bob's public key to generate a shared secret, and uses that shared secret and a salt to create a derived symmetric key. She prepares her message and encrypts it using an AEAD, and sends the encrypted message and the salt to Bob.
3. Bob receives the salt and the encrypted message. He uses his own private key and Alice's public key to generate the shared secret, and uses Alice's salt to create the derived symmetric key. He can then use the derived symmetric key to decrypt and authenticate the message using the AEAD.

The challenge for turning a DH into a KEM is to remove one of the exchanges from the protocol: we can only rely on Bob sending his public key to Alice, but Alice cannot send her public key to Bob before the shared key is generated.

Formulating the problem like this, the solution is almost obvious. We can make it more so by redefining the KEM ciphertext as being "such information as required by Bob to calculate, in combination with his own private key, the same shared key as Alice can calculate with Bob's public key". It should now be obvious that the KEM ciphertext in the context of a DH algorithm is a tuple of Alice's public key, the salt, and any additional information pertinent parameters (such as the hash algorithm used or the number of rounds used in the HKDF).

4.3 Building a KEM using DH

To build a KEM using only an ECDH, the approach is a bit different than it is with RSA and with "real" KEMs: while the generation function still returns a private and a public key, the encryption function actually generates an ephemeral ECDH key pair, of which the private key is used to generate the shared secret, and the public key is presented as the "ciphertext" of that shared secret. While that "ciphertext" has no real relation to the shared secret, it can be used to re-generate the same shared secret when using the original, non-ephemeral private key.

That means that, when using a KEM, Bob still generates a keypair $\langle sk_b, pk_b \rangle$ and provides his public key pk_b to Alice, but Alice now generates her own ephemeral keypair $\langle sk_a, pk_a \rangle$ and generates the shared secret

$$s = X\ sk_a\ pk_b$$

She then provides pk_a to Bob who generates the shared secret

$$s = X\ sk_b\ pk_a$$

With $ct = pk_a$, this is exactly equivalent to

$$E \rightarrow pk_b \rightarrow s, ct: \tag{1}$$

$$sk_a, ct = G \tag{2}$$

$$s = X\ sk_a\ pk_b \tag{3}$$

or in Python:

[9]:
```python
from cryptography.hazmat.primitives.asymmetric import ec

curve = ec.SECP384R1() # the choice of curves is arbitary for this example

def kem_gen():
    sk = ec.generate_private_key(curve)
    pk = sk.public_key()
    return sk, pk

def kem_enc(pk):
    ephemeral_sk = ec.generate_private_key(curve)
    ct = ephemeral_sk.public_key()
    s = ephemeral_sk.exchange(ec.ECDH(), pk)
    return s, ct

def kem_dec(sk, ct):
    s = sk.exchange(ec.ECDH(), ct)
    return s
```

As noted, using DH shared secrets directly is generally frowned upon (with good reason) whereas using KEM secrets directly is not, so a more secure implementation in Python would look like this:

[10]:
```python
import os
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.hkdf import HKDF

curve = ec.SECP384R1()

def kem_gen():
    sk = ec.generate_private_key(curve)
    pk = sk.public_key()
    return sk, pk

def kem_enc(pk):
    ephemeral_sk = ec.generate_private_key(curve)
    salt = os.urandom(32) # 256 bits may be a bit overkill
    ct = ephemeral_sk.public_key(), salt
    ss = ephemeral_sk.exchange(ec.ECDH(), pk)
    hkdf = HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        info=None
    )
    s = hkdf.derive(ss)
    return s, ct

def kem_dec(sk, ct):
    pubkey, salt = ct
    ss = sk.exchange(ec.ECDH(), pubkey)
    hkdf = HKDF(
        algorithm=hashes.SHA256(),
```

```
        length=32,
        salt=salt,
        info=None
    )
    s = hkdf.derive(ss)
    return s
```

This adds the additional HKDF step to the shared secret generation, and adds a salt value to the "ciphertext" sent from Alice to Bob, but does not otherwise change the API.

The accompanying Python module `ecdh_kem` formats the ciphertext as a Base64-encoded JSON object, and includes the SHA algorithm used for the HKDF in that object alongside the ephemeral public key and the salt value.

5 Using a KEM in protocols

Most protocols that use an DH-type algorithm today already also use a digital signature algorithm, if only to sign the DH public key exchanged between "Alice" and "Bob" (i.e. the client and the server). On this section, we will investigate how a KEM can be used to either augment or replace a DH in two related protocols in the OT space: TLS 1.3[15] and DNP3-SAv6 (not published yet).

5.1 Using a KEM in TLS 1.3

In TLS, the client will initiate the connection and request a session, the server will respond to that request with authenticating certificates and cryptographic materials, optionally requesting the client to authenticate itself as well. A mutual shared secret is established using an ECDH key exchange, and the transaction as a whole is authenticated[15].

An approach to introducing a post-quantum KEM into this handshake is discussed in [26] That paper provides a comprehensive overview of peevious work adding post-quantum cryptography (PQC) to protocols like TLS 1.2, TLS 1.3, SSHv2, and IKEv2, as well as "report(ing) on case studies exploring how two major Internet security protocols, Transport Layer Security (TLS) and Secure Shell (SSH), can be adapted to use post-quantum cryptography, both for confidentiality (via post-quantum key exchange) and authentication (via post-quantum digital signatures)"[26].

Some of the more interesting findings of this paper are:

1. Some of the experiments failed due to the sheer size of the public keys and cryptographic materials being used: two-byte size fields, which allow up 65,535 bytes of payload, were insufficient to accomodate some of the cryptographic materials being lugged around.
2. TLS is extensible in that extensions are explicitly allowed on the `ClientHello` and `ServerHello` messages, and TLS 1.3 was designed with the goal of reducing the number of round trips needed to establish and authenticate a session, but that requires the client to send its KEM public key in the `ClientHello` message. This puts a constraint on how PQC can be implemented in TLS without breaking backward comparibility.
3. A hybrid approach, mixing classical cryptography that can still be trusted (even when Q-time has expired) with PQC being used to back up the classical cryptography, but only insofar as the PQC can be trusted in the long term, and the classical cryptography can be trusted now. That is: if the PQC turns out to be unreliable cryptographically, the only risk the communication is exposed to is the "harvest now, decrypt later" attack vector.

Also, while OT has historically put more emphasis on authentication, the proliferation of KEMs to be used for forward confidentiality was explained as follows:

> *Although there have been several Internet-Drafts and experimental implementations of PQ and/or hybrid key exchange in TLS (…), none of those works considered PQ or hybrid authentication. This is likely to due to the general consensus that confidentiality against quantum adversaries is a more urgent need than authenticity, since*

quantum adversaries could retroactively attack confidentiality of any passively recorded communication sessions, but could not retroactively impersonate parties establishing a (completed) communication session.[26]

5.2 Using a KEM in DNP3-SAv6

DNP3-SAv6 is a novel security layer and protocol designed to be embedded in the DNP3 protocol, specified in IEEE Standard 1815TM[17]. It replaces DNP3-SAv5, which was specified in [27], and addresses concerns and weaknesses identified by J. Adam Crain and Sergey Bratus in [28][18] were co-developed and share large parts of their design.

5.2.1 Current use of ECDH in DNP3-SAv6

Contrary to TLS and similar protocols, DNP3-SAv6 does not allow for ciphersuite negotiation: in DNP3-SAv6 the master and outstation are generally expected to be pre-configured to support a certain set of algorithms, and the master dictates which ones will be used. As the master is the party in the association that starts the association (potentially at the request of the outstation) it is equivalent, in TLS parlance, to the client. This is essentially equivalent to the client always dictating which ciphersuite shall be used for a given association, while the server can only either accept the dictate or refuse the association, according to system policy and local implementation.

While this limits some of the flexibility of the protocol, it also resolves one of the design questions raised by [26]: if any hybrid algorithms are to be used, they must be a well-defined hybrid that can be unambiguously identified by both the master and the outstation using only information provided in the `AssociationRequest` and `AssociationResponse` messages. This constraint also avoids the "combinatorial explosion" TLS 1.2 approaches may suffer from[26].

In DNP3-SAv6, the master station (a.k.a. controlling station in IEC 62351-5[18]) sends its certificates, including its ECDH public key, to the outstation (a.k.a. controlled station) in a message called the `AssociationRequest`. The outstation responds with its own certificate, including its ECDH public key, in an `AssociationResponse`. Both sides then calculate the Encryption and Authentication Update Keys and prove to each other that they have the same keys, using the `UpdateKeyChangeRequest` and `UpdateKeyChangeResponse` messages.

Once the Update Keys are established, only symmetric cryptography is used for authentication and encryption: MACs are used to prove ownership of the Update Keys and to authenticate the handshake transcript, AES is used in the keywrap algorithm (KWA) to establish session keys, and an AEAD is used for authenticated encryption during the sessions. All symmetric keys are at least 256 bitsin size, which even in light of Grover's algorithm[29] is sufficient in this contect. Therefore, only the association handshake is affected by PQC considerations.

5.2.2 Introducing a KEM into the association handshake

Replacing this ECDH-based mechanism with a KEM-based mechanism would either have the master send its certificates and KEM public key to the outstation in the `AssociationRequest` message, and have the outstation send its certificates and the *ct* value in the `AssociationResponse` message, or would have the master indicate its intent to use a KEM in the `AssociationRequest` message using an as-yet-to-be-defined object, have the outstation respond with the KEM public key in the `AssociationResponse` message, and have the master send the *ct* in the `UpdateKeyChangeRequest` message. The choice of which party provides the public key and which party provides the *ct* depends on the relative cost of the key encapsulation and decapsulation functions, discussed below.

The first approach, which places key encapsulation on the outstation but gives the choice of the algorithm to use to the master, can be implemented without making any changes to the specified formats of any of the four messages in the handshake: both the `AssociationRequest` and the `AssociationResponse` message have a reserved space for certificate data for which the specification is flexible enough to allow for the addition of PQC cryptographic materials in the

payload. As discussed below, this space may not be large enough to accomodate the change, which can be specified in a DNP Technical Bulletin after the standard's publication if needed. In this case, no new SPDU types and no changes in the SA version field value would be needed.

The format of the data field *is* currently under-specified: the format borrows from RFC 8446 and refers to section 4.4.2 of that specification, but that format is not sufficiently extensible for our purpose and does not allow explicitly signing the ECDH public key, which means ownership of each party's private keys may never be proven and which moves a possible failure due to tampering to the end of the handshake. (This may be a performance issue and, in some cases, a security issue, as was reported to the CSTF on July 28, 2023.) DNP3-SAv6 also calls for an optional use of an OCSP response for OCSP stapling, which is defined as an extension in TLS. The total size of the TLS construct can reach $2^{24} - 1 + 2^8 - 1$ octets, which exceeds the $2^{16} - 1$ octets allocated in DNP3-SAv6[17].

The second approach would require the addition of an as-yet-to-be-defined object in the `AssociationRequest` for the master to indicate a KEM should be used. This is something that cannot currently be accomodated with the specified `AssociationRequest` syntax. It also requires the master to convey the KEM ciphertext to the outstation in the `UpdateKeyChangeRequest` message, which would be a change of format for that message. If this change were introduced after publication of the new standard incorporating SAv6, this would mean either the addition of a new SPDU (Security Protocol Data Unit) type or a change in SA version.

A pre-publication version of this paper was discussed at the Wednesday, July 19, 2023 meeting of the DNP Cybersecurity Task Force (CSTF) where it was decided to add a field to the `UpdateKeyChangeRequest` message allowing the master to convey a KEM ciphertext to the outstation, thus allowing to switch the burden of calculating the ciphertext from the outstation to the master. Subsequent discusiion on the CSTF mailing list leaves the possibility for additional changes open, however.

5.2.3 Size considerations

There are three size-related considerations that need to be taken into account when considering cryptography in the context of DNP3 and, more generally, OT, IoT, and IIoT systems: the amount of bits-on-the-wire data needed to transfer cryptographic data, the amount of on-device non-volatile memory needed to store cryptographic data, and the amount of run-time memory needed to process cryptographic data and cryptographic functions. M. Kumar's survey in [30] includes statistics on the maximum stack and heap usage on a representative device with a number of different algorithms, showing a wide range of values, only some of which are concerning in terms of memory usage in small devices. The choice of algorithms in an OT system can likely fairly easily avoid those more concerning algorithms in favor of less resource-hungry ones.

The general guideline in the design of DNP3 cybersecurity is to limit the need for non-volatile storage aa much as possible, and to restrict the use of such storage to read-only use during normal operation except for the potential storage of the association's Update Keys, which are intended to be long-lived. The general idea being that a device only need to be able to present its "birth certificate", which may be self-signed and a counter-signed version of which may be provisioned to the master by the system's DNP Authority. Outside of that birth certificate, which can be generated at the device's first boot-up in factory, only 64 bytes of writable non-volatile storage are needed by DNP3-SAv6, as that is the size of the two Update Keys combined[17].

The most important size consideration, then, is the a,ount of bits-on-the-wire bandwidth needed to set up an association. This will typically only happen once in the drvice's life-time and once the association is set up only symmetric cryptography is used for any session in that association. The asymmetric-cryptography-related cryptographic material that needs to be transferred consists of X.509 certificates, data needed for key exchange or key encapsulation, and digital aignatures.

At the time of this writing, the total size of both master and outstation certificate data is limited, in both the `AssociationRequest` and the `AssociationResponse` message, to 65,535

bytes. This precludes the use of some PQC algorithms, for which the size of a single public key can exceed 319 kiB and go up to 1.4 MiB for NTS-KEM using certain parameters, for example[31]. This issue can be remediated by adding a third octet to the size fields in those two messages, an option that is currently under discussion with the DNP CSTF. Care will still need to be taken to make sure an association handshake does not interfere with the operational communication requirements of other devices on the same network [32].

Not all PQC algorithms have such large size requirements, however, but there does appear to be a trade-off between the bits-on-the-wire size of cryptographic material and the computational cost of the cryptographic functions to be executed,

5.2.4 Computational cost considerations

Manufacturing cost is always a significant factor when designing and developing OT, IoT, and IIoT devices, especially when those devices are intended to be high-volume and low-cost, and their ability for local processing and communications is seen as secondary-at-best to their primary function of protecting and automating critical infrastructure such as the power system. These considerations weigh heavily in many discussions around how cybersecurity is best designed into protocols such as DNP3. While recent development of ARM CPUs has meant concerns around processing power have become less of an issue for these systems, this "free" processing power has also lead to the use of Linux-based operating systems that are often not as optimized for domain-specific applications as had been the case for earlier generations of devices. Such optimization existed mostly out of necessity whereas, today, it represents a cost that can no longer be justified.

One important aspect of these types of systems, which is in part due to the way critical infrastructure development is financed in many countries. is that the devices that make up these systems are typically installed in the field and not replaced for a decade or more. This means that while it is possible to rely on firmware and software upgrades to improve security it is typically not possible to rely on improvements in hardware. Combining that with the fact that many of these devices have much of their computational resources already allocated to the device's primary function and that communications and cybersecurity are often treated as an after-thought the available resources for cyber-related computation can be extremely limited.

In part due to these considerations, DNP3-SAv6 relies heavily on symmetric cryptography, for which hardware acceleration is ubiquitously available and which remains secure in the presence of adversaries with quantum computers. This again constrains the use of a PQC to the Association handshake and may partly drive the choice of both the algorithms to use and on which device (the master or the outstation) key encapsulation takes place. While M. Kumar's survey of post-quantum peformance in [30] shows, perhaps surprisingly (at least to me) encapsulating keys can be significantly less costly than decapsulating them, in which case it would make sense to perform encapsulating on the outstation, it may be too early to tell whether that will be true for all KEM standards, so it would be advisable to leave the door open to performing encapsulation on the master when needed.

Additionally, it should be noted that, as DNP3-SAv6 supports both mutual authentication and data encryption and, once the association is established, does not require asymmetric cryptography for subsequent sessions that nevertheless retain both mutual authentication and data encryption, the additional overhead a TLS connection would require for its sessions becomes unnecessary. (TLS connections have their own asymmetric cryptography for every session unless the PSK mode is used. The PSK mode is disallowed by IEC 62351-3, which applies in most situation where DNP3 would be used.) I therefore see no reason to recommend using TLS in conjunction with DNP3, and in some cases would actually recommend against it, especially when communicating with resource-constrained devices that are part of critical infrastructure.

5.2.5 Using hybrid approaches

Cryptanalysis takes time. For the next decade or so, we will be in an inter-regnum of sorts before classical cryptography is no longer safe, but while post-quantum cryptographic standard algorithms have not yet been sufficiently analysed yet to know whether they are safe or not. Hopefully, there will be some overlap between the time safe post-quantum cryptographic standards are widely available, and the time quantum computers that can break classical cryptography are widely available. In the mean time, for some systems for which cybersecurity is sufficiently critical, it will make sense to use both classical and post-quantum cryptography, one in combination with the other.

Where digital signatures are concerned, this means that any data that needs to be authenticated should be signed using both a classical algorithm like RSA[33], DSA[34], ECDSA[35], or EdDSA[36], and a post-quantum algorithm like CRYSTALS-Dilithium[37], Falcon[38], or SPHINCS+[39], and both signatures should be verified. Where key exchanges and key encapsulations are concerned, this means that the symmetric key used as a result of key negotiation should be a combination of the result of the classical and post-quantum algorithms used such that even if either one (but not both) of the keys is compromised, the key resulting from the combination is still secure.

It is fairly common practice to use a key derivation function for which all but one inputs are public, and use the derived key for encryption. In such a context, either one of the keys should therefore be consdiered public for the purpose of the choice of a key derivation function. Giacon et al. in [40] describe an efficient yet secure way of combining keys from different KEMs, which XORs the results of any number of standard pseudo-random functions (such as HKDF) that take the KEM ciphertext and output secret as inputs, in lieu of the salt an IKM. A similar approach can be used combining the output of a DH and a KEM, especially if the DH has been previously used to model a KEM, as shown in this paper.

There will, for some time, be systems in which either the master or the outstation will be capable of using PQC while the other station does not have that capability yet. This may require some new, optional extensions in some of the messages exchanged between master and outstation, which will for now be left to future discussion.

5.2.6 Proposed changes to DNP3-SAv6 before publication

To prepare for the addition of PQC to DNP3-SAv6, and to avoid having to develop a DNP3-SAv7 in the near future, some changes to the protocol as currently specified are needed. The recommendations below are intended to (a) leave the door open for some design and implementation choices for which we currently lack sufficient information to be made later, and (b) correct minor issues and future-proof the protocol.

Use ASN.1 for `AssociationRequest` and `AssociationResponse` message payloads: using RFC 8446 notation for the `certificate data` field, while allowing implementations to re-use library code from their TLS implementations where appropriate, restricts the ability for the DNP Cybersecurity Task Force and the DNP Technical Committee to extend on the syntax, and reduces the flexibility of the specification in the long term. (Mea culpa)

Increase the size of the `certificate data length` fields: the `AssociationRequest` and `AssociationResponse` messages currently restrict the size of the certificate data to be transferred to $2^{16} - 1$ octets. This limitation is likely too small for PQC cryptographic data to be transferred. I recommend increasing the data length field to three octets, which will limit the size of $2^{24} - 1$ octets at the cost of one extra octet per message. Using four octets, and allowing $2^{32} - 1$ octets of payload, may be overkill, though it may also be easier to implement for library writers.

Add a one-octet `flags` field to the `UpdateKeyChangeRequest` message: a `flag` field in the `UpdateKeyChangeRequest` would allow the master to indicate that, while it did not receive PQC KEM data in the `AssociationResponse` message, it knows how to handle

such data – much like the "higher available" flag in the `AssociationRequest` message. This would be a one-octet reserved byte near the start of the SPDU that, for now at least, can replace the previously proposed additional length field (the presence of which we can indicate by setting a bit in the flag field)[41].

Require a signature on the ECDH public keys in the `AssociationRequest` and `AssociationResponse` messages, thus proving ownership of the identity certificate's private keys. The Association handshake was based on the TLS handshake including its use of a signature over the handshake transcript to authenticate the handshake but, unlike TLS, it uses an HMAC over the transcript rather than a certificate-based signature. This means that if the ECDH public keys are not signed with the private key of the identity certificate, pownership of that private key is never proven. This is not an issue if the ECDH public key is either embedded in the identity certificate or is the same key, but it is an issue if detached or ephemeral ECDH keys are used -— something I think will be common. We should therefore require a signature on those keys[42].

Clarify the specification w.r.t the `certificate data` fields: clearly specify how the `certificate data` fields in the `AssociationRequest` and `AssociationResponse` messages are to be interpreted, what is optional, and what isn't (especially adding the possibility for optionally-parsed data). Define an extension syntax that the recipient can ignore if it can't parse it, so both parties can signal their capabilities while permitting fall-back.

6 Conclusion and future work

Using the lessons learned from other protocols, and leveraging current libraries to simulate a KEM using existing ECDH functions, it is possible to experiment with new constructs to validate some of the possible ways a KEM can be integrated with a novel protocol like DNP3-SAv6 while retaining its most important performance characteristics. These experiments should include tests in which hybrid scenarios are explored, such as scenarios where the master requests the use of ECDH in addition to a PQC KEM, scenarios where the outstation indicates it can use a PQC KEM even if the master hasn't requested it, scenarios where the master requests the use of PQC KEMs but a man in the middle attempts a downgrade attack (which needs to be overcome at least if classical cryptography is still available), etc. Solutions to such scenarios were discussed but do still need further development.

With the proposed changes to the current standard prior to its publication, however, it should be possible to accomodate such changes as necessary using the normal amendmend procedures available in the DNP Technical Bulletins and subsequent revisions to the standard without having to change the DNP3-SA protocol version or breaking backward compatibility. Such technical bulletins should require post-quantum signatures to be verified when possible, and require that verification to pass (i.e. if the device *can* verify a post-quantum signature and one is presented, it shall do so and shall only accept validity if all available sigantures pass, even if equivalent classical signatures have already passed). The authenticated part of the message should also include an indicator to which signatures should be expected to be present, so a signature cannot simply be omitted from the message. This could be accomplished by indicating the use of a hybrid signature algorithm rather than using two disjoint signatures.

A technical bulletin introducing a hybrid classical-and-PQC key negotiation should also specify how those algorithms are to be implemented, requiring the master and outstation to use both when available, but allowing a fall-back if either station is not capable of PQC. A mechanism to introduce the notion of optional extensions before standardization would be required for forward compatibility, as discussed. A new error code in the error SPDU may also be necessary to restart the handshake if the outstation requires PQC but the master doesn't offer it.

References

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[2] T. Hoefler, T. Häner, and M. Troyer, "Disentangling hype from practicality: on realistically achieving quantum advantage," *Communications of the ACM*, vol. 66, no. 5, pp. 82–87, 2023.

[3] J. R. Biden, "National Security Memorandum on Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems," 2022.

[4] a. E. T. John Preuß Mattsson, Ben Smeets, "Quantum-Resistance Cryptography."

[5] E. Parker and M. J. Vermeer, "Estimating the energy requirements to operate a cryptanalytically relevant quantum computer," *arXiv preprint arXiv:2304.14344*, 2023.

[6] "Post-Quantum Cryptography." https://web.archive.org/web/20230630015245/https://csrc.nist.gov/projects/post-quantum-cryptography. Accessed: 2023-07-24.

[7] P. Gajland, B. de Kock, M. Quaresma, G. Malavolta, and P. Schwabe, "Swoosh: Practical Lattice-Based Non-Interactive Key Exchange," *Cryptology ePrint Archive*, 2023.

[8] "Selected Algorithms 2022." https://web.archive.org/web/20230703110711/https://csrc.nist.gov/projects/post-quantum-cryptography/selected-algorithms-2022. Accessed: 2023-07-24.

[9] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM." Cryptology ePrint Archive, Paper 2017/634, 2017. https://eprint.iacr.org/2017/634.

[10] A. Juels, J. Kelley, R. Tamassia, and N. Triandopoulos, "Falcon Codes: Fast, Authenticated LT Codes (Or: Making Rapid Tornadoes Unstoppable)." Cryptology ePrint Archive, Paper 2014/903.

[11] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS – Dilithium: Digital Signatures from Module Lattices." Cryptology ePrint Archive, Paper 2017/633, 2017. https://eprint.iacr.org/2017/633.

[12] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS+ Signature Framework." Cryptology ePrint Archive, Paper 2019/1086, 2019. https://eprint.iacr.org/2019/1086.

[13] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, IEEE, 1994.

[14] T. Dierks and E. Rescorla, "RFC 5246: The transport layer security (TLS) protocol version 1.2," 2008.

[15] E. Rescorla, "RFC 8446: The Transport Layer Security (TLS) Protocol version 1.3," 2018.

[16] T. Ylonen, "RFC 4253: The secure shell (SSH) transport layer protocol," 2006.

[17] "IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3)," *IEEE Std 1815-202x (Revision of IEEE Std 1815-2012) – work in progress.*

[18] "IEC 62351-5:2023 - Power systems management and associated information exchange - Data and communications security - Part 5: Security for IEC 60870-5 and derivatives," 2023.

[19] S. Fluhrer, D. McGrew, P. Kampanakis, and V. Smyslov, "Postquantum preshared keys for IKEv2," *Internet Engineering Task Force, Internet-Draft draft-ietf-ipsecme-qr-ikev2-08*, 2019.

[20] P. E. Hoffman, "The transition from classical to post-quantum cryptography," *Internet Engineering Task Force, Internet-Draft drafthoffman-c2pq-05*, 2019.

[21] F. Kiefer and K. Kwiatkowski, "Hybrid ECDHE-SIDH key exchange for TLS," 2018.

[22] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, "Post-quantum authentication in TLS 1.3: a performance study," *Cryptology ePrint Archive*, 2020.

[23] D. Ott, C. Peikert, *et al.*, "Identifying research challenges in post quantum cryptography migration and cryptographic agility," *arXiv preprint arXiv:1909.07353*, 2019.
[24] W. Brattain and J. Bardeen, "Quantum and the Cybersecurity Imperative," *Dig tal Debates*, p. 15, 2022.
[25] M. Barenkamp, "„Steal Now, Decrypt Later" Post-Quantum-Kryptografie & KI," *Informatik Spektrum*, vol. 45, no. 6, pp. 349–355, 2022.
[26] E. Crockett, C. Paquin, and D. Stebila, "Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH." Cryptology ePrint Archive, Paper 2019/858, 2019. https://eprint.iacr.org/2019/858.
[27] "IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3)," *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pp. 1–821, 2012.
[28] J. A. Crain and S. Bratus, "Bolt-on security extensions for industrial control system protocols: A case study of DNP3 SAv5," *IEEE Security & Privacy*, vol. 13, no. 3, pp. 74–79, 2015.
[29] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.
[30] M. Kumar, "Post-quantum cryptography Algorithm's standardization and performance analysis," *Array*, vol. 15, pp. 100–242, 2022.
[31] M. Albrecht, C. Cid, K. G. Paterson, C. J. Tjhai, and M. Tomlinson, "NTS-KEM," *NIST PQC Round*, vol. 2, pp. 4–13, 2019.
[32] Éric Thibodeau, "Comment on the question of adding a third octet to the AssociationRequest and AssociationResponse certificate data size fields." https://groups.google.com/g/dnp3sav6/c/-YzYlZutBBM/m/T5hjM8d9AAAJ, 2023.
[33] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
[34] F. PUB, "Digital signature standard (dss)," *FIPS PUB*, pp. 186–192, 2000.
[35] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, pp. 36–63, 2001.
[36] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," tech. rep., 2017.
[37] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.
[38] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon," *Post-Quantum Cryptography Project of NIST*, 2020.
[39] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 2129–2146, 2019.
[40] F. Giacon, F. Heuer, and B. Poettering, "KEM Combiners," in *Public-Key Cryptography – PKC 2018* (M. Abdalla and R. Dahab, eds.), (Cham), pp. 190–218, Springer International Publishing, 2018.
[41] R. Landheer-Cieslak, "Suggestion to the CSTF to add a `flag` field to the `UpdateKeyChangeRequest` message.." https://groups.google.com/g/dnp3sav6/c/AnRriyp5w14/m/atqpYb2GAAAJ, 2023.
[42] R. Landheer-Cieslak, "Suggestion to the CSTF to require signatures on ECDH keys." https://groups.google.com/g/dnp3sav6/c/EwDPogMcsX4/m/Nrlrnig8AgAJ, 2023.
[43] D. McGrew, K. Igoe, and M. Salter, "Fundamental elliptic curve cryptography algorithms," tech. rep., 2011.
[44] H. Krawczyk and P. Eronen, "HMAC-based extract-and-expand key derivation function (HKDF)," tech. rep., 2010.
[45] C. Paquin, D. Stebila, and G. Tamvada, "Benchmarking Post-Quantum Cryptography in TLS." Cryptology ePrint Archive, Paper 2019/1447, 2019. https://eprint.iacr.org/2019/1447.

[46] K. Kwiatkowski, N. Sullivan, A. Langley, D. Levin, and A. Mislove, "Measuring TLS key exchange with post-quantum KEM," in *Workshop Record of the Second PQC Standardization Conference. https://csrc.nist.gov/CSRC/media/Events/Second-PQC-Standardization-Conference/documents/accepted-papers/kwiatkowski-measuring-tls.pdf*, 2019.

[47] L. Malina, L. Popelova, P. Dzurenda, J. Hajny, and Z. Martinasek, "On feasibility of post-quantum cryptography on small devices," *IFAC-PapersOnLine*, vol. 51, no. 6, pp. 462–467, 2018.

[48] P. Kampanakis, P. Panburana, E. Daw, and D. Van Geest, "The viability of post-quantum X.509 certificates," *Cryptology ePrint Archive*, 2018.

[49] A. A. Giron, "Migrating applications to post-quantum cryptography: Beyond algorithm replacement," *Cryptology ePrint Archive*, 2023.

[50] O. S. Althobaiti and M. Dohler, "Quantum-resistant cryptography for the internet of things based on location-based lattices," *IEEE Access*, vol. 9, pp. 133185–133203, 2021.