

# Post Quantum KEM authentication in SPDm for secure session establishment

Jiewen Yao\*, Anas Hlayhel, Krystian Matusiewicz

**Abstract**—The Secure Protocol and Data Model (SPDM) is a protocol that can be used to authenticate a hardware identity and to establish secure communication sessions between devices. A number of recent research efforts has been investigating ways of enabling post-quantum cryptography (PQC) in the current SPDM version 1.2. However, measurements show that the current PQC digital signature algorithms are a major performance bottleneck, which may become an obstacle to the adoption of quantum-secure SPDM versions. This paper describes a new way to use PQC key establishment algorithms to perform identity authentication. Our experimental results show it can achieve much better performance, especially on the device side.

**Index Terms**—SPDM, post-quantum, key encapsulation method, KEM, device authentication, device secure session.

## I. INTRODUCTION

The Secure Protocol and Data Model (SPDM) 1.2 specification<sup>1</sup> defines a protocol that can be used for device identity establishment, device authentication, measurement collection and device secure session establishment. The SPDM protocol is a standard for the device community and has been adopted by multiple other standard groups, including Peripheral Component Interconnect (PCI), Compute Express Link (CXL), Mobile Industry Processor Interface (MIPI), and Trusted Computing Group (TCG).

The current SPDM 1.2 specification uses traditional asymmetric cryptographic algorithms, such as RSA, ECDSA, and EdDSA for digital signatures, and ephemeral Diffie-Hellman over finite fields (FFDHE) or elliptic curves (ECDHE) for key establishment. A simple way to add PQC support for SPDM is to use NIST defined PQC digital signature algorithms to replace the traditional asymmetric algorithms, and use NIST key establishment algorithms to replace the ephemeral Diffie-Hellman. However, the PQ-SPDM prototype [1] that implements NIST PQC algorithms shows that the signing with PQC algorithms takes significantly longer than with classic asymmetric algorithms.

### A. Our Contributions

In this work we define a key encapsulation method (KEM) based authentication mechanism for SPDM protocol (we call it KEM-SPDM) in order to replace digital signatures in authenticated secure session establishment in the current SDPM.

We redesigned SPDM messages to make use of KEM algorithms and achieved the same goal without digital signatures. We also prototyped the solution implementing the new method and collected performance benchmarks.

The paper is organized as follows. We start with a brief recap of SPDM in Section II. Section III discusses the KEM-SPDM Secure Session Establishment and Section IV provides security analysis for the KEM-SPDM. Finally, we report the results of our proof-of-concept implementation in Section V. Security proofs are provided in the full version of the paper<sup>2</sup>.

### B. Related Work

Authenticated key exchange is an important topic in system security and has been studied for a long time. We notice that KEM-based authentication [2] was adopted by several standard proposals, such as PQ-Wireguard [3], KEMTLS [4]–[6], and Post-Quantum Hybrid Public Key Encryption (PQ-HPKE) [7]. We compare KEM-SPDM with those in Section IV-C. Authentication without digital signatures can be a solution to latency issues. Such work motivates us to create KEM-SPDM. Multi-Stage based proofs were introduced in the analysis of the QUIC protocol [8]. It was also used to prove the security of TLS1.3 [9] and KEMTLS [5], [6]. Our proof in the full version largely adopts the multi-stage security model of KEMTLS, with adjustments made to fit the KEM-SPDM scheme.

## II. SPDm BACKGROUND

### A. SPDm Device Identification

During the device manufacturing phase, each device is provisioned with a device certificate chain. This certificate chain can be treated as the device identity. The device certificate chain includes all the certificates from a trusted root certificate authority (CA) certificate to a device specific leaf certificate. The device certificate contains the public key that corresponds to the device's private key. The root CA endorses the device's key pair through the certificate chain. At runtime, an SPDm initiator (requester) may use a GET\_CERTIFICATE message to ask an SPDm device (responder) to return its certificate chain as the identity. Vice versa, the SPDm responder may use a GET\_CERTIFICATE to get the certificate chain from the SPDm initiator.

As an alternative to the certificate chain, the SPDm entity may use a raw public key, as the identity information, to establish trust without the certificate. For example, an SPDm initiator can provision an SPDm responder's raw public key in a trusted environment during the manufacturing phase.

<sup>2</sup>For proofs submitted with the paper, see the Supplementary Materials in this submission. Proofs will be also part of the full version of the paper we plan to post on eprint.iacr.org if the paper gets accepted.

<sup>1</sup><https://www.dmtf.org/dsp/DSP0274>

## B. SPDM Secure Session Establishment

The SPDM specification allows two devices to establish an authenticated secure communication channel, similar to the network Transport Layer Security (TLS) 1.3. An SPDM requester and an SPDM responder may use an authenticated key exchange protocol to derive a set of session keys.

According to the SPDM specification, the authenticated secure session could be one-way authentication, where the initiator authenticates the responder, or mutual authentication, where both entities authenticate each other.

SPDM defines two ways to build an authenticated secure session, based on either a pre-shared key (PSK) or an asymmetric key exchange. Since the PSK based key exchange involves only hash-based message authentication code (HMAC), which is still secure against quantum adversaries, we focus only on asymmetric key exchange in this paper.

The asymmetric key exchange in SPDM 1.2 uses either the finite-field-based Diffie-Hellman ephemeral (FFDHE) key exchange or the Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange.

## III. KEM-SPDM SECURE SESSION ESTABLISHMENT

In KEM-SPDM we replace DHE with a key encapsulation mechanism (KEM) based authenticated key exchange. Fig. 2 shows the high-level view of KEM-SPDM handshake message flows with mutual authentication. The steps that are marked with an asterisk are not required for one-way authentication.

There are three invocations of KEM algorithms in the flow:

- $KEM_e$ : The ephemeral KEM that is used to ensure forward secrecy. The ephemeral secret key ( $sk_e$ ) and public key ( $pk_e$ ) pair are generated every session by the initiator. Its Encap() function returns a secret ( $ss_e$ ) and its corresponding cipher text ( $ct_e$ ).
- $KEM_r$ : The static KEM on the responder side authenticates the responder. The secret key ( $sk_r$ ) and public key ( $pk_r$ ) are generated once. It has the corresponding cipher text ( $ct_r$ ) and shared secret ( $ss_r$ ).
- $KEM_i$ : The static KEM on the initiator side authenticates the initiator. The initiator's secret key ( $sk_i$ ) and public key ( $pk_i$ ) are generated once for optional mutual authentication case. It has the corresponding ciphertext ( $ct_i$ ) and shared secret ( $ss_i$ ).

The KEM-related actions are highlighted in blue, secret keys ( $sk_e$ ,  $sk_r$ ,  $sk_i$ ) are highlighted in red while public keys ( $pk_e$ ,  $pk_r$ ,  $pk_i$ ) and ciphertexts ( $ct_e$ ,  $ct_r$ ,  $ct_i$ ) are highlighted in violet.

In the description we omit some of the protocol-specific fields (like opaque data or transcripts of commands like GET\_VERSION) that are not crucial to the security analysis of the enhanced protocol.

### A. Key Derivation

The SPDM key derivation relies on a standard HMAC-based key derivation function (HKDF) to derive the set of necessary symmetric keys out of established shared secrets. HKDF consists of two functions: HKDF-Extract() that

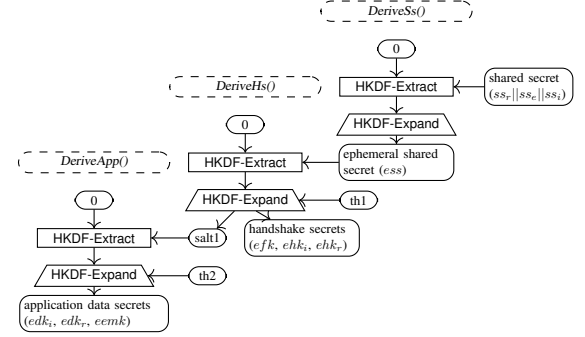


Fig. 1. KEM-SPDM key derivation scheme. A new stage  $DeriveSs()$  is added to the existing SPDM steps  $DeriveHs()$  and  $DeriveApp()$ .

takes an optional random salt value and an Input Keying Material (IKM) and outputs a Pseudo-Random Key (PRK) and HKDF-Expand that uses the PRK and an optional context-specific information (info), and outputs the final Output Keying Material (OKM) as the derived key.

KEM-SPDM secure session establishment employs a couple of key derivation steps using the above functions. Fig. 1 shows the high-level flow for those derivation steps.  $DeriveHs$  and  $DeriveApp$  already exist in SPDM specification and  $DeriveSs$  is a new step introduced in this work.

- $DeriveSs()$  generates ephemeral shared secret ( $ess$ ) for the session.
  - input: KEM-generated shared secrets ( $ss_r$ ,  $ss_e$ ,  $ss_i$ )
  - output: ephemeral shared secret ( $ess$ )
- $DeriveHs()$  generates handshake secrets.
  - input: The ephemeral shared secret ( $ess$ ).
  - input: The transcript hash 1 ( $th1$ ) for handshake. It is a digest of the negotiated protocol version, capability and algorithm as the field  $vca$ , identities of the responder and initiator as their public keys  $pk_r$ ,  $pk_i$ , the KEY\_EXCHANGE message, the KEY\_EXCHANGE\_RSP message without HMAC.
  - output: The initiator-direction ephemeral handshake key ( $ehk_i$ ) is to encrypt the handshake message from the initiator.
  - output: The responder-direction ephemeral handshake key ( $ehk_r$ ) is to encrypt the handshake message from the responder.
  - output: The ephemeral finished key ( $efk$ ) is used to provide HMAC in handshake message for explicate authentication.
  - output: The salt ( $salt1$ ) is used as input for  $DeriveApp()$ .
- $DeriveApp()$  generates the application data secret keys.
  - input: The salt ( $salt1$ ).
  - input: The transcript hash 2 ( $th2$ ) for application data. It is the concatenation of the negotiated protocol version, capability and algorithm as the field  $vca$ , identities of the responder and initiator as their public keys  $pk_r$ ,  $pk_i$ , the KEY\_EXCHANGE message,

the KEY\_EXCHANGER\_RSP message, the plaintext FINISH message and the plaintext FINISH\_RSP message.

- output: The initiator-direction ephemeral application data key ( $edk_i$ ) is to encrypt the application data from the initiator.
- output: The responder-direction ephemeral application data key ( $edk_r$ ) is to encrypt the application data from the responder.
- output: The ephemeral export master key ( $eemk$ ) is for application specific usage.

## B. Message Flows

Figure 2 shows secure session establishment flows of SPDM modified to use KEM-based authentication. Since SPDM relies on both entities to get the identification information earlier, (e.g. via SPDM GET\_CERTIFICATE or public key provisioning) the session establishment messages do not include initial identity transmission. The identity information is only included in the transcripts for identity binding.

The protocol steps are as follows.

1) *First Message*: The initiator uses  $KEM_e$  algorithm to generate an ephemeral key pair ( $sk_e$  and  $pk_e$ ) and uses the responder's  $KEM_r$  algorithm to generate a secret ( $ss_r$ ) and its encapsulated ciphertext form ( $ct_r$ ) using the responder's public key ( $pk_r$ ). Then it generates its part of session ID ( $reqid$ ) and a random number ( $rand_i$ ) that is used to prevent reply attacks. The first message  $keyex_i$  is the sequence of  $reqid$ ,  $rand_i$ , the ephemeral public key ( $pk_e$ ), the initiator-specific opaque data, and the encapsulation of the shared secret ( $ct_r$ ).

2) *Second Message*: After receiving  $keyex_i$  message, the responder uses its private key ( $sk_r$ ) to decapsulate the ciphertext ( $ct_r$ ) and get the shared secret ( $ss_r$ ). It uses the ephemeral public key ( $pk_e$ ) to generate an ephemeral shared secret ( $ss_e$ ) and its encapsulation into a ciphertext ( $ct_e$ ). If mutual authentication is required, it also uses initiator's  $KEM_i$  algorithm to generate and encapsulate the third shared secret ( $ss_i$ ) into ciphertext ( $ct_i$ ) using initiator's public key ( $pk_i$ ). At this point, the responder has all three shared secrets and can feed them to  $DeriveSs$  to derive the final ephemeral shared secret ( $ess$ ).

The responder also generates a random value ( $rspid$ ) that is the second half of the 32-bit session ID ( $reqid||rspid$ ) and another reply-protecting 256-bit random number ( $rand_r$ ).

The first part of responder's message  $keyex1_r$  is formed as the concatenation of  $rspid$ ,  $rand_r$  and ciphertexts  $ct_e$ ,  $ct_i$ .

To bind to the session data, the responder calculates the transcript hash ( $th1$ ). The  $th1$  and the ephemeral secret ( $ess$ ) are used as inputs to handshake key derivation process  $DeriveHs$  that produces the handshake keys ( $efk$ ,  $ehk_i$ ,  $ehk_r$ , and  $salt1$ ).

The next step is to MAC the transcript with the ephemeral finished key ( $efk$ ) and generate the MAC ( $mac_r$ ). Now, the second message  $keyex_r$  sent back to the initiator is  $keyex1_r$  with appended authentication tag  $mac_r$ .

3) *Third Message*: Once the initiator receives the  $keyex_r$  message, it decapsulates the ciphertext with its ephemeral

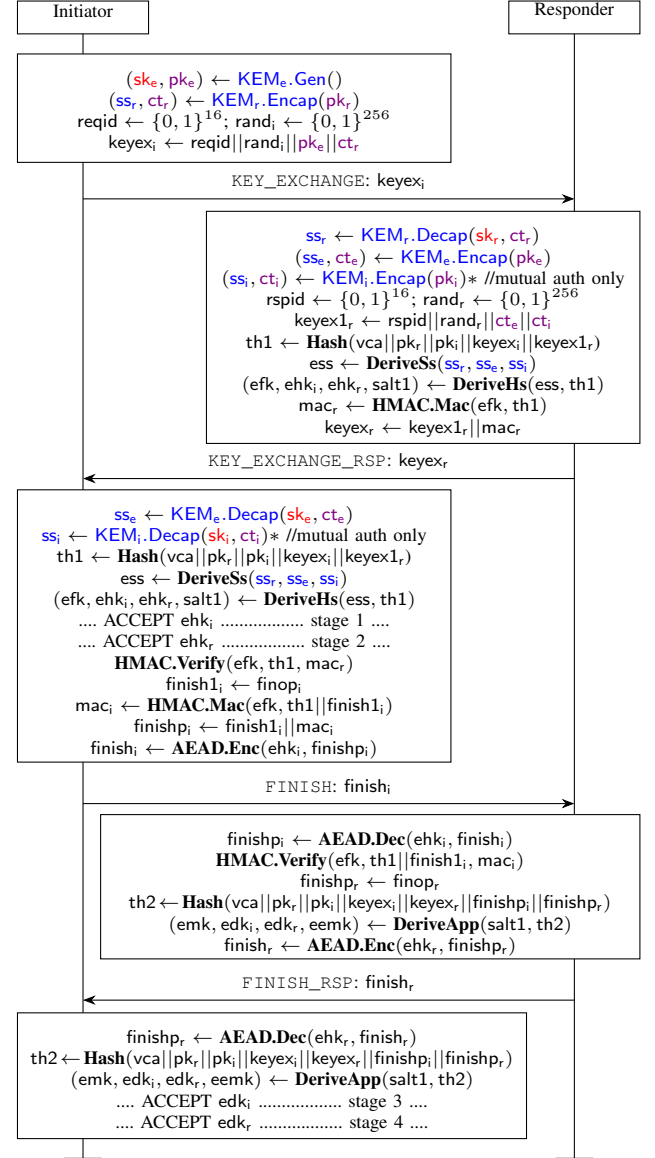


Fig. 2. KEM-SPDM Secure Session Establishment Flow

private key ( $sk_e$ ) to get the ephemeral shared secret ( $ss_e$ ). If mutual authentication is required, the initiator also decapsulates the ciphertext ( $ct_i$ ) with its long term private key ( $sk_i$ ) and gets the third shared secret ( $ss_i$ ). Having all three shared secrets,  $DeriveSs$  key derivation is used to derive the shared secret ( $ess$ ). The same SPDM handshake key derivation ( $DeriveHs$ ) scheme is used by the initiator that takes ( $ess$ ) and ( $th1$ ) to derive the handshake keys ( $efk$ ,  $ehk_i$ ,  $ehk_r$ , and  $salt1$ ). These keys should match the ones derived by the responder. We call “stage 1” the point after  $ehk_i$  is accepted and “stage 2” the point after  $ehk_r$  is accepted.

Then, the initiator follows the same process to compute the MAC of the transcript and verifies the received MAC ( $mac_r$ ) with the key  $efk$ . If the verification fails, the session is terminated.

At this point the initiator starts preparing the FINISH message to complete the handshake: it starts with a fixed opcode string ( $finop_i$ ), creates the transcript hash  $th1$  and computes a MAC over a message which is the concatenation of  $th1$  and the initiator generated FINISH message ( $finish1_i$ ). The initiator needs to MAC the transcript with the ephemeral finished key ( $efk$ ) and generate the MAC ( $mac_i$ ). The full FINISH message ( $finishp_i$ ) is the concatenation of the requester generated FINISH message ( $finish1_i$ ) and the MAC ( $mac_i$ ). The final FINISH message ( $finish_i$ ) is the AEAD of the full FINISH message ( $finishp_i$ ) with initiator-direction handshake key ( $ehk_i$ ).

#### 4) Fourth Message: FINISH\_RSP (responder to initiator)

Once the responder receives FINISH message, it performs AEAD decryption and verifies the AEAD MAC with the ephemeral finished key ( $efk$ ). If the MAC verification fails, the responder terminates the session.

As the final step, the responder creates the FINISH\_RSP message ( $finishp_r$ ). With the transcript hash for the application data keys ( $th2$ ) and the salt ( $salt1$ ), the responder uses *DeriveApp* to derive the application data keys ( $edk_i$ ,  $edk_r$ ,  $eemk$ ). Then the responder sends the final FINISH\_RSP message ( $finish_r$ ), which is the AEAD of the responder generated FINISH\_RSP message ( $finishp_r$ ) with the responder-direction handshake key ( $ehk_r$ ).

5) *Initiator Finalization*: After the initiator receives the FINISH\_RSP, it performs AEAD decryption and verifies the AEAD MAC. If the AEAD MAC verification passes, then the secure session is established between the initiator and the responder. The initiator follows the same process to calculate the transcript hash for the application data ( $th2$ ) then uses *DeriveApp* and  $salt1$  to derive the application data keys ( $edk_i$ ,  $edk_r$ ,  $eemk$ ). When  $edk_i$  is accepted, the protocol enters “stage 3”. When  $edk_r$  is accepted, the protocol enters “stage 4”.

## IV. SECURITY ANALYSIS

In this section, we discuss the security of KEM-SPDM and compare it with other similar solutions.

### A. Security Properties

The KEM authentication SPDM can achieve the following security properties:

1) *Key Indistinguishability*: The adversary should not be able to distinguish a real session key from a random key. The goal of the adversary is to use the Test() query to guess the hidden random chosen bit. There is a restriction that the adversary cannot perform Test() query on a revealed session.

A set of unique session keys are created for every SPDM session. Revealing other sessions should not bring any advantage to the adversary.

2) *Forward Secrecy*: Forward secrecy is a way to guarantee the session data is still secure even if one of the session parties is corrupted. The security of session data only relies on the security of session keys. As long as the session key and session key related information are erased, the session data is secure.

In stage 1, initiator-direction ephemeral handshake key ( $ehk_i$ ) is derived. In stage 2, responder-direction ephemeral handshake key ( $ehk_r$ ) is derived. Because the ephemeral  $KEM_e$  is involved in the key derivation, we achieved forward secrecy in stage 1 and stage 2. However, because keys are not confirmed yet, the explicit authentication is not achieved yet.

In stage 3 and stage 4, the forward secrecy is still maintained, because the salt ( $salt1$ ) from stage 1 is input for the key in stage 3 and stage 4. With MAC verification, the explicit authentication is also achieved.

3) *Authentication*: The KEM authentication supports one-way or mutual authentication. In one-way authentication cases, the initiator needs to authenticate the responder. But the responder does not know the identity of the initiator, which might be an active attacker. In mutual authentication, both entities need to authenticate each other.

The initiator authenticates the responder in below phases:

- 1) *Implicit authentication*: In first KEY\_EXCHANGE message, the initiator uses the responder's public key to perform KEM encapsulation. Only the private owner can decapsulate and derive the  $ss_r$ .
- 2) *Key confirmation*: In second KEY\_EXCHANGE\_RSP message, the responder uses its own private key to perform KEM decapsulation, and derives the  $ss_r$ . This  $ss_r$  will be used to derive the finished key  $efk$ , which will be used to MAC the transcript for KEY\_EXCHANGE\_RSP.
- 3) *Explicit authentication*: After the initiator receives the KEY\_EXCHANGE\_RSP, it verifies the MAC to ensure that the responder does have the private key.

If mutual authentication is required, the responder authenticates the initiator in a similar fashion.

In summary, Stage 1 and stage 2 provide forward secrecy but no authentication. Stage 3 and stage 4 provide unilateral authentication, and mutual authentication respectively.

The security of KEM-SPDM relies on the security of KEM, HKDF and HMAC functions. Security proofs are provided in the full version of the paper.

### B. Other Security Properties

1) *KEM Algorithm*: KEM-SPDM protocol is designed to resist active attacks. This is obtained through the choice of a KEM algorithm that is secure against an active adversary. The notion of indistinguishability of KEMs against an active and adaptive attack (i.e. IND-CCA2) was introduced and proven in [10]. Later it was extended to CPA and CCA1 attacks by [11]. In our protocol, the ephemeral KEM used for key exchange and the long term KEM used for authentication have a slightly different strength against the active adversary.

For key exchange, the ephemeral KEM ( $KEM_e$ ) needs to be IND-ICCA-secure in order to resist active attacks. Since an active attacker has one chance to replace the ciphertext  $CT_e$  of the responder (since  $CT_e$  is unique per session),  $KEM_e$  needs only be secure against a single decapsulation query; hence the IND-ICCA notion.

For entity authentication, the long term KEMs ( $KEM_i$  and  $KEM_r$ ) need to be IND-CCA-secure in order to resist active attacks. Specifically, if  $KEM_i$  is IND-CCA-secure,

only the intended initiator should be able to decapsulate and recover  $ss_i$ . What follows is that all keys derived from  $ss_i$  are implicitly authentic. Likewise, if  $KEM_r$  is IND-CCA-secure, only the intended responder should be able to decapsulate and recover  $ss_r$ . This implies that all keys derived from  $ss_r$  are implicitly authentic.

In conclusion, what matters for our protocol is the number of decapsulation queries (one versus many or IND-1CCA versus IND-CCA) not whether they're adaptive or not (IND-CCA1 versus IND-CCA2). This security property is inherited from the KEMTLS protocol [6].

2) *Quantum Security*: Although the idea of using KEM for authentication is independent of PQC considerations, our intention is to use the post-quantum KEM algorithms to resist quantum adversaries. The KEM-based protocol using PQ-resistant KEMs should be safe against both classical and quantum computing adversaries.

3) *Downgrade Protection*: The SPDM initiator and responder need to negotiate the SPDM version, post-quantum capabilities (such as traditional mode, PQC mode, or hybrid mode), and post-quantum algorithms (such as Kyber-512 or Kyber-1024) [1]. Those six messages (*vca*) are mandatory and always included in the transcript to prevent downgrade attacks.

4) *Replay Protection*: The first message `KEY_EXCHANGE` includes a random number from the initiator and the second message `KEY_EXCHANGE_RSP` includes a random number from the responder. Both messages are included in the transcript hash.

The adversary may replay the first message to the responder. Then the responder may generate same  $ss_e$  and same  $ss_r$ . Those keys are not replay protected. However, when the responder derives the handshake keys, it will include the transcript hash (*th1*), which includes a unique random number on responder side. The result is that the handshake keys are different and all remaining steps are replay-protected.

5) *Anonymity*: Anonymity is not a design goal for SPDM. The public certificate of the initiator and responder may be transported in plain text. However, if the public certificate or public key is provisioned directly, anonymity can be potentially achieved. KEM-SPDM inherits this property.

6) *Deniability*: Deniability is also not a design goal. Standard SPDM uses digital signatures which actually ensure the opposite property of non-repudiation (c.f. [12]). KEM-SPDM may potentially provide deniability since an ephemeral public/private key pair is used to generate the shared symmetric key and the authentication is based upon the symmetric key.

## C. Comparison with other algorithms

1) *Comparison with PQ-SPDM*: PQ-SPDM [1] added support for quantum resistant algorithms to SPDM specification (DSP0274). It replaced traditional digital signatures with PQC or hybrid signature algorithms, and replaced traditional DHE algorithm with PQC KEM algorithm or a hybrid algorithm.

KEM-SPDM does not use digital signatures at all, both key establishment and authentication are based on KEMs only.

2) *Comparison with PQ-KEM-TLS*: KEM-TLS [6] or KEM-TLS with pre-distributed key (PDK) [5] were the alternatives to the RFC 8446 TLS 1.3 handshake that uses KEM instead of digital signature for authentication, which can be used for performance improvement.

Feature-wise, KEM-TLS supports encrypted server certificate transport, while the KEM-SPDM only supports plain-text responder certificate transport. KEM-TLS with PDK supports server certificates only and it still requires client certificate transport. KEM-SPDM supports certificate provisioning at both endpoints.

The authors of KEM-TLS used multistage based analysis and they defined 6 stages for KEM-TLS [6] and 5 stages for KEM-TLS with PDK [5]. We defined 4 stages for KEM-SPDM, because the early handshake secret for certificate transfer is not required for SPDM protocol.

3) *Comparison with PQ-WireGuard*: PQ-WireGuard [3] added post-quantum support for WireGuard protocol by replacing Diffie-Hellman with a KEM. Beyond that, PQ-WireGuard added key confirmation for explicit authentication.

PQ-WireGuard require both entities to provision the long term asymmetric key to each other. The plain text public key is not transported, only encrypted and authenticated hash of the public key. However, both the original SPDM and KEM-SPDM do not have such a requirement. The long term asymmetric key can be transported at runtime in SPDM or KEM-SPDM. Since PQ-WireGuard required long term key provisioning, that means mutual authentication is required. SPDM or KEM-SPDM may support one way authentication, which means the responder might not know the initiator's public key.

4) *Comparison with PQ-HPKE*: Post-Quantum Hybrid Public Key Encryption (PQ-HPKE) [7] adds post-quantum support for HPKE in RFC 9180.

PQ-HPKE only described the base mode in which only the receiver is authenticated and the sender is not authenticated, while KEM-SPDM supports mutual authentication mode. PQ-HPKE only supports implicit authentication. KEM-SPDM uses key confirmation for explicit authentication.

## V. IMPLEMENTATION RESULTS

We have developed a prototype that implements the above PQ-enabled SPDM variant using post-quantum algorithm library `liboqs`<sup>3</sup> on top of an SPDM implementation<sup>4</sup>. The implementation can run in both the Windows and Linux SPDM emulators. It can run in a Field Programmable Gate Array (FPGA) smart card and communicate with a host system on Intel Core CPU. We collected data for the winning PQC algorithms (Kyber as a KEM and Dilithium, Falcon, SPHINCS+ as digital signature primitives) described in the status report of NIST PQC 3rd round finalists and compare them with the RSA and ECC in traditional mode. The data has been collected on Intel® Core™ i7-8665U CPU @ 1.90 GHz and Arm® Cortex®-A53 64bit processor @ 190 MHz. We use the default messages defined in SPDM DSP0274 and DSP0277

<sup>3</sup><https://github.com/open-quantum-safe/liboqs>

<sup>4</sup><https://github.com/jyao1/openspdm-pqc-kem>

specification, where the opaque payload data only contains the 20 bytes secured version information. We also tried to increase the opaque payload data to the maximum 1024 bytes and get consistent result, because the SPDm specification requires the asymmetric operation (signing or verification) on the hash of the message transcript.

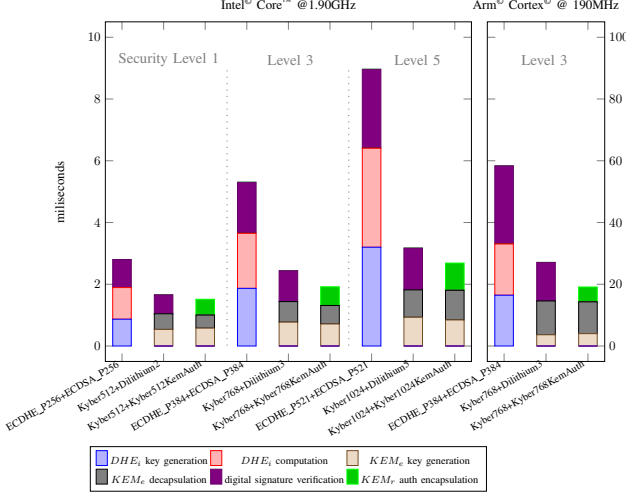


Fig. 3. SPDm Initiator performance comparison of unmodified SPDm using ECDHE+ECDSA, PQ-SPDM using Kyber+Dilithium and KEM-SPDM using Kyber for both key transport and authentication.

Usually, a device is an SPDm responder and a host operating system is an SPDm requester so responder performance is the main concern. We collected data from SPDm requester and responder separately in one-way authenticated secure session establishment for both digital signature and KEM authentication. We only listed traditional mode and PQC mode, because that helps us better understand the performance difference. In traditional mode, we always use digital signature-based authentication defined in SPDm specification. In PQC mode, we use the signature-based authentication from [1] or KEM-based authentication without signatures described in this paper.

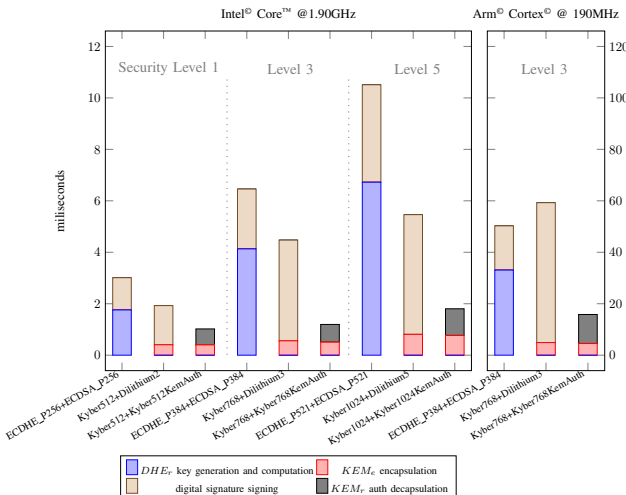


Fig. 4. SPDm Responder performance of SPDm using ECDHE+ECDSA, PQ-SPDM using Kyber+Dilithium and KEM-SPDM using Kyber.

Fig. 3 and Fig. 4 show that the KEM-based authentication (Kyber) is faster than the digital signature-based authentication (Dilithium) in general. PQ-SPDM using Falcon signatures (not shown in the figure) is sometimes slightly faster than Kyber on the initiator but much worse on the responder. Our results show also that SPHINCS+ takes significantly longer time so we did not include it in the figures either.

## VI. DISCUSSION AND FUTURE WORK

The KEM-based authentication mechanism is different from digital signature, which is widely adopted in current public key infrastructure (PKI), especially the device manufacturer. If the device certificate is changed from a digital signature public key to a KEM public key, the supply chain tool needs to be changed accordingly. That will bring a big change to current implementations.

Hybrid mode certificates are also a topic for further investigation. Combining one digital signature public key with one KEM public key into one certificate and performing hybrid authentication may not be adequate since the authentication mechanism is different.

In many use cases, the digital signature is used to provide the integrity of data, not only authentication. The PQC hash-based signature (HBS) algorithm, such as NIST approved Leighton-Micali hash-based signatures (LMS) or extended Merkle signature scheme (XMSS), can be used to support the integrity of static data, such as secure boot or image update use case. However, it is hard to use HBS to support dynamic data integrity, such as runtime measurement data collection with nonce. As such, we have to use public key encryption or data transfer inside the secure session.

Deniability of KEM-based authentication may be considered as a feature or a concern based on the use case of SPDm protocol.

In this paper, we assume both entities already have the peer's public key before the secure session setup. It is always true for the initiator's side, because the initiator should use GET\_CERTIFICATE to get the responder's certificate. But the responder may defer the certificate retrieval after secure session setup, if the initiator does not know that the mutual authentication is required by the responder. This special case needs further investigation.

## VII. CONCLUSIONS

In this paper, we introduced a KEM-based authentication mechanism for SPDm protocol (KEM-SPDM). It can eliminate the digital signature usage in SPDm device authentication and secure session establishment, to potentially improve the performance of quantum-secure communication. We performed security analysis of the protocol and implemented a proof-of-concept of the solution that was used to collect performance measurements on two platforms. The resulting data show that SPDm based on NIST selected PQC KEM algorithms achieve better performance than that based on PQC digital signatures. It proves the feasibility and performance benefit of using KEM-based authentication.

## REFERENCES

- [1] J. Yao, K. Matusiewicz, and V. Zimmer, "Post quantum design in spdm for device authentication and key establishment," *Cryptography*, vol. 6, no. 4, 2022. [Online]. Available: <https://www.mdpi.com/2410-387X/6/4/48>
- [2] A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama, "Strongly secure authenticated key exchange from factoring, codes, and lattices," in *PKC'12: Proceedings of the 15th international conference on Practice and Theory in Public Key Cryptography*. Springer International Publishing, 2012, pp. 467–484.
- [3] A. Hülsing, K.-C. Ning, P. Schwabe, F. Weber, and P. R. Zimmermann, "Post-quantum wireguard," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 304–321.
- [4] S. Celi, P. Schwabe, D. Stebila, N. Sullivan, and T. Wiggers, "KEM-based Authentication for TLS 1.3," Internet Engineering Task Force, Internet-Draft draft-celi-wiggers-tls-authkem-01, Mar. 2022, work in Progress. [Online]. Available: <https://datatracker.ietf.org/doc/draft-celi-wiggers-tls-authkem>
- [5] P. Schwabe, D. Stebila, and T. Wiggers, "More efficient post-quantum KEMTLS with pre-distributed public keys," in *Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 12972. Springer, 2021, pp. 3–22.
- [6] —, *Post-Quantum TLS Without Handshake Signatures*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1461–1480.
- [7] M. Anastasova, P. Kampanakis, and J. Massimo, "Pq-hpke: Post-quantum hybrid public key encryption," *Cryptology ePrint Archive*, Paper 2022/414, 2022, <https://eprint.iacr.org/2022/414>. [Online]. Available: <https://eprint.iacr.org/2022/414>
- [8] M. Fischlin and F. Günther, "Multi-stage key exchange and the case of google's quic protocol," in *CCS '14: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. Springer International Publishing, 2014, pp. 1193–1204.
- [9] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the tls 1.3 handshake protocol candidates," in *CCS '15: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. Springer International Publishing, 2015, pp. 1197–1210.
- [10] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," *SIAM J. Comput.*, vol. 33, no. 1, pp. 167–226, 2003.
- [11] J. Herranz, D. Hofheinz, and E. Kiltz, "Kem/dem: Necessary and sufficient conditions for secure hybrid encryption," 2006.
- [12] H. Krawczyk, "Skeme: A versatile secure key exchange for internet," in *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*. IEEE, 1996.

**Jiewen Yao** is a Principal Engineer in Intel corporation. His interest is firmware security, secure boot, trusted computing, confidential computing, device attestation, etc. He is a co-chair in multiple industry standard task groups, including TCG, DMTF, and RISC-V. Jiewen got Master of Engineering degree from Shanghai Jiaotong University, China.

**Anas Hlayhel** is a member of Intel's cryptography assurance team. His interests include performance optimizations of cryptographic implementations, cryptography-based hardware protection and key management. Anas graduated with B.Sc. in Electrical Engineering from the University of Houston, Texas and has worked at AMD and Intel. Anas is a member of IACR.

**Krystian Matusiewicz** received his Ph.D. degree in Computer Science from Macquarie University, Australia for his work on cryptanalysis of symmetric cryptosystems. Currently, he is a Principal Engineer leading cryptography assurance team at Intel Corporation. His interests focus on the design, analysis and deployment of cryptographic schemes that suit specific industry needs.