JOSE
Internet-Draft
Intended status: Standards Track
Expires: 27 July 2026

L. Prabel
S. Sun
Huawei
J. Gray
Entrust
T. Reddy
Nokia

23 January 2026

PQ/T Hybrid Composite Signatures for JOSE and COSE
draft-ietf-jose-pq-composite-sigs-00

Abstract

This document describes JSON Object Signing and Encryption (JOSE) and CBOR Object Signing and Encryption (COSE) serializations for PQ/T hybrid composite signatures. The composite algorithms described combine ML-DSA as the post-quantum component and either ECDSA or EdDSA as the traditional component.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-jose-pq-composite-sigs/.

Discussion of this document takes place on the Javascript Object Signing and Encryption Working Group mailing list (mailto:jose@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/jose/. Subscribe at https://www.ietf.org/mailman/listinfo/jose/.

Source for this draft and an issue tracker can be found at https://github.com/lucasprabel/draft-jose-pq-composite-sigs.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The impact of a potential Cryptographically Relevant Quantum Computer (CRQC) on algorithms whose security is based on mathematical problems such as integer factorisation or discrete logarithms over finite fields or elliptic curves raises the need for new algorithms that are perceived to be secure against CRQC as well as classical computers. Such algorithms are called post-quantum, while algorithms based on integer factorisation or discrete logarithms are called traditional.

While switching from a traditional algorithm to a post-quantum one intends to strengthen the security against an adversary possessing a quantum computer, the lack of maturing time of post-quantum algorithms compared to traditional algorithms raises uncertainty about their security.

Thus, the joint use of a traditional algorithm and a post-quantum algorithm in protocols represents a solution to this problem by providing security as long as at least one of the traditional or post-quantum components remains secure.

This document describes JSON Object Signing and Encryption (JOSE) and CBOR Object Signing and Encryption (COSE) serializations for hybrid composite signatures. The composite algorithms described combine ML-DSA as the post-quantum component and either ECDSA or EdDSA as the traditional component.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document follows the terminology for post-quantum hybrid schemes defined in [I-D.draft-ietf-pquip-pqt-hybrid-terminology].

This section recalls some of this terminology, but also adds other definitions used throughout the whole document:

"Asymmetric Traditional Cryptographic Algorithm": An asymmetric cryptographic algorithm based on integer factorisation, finite field discrete logarithms, elliptic curve discrete logarithms, or related mathematical problems. A related mathematical problem is one that can be solved by solving the integer factorisation, finite field discrete logarithm or elliptic curve discrete logarithm problem. Where there is little risk of confusion asymmetric traditional cryptographic algorithms can also be referred to as traditional algorithms for brevity.

"Post-Quantum Algorithm": An asymmetric cryptographic algorithm that is intended to be secure against attacks using quantum computers as well as classical computers. As with all cryptography, it always remains the case that attacks, either quantum or classical, may be found against post-quantum algorithms. Therefore it should not be assumed that just because an algorithm is designed to provide post-quantum security it will not be compromised.

"Post-Quantum Traditional (PQ/T) Hybrid Scheme": A multi-algorithm scheme where at least one component algorithm is a post-quantum algorithm and at least one is a traditional algorithm.

"PQ/T Hybrid Digital Signature": A multi-algorithm digital signature scheme made up of two or more component digital signature algorithms where at least one is a post-quantum algorithm and at least one is a traditional algorithm.

"Composite Algorithm": An algorithm which is a sequence of two component algorithms, as defined in [I-D.draft-ietf-lamps-pq-composite-sigs].

"Component Algorithm": Each cryptographic algorithm that forms part of a cryptographic scheme.

3. Algorithm Key Pair (AKP) Type

This document makes use of the Algorithm Key Pair (AKP) type which is defined in [I-D.draft-ietf-cose-dilithium].

As a reminder, the AKP type is used to express public and private keys for use with algorithms. The parameters for public and private keys contain byte strings.

This document makes use of the serialization routines defined in [I-D.draft-ietf-lamps-pq-composite-sigs] to obtain the byte string encodings of the composite public and private keys.

The process to compute JWK Thumbprint and COSE Key Thumbprint as described in [RFC7638] and [RFC9679] is detailed in [I-D.draft-ietf-cose-dilithium].

4. Composite Signature Algorithm

The structures of the composite keys and composite signatures follow an approach similar to [I-D.draft-ietf-lamps-pq-composite-sigs]. The composite design is chosen so that composite keys and signatures can be used as a drop-in replacement in JOSE / COSE object formats. This section gives some details about their construction.

4.1. Composite Key Generation

Composite public and private keys are generated by calling the key generation functions of the two component algorithms and concatenating the keys in an order given by the registered composite algorithm.

Composite Public Key <- Public Key of the 1st Algorithm || Public Key of the 2nd Algorithm and

Composite Private Key <- Private Key of the 1st Algorithm || Private Key of the 2nd Algorithm

For the composite algorithms described in this document (ML-DSA with ECDSA or EdDSA), the Key Generation process is as follows:

1. Generate component keys

```
mldsaSeed = Random(32)
(mldsaPK, mldsaSK) = ML-DSA.KeyGen_internal(mldsaSeed)

(tradPK, tradSK) = Trad.KeyGen()
```

2. Check for component key generation failure

```
if NOT (mldsaPK, mldsaSK) or NOT (tradPK, tradSK):
    output "Key generation error"
```

3. Serialize keys into composite form

```
Composite Public Key  <- SerializePublicKey(mldsaPK, tradPK)
Composite Private Key <- SerializePrivateKey(mldsaSeed, tradSK)
```

As in [I-D.draft-ietf-lamps-pq-composite-sigs], this keygen routine uses the seed-based ML-DSA.KeyGen_internal defined in Algorithm 6 of [FIPS.204].

This document makes use of the serialization routines from [I-D.draft-ietf-lamps-pq-composite-sigs] to obtain the byte string encodings of the composite public and private keys. These encodings are then directly used with the AKP Key Type. For more information, see the SerializePublicKey, DeserializePublicKey, SerializePrivateKey and DeserializePrivateKey algorithms from [I-D.draft-ietf-lamps-pq-composite-sigs].

Point compression for the ECDSA or EdDSA component is not performed for the AKP JSON Web Key Type but can be performed for the AKP COSE Key Type.

In this document, as in [I-D.draft-ietf-cose-dilithium], the ML-DSA private key MUST be a 32-bytes seed.

4.2. Composite Sign

When signing a message M with the composite Sign algorithm, the signature combiner prepends a prefix as well as a label value specific to the composite algorithm used to bind the two component signatures to the composite algorithm and achieve weak non-separability, as defined in [I-D.draft-ietf-pquip-hybrid-signature-spectrums].

However, only the pure ML-DSA component algorithm is used internally.

A composite signature’s value MUST include the two signature components and the two components MUST be in the same order as the components from the corresponding signing key.

A composite signature for the message M is generated by:

* computing the pre-hash of the message M;
* concatenating the prefix, the label, a byte 0x00 and the pre-hash;
* encoding the resulting message;
* calling the two signature component algorithms on this new message;
* concatenating the two output signatures.

For the composite algorithms described in this document (ML-DSA with ECDSA or EdDSA), the signature process of a message M is as follows:

1. Compute the Message representative M’

```
M’ <- Prefix || Label || 0x00 || PH(M)
M’ <- Encode(M’)
```

2. Separate the private key into component keys and re-generate the ML-DSA key from seed

```
(mldsaSeed, tradSK) = DeserializePrivateKey(sk)
(_, mldsaSK) = ML-DSA.KeyGen_internal(mldsaSeed)
```

3. Generate the two component signatures

```
sig_1 <- ML-DSA.Sign(mldsaSK, M’, ctx=Label)
sig_2 <- Trad.Sign(tradSK, M’)
```

4. If either ML-DSA.Sign() or Trad.Sign() return an error, then this process MUST return an error.

```
if NOT mldsaSig or NOT tradSig:
  output "Signature generation error"
```

5. Output the encoded composite signature value.

```
CompositeSignature <- SerializeSignatureValue(sig_1, sig_2)
return CompositeSignature
```

The serialization routines from [I-D.draft-ietf-lamps-pq-composite-sigs] are again used to obtain the byte string encoding of the composite signature. The SerializeSignatureValue routine simply concatenates the fixed-length ML-DSA signature value and the signature value from the traditional algorithm. For more information, see the SerializeSignatureValue and DeserializeSignatureValue algorithms from [I-D.draft-ietf-lamps-pq-composite-sigs].

The prefix "Prefix" string is defined as in [I-D.draft-ietf-lamps-pq-composite-sigs] as the byte encoding of the string "CompositeAlgorithmSignatures2025", which in hex is 436F6D706F73697465416C676F726974686D5369676E61747572657332303235. It can be used by a traditional verifier to detect if the composite signature has been stripped apart.

A signature label "Label" is defined in the same way as [I-D.draft-ietf-lamps-pq-composite-sigs]. It is specific to each composite algorithm. Additionally, the composite label is passed into the underlying ML-DSA primitive as the ctx. Signature Label values can be found in Table 4.

Similarly to [I-D.draft-ietf-cose-dilithium] which indicates that the ctx parameter MUST be the empty string, the application context passed in to the composite signature algorithm MUST be the empty string. To align with the structure of the [I-D.draft-ietf-lamps-pq-composite-sigs] combiner, the byte 0x00 is appended in the message M' after the label to indicate the context has length 0. However, a second non-empty context, defined as the label, is passed down into the underlying pure ML-DSA component algorithm, to bind the Composite-ML-DSA algorithm used.

Table 2 (resp. Table 3) indicates the pre-hash algorithms to use for JOSE (resp. COSE).

For JOSE (resp. COSE), M' is base64url-encoded (resp. binary encoded) before signature computations.

4.3. Composite Verify

The Verify algorithm MUST validate a signature only if all component signatures were successfully validated.

The verification process of a signature sig is as follows:

* separate the composite public key into the component public keys;
* separate the composite signature into its 2 component signatures;
* compute the message M' from the message M whose signature is to be verified;
* encode the resulting message M';
* verify each component signature.

1. Separate the keys and signatures

```
    (mldsaPK, tradPK) <- DeserializePublicKey(pk)
    (sig_1, sig_2) <- DeserializeSignatureValue(sig)
```

If Error during deserialization, or if any of the component keys or signature values are not of the correct type or length for the given component algorithm then output "Invalid signature" and stop.

2. Compute the message representative M’

```
    M’ <- Prefix || Label || 0x00 || PH(M)
    M’ <- Encode(M’)
```

3. Check each component signature individually, according to its algorithm specification.

```
    if NOT ML-DSA.Verify(mldsaPK, M’, ctx=Label)
        output "Invalid signature"
    if NOT Trad.Verify(tradPK, M’)
        output "Invalid signature"
    if all succeeded, then
        output "Valid signature"
```

The DeserializePublicKey and DeserializeSignatureValue serialization routines from [I-D.draft-ietf-lamps-pq-composite-sigs] are used to get the component public keys and the component signatures. For more information, see the DeserializePublicKey and DeserializeSignatureValue algorithms from [I-D.draft-ietf-lamps-pq-composite-sigs].

4.4. Encoding Rules

In each combination, the byte streams of the keys and of the signatures are directly concatenated.

Signature of the 1st Algorithm || Signature of the 2nd Algorithm

Since all combinations presented in this document start with the ML-DSA algorithm and the key or signature sizes are fixed as defined in [FIPS.204], it is unambiguous to encode or decode a composite key or signature.

Table 1 lists sizes of the three parameter sets of the ML-DSA algorithm.

	Private Key (seed)	Private Key	Public Key	Signature Size
ML-DSA-44	32	2560	1312	2420
ML-DSA-65	32	4032	1952	3309
ML-DSA-87	32	4896	2592	4627

Table 1: Sizes (in bytes) of keys and signatures of ML-DSA

Note that the seed is always 32 bytes, and that ML-DSA.KeyGen_internal from [FIPS.204] is called to produce the expanded private key from the seed, whose size corresponds to the sizes of the private key in the table above.

5. Composite Signature Instantiations

The ML-DSA signature scheme supports three possible parameter sets, each of which corresponding to a specific security strength. See [FIPS.204] for more considerations on that matter.

The traditional signature algorithm for each combination in Table 2 and Table 3 was chosen to match the security level of the ML-DSA post-quantum component.

The [FIPS.204] specification defines both pure and pre-hash modes for ML-DSA, referred to as "ML-DSA" and "HashML-DSA" respectively. This document only specifies a single mode which is similar in construction to HashML-DSA. However, because the pre-hashing is done at the composite level, only the pure ML-DSA algorithm is used as the underlying ML-DSA primitive.

5.1. JOSE algorithms

The following table defines a list of algorithms associated with specific PQ/T combinations to be registered in [IANA.JOSE].

Name	First Algorithm	Second Algorithm	Pre-Hash	Description
ML-DSA-44-ES256	ML-DSA-44	ecdsa-with-SHA256 with secp256r1	SHA256	Composite Signature with ML-DSA-44 and

				ECDSA using P-256 curve and SHA256
ML-DSA-65-ES256	ML-DSA-65	ecdsa-with-SHA256 with secp256r1	SHA512	Composite Signature with ML-DSA-65 and ECDSA using P-256 curve and SHA256
ML-DSA-87-ES384	ML-DSA-87	ecdsa-with-SHA384 with secp384r1	SHA512	Composite Signature with ML-DSA-87 and ECDSA using P-384 curve and SHA384
ML-DSA-44-Ed25519	ML-DSA-44	Ed25519	SHA512	Composite Signature with ML-DSA-44 and Ed25519
ML-DSA-65-Ed25519	ML-DSA-65	Ed25519	SHA512	Composite Signature with ML-DSA-65 and Ed25519
ML-DSA-87-Ed448	ML-DSA-87	Ed448	SHAKE256	Composite Signature with ML-DSA-87 and Ed448

Table 2: JOSE Composite Signature Algorithms for ML-DSA

Examples can be found in Appendix A.1.

5.2. COSE algorithms

The following table defines a list of algorithms associated with specific PQ/T combinations to be registered in [IANA.COSE].

Name	COSE Value	First Algorithm	Second Algorithm	Pre-Hash	Description
ML-DSA-44-ES256	TBD (request assignment -51)	ML-DSA-44	ecdsa-with-SHA256 with secp256r1	SHA256	Composite Signature with ML-DSA-44 and ECDSA using P-256 curve and SHA256
ML-DSA-65-ES256	TBD (request assignment -52)	ML-DSA-65	ecdsa-with-SHA256 with secp256r1	SHA512	Composite Signature with ML-DSA-65 and ECDSA using P-256 curve and SHA256
ML-DSA-87-ES384	TBD (request assignment -53)	ML-DSA-87	ecdsa-with-SHA384 with secp384r1	SHA512	Composite Signature with ML-DSA-87 and ECDSA using P-384 curve and SHA384
ML-DSA-44-Ed25519	TBD (request assignment -54)	ML-DSA-44	Ed25519	SHA512	Composite Signature with ML-DSA-44 and Ed25519
ML-DSA-65-Ed25519	TBD (request assignment -55)	ML-DSA-65	Ed25519	SHA512	Composite Signature with ML-DSA-65 and Ed25519
ML-DSA-87-Ed448	TBD (request assignment -56)	ML-DSA-87	Ed448	SHAKE256	Composite Signature with ML-DSA-87 and Ed448

Table 3: COSE Composite Signature Algorithms for ML-DSA

Examples can be found in Appendix A.2.

5.3. Composite Labels for JOSE and COSE

The JOSE and COSE composite label values are listed in Table 4.

They are represented here as ASCII strings, but implementers MUST convert them to byte strings using the obvious ASCII conversions prior to concatenating them with other byte values, as in [I-D.draft-ietf-lamps-pq-composite-sigs].

"alg" Header Parameter	Label (in ASCII)	Label (in Hex encoding)
ML-DSA-44-ES256	COMPSIG-MLDSA44-ECDSA-P256-SHA256	434F4D505349472D4D4C44534134342D45434453412D503235362D534841323536
ML-DSA-65-ES256	COMPSIG-MLDSA65-ECDSA-P256-SHA512	434F4D505349472D4D4C44534136352D45434453412D503235362D534841353132
ML-DSA-87-ES384	COMPSIG-MLDSA87-ECDSA-P384-SHA512	434F4D505349472D4D4C44534138372D45434453412D503338342D534841353132
ML-DSA-44-Ed25519	COMPSIG-MLDSA44-Ed25519-SHA512	434F4D505349472D4D4C44534134342D456432353531392D534841353132
ML-DSA-65-Ed25519	COMPSIG-MLDSA65-Ed25519-SHA512	434F4D505349472D4D4C44534136352D456432353531392D534841353132
ML-DSA-87-Ed448	COMPSIG-MLDSA87-Ed448-SHAKE256	434F4D505349472D4D4C44534138372D45643434382D5348414B45323536

Table 4: JOSE/COSE Composite Label Values

6. Security Considerations

The security considerations of [RFC7515], [RFC7517], [RFC9053] and [FIPS.204] also apply to this document.

All security issues that are pertinent to any cryptographic application must be addressed by JWS/JWK agents. Protecting the user's private key and employing countermeasures to various attacks constitute a priority.

In particular, to avoid key reuse, when generating a new composite key, the key generation functions for both component algorithms MUST be executed. Compliant parties MUST NOT use, import or export component keys that are used in other contexts, combinations, or by themselves as keys for standalone algorithm use.

For security properties and security issues related to the use of a hybrid signature scheme, the user can refer to [I-D.draft-ietf-pquip-hybrid-signature-spectrums]. For more information about hybrid composite signature schemes and the different hybrid combinations that appear in this document, the user can read [I-D.draft-ietf-lamps-pq-composite-sigs].

In terms of security properties, Composite ML-DSA will be EUF-CMA secure if at least one of its component algorithms is EUF-CMA secure and the message hash PH is collision resistant.

Ed25519 and Ed448 are SUF-CMA secure, making the composite SUF-CMA secure against classical adversaries.

However, Composite ML-DSA will not be SUF-CMA secure against a quantum adversary, since a quantum adversary will be able to break the SUF-CMA security of the traditional component. Consequently, applications where SUF-CMA security is critical SHOULD NOT use Composite ML-DSA.

7. IANA Considerations

7.1. JOSE Algorithms

The following values of the JWS "alg" (algorithm) are requested to be added to the "JSON Web Signature and Encryption Algorithms" registry. They are represented following the registration template provided in [RFC7518].

7.1.1. ML-DSA-44-ES256

* Algorithm Name: ML-DSA-44-ES256
* Algorithm Description: Composite Signature with ML-DSA-44 and ECDSA using P-256 curve and SHA-256
* Algorithm Usage Location(s): alg
* JOSE Implementation Requirements: Optional
* Change Controller: IETF

* Specification Document(s): n/a

* Algorithm Analysis Documents(s): TBD

7.1.2. ML-DSA-65-ES256

* Algorithm Name: ML-DSA-65-ES256

* Algorithm Description: Composite Signature with ML-DSA-65 and ECDSA using P-256 curve and SHA-256

* Algorithm Usage Location(s): alg

* JOSE Implementation Requirements: Optional

* Change Controller: IETF

* Specification Document(s): n/a

* Algorithm Analysis Documents(s): TBD

7.1.3. ML-DSA-87-ES384

* Algorithm Name: ML-DSA-87-ES384

* Algorithm Description: Composite Signature with ML-DSA-87 and ECDSA using P-384 curve and SHA-384

* Algorithm Usage Location(s): alg

* JOSE Implementation Requirements: Optional

* Change Controller: IETF

* Specification Document(s): n/a

* Algorithm Analysis Documents(s): TBD

7.1.4. ML-DSA-44-Ed25519

* Algorithm Name: ML-DSA-44-Ed25519

* Algorithm Description: Composite Signature with ML-DSA-44 and Ed25519 using SHA-512

* Algorithm Usage Location(s): alg

* JOSE Implementation Requirements: Optional

* Change Controller: IETF

* Specification Document(s): n/a

* Algorithm Analysis Document(s): TBD

7.1.5. ML-DSA-65-Ed25519

* Algorithm Name: ML-DSA-65-Ed25519

* Algorithm Description: Composite Signature with ML-DSA-65 and Ed25519 using SHA-512

* Algorithm Usage Location(s): alg

* JOSE Implementation Requirements: Optional

* Change Controller: IETF

* Specification Document(s): n/a

* Algorithm Analysis Document(s): TBD

7.1.6. ML-DSA-87-Ed448

* Algorithm Name: ML-DSA-87-Ed448

* Algorithm Description: Composite Signature with ML-DSA-87 and Ed448 using SHAKE-256

* Algorithm Usage Location(s): alg

* JOSE Implementation Requirements: Optional

* Change Controller: IETF

* Specification Document(s): n/a

* Algorithm Analysis Document(s): TBD

7.2. COSE Algorithms

The following values are requested to be added to the "COSE Algorithms" registry. They are represented following the registration template provided in [RFC9053], [RFC9054].

7.2.1. ML-DSA-44-ES256

* Name: ML-DSA-44-ES256
* Value: TBD (request assignment -51)
* Description: Composite Signature with ML-DSA-44 and ECDSA using P-256 curve and SHA-256
* Capabilities: [kty]
* Change Controller: IETF
* Reference: n/a
* Recommended: Yes

7.2.2. ML-DSA-65-ES256

* Name: ML-DSA-65-ES256
* Value: TBD (request assignment -52)
* Description: Composite Signature with ML-DSA-65 and ECDSA using P-256 curve and SHA-256
* Capabilities: [kty]
* Change Controller: IETF
* Reference: n/a
* Recommended: Yes

7.2.3. ML-DSA-87-ES384

* Name: ML-DSA-87-ES384
* Value: TBD (request assignment -53)
* Description: Composite Signature with ML-DSA-87 and ECDSA using P-384 curve and SHA-384
* Capabilities: [kty]
* Change Controller: IETF
* Reference: n/a

* Recommended: Yes

7.2.4. ML-DSA-44-Ed25519

* Name: ML-DSA-44-Ed25519
* Value: TBD (request assignment -54)
* Description: Composite Signature with ML-DSA-44 and Ed25519 using SHA-512
* Capabilities: [kty]
* Change Controller: IETF
* Reference: n/a
* Recommended: Yes

7.2.5. ML-DSA-65-Ed25519

* Name: ML-DSA-65-Ed25519
* Value: TBD (request assignment -55)
* Description: Composite Signature with ML-DSA-65 and Ed25519 using SHA-512
* Capabilities: [kty]
* Change Controller: IETF
* Reference: n/a
* Recommended: Yes

7.2.6. ML-DSA-87-Ed448

* Name: ML-DSA-87-Ed448
* Value: TBD (request assignment -56)
* Description: Composite Signature with ML-DSA-87 and Ed448 using SHAKE-256
* Capabilities: [kty]
* Change Controller: IETF

* Reference: n/a

* Recommended: Yes

8. References

8.1. Normative References

[FIPS.204] National Institute of Standards and Technology (NIST), "Module-Lattice-Based Digital Signature Standard", August 2024, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf>.

[IANA.COSE] IANA, "CBOR Object Signing and Encryption (COSE)", n.d., <https://www.iana.org/assignments/cose/cose.xhtml>.

[IANA.JOSE] IANA, "JSON Object Signing and Encryption (JOSE)", n.d., <https://www.iana.org/assignments/jose/jose.xhtml>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/rfc/rfc7518>.

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <https://www.rfc-editor.org/rfc/rfc7638>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9679] Isobe, K., Tschofenig, H., and O. Steele, "CBOR Object Signing and Encryption (COSE) Key Thumbprint", RFC 9679, DOI 10.17487/RFC9679, December 2024, <https://www.rfc-editor.org/rfc/rfc9679>.

8.2. Informative References

[I-D.draft-ietf-cose-dilithium]
Prorock, M. and O. Steele, "ML-DSA for JOSE and COSE", Work in Progress, Internet-Draft, draft-ietf-cose-dilithium-11, 15 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-dilithium-11>.

[I-D.draft-ietf-lamps-pq-composite-sigs]
Ounsworth, M., Gray, J., Pala, M., Klauテ殤er, J., and S. Fluhrer, "Composite ML-DSA for use in X.509 Public Key Infrastructure", Work in Progress, Internet-Draft, draft-ietf-lamps-pq-composite-sigs-14, 7 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-pq-composite-sigs-14>.

[I-D.draft-ietf-pquip-hybrid-signature-spectrums]
Bindel, N., Hale, B., Connolly, D., and F. D, "Hybrid signature spectrums", Work in Progress, Internet-Draft, draft-ietf-pquip-hybrid-signature-spectrums-07, 20 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-pquip-hybrid-signature-spectrums-07>.

[I-D.draft-ietf-pquip-pqt-hybrid-terminology]
D, F., P, M., and B. Hale, "Terminology for Post-Quantum Traditional Hybrid Schemes", Work in Progress, Internet-Draft, draft-ietf-pquip-pqt-hybrid-terminology-06, 10 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-pquip-pqt-hybrid-terminology-06>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[RFC9054] Schaad, J., "CBOR Object Signing and Encryption (COSE): Hash Algorithms", RFC 9054, DOI 10.17487/RFC9054, August 2022, <https://www.rfc-editor.org/rfc/rfc9054>.

Appendix A. Examples

A.1. JOSE

Will be completed in later versions.

A.2. COSE

Will be completed in later versions.

Appendix B. Acknowledgments

We thank Orie Steele for his valuable comments on this document.

Authors' Addresses

Lucas Prabel
Huawei
Email: lucas.prabel@huawei.com

Sun Shuzhou
Huawei
Email: sunshuzhou@huawei.com

John Gray
Entrust Limited
Email: john.gray@entrust.com

Tirumaleswar Reddy
Nokia
Bangalore
Karnataka
India
Email: kondtir@gmail.com