PQ/T Hybrid Composite Signatures for JOSE and COSE

draft-prabel-jose-pq-composite-sigs-03

Lucas PRABEL, Shuzhou SUN, John GRAY

IETF 123, Madrid

Major recent updates following feedback

- Support only the **seed representation** for ML-DSA private keys
- Change the key type to **AKP**
  - Issue #4 · lucasprabel/draft-jose-pq-composite-sigs (Change key type and map signature construction from LAMPS composite draft)
- Add JOSE test vectors
- Align with the LAMPS composite draft for the **signature combiner**, **domain separators, …**
  - Issue #1 · lucasprabel/draft-jose-pq-composite-sigs (Alignment with LAMPS Composite Signatures)
  - Issue #3 · lucasprabel/draft-jose-pq-composite-sigs (Using similar domain separators as LAMPS Composite Draft)

Composite Signatures at IETF

draft-ietf-lamps-pq-composite-sigs-07

Workgroup: LAMPS
Internet-Draft: draft-ietf-lamps-pq-composite-sigs-07
Published: 7 July 2025
Intended Status: Standards Track
Expires: 8 January 2026
Authors: M. Ounsworth (*Entrust*), J. Gray (*Entrust*), M. Pala (*OpenCA Labs*), J. Klaussner (*Bundesdruckerei GmbH*), S. Fluhrer (*Cisco Systems*)

Composite ML-DSA for use in X.509 Public Key Infrastructure

Abstract

This document defines combinations of ML-DSA [FIPS.204] in hybrid with traditional algorithms RSASSA-PKCS1-v1_5, RSASSA-PSS, ECDSA, Ed25519, and Ed448. These combinations are tailored to meet security best practices and regulatory guidelines. Composite ML-DSA is applicable in any application that uses X.509 or PKIX data structures that accept ML-DSA, but where the operator wants extra protection against breaks or catastrophic bugs in ML-DSA.

draft-reddy-tls-composite-mldsa-05

Workgroup: TLS
Internet-Draft: draft-reddy-tls-composite-mldsa-05
Published: 4 July 2025
Intended Status: Standards Track
Expires: 5 January 2026
Authors: T. Reddy (*Nokia*), T. Hollebeek (*DigiCert*), J. Gray (*Entrust*), S. Fluhrer (*Cisco Systems*)

Use of Composite ML-DSA in TLS 1.3

Abstract

Compositing the post-quantum ML-DSA signature with traditional signature algorithms provides protection against potential breaks or critical bugs in ML-DSA or the ML-DSA implementation. This document specifies how such a composite signature can be formed using ML-DSA with RSA-PKCS#1 v1.5, RSA-PSS, ECDSA, Ed25519, and Ed448 to provide authentication in TLS 1.3.

draft-prabel-jose-pq-composite-sigs-03

Workgroup: JOSE
Internet-Draft: draft-prabel-jose-pq-composite-sigs-03
Published: 7 July 2025
Intended Status: Standards Track
Expires: 8 January 2026
Authors: L. Prabel (*Huawei*), S. Sun (*Huawei*), J. Gray (*Entrust*)

PQ/T Hybrid Composite Signatures for JOSE and COSE

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 ECDSA as the traditional component.

draft-hu-ipsecme-pqt-hybrid-auth-02

Workgroup: ipsecme
Internet-Draft: draft-hu-ipsecme-pqt-hybrid-auth-02
Published: 1 May 2025
Intended Status: Standards Track
Expires: 2 November 2025
Authors: H. Jun (*Nokia*), Y. Morioka (*NTT DOCOMO, INC.*), W. Guilin (*Huawei*)

Post-Quantum Traditional (PQ/T) Hybrid PKI Authentication in the Internet Key Exchange Version 2 (IKEv2)

Abstract

One IPsec area that would be impacted by Cryptographically Relevant Quantum Computer (CRQC) is IKEv2 authentication based on traditional asymmetric cryptographic algorithms: e.g RSA, ECDSA; which are widely deployed authentication options of IKEv2. There are new Post-Quantum Cryptographic (PQC) algorithms for digital signature like NIST [ML-DSA], however it takes time for new cryptographic algorithms to mature, so there is security risk to use only the new algorithm before it is field proven. This document describes a IKEv2 hybrid authentication scheme that could contain both traditional and PQC algorithms, so that authentication is secure as long as one algorithm in the hybrid scheme is secure.

draft-sun-ssh-composite-sigs-01

Workgroup: sshm
Internet-Draft: draft-sun-ssh-composite-sigs-01
Published: 4 July 2025
Intended Status: Standards Track
Expires: 5 January 2026
Authors: S. Sun (*Huawei*), L. Prabel (*Huawei*)

Composite ML-DSA Signatures for SSH

Abstract

This document describes the use of PQ/T composite signatures for the Secure Shell (SSH) protocol. The composite signatures described combine ML-DSA as the post-quantum part and the elliptic curve signature schemes ECDSA, Ed25519 and Ed448 as the traditional part.

The draft aligns with the LAMPS composite WG document.

JOSE & COSE algorithms using ML-DSA with ECDSA

JOSE Composite Signature Algorithms for ML-DSA

Name	First Algorithm	Second Algorithm	Pre-Hash	Description
MLDSA44-ES256	ML-DSA-44	ecdsa-with-SHA256 with secp256r1	id-sha256	Composite Signature with ML-DSA-44 and ECDSA using P-256 curve and SHA-256
MLDSA65-ES512	ML-DSA-65	ecdsa-with-SHA512 with secp256r1	id-sha512	Composite Signature with ML-DSA-65 and ECDSA using P-256 curve and SHA-512
MLDSA87-ES512	ML-DSA-87	ecdsa-with-SHA512 with secp384r1	id-sha512	Composite Signature with ML-DSA-87 and ECDSA using P-384 curve and SHA-512

COSE Composite Signature Algorithms for ML-DSA

Name	COSE Value	First Algorithm	Second Algorithm	Pre-Hash	Description
MLDSA44-ES256	TBD (request assignment -51)	ML-DSA-44	ecdsa-with-SHA256 with secp256r1	id-sha256	Composite Signature with ML-DSA-44 and ECDSA using P-256 curve and SHA-256
MLDSA65-ES512	TBD (request assignment -52)	ML-DSA-65	ecdsa-with-SHA512 with secp256r1	id-sha512	Composite Signature with ML-DSA-65 and ECDSA using P-256 curve and SHA-512
MLDSA87-ES512	TBD (request assignment -53)	ML-DSA-87	ecdsa-with-SHA512 with secp384r1	id-sha512	Composite Signature with ML-DSA-87 and ECDSA using P-384 curve and SHA-512

Composite Key Generation and Signature Flow

KeyGen

```
(pk_1, sk_1) <- ML-DSA.KeyGen()
(pk_2 = (x,y), sk_2 = d) <- ECDSA.KeyGen()

Composite Public Key <- SerializePublicKey(pk_1, pk_2)
Composite Private Key <- SerializePrivateKey(sk_1, sk_2)
```

Sign

```
M' <- Prefix || Domain || 0x00 || r || PH(M)
M' <- Encode(M')

sig_1 <- ML-DSA.Sign(sk_1, M', ctx=Domain)
sig_2 <- ECDSA.Sign(sk_2, M')

Composite Signature <- SerializeSignatureValue(r, sig_1, sig_2)
```

Verify

```
(pk_1, pk_2) <- DeserializePublicKey(pk)
(r, sig_1, sig_2) <- DeserializeSignatureValue(sig)

M' <- Prefix || Domain || 0x00 || r || PH(M)
M' <- Encode(M')

if not ML-DSA.Verify(pk_1, M', ctx=Domain)
    output "Invalid signature"
if not ECDSA.Verify(pk_2, M')
    output "Invalid signature"
if all succeeded, then
    output "Valid signature"
```

Composite Key Generation and Signature Flow

KeyGen

```
(pk_1, sk_1) <- ML-DSA.KeyGen()
(pk_2 = (x,y), sk_2 = d) <- ECDSA.KeyGen()

Composite Public Key <- SerializePublicKey(pk_1, pk_2)
Composite Private Key <- SerializePrivateKey(sk_1, sk_2)
```

Sign

```
M' <- Prefix || Domain || 0x00 || r || PH(M)
M' <- Encode(M')

sig_1 <- ML-DSA.Sign(sk_1, M', ctx=Domain)
sig_2 <- ECDSA.Sign(sk_2, M')

Composite Signature <- SerializeSignatureValue(r, sig_1, sig_2)
```

Verify

```
(pk_1, pk_2) <- DeserializePublicKey(pk)
(r, sig_1, sig_2) <- DeserializeSignatureValue(sig)

M' <- Prefix || Domain || 0x00 || r || PH(M)
M' <- Encode(M')

if not ML-DSA.Verify(pk_1, M', ctx=Domain)
    output "Invalid signature"
if not ECDSA.Verify(pk_2, M')
    output "Invalid signature"
if all succeeded, then
    output "Valid signature"
```

Serialization subroutines from the LAMPS composite draft are used to serialize and deserialize composite values to **bytes** via simple **concatenation** of the underlying encodings of the component.

Composite Key Generation and Signature Flow

KeyGen

```
(pk_1, sk_1) <- ML-DSA.KeyGen()
(pk_2 = (x,y), sk_2 = d) <- ECDSA.KeyGen()

Composite Public Key <- SerializePublicKey(pk_1, pk_2)
Composite Private Key <- SerializePrivateKey(sk_1, sk_2)
```

Sign

```
M' <- Prefix || Domain || 0x00 || r || PH(M)
M' <- Encode(M')

sig_1 <- ML-DSA.Sign(sk_1, M', ctx=Domain)
sig_2 <- ECDSA.Sign(sk_2, M')

Composite Signature <- SerializeSignatureValue(r, sig_1, sig_2)
```

The computation of the **message representative** aligns with LAMPS:

- use of a **prefix** and **domain separator**
- **randomizer r** for extra security
- **pre-hashing** always done

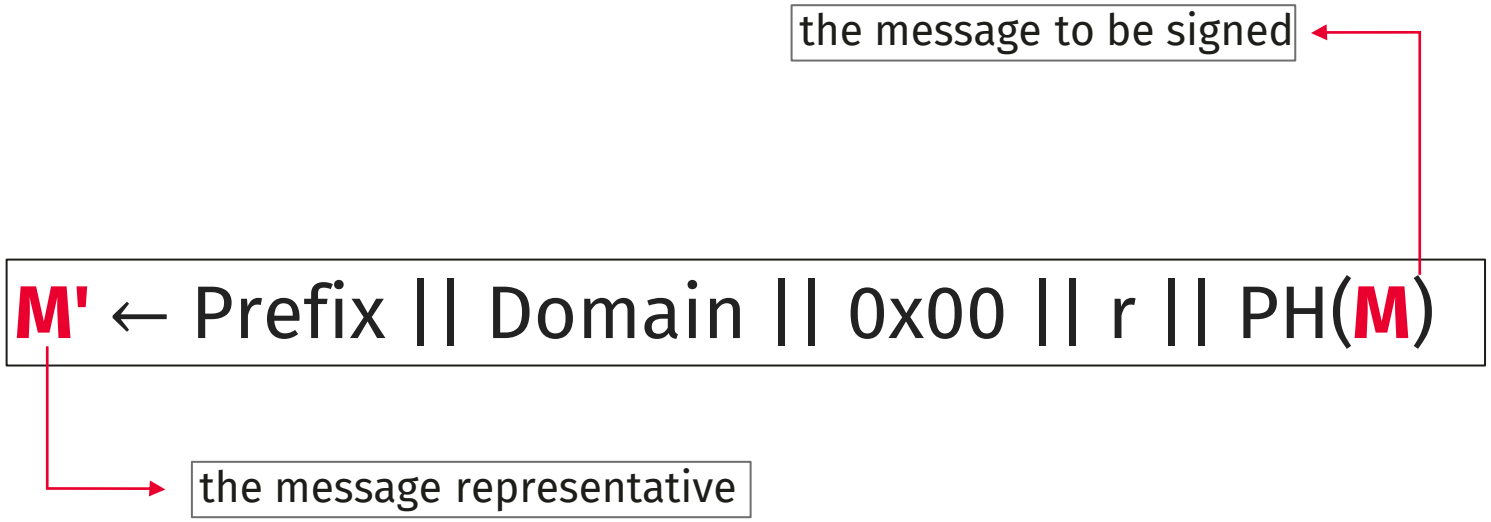
Verify

```
(pk_1, pk_2) <- DeserializePublicKey(pk)
(r, sig_1, sig_2) <- DeserializeSignatureValue(sig)

M' <- Prefix || Domain || 0x00 || r || PH(M)
M' <- Encode(M')

if not ML-DSA.Verify(pk_1, M', ctx=Domain)
    output "Invalid signature"
if not ECDSA.Verify(pk_2, M')
    output "Invalid signature"
if all succeeded, then
    output "Valid signature"
```

"alg" Header Parameter	Domain Separator (in Hex encoding)
ML-DSA-44-ES256	060B6086480186FA6B50090103
ML-DSA-65-ES256	060B6086480186FA6B50090108
ML-DSA-87-ES384	060B6086480186FA6B5009010C

Table 4: JOSE/COSE Composite Domain Separators

Construction of the message representative



Construction of the message representative

The Prefix string

It is an extra protection against stripping a component signature from the composite signature.

M' ← **Prefix** || Domain || 0x00 || r || PH(M)

Prefix – Definition

Prefix is defined as the byte encoding of the string "CompositeAlgorithmSignatures2025", which in hex is:
436F6D706F73697465416C676F726974686D5369676E617475726573323032 35

Construction of the message representative

The Domain separator

It serves to bind the signature to the specific composite algorithm used.
It helps protect against component signature values being removed from their composite and used out of context.

M' ← Prefix || **Domain** || 0x00 || r || PH(M)

Domain – Definition

Domain is defined as the DER encoding of the OID of the specific composite algorithm.
The specific values in hex encoding can be found in the following Table.

"alg" Header Parameter	Domain Separator (in Hex encoding)
ML-DSA-44-ES256	060B6086480186FA6B50090103
ML-DSA-65-ES256	060B6086480186FA6B50090108
ML-DSA-87-ES384	060B6086480186FA6B5009010C

Table 4: JOSE/COSE Composite Domain Separators

Construction of the message representative

Context

The context is empty, as represented by the 0x00 byte.

Construction of the message representative

Signature Randomizer

A 32 byte signature randomizer, which prevents a class of "mixed-key forgery attacks", specific to composites.

Construction of the message representative

$$M' \leftarrow \text{Prefix} \,||\, \text{Domain} \,||\, 0x00 \,||\, r \,||\, \mathbf{PH}(M)$$

Pre-hashing

Used to perform only a single pass over the potentially large input message M.

JOSE Test vectors


```
{
  "priv": "0000000000000000000000000000000000000000000000000000000000000000",
  "jwk": {
    "kid": "PL7f9-uTJKx0Q_21YdJ4zGEHPcHdqJNACd_fLup8VrE",
    "kty": "AKP",
    "alg": "ML-DSA-44-ES256",
    "pub": "unH59k4RuutY-pxvu24U5h8YZD2rSVtHU5qRZsoBmBMcRPgmu9VuNOVdteXi1zNIXjnqJg_GAAxep
    "priv": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
  },
  "jws": "eyJhbGciOiJNTC1EU0EtNDQtRVMyNTYiLCJraWQiOiJQTDdmOS11VEpLeDBRXzIxWWRKNHpHRUhQY0h
  "raw_randomizer": "0fd19763f5682f77d81c75c6aa9317972a52644a2847f778aa31e93f599a544b",
  "raw_to_be_signed": "436f6d706f73697465416c676f726974686d5369676e617475726573323032350
  "raw_composite_signature": "0fd19763f5682f77d81c75c6aa9317972a52644a2847f778aa31e93f599
  "raw_composite_public_key": "ba71f9f64e11baeb58fa9c6fbb6e14e61f18643dab495b47539a9166ca
}
```


Figure 1: ML-DSA-44-ES256


```
{
  "priv": "0000000000000000000000000000000000000000000000000000000000000000",
  "jwk": {
    "kid": "4cT9Q1VAUkl8mhuxioA9ZViGLsDoySnT0ZdDr4yHkyo",
    "kty": "AKP",
    "alg": "ML-DSA-65-ES256",
    "pub": "QksvJn5Y1bO0TXGs_Gpla7JpUNV8YdsciAvPof6rRD8JQquL2619cIq7w1YHj22ZolInH-YsdAkeu
    "priv": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
  },
  "jws": "eyJhbGciOiJNTC1EU0EtNjUtRVMyNTYiLCJraWQiOiI0Y1Q5UTFWQVVrbDhtaHV4aW9BOVpWaUdMc0R
  "raw_randomizer": "fa0ee107093ca9581bce637aa8f6e632b072da76d96513f432650c7d1bd2c69c",
  "raw_to_be_signed": "436f6d706f73697465416c676f726974686d5369676e617475726573323032350
  "raw_composite_signature": "fa0ee107093ca9581bce637aa8f6e632b072da76d96513f432650c7d1bd
  "raw_composite_public_key": "424b2f267e58d5b3b44d71acfc6a656bb26950d57c61db1c880bcfa1fe
}
```


Figure 2: ML-DSA-65-ES256


```
{
  "priv": "0000000000000000000000000000000000000000000000000000000000000000",
  "jwk": {
    "kid": "p1MMg8xj6mCplHRRACr5Afj_-4etB4DQLeRyFOMG1cQ",
    "kty": "AKP",
    "alg": "ML-DSA-87-ES384",
    "pub": "5F_8jMc9uIXcZi5ioYzY44AylxF_pWWIFKmFtf8dt7Roz8gruSnx2Gt37RT1rhamU2h3LOUZEkEBB
    "priv": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
  },
  "jws": "eyJhbGciOiJNTC1EU0EtODctRVMzODQiLCJraWQiOiJwMU1NZzh4ajZtQ3BsSFJSQUNyNUFmal8tNGV
  "raw_randomizer": "68ea12f8d96d4eebf1fb5251a48630216067ec02e2f2bc313344df36d381834e",
  "raw_to_be_signed": "436f6d706f73697465416c676f726974686d5369676e617475726573323032350
  "raw_composite_signature": "68ea12f8d96d4eebf1fb5251a48630216067ec02e2f2bc313344df36d38
  "raw_composite_public_key": "e45ffc8cc73db885dc662e62a18cd8e3803297117fa5658814a985b5ff
}
```


Figure 3: ML-DSA-87-ES384

Next Steps

- **Composite algorithms:** which one to register?
- **Message representative:** alignment with LAMPS?

WG Call for Adoption?

Comments and suggestions are welcome