OpenSSL CONFERENCE

Composites Aren't Optional: Designing PKI for the Post-Quantum Transition and Beyond

John Gray
Entrust

§ PRIVACY AS A FUNDAMENTAL HUMAN RIGHT

20.10.2026

Topics

- What are Composites
- Status of Composites in Standards
- Composite Design Patterns
- Implementation Considerations
- Interoperability Testing
- Composites beyond the PQ Transition

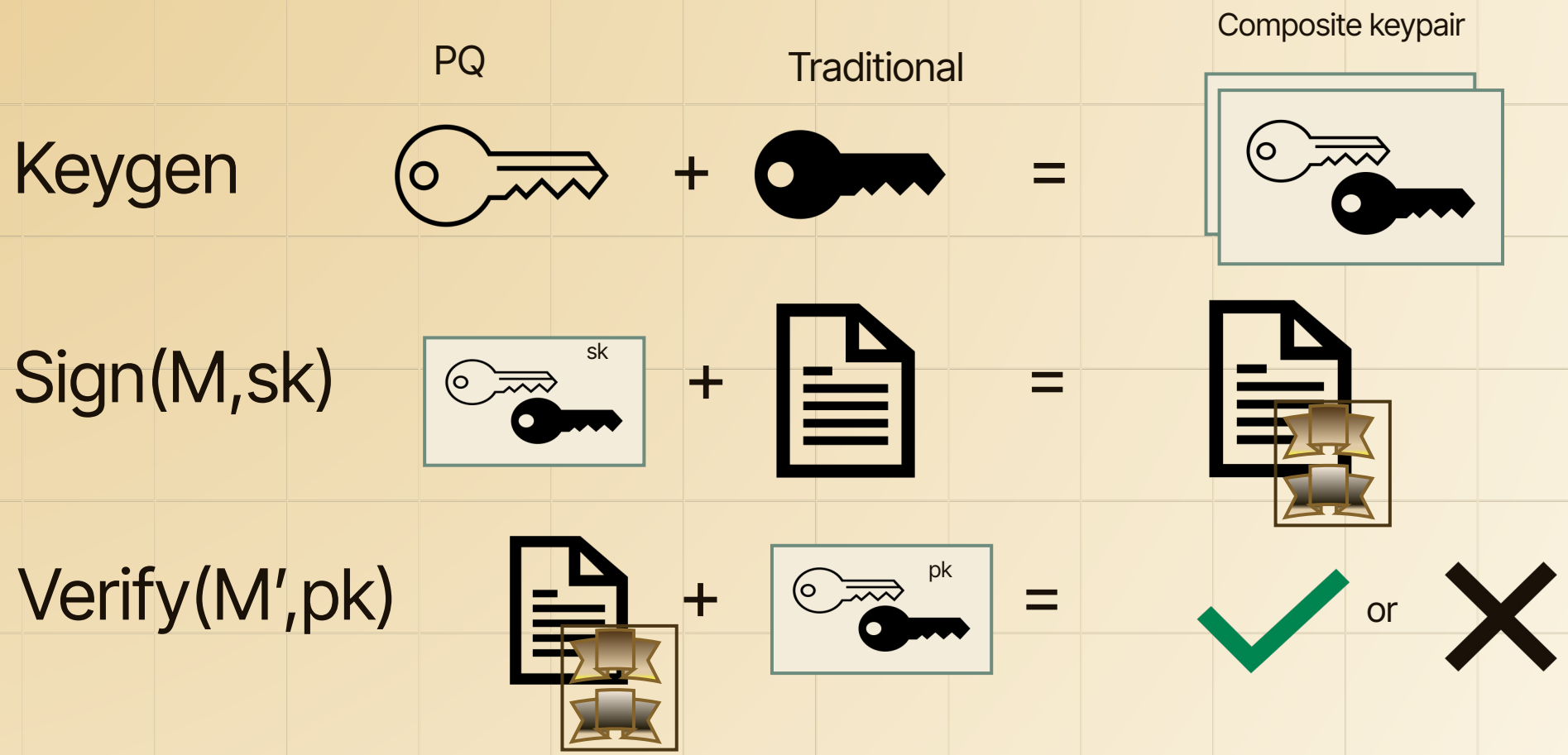
20.10.2026

What are Composites?

"A multi-algorithm scheme where at least one component algorithm is a post-quantum algorithm and at least one is a traditional algorithm, and where the resulting composite scheme is ***exposed as a singular interface of the same type*** *as the component algorithms."*

RFC 9794 definition of a Post-Quantum Traditional (PQ/T) hybrid composite scheme

COMPOSITE ML-DSA - EXPOSED AS A SIGNATURE

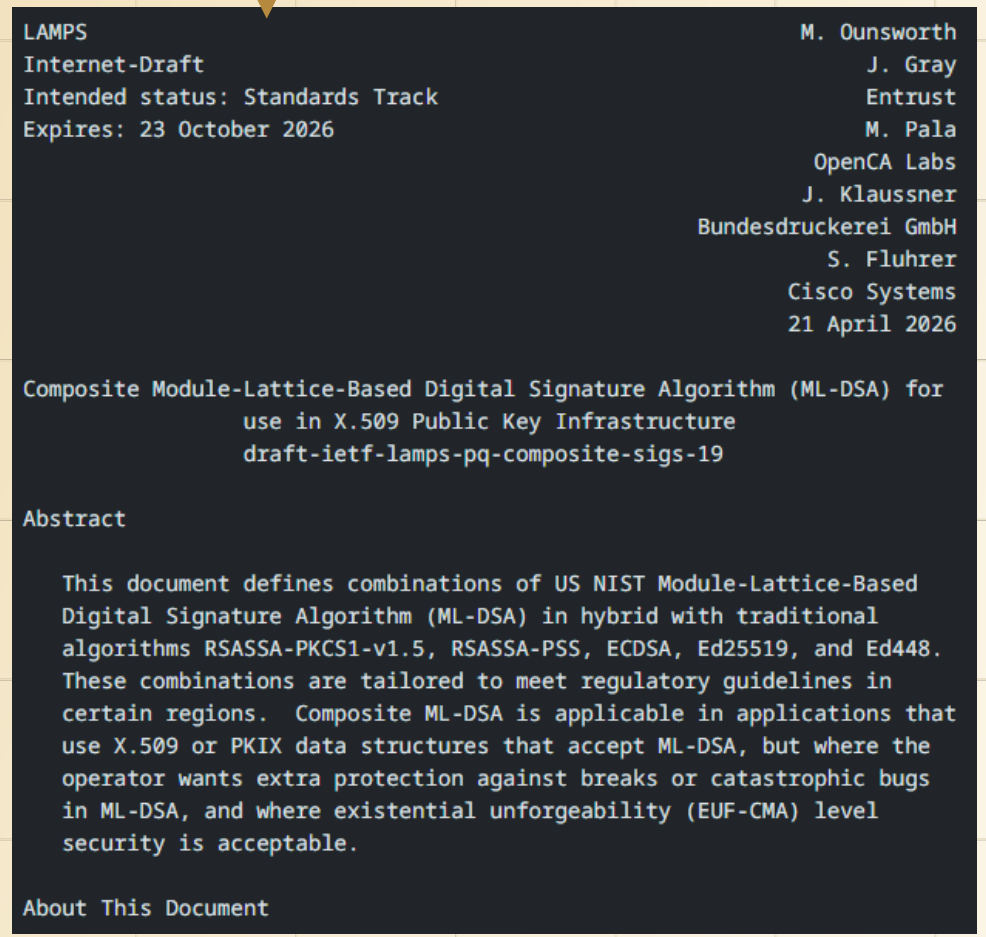


COMPOSITE ML-DSA

Combines ML-DSA with RSA or ECDSA

- resilience by combining traditional and PQ security
- Pre-hashed design fits well with existing RSA/ECDSA systems.
- Not much larger than ML-DSA, and smaller and faster than SLH-DSA
- Performance can be parallelized
- Very soon to be an RFC (final Queue)

Developed at IETF



LAMPS
Internet-Draft
Intended status: Standards Track
Expires: 23 October 2026

M. Ounsworth
J. Gray
Entrust
M. Pala
OpenCA Labs
J. Klaussner
Bundesdruckerei GmbH
S. Fluhrer
Cisco Systems
21 April 2026

Composite Module-Lattice-Based Digital Signature Algorithm (ML-DSA) for use in X.509 Public Key Infrastructure
draft-ietf-lamps-pq-composite-sigs-19

Abstract

This document defines combinations of US NIST Module-Lattice-Based Digital Signature Algorithm (ML-DSA) in hybrid with traditional algorithms RSASSA-PKCS1-v1.5, RSASSA-PSS, ECDSA, Ed25519, and Ed448. These combinations are tailored to meet regulatory guidelines in certain regions. Composite ML-DSA is applicable in applications that use 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, and where existential unforgeability (EUF-CMA) level security is acceptable.

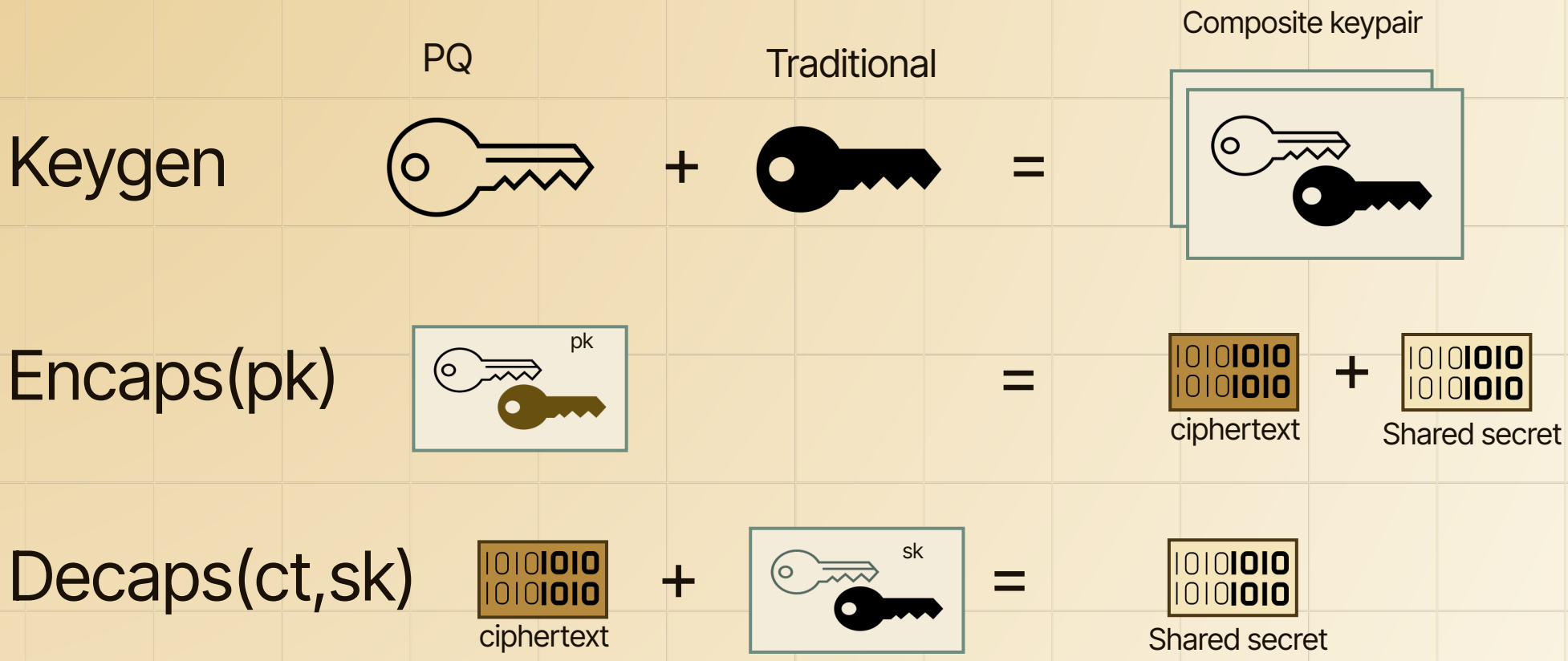
About This Document

Composite ML-DSA

Note: Certificate size estimates are using the same signature algorithm to sign the certificate.
C* is a constant representing the rest of the certificate data

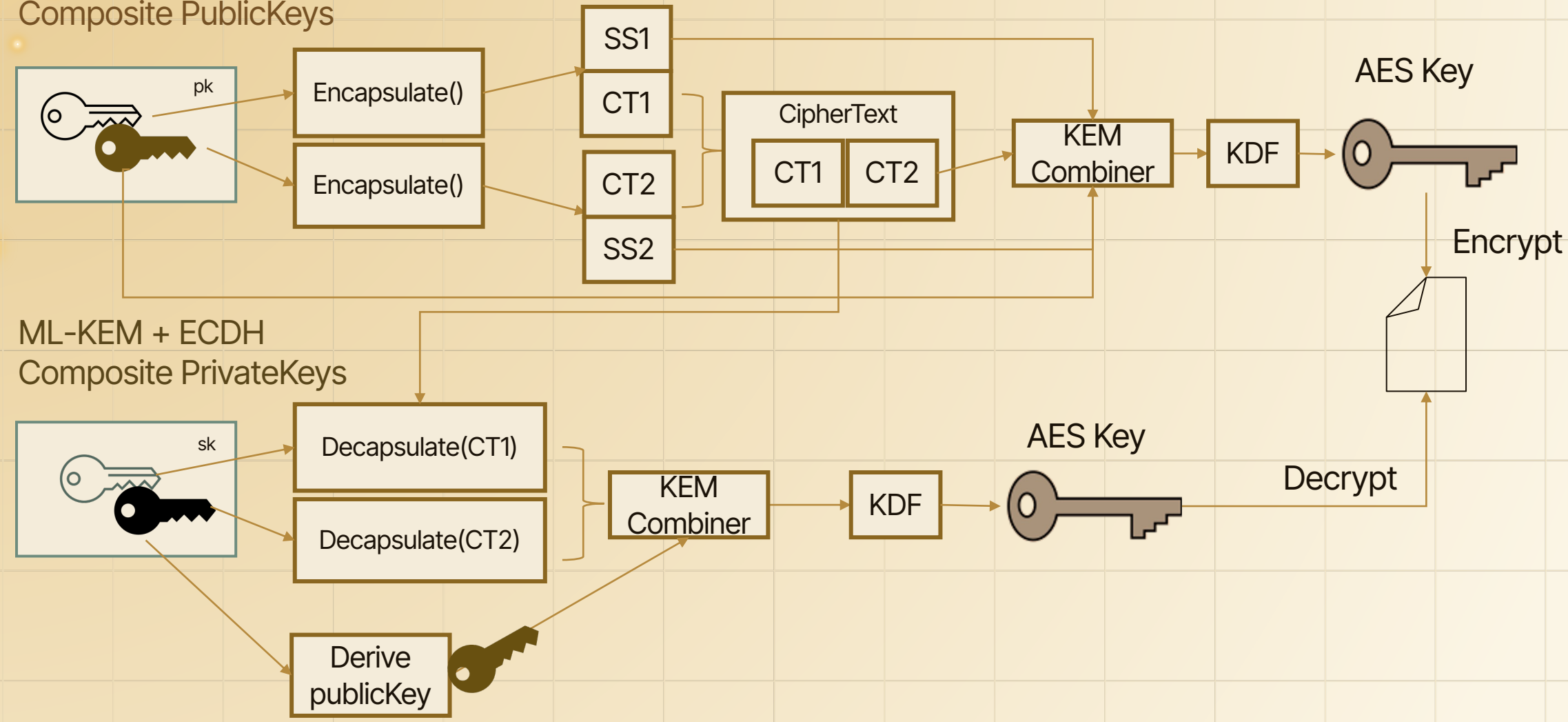
Composite Algorithm	Public Key (Bytes)	Private Key (Bytes)	Signature (Bytes)	Estimated Certificate (Bytes)
MLDSA44-RSA2048-PSS-SHA256	1,582	1,226	2,676	4,258 + C*
MLDSA44-RSA2048-PKCS15-SHA256	1,582	1,226	2,676	4,258 + C
MLDSA44-Ed25519-SHA512	1,344	64	2,484	3,828 + C
MLDSA44-ECDSA-P256-SHA256	1,377	83	2,492	3,869 + C
MLDSA65-RSA3072-PSS-SHA512	2,350	1,802	3,693	6,043 + C
MLDSA65-RSA3072-PKCS15-SHA512	2,350	1,802	3,693	6,043 + C
MLDSA65-RSA4096-PSS-SHA512	2,478	2,383	3,821	6,299 + C
MLDSA65-RSA4096-PKCS15-SHA512	2,478	2,383	3,821	6,299 + C
MLDSA65-ECDSA-P256-SHA512	2,017	83	3,381	5,398 + C
MLDSA65-ECDSA-P384-SHA512	2,049	96	3,413	5,462 + C
MLDSA65-ECDSA-brainpoolP256r1-SHA512	2,017	84	3,381	5,398 + C
MLDSA65-Ed25519-SHA512	1,984	64	3,373	5,357 + C
MLDSA87-ECDSA-P384-SHA512	2,689	96	4,731	7,420 + C
MLDSA87-ECDSA-brainpoolP384r1-SHA512	2,689	100	4,731	7,420 + C
MLDSA87-Ed448-SHAKE256	2,649	89	4,741	7,390 + C
MLDSA87-RSA3072-PSS-SHA512	2,990	1,802	5,011	8,001 + C
MLDSA87-RSA4096-PSS-SHA512	3,118	2,383	5,139	8,257 + C
MLDSA87-ECDSA-P521-SHA512	2,725	114	4,766	7,491 + C

COMPOSITE ML-KEM - EXPOSED AS A KEM



COMPOSITE KEM

ML-KEM + ECDH
Composite PublicKeys

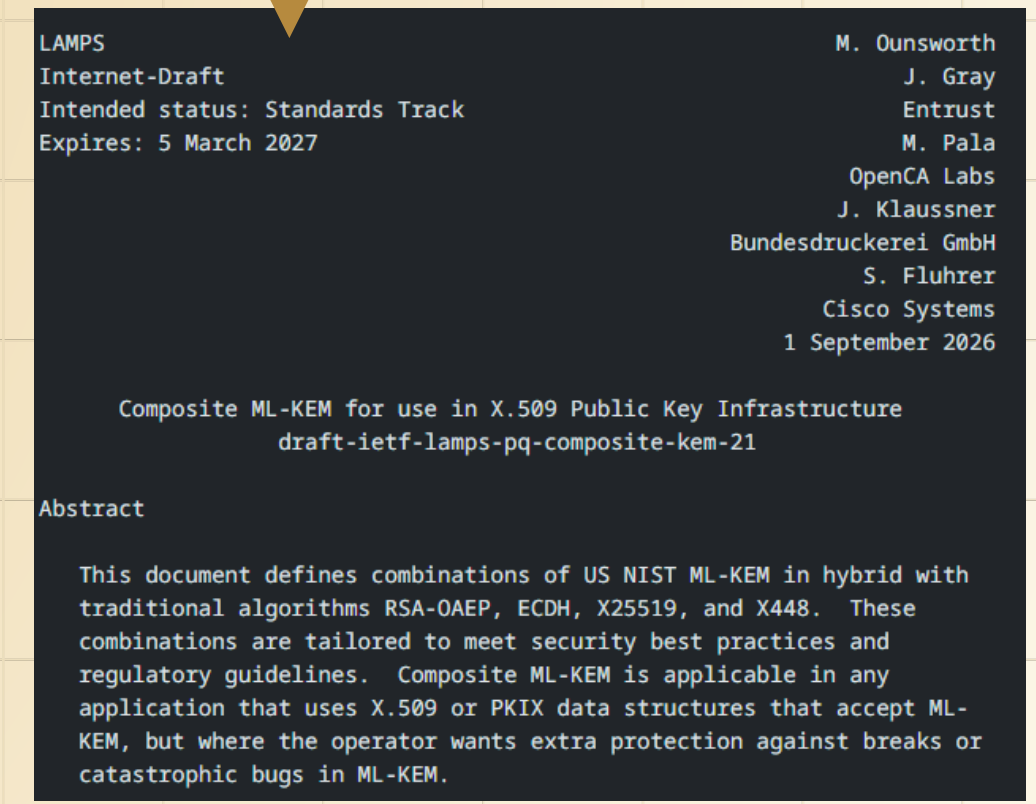


COMPOSITE ML-KEM

Developed at IETF

Combines ML-KEM with RSA-OAEP or ECDH

- Uses a concatenation combiner
- Aligns with the XWing drafts
  - ML-KEM ct and pk not needed in combiner
- Exposes RSA and ECDH as KEMs*
- Also soon to be an RFC!



LAMPS — M. Ounsworth
Internet-Draft — J. Gray
Intended status: Standards Track — Entrust
Expires: 5 March 2027 — M. Pala
OpenCA Labs
J. Klaussner
Bundesdruckerei GmbH
S. Fluhrer
Cisco Systems
1 September 2026

Composite ML-KEM for use in X.509 Public Key Infrastructure
draft-ietf-lamps-pq-composite-kem-21

Abstract

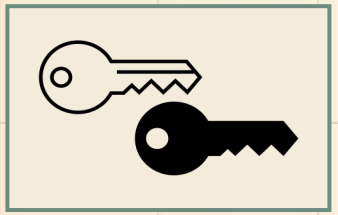
This document defines combinations of US NIST ML-KEM in hybrid with traditional algorithms RSA-OAEP, ECDH, X25519, and X448. These combinations are tailored to meet security best practices and regulatory guidelines. Composite ML-KEM is applicable in any application that uses X.509 or PKIX data structures that accept ML-KEM, but where the operator wants extra protection against breaks or catastrophic bugs in ML-KEM.

Composite KEM

Note: Certificate size estimates are using ML-DSA-65 Signatures for ML-KEM-768 composites, or ML-DSA-87 signature for ML-KEM-1024 composites

Composite KEM Algorithm	Public Key (B)	Private Key (B)	Ciphertext (B)	SS (B)	Estimated Certificate (B)
MLKEM768-RSA2048-SHA3-256	1,454	1,258	1,344	32	4,763 + C
MLKEM768-RSA3072-SHA3-256	1,582	1,834	1,472	32	4,891 + C
MLKEM768-RSA4096-SHA3-256	1,710	2,415	1,600	32	5,019 + C
MLKEM768-X25519-SHA3-256	1,216	96	1,120	32	4,525 + C
MLKEM768-ECDH-P256-SHA3-256	1,249	115	1,153	32	4,558 + C
MLKEM768-ECDH-P384-SHA3-256	1,281	128	1,185	32	4,590 + C
MLKEM768-ECDH-brainpoolP256r1-SHA3-256	1,249	116	1,153	32	4,558 + C
MLKEM1024-RSA3072-SHA3-256	1,966	1,834	1,952	32	6,593 + C
MLKEM1024-ECDH-P384-SHA3-256	1,665	128	1,665	32	6,292 + C
MLKEM1024-ECDH-brainpoolP384r1-SHA3-256	1,665	132	1,665	32	6,292 + C
MLKEM1024-X448-SHA3-256	1,624	120	1,624	32	6,251 + C
MLKEM1024-ECDH-P521-SHA3-256	1,701	146	1,701	32	6,328 + C

Composite Algorithms



Composite Signatures and KEM were developed at the IETF

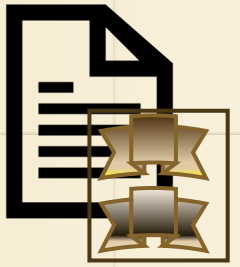
Composite Signatures

https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-sigs/19/

Composite KEM

https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-kem/22/

Why use composite Signatures?



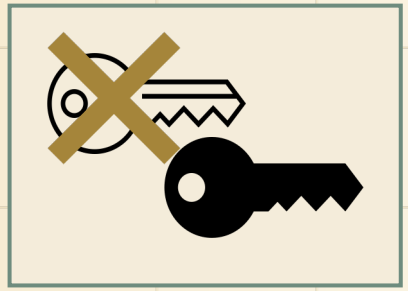
- Composite provides resilient security as multiple cryptographic primitives need to be broken (either by classical or quantum computational means).
- Currently they combine ML-DSA with RSA or EC algorithms
- They are smaller and more performant than SLH-DSA
- They are a “pre-hash” algorithm (like RSA and EC) and easily fit in existing systems

More conservative security

- SLH-DSA (hash based) - Most trusted, but very large signatures
- **Composite Signatures** – resilience with smaller signatures than SLH-DSA
- ML-DSA (lattice based, lattice attacks ongoing)
- LMS / XMSS* – Hash based (very trusted), but state management a problem
- RSA / EC – classically resistant, but not Quantum Resistant

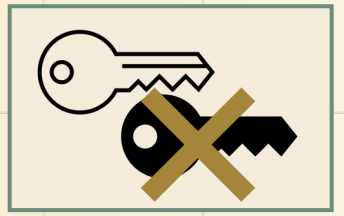
Least conservative security

What happens if RSA/EC are compromised?



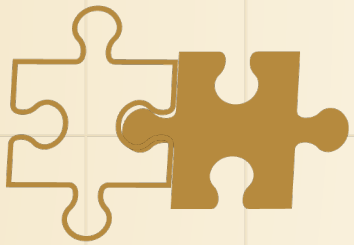
- Using a composite gives you a choice:
  - **Do nothing**
    - If the ML-DSA or ML-KEM component are not broken, you can continue using the composite.
      - You lose the extra layer of security, but the data being protected is still secure.
      - The RSA/ECDSA parts are deadweight (but small)
    - Unless the encumbrance of carrying the extra RSA/EC bits is more expensive than a quickly forced migration, you don't need to move.
  - **Move to another algorithm**
    - Move to whatever Signature or KEM algorithm is secure and meets your requirements at that time.
    - If Hybrid security is required, a future PQ/PQ composite construction could be used.

What if ML-DSA/ML-KEM are compromised?



- Using a composite still gives you a choice:
  - **Do nothing**
    - If the RSA / EC component are not broken, you can continue using the composite.
      - If lifetime of the key is within your quantum risk tolerance, the key can expire naturally
      - You lose the extra layer of security, but the data being protected is still secure.
      - ML-DSA / ML-KEM are much larger deadweight
  - **Plan your Migration to another algorithm**
    - Considering the Quantum computing risks, plan to migrate to a Quantum Resistant algorithm.
  - **Move to another algorithm**
    - Move to whatever Signature or KEM algorithm is secure and meets your requirements at that time.
    - If Hybrid security is required, a future PQ/PQ composite construction could be considered.

Composite Design Patterns



- Composite ML-DSA and Composite ML-KEM are designed to allow black-box signatures or KEM's to be combined at the crypto layer.
- They expose a common Signature or KEM interface
- To define new composites, start with the following:

Choose the set of algorithms (ex: ML-DSA, ECDSA)	Lock down specific parameter set (ex: ML-DSA-65, ECDSA-P-384)	Define an Object ID to fully represent the composite and its parameters

Composite Design Patterns - Signatures

$$M' := \text{Prefix} \,||\, \text{Label} \,||\, \text{len(ctx)} \,||\, \text{ctx} \,||\, \text{PH}(M)$$

Prefix:

A fixed octet string byte encoding in ASCII: "CompositeAlgorithmSignatures2025"

Label

A domain context which is unique to each composite

Len(ctx)

A single unsigned byte encoding the length of the context

ctx

Optional: The context bytes, allows binding the signature to an application context. Max length 255 bytes

Composite Design Patterns - Signatures

M' := Prefix || Label || len(ctx) || ctx || PH(M)

PH (M)

The prehash of the Message M

For example: SHA512

- **The message is hashed once before it is signed by each component**
  - Same design pattern as RSA / ECDSA, efficient for large data
  - No need to do the ML-DSA “external-mu” pre-hash

    SHAKE256(SHAKE256(pk,64) || M’,64)

- **Design considerations for the pre-hash:**
  - Security level of the components (SHA256, SHA512 or SHAKE256)
    - ex. ML-DSA-44, RSA2048 use SHA256
  - What hash functions do the underlying component algorithms use?
    - RSA / EC / ML-DSA
  - Certification considerations (example FIPS)

Composite Design Patterns - Signatures

Use the Label in underlying component algorithms context for stronger non-separability

Label

A domain context unique to each composite

- **ML-DSA supports a cryptographic Context**
  - For non-separability, use the Label as the context
  - ML-DSA.Sign(sk, M′, mldsa_ctx=Label)
  - ML-DSA.Verify (mldsaPK, M′, mldsaSig, mldsa_ctx=Label)
- **RSA/ECDSA** do not support a context, which limits Composite ML-DSA to the weak non-separability property overall.
- **Future PQ/PQ composites**, where both components support a context could achieve a strong non-separability property.

Composite Design Patterns - KEM

combined_input = mlkemss || tradss || tradCT || tradPK || Label

mlkemss: The ML-KEM Shared secret
tradss: The traditional shared secret
tradCT: The traditional algorithms Ciphertext
tradPK: The traditional algorithm Public Key. During decapsulation it is regenerated from the traditional private key.
Label: A domain context unique to each composite

[see https://datatracker.ietf.org/doc/draft-irtf-cfrg-hybrid-kems/]
[see https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-227.pdf section 4.6.3]

Composite ML-KEM uses the C2PRI Combiner with a Nominal Group. KEMs without C2PRI (Ciphertext Second Preimage Resistance) should use the UniversalCombiner (IETF) / KeyCombiner (SP800-227)

Composite Design Patterns - KEM

ss = KDF (combined_input)

KDF: The Key Derivation function takes as input the combined data and uses **SHA3-256** to produce a 32-byte shared secret.

SHA3-256 is an allowed Key Derivation Algorithm (KDF) as specified in SP 800-56C as a one step KDF.

[see https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf]

Composite Design Patterns - KEM

RSA-OAEP and ECDH are "promoted" to KEMs

RSA-OAEP: a randomly shared-secret (ss) is generated and encrypted with RSA-OAEP.

RSAOAEPKEM.Encaps(pkR):

ss = SecureRandom(ss_len)
ct = RSAES-OAEP-ENCRYPT(pkR, ss)

RSAOAEPKEM.Decaps(skR, enc):

ss= RSAES-OAEP-DECRYPT(skR, ct)

ECDH: An ephemeral ECDH keypair is generated, the encapsulation ECDH public key is combined with the ephemeral ECDH private key to produce the shared-secret. The ciphertext is just the serialized ephemeral ECDH public key

DHKEM.Encaps(pkR):

(skE, pkE) = GenerateKeyPair()
ss = DH(skE, pkR)
ct = SerializePublicKey(pkE)

DHKEM.Decaps(skR, ct):

pkE = DeserializePublicKey(ct)
ss = DH(skR, pkE)

[see https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-kem/22/ section 2.1 and 2.2]

Let's design a PQ/PQ composite Signature!

1. **Choose algorithms** - ML-DSA, SQISign
   - Does this choice make sense? Yes – 2 different hardness problems

2. **Select complementary security levels** (Lets choose NIST level 3):
   - ML-DSA-65 + SQISign-L3

3. **Assign a prototype OID for testing**
   - Let's pick 1.2.3.4.5.6.7.8 for fun!

Let's design a PQ/PQ composite Signature!

$$M' := \text{Prefix} \,||\, \text{Label} \,||\, \text{len(ctx)} \,||\, \text{ctx} \,||\, \text{PH}(M)$$

4. **Which pre-hash function**?
   - ML-DSA-65 and SQISign both use SHAKE256 → Use SHAKE-256 (with 512-bit length)
5. **Prefix** = "CompositeAlgorithmSignatures2025"
6. **Label** = "MLDSA65-SQISignL3-SHAKE256"
7. **len(ctx)** = optional
8. **ctx** = optional

Let's design a PQ/PQ composite Signature!

9. **Non-separability** (assume SQISign API will include a cryptographic context like ML-DSA, SLH-DSA and newer signature designs).
    - Pass in the label so ctx = Label for both ML-DSA and SQISign
    - ML-DSA.Sign(sk, M′, mldsa_ctx=Label)
    - SQISign.Sign(sk, M′, sqisign_ctx=Label)
10. **Public and Private key format**:
    - Private: ML-DSA Seed format, SQISign (secret isogeny)
    - Public: ML-DSA public key, SQISign (public curve)

We just designed a composite!

Characteristics of our PQ/PQ composite

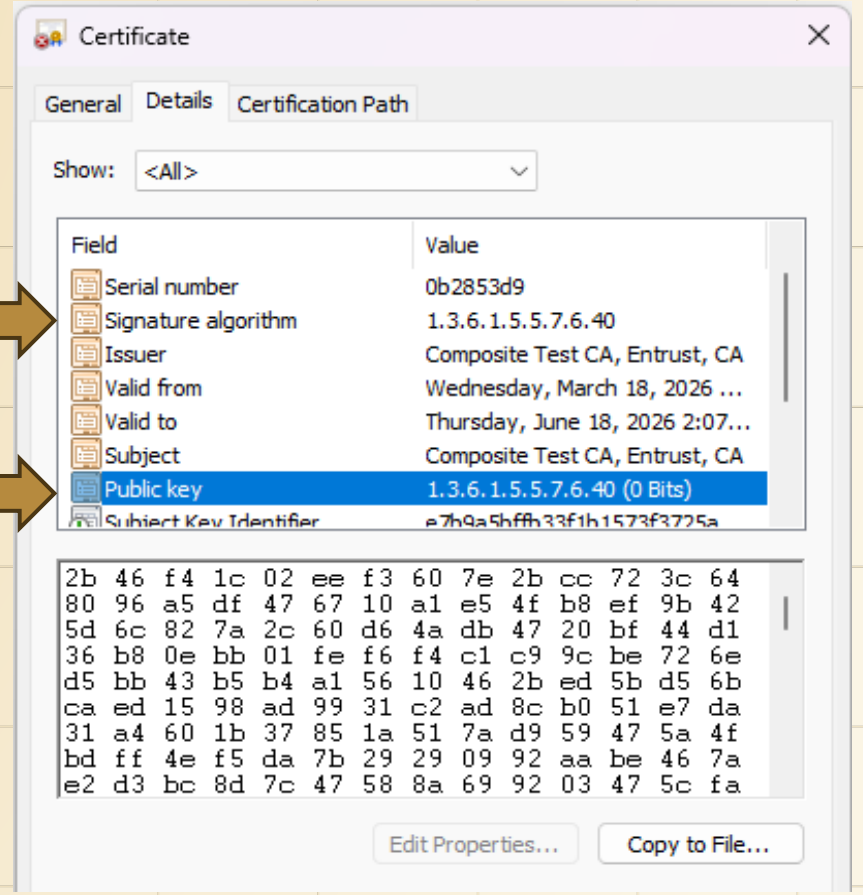
Composite Algorithm	Public Key (Bytes)	Private Key (Bytes)	Signature (Bytes)	Estimated Certificate (Bytes)
• MLDSA65-SQISignL3-SHAKE256	1,952 + 129 **2,081 B**	4,032 + 1,138 **5,170**	3,309 + 306 **3,615 B**	2,081 + 3,615 **5,696 + C**

- PQ/PQ composite signature algorithm
- Resilience if either ML-DSA or SQISign-L3 are broken
- Combines lattice and isogeny hardness problem
- Strong Non-separability (assuming SQISign will accept a context value that can be used)
- Pre-Hashed with SHAKE256 for performance
- Key sizes still smaller than SLH-DSA
- This can be done for other combinations

Note: The above was just for fun! SQISign has not been selected for standardization.

Composite Use in Protocols - Signatures

- PKI Protocols like TLS, CMP, CMS, S/MIME, etc can simply drop in a composite signature following this simple design pattern:
- **Use the algorithm extensibility built into PKI systems**:
  - Allocate an identifier that represents a signature algorithm
    - TLS has “code points”, Certificates “Object IDs”, JOSE uses JWS “alg”
- **Update the implementation to recognize the identifier**
  - This is a simple update to most protocol libraries; some may not even need an update if they simply hand the identifier to a cryptographic library.
  - **All the cryptography is performed at the cryptographic layer, no protocol updates needed!**



Composite Use in Protocols - KEM

- Many PKI protocols are being updated to support the KEM API as it is the way forward for key establishment mechanisms
- **Examples**: CMS (RFC 9629 defines KEMRecipientInfo),
  - CMP (RFC 9810) defines KEM enrollment and protection mechanisms.
  - Unfortunately, not all protocols have been standardized to support KEM
    - S/MIME (see https://github.com/lamps-wg/smime-v5)
    - XML-Encryption (see draft-eastlake-rfc9231bis-xmlsec-uris)
- The good news is that protocols that are updated to accept KEM algorithms will be easy to update to support composite KEM.
  - **Use the cryptographic extensibility built into existing systems**:
  - **Update the protocol implementation to recognize the identifier**

Composite Use in Protocols - KEM

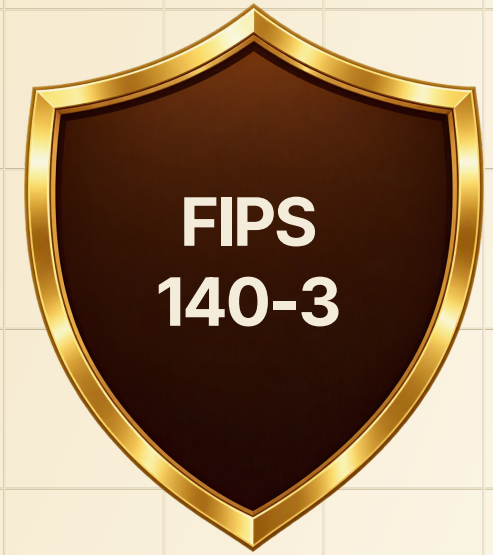
- **KEM can also be used for authentication!**
  - At this time KEM authentication is being explored for existing protocols
  - Examples:
    - IKEv2 KEM Authentication draft-wang-ipsecme-kem-auth-ikev2
    - TLS authKem - draft-celi-wiggers-tls-authkem
    - Protocols for PIV (Emmanuelle Dottax – ICMC - Q22b-DottaxE-UPDATED.pdf)
- **Composite KEM could likewise be used for KEM authentication**
  - Protocols will first need to be updated to support KEM authentication

Composite Implementation Details

- **Use of Existing RSA / ECDSA implementations**
  - PublicKey, PrivateKey formats follow existing standards:
    - RSA encoded as an RSAPublicKey or RSAPrivateKey [RFC 8017 A.1.1 and A.1.2] (
    - ECDSA / ECDH – uncompressed X9.62 [X9.62_2005] / ECPoint [RFC 5480 section 2.2]
    - EdDSA Raw values [RFC 8032], X25519 and X448 section 5 of [RFC 7748]
- **ML-DSA, ML-KEM private keys use only the seed format.**
  - ML-DSA Section 7.2 of FIPS.204 and ML-KEM of Section 7.1 and 7.2 of [FIPS.203]
  - Only 1 format of private and public keys specified for better interoperability
- **Black box design allows parallelization:**
  - keygen(), sign(), verify(), encaps() and decaps() routines can be parallelized when running the component algorithms.

FIPS certification status – Signatures

- Composites ML-DSA signatures can be considered FIPS certified as long as one of the two component algorithms is FIPS certified
  - Signatures use two parallel executions
  - The verification routine only succeeds if all component signatures are successfully verified
    - As long as one of them is FIPS approved, the FIPS requirement is met

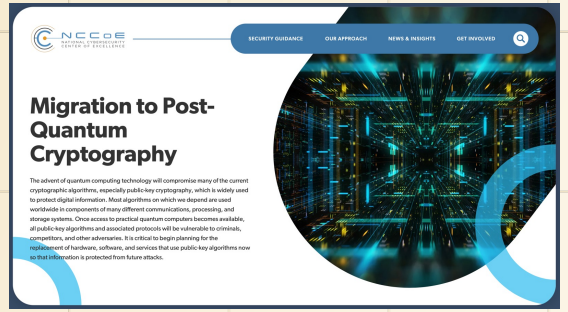


FIPS certification status – KEM

- SP-800-56Cr2 allowed ss = KDF (**mlkemSS**, **tradSS**, fixedInfo)
  - First item (mlkemSS) had to be the FIPS certified one.
  - This was problematic and would have required another composite definition to fix ss = KDF (**tradSS**, **mlKEMSS**, fixedInfo)
- SP-800-227 section 4.6.2 allows either one in either order to be used as the KEM combiner
- This means a composite ML-KEM using a FIPS certified RSA-3072 but uses an ML-KEM-768 implementation waiting in the queue can still be FIPS approved!

Composite Interoperability Testing

- **IETF PQC PKI Interoperability** project has tested many different composite implementations
  - See https://github.com/IETF-Hackathon/pqc-certificates
  - Most interoperability issued had to do with ASN.1 encoding (we still get it wrong after all these years)!
  - A few issues with composite key format interoperability
    - Most implementations were found to be accepting of historical things like the ASN.1 NULL parameter (00 05) in PQ component
    - Most produced compliant key format outputs



Composite Interoperability Testing

- See: https://ietf-hackathon.github.io/pqc-certificates/pqc_hackathon_results_certs_r5_automated_tests.html

✅ = passing all verifiers
◒ = passing some verifiers
○ = not passing any verifiers
Columns represent producers who submitted artifacts. Verifiers are not listed in this table, but are listed in the broken-out tables below.

-	bc	botan	carl-redhound	cht	composite-crypto	composite-kem-ref-impl	composite-sigs-ref-impl	corey-digicert	crypto4a	cryptonext	entrust	entrust-pkihub	leancrypto	openjdk	openssl-composite-preliminary-impl	ossl35	safelogic	sanctum-sec
id-MLDSA44-RSA2048-PSS-SHA256-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA44-RSA2048-PKCS15-SHA256-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA44-Ed25519-SHA512-cert	✅ 3/3			✅ 3/3	✅ 3/3		✅ 3/3		✅ 3/3	✅ 3/3	✅ 3/3	✅ 3/3	✅ 3/3		✅ 3/3		✅ 3/3	✅ 3/3
id-MLDSA44-ECDSA-P256-SHA256-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA65-RSA3072-PSS-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA65-RSA3072-PKCS15-SHA512-cert	✅ 2/2			✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA65-RSA4096-PSS-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA65-RSA4096-PKCS15-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA65-ECDSA-P256-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA65-ECDSA-P384-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA65-ECDSA-brainpoolP256r1-SHA512-cert	✅ 2/2			✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2				✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA65-Ed25519-SHA512-cert	✅ 3/3			✅ 3/3	✅ 3/3		✅ 3/3		✅ 3/3	✅ 3/3	✅ 3/3	✅ 3/3	✅ 3/3		✅ 3/3		✅ 7/7	✅ 3/3
id-MLDSA87-ECDSA-P384-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA87-ECDSA-brainpoolP384r1-SHA512-cert	✅ 2/2			✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2				✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA87-Ed448-SHAKE256-cert	✅ 2/2			✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA87-RSA3072-PSS-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA87-RSA4096-PSS-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLDSA87-ECDSA-P521-SHA512-cert	✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2		✅ 2/2		✅ 2/2	✅ 2/2	✅ 2/2	✅ 2/2			✅ 2/2		✅ 2/2	✅ 2/2
id-MLKEM768-RSA2048-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM768-RSA3072-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM768-RSA4096-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM768-X25519-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM768-ECDH-P256-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM768-ECDH-P384-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM768-ECDH-brainpoolP256r1-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM1024-RSA3072-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM1024-ECDH-P384-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM1024-ECDH-brainpoolP384r1-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM1024-X448-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							
id-MLKEM1024-ECDH-P521-SHA3-256-cert	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1			✅ 1/1	✅ 1/1	✅ 1/1							

Composite deployment in progress

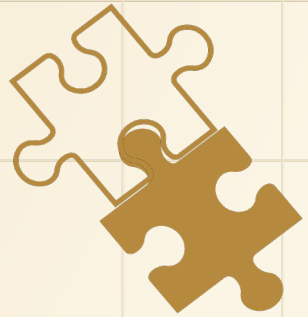
- **Composites are ready for production!**
  - Supported in Windows (https://devblogs.microsoft.com/dotnet/post-quantum-cryptography-in-dotnet/)
  - Apple is using composites (https://security.apple.com/blog/apple-reference-image/)
  - PKCS#11 3.3 adding Composites (https://github.com/oasis-tcs/pkcs11/tree/master/working/doc/spec)
  - 13+ implementations Composite ML-DSA signatures (IETF PQC hackathon project)
  - 7+ implementations Composite KEM (IETF PQC hackathon project)
  - Open Source:
    - **Bouncy Castle** (https://www.bouncycastle.org/resources/new-release-composite-signatures_unsigned-certificates_and_crmf-cmp-challenge-response/)
    - **OpenSSL** has a pull request in progress (https://github.com/openssl/openssl/pull/32886)

OpenSSL CONFERENCE

German Bundesdruckerei are prototyping composites in national ID cards

20.10.2026

Composites beyond the PQ transition

- In places that require hybrid cryptography, PQ / PQ composites will likely be a requirement
- The black-box design pattern used in Composite Signatures and KEM can be used as a starting point for the development of future composites
- Composite fit into existing security protocols and PKI systems
- Composites combined with cryptographic agility give a robust security framework that can help prevent unexpected compromise while giving time to move to something newer.

"Using a composite can be the difference between chaos and calm"