Composite ML-DSA
Composite ML-KEM

Mike Ounsworth, John Gray, Jan Klaussner, Max Pala, Scott Fluhrer

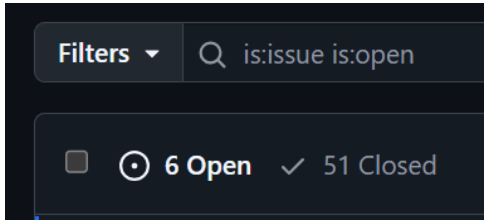
November 2024

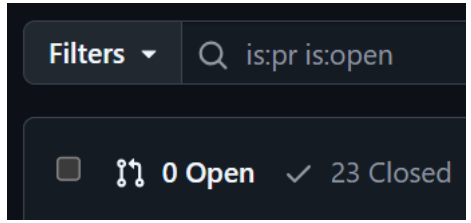
Background / Refresher

History / where we are today

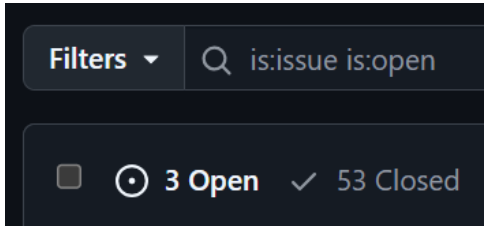
- These drafts have taken a lot of shapes since first introduced in 2019. A lot of simplifying and focusing.
- We have done another HUGE round of edits since IETF 120. (We basically dealt with and closed 1 issue / day since August)

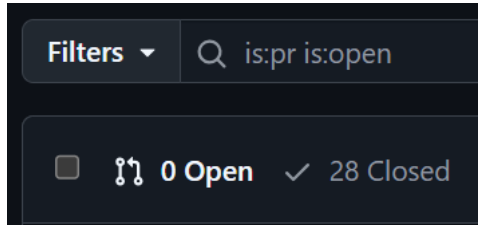
Sigs:





KEMs:





- Today, these drafts are “Composite ML-DSA” and “Composite ML-KEM”. IE they are now direct mirrors of:

FIPS 204
draft-ietf-lamps-dilithium-certificates
draft-salter-lamps-cms-ml-dsa

FIPS 203
draft-ietf-lamps-kyber-certificates
draft-ietf-lamps-cms-kyber

ENTRUST

Change log since IETF 120

- Both documents start with a “Changes in this version” section.
  - https://datatracker.ietf.org/doc/html/draft-ietf-lamps-pq-composite-sigs-03
  - https://datatracker.ietf.org/doc/html/draft-ietf-lamps-pq-composite-kem-05
- We will only touch on the major items in this presentation.

- **Summary:**
  - We think the composite drafts have the right “shape” now, but we still have details to work out and implementation experience to gain with the new changes.

ENTRUST

Cryptographic changes to Composite ML-DSA since IETF 120

draft-ietf-lamps-pq-composite-sigs

ENTRUST

Cryptographic changes to Composite ML-DSA since IETF 120
Pure Signature Mode – “Composite-ML-DSA.Sign()”

```
Composite-ML-DSA.Sign(sk, M, ctx):
1. If |ctx| > 255: return error
2. Compute a hash of the Message
        M' = Domain || len(ctx) || ctx || M
3.Separate the private key into component keys:
        (mldsaSK, tradSK) = sk
4. Generate the 2 component signatures independently, calculating
   signature over M’ according to specification
        mldsaSig = ML-DSA.Sign( mldsaSK, M’, ctx=Domain)
        tradSig = Trad.Sign( tradSK, M’)
5. If either ML-DSA.Sign() or Trad.Sign() fails, then return an error
6. Encode each component signature into a CompositeSignatureValue
        signature ::= CompositeSignatueValue(mldsaSig, tradSig)
7. Output signature
```

API matches FIPS 204 s. 5.2 Algorithm 2

Now exactly matches FIPS 204 s. 5.2 Algorithm 2 line 10

Make use of ML-DSA ctx for better non-separability.

Note: RSAv1.5/PSS and ECDSA don’t have a ctx. EdDSA does, but the “EdDSA in X.509 / CMS” RFCs explicitly disallow using it.
Therefore the Trad component does not get the extra non-separability protection.

ENTRUST

Cryptographic changes to Composite ML-DSA since IETF 120
Pre-hashed Signature Mode -- “HashComposite-ML-DSA.Sign()”

```
HashComposite-ML-DSA.Sign(sk, M, ctx, PH):
1. If |ctx| > 255: return error
2. Compute a hash of the Message
        M' :=  Domain || len(ctx) || ctx || HashOID || PH(M)
3.Separate the private key into component keys:
        (mldsaSK, tradSK) = sk
4. Generate the 2 component signatures independently, calculating
   signature over M’ according to specification
        mldsaSig = ML-DSA.Sign( mldsaSK, M’, ctx=Domain)
        tradSig = Trad.Sign( tradSK, M’)
5. If either ML-DSA.Sign() or Trad.Sign() fails, then return an error
6. Encode each component signature into a CompositeSignatureValue
        signature ::= CompositeSignatueValue(mldsaSig, tradSig)
7. Output signature
```

API matches FIPS 204 s. 5.4 Algorithm 4

Now exactly matches FIPS 204 s. 5.4 Algorithm 5 line 24

Since we’ve already pre-hashed, use pure ML-DSA.Sign().

Make use of ML-DSA ctx for better non-separability.

Note: RSAv1.5/PSS and ECDSA don’t have a ctx. EdDSA does, but the “EdDSA in X.509 / CMS” RFCs explicitly disallow using it. Therefore the Trad component does not get the extra non-separability protection.

Cryptographic changes to Composite ML-KEM since IETF 120

draft-ietf-lamps-pq-composite-kem

ENTRUST

Cryptographic changes to Composite ML-KEM since IETF 120

```
Composite-ML-KEM.Encap(pk) :
1. Separate the public keys.
      (mlkemPK, tradPK) = pk
2.  Perform the respective component Encap operations according to
    their algorithm specifications.
      (mlkemCT, mlkemSS) = MLKEM.Encaps(mlkemPK)
      (tradCT, tradSS)   = TradKEM.Encap(tradPK)
3. If either ML-KEM.Encaps() or TradKEM.Encap() return an error,
     then this process must return an error.
4. Encode and combine
     ct = CompositeCiphertextValue(mlkemCT, tradCT)
     ss = KDF(mlkemSS || tradSS || tradCT || tradPK || Domain)
5. return (ss, ct)
```

KDF is either HKDF-SHA256 or SHA3-256 (aligns with X-Wing).

Authors are in discussion with NIST about whether this can be FIPS-certified under SP 800-56Cr2, or SP 800-227.

ENTRUST

Combinations list

Composite ML-DSA – Pure

#	Composite Signature AlgorithmID	ObjectID	First Algorithm	Second Algorithm
1	id-MLDSA44-RSA2048-PSS	<CompSig>.21	id-ML-DSA-44	id-RSASA-PSS with id-sha256
2	id-MLDSA44-RSA2048-PKCS15	<CompSig>.22	id-ML-DSA-44	sha256WithRSAEncryption
3	id-MLDSA44-Ed25519	<CompSig>.23	id-ML-DSA-44	id-Ed25519
4	id-MLDSA44-ECDSA-P256	<CompSig>.24	id-ML-DSA-44	ecdsa-with-SHA256 with secp256r1
5	id-MLDSA65-RSA3072-PSS	<CompSig>.26	id-MLDSA65	id-RSASA-PSS with id-sha256
6	id-MLDSA65-RSA3072-PKCS15	<CompSig>.27	id-MLDSA65	sha256WithRSAEncryption
7	id-MLDSA65-RSA4096-PSS	<CompSig>.34	id-MLDSA65	id-RSASA-PSS with id-sha384
8	id-MLDSA65-RSA4096-PKCS15	<CompSig>.35	id-MLDSA65	sha384WithRSAEncryption
9	id-MLDSA65-ECDSA-P384	<CompSig>.28	id-MLDSA65	ecdsa-with-SHA384 with secp384r1
10	id-MLDSA65-ECDSA-brainpoolP256r1	<CompSig>.29	id-MLDSA65	ecdsa-with-SHA256 with brainpoolP256r1
11	id-MLDSA65-Ed25519	<CompSig>.30	id-MLDSA65	id-Ed25519
12	id-MLDSA87-ECDSA-P384	<CompSig>.31	id-MLDSA87	ecdsa-with-SHA384 with secp384r1
13	id-MLDSA87-ECDSA-brainpoolP384r1	<CompSig>.32	id-MLDSA87	ecdsa-with-SHA384 with brainpoolP384r1
14	id-MLDSA87-Ed448	<CompSig>.33	id-MLDSA87	id-Ed448

ENTRUST

Composite ML-DSA – Pre-Hash

#	Composite Signature AlgorithmID	ObjectID	First Algorithm	Second Algorithm	Pre-Hash
1	id-MLDSA44-RSA2048-PSS-SHA256	<CompSig>.40	id-ML-DSA-44	id-RSASA-PSS with id-sha256	id-sha256
2	id-MLDSA44-RSA2048-PKCS15-SHA256	<CompSig>.41	id-ML-DSA-44	sha256WithRSAEncryption	id-sha256
3	id-MLDSA44-Ed25519-SHA512	<CompSig>.42	id-ML-DSA-44	id-Ed25519	id-sha512
4	id-MLDSA44-ECDSA-P256-SHA256	<CompSig>.43	id-ML-DSA-44	ecdsa-with-SHA256 with secp256r1	id-sha256
5	id-MLDSA65-RSA3072-PSS-SHA512	<CompSig>.44	id-MLDSA65	id-RSASA-PSS with id-sha256	id-sha512
6	id-MLDSA65-RSA3072-PKCS15-SHA512	<CompSig>.45	id-MLDSA65	sha256WithRSAEncryption	id-sha512
7	id-MLDSA65-RSA4096-PSS-SHA512	<CompSig>.46	id-MLDSA65	id-RSASA-PSS with id-sha384	id-sha512
8	id-MLDSA65-RSA4096-PKCS15-SHA512	<CompSig>.47	id-MLDSA65	sha384WithRSAEncryption	id-sha512
9	id-MLDSA65-ECDSA-P384-SHA512	<CompSig>.48	id-MLDSA65	ecdsa-with-SHA384 with secp384r1	id-sha512
10	id-MLDSA65-ECDSA-brainpoolP256r1-SHA512	<CompSig>.49	id-MLDSA65	ecdsa-with-SHA256 with brainpoolP256r1	id-sha512
11	id-MLDSA65-Ed25519-SHA512	<CompSig>.50	id-MLDSA65	id-Ed25519	id-sha512
12	id-MLDSA87-ECDSA-P384-SHA512	<CompSig>.51	id-MLDSA87	ecdsa-with-SHA384 with secp384r1	id-sha512
13	id-MLDSA87-ECDSA-brainpoolP384r1-SHA512	<CompSig>.52	id-MLDSA87	ecdsa-with-SHA384 with brainpoolP384r1	id-sha512
14	id-MLDSA87-Ed448-SHA512	<CompSig>.53	id-MLDSA87	id-Ed448	id-sha512

Composite ML-KEM

#	Composite ML-KEM Algorithm	ObjectID	First Algorithm	Second Algorithm	KDF
1	id-MLKEM768-RSA2048	<CompKEM>.21	MLKEM768	RSA-OAEP 2048	HKDF-SHA256/256
2	id-MLKEM768-RSA3072	<CompKEM>.22	MLKEM768	RSA-OAEP 3072	HKDF-SHA256/256
3	id-MLKEM768-RSA4096	<CompKEM>.23	MLKEM768	RSA-OAEP 4096	HKDF-SHA256/256
4	id-MLKEM768-X25519	<CompKEM>.24	MLKEM768	X25519	SHA3-256
5	id-MLKEM768-ECDH-P384	<CompKEM>.25	MLKEM768	ECDH-P384	HKDF-SHA256/256
6	id-MLKEM768-ECDH-brainpoolP256r1	<CompKEM>.26	MLKEM768	ECDH-brainpoolp256r1	HKDF-SHA256/256
7	id-MLKEM1024-ECDH-P384	<CompKEM>.27	MLKEM1024	ECDH-P384	SHA3-256
8	id-MLKEM1024-ECDH-brainpoolP384r1	<CompKEM>.28	MLKEM1024	ECDH-brainpoolP384r1	SHA3-256
9	id-MLKEM1024-X448	<CompKEM>.29	MLKEM1024	X448	SHA3-256

Now outputs a 32-byte shared secret key at all security levels, same as ML-KEM (FIPS 203).

ENTRUST

Miscellaneous other changes

Changes affecting Implementation Interoperability

- Compacted CompositeSignaturePrivateKey
  - OLD:
    ```
    CompositeSignaturePrivateKey ::= SEQUENCE SIZE (2) OF OneAsymmetricKey
    ```
  - NEW:
    ```
    CompositeSignaturePrivateKey ::= SEQUENCE SIZE (2) OF OCTET STRING
    ```
  - This was a hold-over from generic composite when we needed to carry the component AlgorithmIDs.
  - Open Question: How exactly do we carry ML-* keys? Seed? Expanded? See Open Questions section.
    X-Wing team points out that ML-* and Ed keys can both be generated on-the-fly from a single seed.
- Fully aligned “Use in CMS” sections to `draft-salter-lamps-cms-ml-dsa` & `draft-ietf-lamps-cms-Kyber`.

ENTRUST

Working Implementations – Dynamic Hackathon Results

Note: interop for composites -03 done at Hackathon – more implementations being worked on (oqsprovider, cryptonext, Entrust, Carl and others)

✅ = 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	carl-redhound	cht	cnsprovider	corey-digicert	cryptonext	cryptonext-cnsprovider	kris	seventhsense.ai
ML-DSA-44	✅	✅	✅	✅	✅	✅	✅	✅	✅
ML-DSA-65	✅	✅	✅	✅	✅	✅	✅	✅	✅
ML-DSA-87	✅	✅	✅	✅	✅	✅	✅	✅	✅
id-MLDSA44-RSA2048-PSS	✅								✅
id-MLDSA44-RSA2048-PKCS15	✅								✅
id-MLDSA44-Ed25519	✅								✅
id-MLDSA44-ECDSA-P256	✅								✅
id-MLDSA65-RSA3072-PSS	✅								✅
id-HashMLDSA65-RSA4096-PSS-SHA512	✅								✅
id-HashMLDSA65-RSA4096-PKCS15-SHA512	✅								✅
id-HashMLDSA65-ECDSA-P384-SHA512	✅								✅
id-HashMLDSA65-Ed25519-SHA512	✅								✅
id-HashMLDSA87-ECDSA-P384-SHA512	✅								✅

ENTRUST

Open Questions and Next Steps

Open Questions – ML-* private key format?

- 3 options
  - 1) `ML-*PrivateKeyFull ::= OCTET STRING`
  - 2) `ML-*PrivateKeySeed ::= OCTET STRING (SIZE X) -- X is 32 for ML-DSA and 64 for ML-KEM`
  - 3)
```
ML-*PrivateKey ::= CHOICE {
          ML-*PrivateKeySeed,
      [0] ML-*PrivateKeyFull }
```
- The argument for Seed is bandwidth and security – you can't play games where you mismatch public and private key if the signer / decryptor re-derives the keypair from seed on each use.
- The argument for Full (or Both) is that no matter how nicely we ask, some hardware modules will store the seed expanded for performance reasons, and will not bother to keep the seed around, and will eventually need an interoperable way to export that.
- **Question:** The Seed-based private key assumes runtime-access to ML-DSA.KeyGen_internal(ξ) which FIPS 203 / 204 make clear **"should not be made available to applications"**... so can you even FIPS-certify a seed-based private key? Or do we need to provide a FullPrivKey p12 format for FIPS-mode applications?

ENTRUST

Open Questions – ML-* private key format for composites?

- 3 options
  - 1) `ML-*PrivateKeyFull ::= OCTET STRING`
  - 2) `ML-*PrivateKeySeed ::= OCTET STRING (SIZE X) -- X is 32 for ML-DSA and 64 for ML-KEM`
  - 3)
```
ML-*PrivateKey ::= CHOICE {
          ML-*PrivateKeySeed,
      [0] ML-*PrivateKeyFull }
```
- So what does that mean for Composite ML-*?
- The obvious answer is that whatever we choose for pure ML-* priv keys just becomes the ML-* component of a CompositePrivateKey.
- But the X-Wing team points out that for ML-* + Ed/X curves, you can efficiently regen both components from a single 32-byte seed. Which is nice. So should those combos have an optimized private key and corresponding optimized KeyGen()?

```
CompositeMLKEMX25519PrivateKey ::= OCTET STRING (SIZE 32)
```


ENTRUST

Is HKDF FIPS-approved?

- **Background**, for the KEM Combiner, the “higher” combinations for Composite ML-KEM use SHA3-256 while the “lower” combinations use HKDF-SHA2-256.
- Problem: HKDF (RFC 5869) has two steps:
  - Step 1: HKDF-Extract(salt, IKM) -> PRK
    PRK = HMAC-Hash(salt, IKM)
  - Step 2: HKDF-Expand(PRK, info, L) -> OKM
    … iterated applications of HMAC-Hash() until you get the required amount of output.
- **Problem**: Step 1 is ok as a post-ML-KEM KDF as per SP 800-56Cr2 Section 4 Option 2. However, **Step 2 is not FIPS-approved at all**. (thanks Panos for confirmation).
- **Solution**: (I think) is to change the Composite ML-KEM draft to only use the HKDF-Extract step, but I would like a second pair of eyes on this before I make the change.

ENTRUST

Next Steps

- After this re-vamp, `composite-sigs-03` and `composite-kems-05` need soaking time and implementation experience (hackathon).
- Our hope is that these 6 drafts will be ready around the same time:
  - ML-DSA: draft-ietf-lamps-dilithium-certificates, draft-salter-lamps-cms-ml-dsa, draft-ietf-lamps-pq-composite-sigs.
  - ML-KEM: draft-ietf-lamps-kyber-certificates, draft-ietf-lamps-cms-kyber, draft-ietf-lamps-pq-composite-kem

ENTRUST