FIPS “issues” with deploying ML-KEM and ML-DSA

PrivKeys, P12, P11, Hybrids, and beyond!

Mike Ounsworth
PQUIP, IETF 121

With input (but maybe not full agreement) from:
Paul Hoffman, Bas Westerbaan,
Deirdre Connolly, Sophie Schmieg, Gorjan Alagic

!!! Disclaimers !!!

- Purpose of this presentation: to summarize the current status of ongoing discussions relating to compatibility of IETF drafts with FIPS 203 / 204.

- My personal understanding may be incorrect
- Info is very fresh and still evolving:
  - pqc-forum mailing list discussions last week and ongoing.
  - Meeting Oct 17 between IETF hybrid KEM authors + NIST PQC team, follow-up meeting maybe next week

- Deirdre Connolly's blog is seriously better than my slides, read it.
  - https://durumcrustulum.com/2024/02/24/how-to-hold-kems/

Topics to cover

1. ML-KEM, ML-DSA private keys: “seed” vs “expanded”
    a. Cryptographic pros / cons
    b. Implications for FIPS certification
        i. What do FIPS 203 / 204 say, **exactly*,* about using *KeyGen_internal(seed)* from application layer.
2. Direct seed vs derived seed, and implications for IETF protocols
3. Interaction with PKCS#12 files (IETF) and PKCS#11 HSM API (OASIS)
    a. Basically, once there exist HSMs that store expanded key and do not keep the seed, then we will need to support expanded keys forever. (or at least, owners of such hardware need to use non-standard export / import mechanisms).
4. Hybrid ML-KEM combiner: SP 800-56Cr2, SP 800-227 and issue with KDF(ec || mlkem) vs KDF(mlkem || ec)
5. ML-DSA vs HashML-DSA vs “RealHash-ML-DSA”

Topic #1
Lattice Private Keys: Seed vs Expanded

KeyGen() vs KeyGen_internal(seed)

- ML-DSA.KeyGen_internal(ξ) ← 32 bytes
- ML-KEM.KeyGen_internal(d, z) ← 2x32 bytes = 64 bytes of seed.

- Obviously, size / bandwidth 👍
- Security: Sophie Schmieg dubbed the “Unbindable Kemmy Schmidt” attacks [1] where (simplified) expanded private keys can be manipulated to force exploitable conditions leading to “re-encapsulation attacks” where Mallory can force two separate sessions Alice <-> Mallory and Mallory <-> Bob to arrive at the same shared secret key.
- If the honest decryptor re-derives their priv key from seed at decrypto time, then these attacks go away.
    - Especially if (d, z) are both derived from a single 32-byte seed.

[1]: S. Schmieg, “Unbindable Kemmy Schmid”, 2024 https://eprint.iacr.org/2024/523.pdf

KeyGen() vs KeyGen_internal(seed)

- ML-DSA.KeyGen_internal(ξ) ← 32 bytes
- ML-KEM.KeyGen_internal(d, z) ← 2x32 bytes = 64 bytes of seed.

- Obviously, size / bandwidth 👍
- Security: Sophie Schmieg dubbed the "Unb
  where (simplified) expanded private keys c
  conditions leading to "re-encapsulation atta
  separate sessions Alice <-> Mallory and Ma
  shared secret key.
- If the honest decryptor re-derives their priv key from seed at decrypto time, then these attacks go away.
  - Especially if (d, z) are both derived from a single 32-byte seed.

So seed private keys are all up-side with no down-side.

Next: are they allowed in FIPS mode?
Answer: yes (but confusing).
Let's take a look.

[1]: S. Schmieg, 2024 https://eprint.iacr.org/2024/523.pdf

KeyGen() vs KeyGen_internal(seed)

- ML-DSA.KeyGen_internal(ξ) ← 32 bytes
- ML-KEM.KeyGen_internal(d, z) ← 2x32 bytes = 64 bytes of seed.

This part, while academically true, unclear if it's worth the effort.

General consensus: 64-byte ML-KEM seeds are not worth the hassle. More on this later…

So seed private keys are all up-side with no down-side.

Next: are they allowed in FIPS mode?
Answer: yes (but confusing).
Let's take a look.

- 👍 bbed the "Unb
  - private keys c
  - apsulation atta
  - Mallory and Ma
  - sha secret key.
- If the honest decryptor re-derives their priv key from seed at decrypto time, then these attacks go away.
  - Especially if (d, z) are both derived from a single 32-byte seed.

[1]: S. Schmieg, 2024 https://eprint.iacr.org/2024/523.pdf

Private key seed mode

FIPS 203 (ML-KEM) section 6:

> *"The key generation [internal] function in this section may also be used to re-expand a key from a seed (see Section 3.3), including when obtaining assurance of private key possession via regeneration.*
>
> *The [KeyGen_internal] interface specified in this section **should not** be made available to applications other than for testing purposes, and the random seeds shall be generated by the cryptographic module."*

So it sounds like I can do private key seeds within the crypto module, but not importing a seed from, for example, a PEM file or JWK because that would require KeyGen_internal to be made available to the application.

> … but wait, there's more 🤔

Private key seed mode
KeyGen() vs KeyGen_internal(seed)

FIPS 203 (ML-KEM) section 3.3 says:

“Destruction of intermediate values.

…

1. *The seed (d, z) generated in steps 1 and 2 of ML-KEM.KeyGen can be stored for later expansion using ML-KEM.KeyGen_internal. As this seed can be used to compute the decapsulation key, it is sensitive data and shall be treated with the same safeguards as a decapsulation key (see SP 800-227 [1]).”*

Private key seed mode KeyGen() vs KeyGen_internal(seed)

But FIPS 203 (ML-KEM) section 3.3 says:

"Destruction of intermediate values.

…

1. *The seed (d, z) generated in steps 1 and 2 of ML-KEM.KeyGen can be stored for later expansion using ML-KEM.KeyGen_internal. As this seed can be used to compute the decapsulation key, it is sensitive data and shall be treated with the same safeguards as a decapsulation key (see SP 800-227 [1]).”*

I guess this means that anywhere an expanded priv key can be stored, also a seed key can be stored in the same way? So seed keys in .p12 files (ie imported from outside the crypto module) are ok?

How does that line up with section 6's "KeyGen_internal **should not** be made available to applications" ??

Private key seed mode KeyGen() vs KeyGen_internal(seed)

But FIPS 203 (ML-KEM) section 3.3 says:

"Destruction of intermediate values.

…

1. *The seed (d, z) generated in steps 1 and 2 of ML-KEM.KeyGen can be stored for later expansion using ML-KEM.KeyGen_internal. As this seed can be used to compute the decapsulation key, it is sensitive data and shall be treated with the same safeguards as a decapsulation key (see SP 800-227 [1]).*"

I guess this means that anywhere an expanded priv key can be stored, also a seed key can be stored in the same way? So seed keys in .p12 files (ie imported from outside the crypto module) are ok?

How does that line up with section 6's "KeyGen_internal **should not** be made available to applications" ??

[1] National Institute of Standards and Technology (2024) Recommendations for key-encapsulation mechanisms, (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-227. [Forthcoming; will be available at https://csrc.nist.gov/publications].

Private key seed mode KeyGen() vs KeyGen_internal(seed)

But FIPS 203 (ML-KEM) section 3.3 says:

"Destruction of intermediate values.

…

1. *…nd 2 of ML-KEM.KeyGen can be stored for later …ternal. As this seed can be used to compute the …and shall be treated with the same safeguards as …[1]).”*

…nd Techno…y (2024) Recommendations for key-encapsulation mechanis…, (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-227. [Forthcoming; will be available at https://csrc.nist.gov/publications].

I guess this means that anywhere an expanded priv key can be stored, also a seed key can be stored in the same way? So seed keys in .p12 files (ie imported from outside the crypto module) are ok?

How does that line up with section 6's "KeyGen_internal **should not** be made available to applications" ??

As of this presentation, we haven't yet seen a public draft of SP 800-227.

NIST has scheduled a "Virtual Workshop on 800-227" on Feb 24, 2025, so we'll get details then.

Private key seed mode
… but wait there's more

FIPS 204 (ML-DSA) section 6:

> *"Other than for testing purposes, the interfaces for key generation and signature generation specified in this section should not be made available to applications, as any random values required for key generation and signature generation shall be generated by the cryptographic module."*

That first sentence in FIPS 203 that specifically allows re-expansion from a seed is just missing from FIPS 204 section 6.

Private key seed mode
… but wait there's more

Fortunately, FIPS 204 section 3.6.3 has the more detailed text that we care about:

> *"The seed ξ generated in step 1 of ML-DSA.KeyGen can be stored for the purpose of later expansion using ML-DSA.KeyGen_internal. As the seed can be used to compute the private key, it is sensitive data and shall be treated with the same safeguards as a private key."*

Ok, fine, Let's chalk this up to **"Poorly-worded, but intention is clear"**. Seeds are allowed anywhere that an expanded key is.

… moving on.

Topic #2
Direct Seed vs Derived Seed

Direct seed vs Derived seed

ML-KEM.KeyGen_internal(d, z)

- We mentioned above [1], [2] that ML-KEM gains security benefits from

```
byte[32] seed = loadSeedFromWherever();
byte[64] (d,z) = KDF(seed,64);
ML-KEM.KeyGen_internal(d,z);
```

[1]: Deirdre Connolly's blog: https://durumcrustulum.com/2024/02/24/how-to-hold-kems/
[2]: S. Schmieg, "Unbindable Kemmy Schmid", 2024 https://eprint.iacr.org/2024/523.pdf

Direct seed vs Derived seed

ML-KEM.KeyGen_internal(d, z)

- We mentioned above [1] that ML-KEM security benefits from

```
byte[32] seed = loadSeedFromWherever();
byte[64] (d,z) = KDF(seed,64);
ML-KEM.KeyGen_internal(dz);
```

- Unfortunately, this is explicitly FIPS disallowed [2] 😢 (but NIST "is considering it")

Alagic, Gorjan (Assoc)

to Phillip Hallam-Baker, Tony Arcieri, pqc-forum

Using only a 32-byte seed for ML-KEM is not allowed by FIPS 203, and we have no plans to allow it. I
applications. However, we do not see this as a sufficient reason to change FIPS 203, either directly o

[1]: Deirdre Connolly's blog: https://durumcrustulum.com/2024/02/24/how-to-hold-kems/
[2]: Gorjan Alagic (NIST) response on NIST pqc-forum:
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/1r6FnG0coiM/m/3JQKgU2iAgAJ

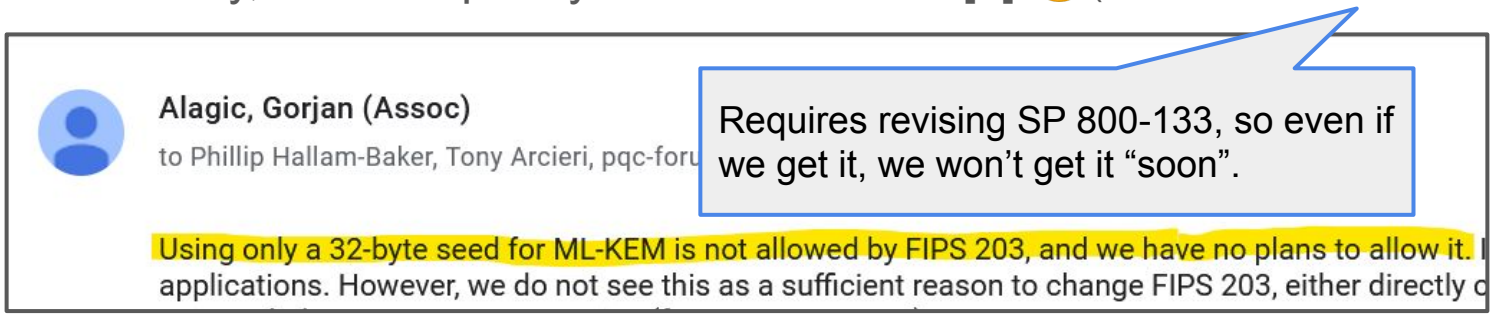
Direct seed vs Derived seed

ML-KEM.KeyGen_internal(d, z)

- We mentioned above [1] that ML-KEM security benefits from

```
byte[32] seed = loadSeedFromWherever();
byte[64] (d,z) = KDF(seed,64);
ML-KEM.KeyGen_internal(dz);
```

- Unfortunately, this is explicitly FIPS disallowed [2] 😢 (but NIST "is considering it")



[1]: Deirdre Connolly's blog: https://durumcrustulum.com/2024/02/24/how-to-hold-kems/
[2]: Gorjan Alagic (NIST) response on NIST pqc-forum:
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/1r6FnG0coiM/m/3JQKgU2iAgAJ

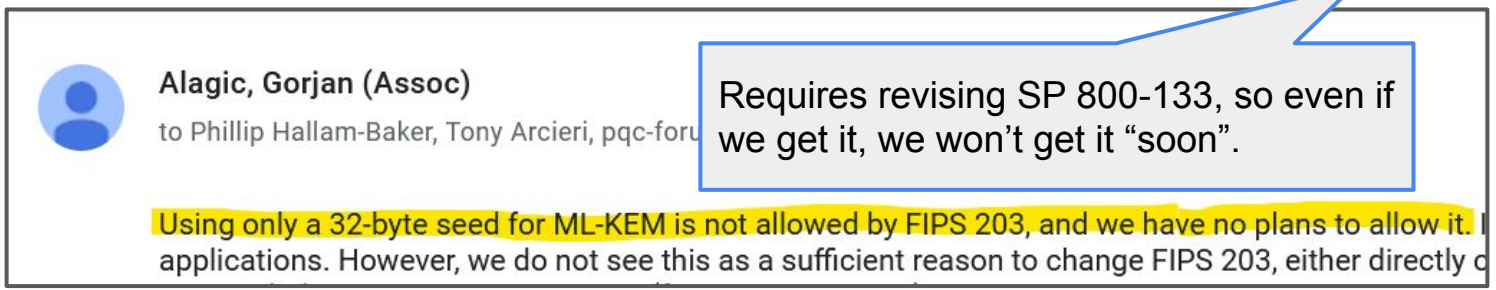
Direct seed vs Derived seed

ML-KEM.KeyGen_internal(d, z)

- We mentioned above [1] that ML-KEM
  ```
  byte[32] seed = loadSeedFrom
  byte[64] (d,z) = KDF(seed,64
  ML-KEM.KeyGen_internal(dz);
  ```
- Unfortunately, this is explicitly FIPS disallowed [2] 😢 (but NIST "is considering it")

Specifically, the seed used with KeyGen_internal() must be the DIRECT output of a NIST-approved DRBG … no hash functions or KDFs in between.

Let's explore some other implications of this …



Requires revising SP 800-133, so even if we get it, we won't get it "soon".

[1]: Deirdre Connolly's blog: https://durumcrustulum.com/2024/02/24/how-to-hold-kems/
[2]: Gorjan Alagic (NIST) response on NIST pqc-forum:
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/1r6FnG0coiM/m/3JQKgU2iAgAJ

Direct seed vs Derived seed

Implication for hybrids

Deirdre Connolly and Bas Westerbaan came up with this clever observation [1] that since ML-KEM-768 and X25519 each claim <= 256 bits security, why not derive all of it from a single 256-bit (32-byte) seed?

```
byte[32] seed = loadSeedFromWherever();
byte[96] (d,z,skX) = KDF(seed,96);
XWing.KeyGen(d,z,skX);
```

It’s good for bandwidth, KeyGen() is efficient for both schemes, and no possibility of mismatch attacks between the ML-KEM and X25519 components.

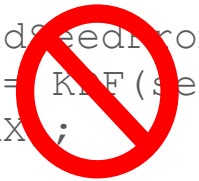
[1]: https://datatracker.ietf.org/doc/html/draft-connolly-cfrg-xwing-kem-04#name-key-generation

Direct seed vs Derived seed

Implication for hybrids

Deirdre Connolly and Bas Westerbaan came up with this clever observation [1] that since ML-KEM-768 and X25519 each claim <= 256 bits security, why not derive all of it from a single 256-bit (32-byte) seed?

```
byte[32] seed = loadSeedFromWherever();
byte[96] (d,z,skX) = KDF(seed,96);
XWing.KeyGen(d,z,skX);
```



It's good for bandwidth, KeyGen() is efficient for both schemes, and no possibility of mismatch attacks between the ML-KEM and X25519 components.

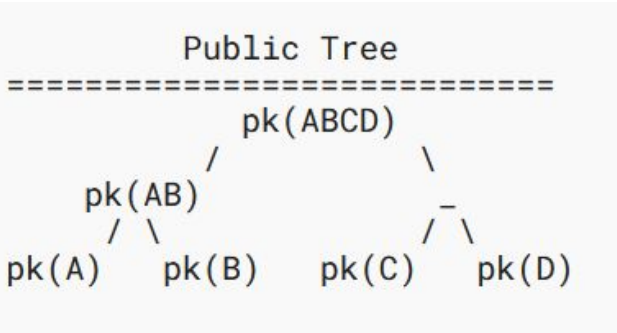
Unfortunately, this is explicitly FIPS disallowed [2]😢 (but NIST "is considering it")

[1]: https://datatracker.ietf.org/doc/html/draft-connolly-cfrg-xwing-kem-04#name-key-generation
[2] Gorjan Alagic (NIST) response on NIST pqc-forum:
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/1r6FnG0coiM/m/3JQKgU2iAgAJ

Direct seed vs Derived seed

Implication for ratcheting protocols

IETF's MLS has a ratchet tree that feeds derived secrets into KEM.KeyGen().




```
node_secret[n] = DeriveSecret(path_secret[n], "node")
node_priv[n], node_pub[n] = KEM.DeriveKeyPair(node_secret[n])
```

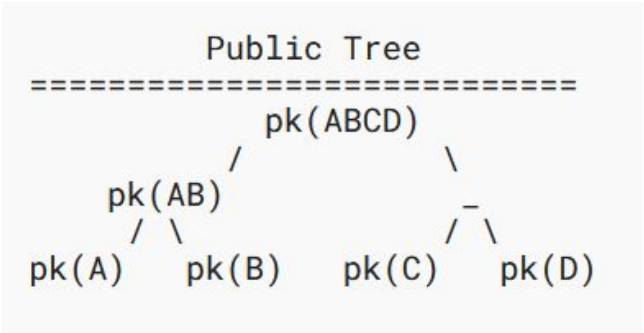

[1]: RFC 9420

[2] Gorjan Alagic (NIST) response on NIST pqc-forum:
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/1r6FnG0coiM/m/3JQKgU2iAgAJ

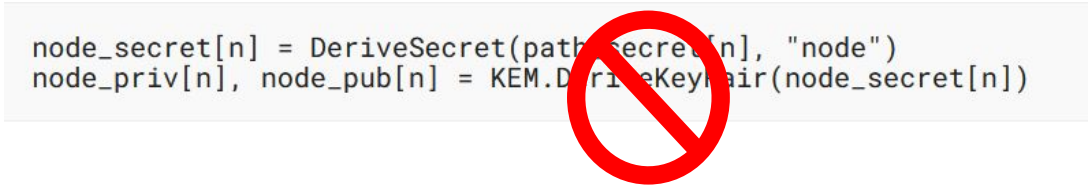
Direct seed vs Derived seed

Implication for ratcheting protocols

IETF's MLS also has a ratchet tree that feeds derived secrets into KEM.KeyGen().



Unfortunately, this is explicitly FIPS disallowed [2]😢



Certain members of the IETF community most certainly care about having FIPS-certified MLS implementations.

[1]: RFC 9420
[2] Gorjan Alagic (NIST) response on NIST pqc-forum:
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/1r6FnG0coiM/m/3JQKgU2iAgAJ

Topic #3
Lattice private keys, HSMs, PKCS#11, and PKCS#12

The horror scenario

- **Fact #1**: If you implement ML-KEM / ML-DSA in cryptographic hardware looking only at FIPS 203 / 204, you may choose to only store the expanded key.
  - FIPS 203 Section 6: *“KeyGen_internal **should not** be made available to applications, other than for testing purposes.”*
- **Fact #2**: If you did not store the seed at KeyGen time, you can never get it back because Seed → Expanded is a one-way-function.
- **Fact #3**: Some keys have extremely high value; in disaster scenarios or when hardware hits end-of-life, or whatever, we need to be able to get keys off hardware onto new hardware.
- **Fact #4**: All IETF WGs are leaning towards Seed-only private key formats.

1 + 2 + 3 + 4 = We will end up with important production keys that we cannot export and transport over any standard key format.

Our only hope is to get in touch with the HSM people NOW and make sure that all implementations keep the seeds around.

IETF + OASIS PKCS#11 Liaison effort?

I think we need some ASAP collaboration with our counterparts at the OASIS PKCS#11 TC so that we all agree to the same ML-KEM and ML-DSA private key formats, and we avoid the one-way-trap problem.

I’m willing to spearhead this (I already started collecting volunteers on the NIST pqc-forum [1])

[1]: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/2EWXvBaE-Yw/m/wMhvJUaiAAAJ

Topic #4
Hybrid KEM Combiner

SP 800-56Cr2, SP 800-227 and issue with
KDF(ec || mlkem) vs KDF(mlkem || ec)

Background: FIPS implications of Hybrid KEMs

One of the large drivers of PQ/T Hybrids is certification requirements.

- The queue for FIPS certification is years-long, so a hybrid with your already-certified RSA or EC code allows you to deploy a FIPS-certified ML-KEM hybrid before you have a FIPS-certified pure ML-KEM implementation.
- … or you can deploy a FIPS-certified ML-KEM with a non-FIPS X25519.
- … or when eventually RSA / EC are removed from the FIPS-approved list, you can just leave your deployed hybrids there by resting on the FIPS-ML-KEM.

Notice that in some scenarios, you want to claim FIPS for the RSA / EC part, while others want to claim FIPS for ML-KEM.

So we want to have a single hybrid scheme that can claim the FIPS-allowed component in either “slot”. Having to re-deploy to swap the order of inputs sounds like a nightmare.

How we claim FIPS for a hybrid KEM

SP 800-56Cr2:

> 6. For $i = 1$ to *reps*, do the following:
>
> 6.1 Increment *counter* by 1.
>
> 6.2 Compute $K(i) = \mathrm{H}(counter \,||\, Z \,||\, FixedInfo)$.
>
> 6.3 Set $Result(i) = Result(i-1) \,||\, K(i)$.

800-56Cr2 specifically calls out:

> *"this Recommendation permits the use of a "hybrid" shared secret of the form Z' = Z || T"*
> Where Z is from a FIPS-approved key establishment method, and T is not.

(conceptually) T just counts as the beginning of "*FixedInfo*".

(technically) SP 800-56Cr2 allows for an "auxiliary shared secret".

How we claim FIPS for a hybrid KEM

800-56Cr2 specifically calls out:

> *“this Recommendation permits the use of a “hybrid” shared secret of the form Z' = Z || T”*

HOWEVER, this *requires* the FIPS-approved thing to be in the first position, despite the fact that

KDF(ec || mlkem) vs KDF(mlkem || ec)

are cryptographically equivalent for a “well-behaved” KDF, or (I believe) when both shared secret parts are fixed-length.

- Note: HKDF-SHA2 and SHA3 family are “well-behaved” in this regard.
- Note: for all hybrid KEMs under consideration by the IETF, both shared secret parts are fixed-length.

KDF(ec || mlkem) vs KDF(mlkem || ec)

There has already been one phone call between IETF hybrid KEM authors, and NIST PQC team. We are hoping to have a second follow-up call soon.

We are hoping that SP 800-227 (public draft expected in Feb. 2025) will allow us to claim FIPS regardless of order of inputs. NIST is currently exploring what “guard rails” are needed to do this safely.

Topic #5
ML-DSA vs HashML-DSA vs “RealHash-ML-DSA”

ML-DSA vs HashML-DSA vs “RealHash-ML-DSA”

- A naive reader of FIPS 204 will think that it defines **two** ML-DSA “modes”:

Algorithm 2 ML-DSA.Sign(sk, M, ctx)

Generates an ML-DSA signature.

Algorithm 4 HashML-DSA.Sign$(sk, M, ctx,$ PH$)$

Generate a “pre-hash” ML-DSA signature.

ML-DSA vs HashML-DSA vs “RealHash-ML-DSA”

- A naive reader of FIPS 204 will think that it defines **two** ML-DSA “modes”:

Algorithm 2 ML-DSA.Sign(sk, M, ctx)

Generates an ML-DSA signature.

Algorithm 4 HashML-DSA.Sign$(sk, M, ctx,$ PH$)$

Generate a “pre-hash” ML-DSA signature.

- But in fact there is a **third** mode implied by a comment in Algorithm 7:

Algorithm 7 ML-DSA.Sign_internal(sk, M', rnd)

6: $\mu \leftarrow$ H(BytesToBits$(tr)||M', 64)$ ▷ message representative that may optionally be computed in a different cryptographic module

ML-DSA vs HashML-DSA vs “RealHash-ML-DSA”

- The existence of this third mode is only implied by a comment in Algorithm 7, and therefore it does not have a name. **Let’s call it “RealHash-ML-DSA”**.
- It is, however, a distinct signing routine which requires a different API:

  Algorithm 2a RealHash-ML-DSA.Sign(*sk, μ)*

  Algorithm 7a RealHash-ML-DSA.Sign_internal(sk, μ, rnd)

  where Algorithm 7a is identical to Algorithm 7, but with Step 6 removed.

- Good news is that signatures produced by RealHash-ML-DSA.Sign() can be verified by regular ML-DSA.Verify(), so both modes can use the same AlgID OID.

ML-DSA vs HashML-DSA vs “RealHash-ML-DSA”

- The existence of this third mode is only implied by a comment in Algorithm 7, and therefore it does not have a name. **Let’s call it “RealHash-ML-DSA”**.
- It is, however, a distinct signing routine which requires a different API:

  Algorithm 2a RealHash-ML-DSA.Sign(*sk, μ)*

  Algorithm 7a RealHash-ML-DSA.Sign_interna

  where Algorithm 7a is identical to Algorithm 7, but v

Note: no *ctx* param since in this version, *ctx* needs to be dealt with in the external pre-hash step.

- Good news: signatures produced by RealHash-ML-DSA.Sign() can be verified by regular ML-DSA.Verify(), so both modes can use the same AlgID OID.
- Good news: writing down algorithms and giving them names is literally what the IETF does, so we can probably solve this problem ourselves.

ML-DSA vs HashML-DSA vs “RealHash-ML-DSA”

- Some protocols will require signing large amounts of data where the protocol does not provide a way to pre-hash.
  - Two examples can be found in X.509 (RFC 5280) itself: CRLs, and X.509 certs containing large public keys such as Classic McEcliece (or even ML-DSA-87).
- It was discussed at LAMPS this week that “RealHash-ML-DSA” is LAMPS’ preferred way to handle this and LAMPS would prefer to forget that HashML-DSA (FIPS 204 section 5.4) even exists.
- **Problem**: ML-DSA.Sign() vs RealHash-ML-DSA.Sign() are different APIs that will need to be implemented separately by HSM / smartcard vendors.
- Sophie Schmieg argues that the market will self-correct this if, for example, a smartcard product can not be used to sign CRLs. So maybe we can ignore this problem?
- But still, the fact that this “mode” does not have a name makes it very frustrating to refer to it from an RFC.

Summary

1. ML-KEM, ML-DSA private keys: “seed” vs “expanded” 🙂
   a. This is confusing on a “strict reading” of FIPS 203 / 204, but NIST’s intention is clearly to freely allow use of seed private keys everywhere. Good.
2. Direct seed vs derived seed 😢
   a. Currently, a KDF is NOT ALLOWED in between the Seed and the KeyGen(), which breaks the way a number of IETF protocols want to use ML-KEM. NIST is considering relaxing this, hopefully, in SP 800-227.
3. Interaction with PKCS#12 files (IETF) and PKCS#11 HSM API (OASIS) 🙃
   a. Basically, *everyone* needs to do Seed keys. If IETF does seeds and hardware vendors do expanded, then we’re gonna have problems.
4. SP 800-56Cr2, SP 800-227 and issue with
   KDF(ec || mlkem) vs KDF(mlkem || ec) 😢
   a. NIST is currently studying this and we expect relaxation of this in SP 800-227.
5. ML-DSA vs HashML-DSA vs “RealHash-ML-DSA” 🙃
   a. Maybe we can ignore this? But without a name, it’s hard to refer to it from an RFC.