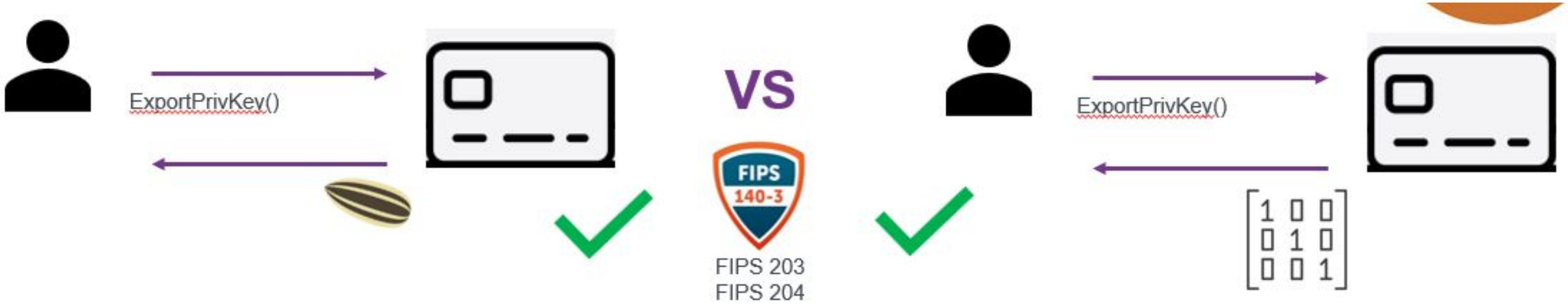
Mike Ounsworth
PQUIP 122

Background: KEY vs SEED



… maybe?

Some seemingly-conflicting wording in FIPS 203 / 204.

FIPS 203

Section 6:

“Controlled access to internal functions. The key-encapsulation mechanism ML-KEM makes use of internal, “derandomized” functions ML-KEM.KeyGen_internal, …. The interfaces for these functions ***should not be made available to applications****”*

But also – Section 3.3:

“There are two exceptions to this rule:

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]).”*

… so … are seed-based private keys allowed?

Around **October 2024**, NIST's CMVP team ruled that
"*can be stored for later expansion using ML-KEM.KeyGen_internal.*"

did not overrule the prohibition on exposing `_internal()` functions outside the crypto module, and therefore you can do this trick internally to the module, but you cannot accept a seed from outside the module.

On **January 30, 2025** NIST posted an FAQ ruling the other way:

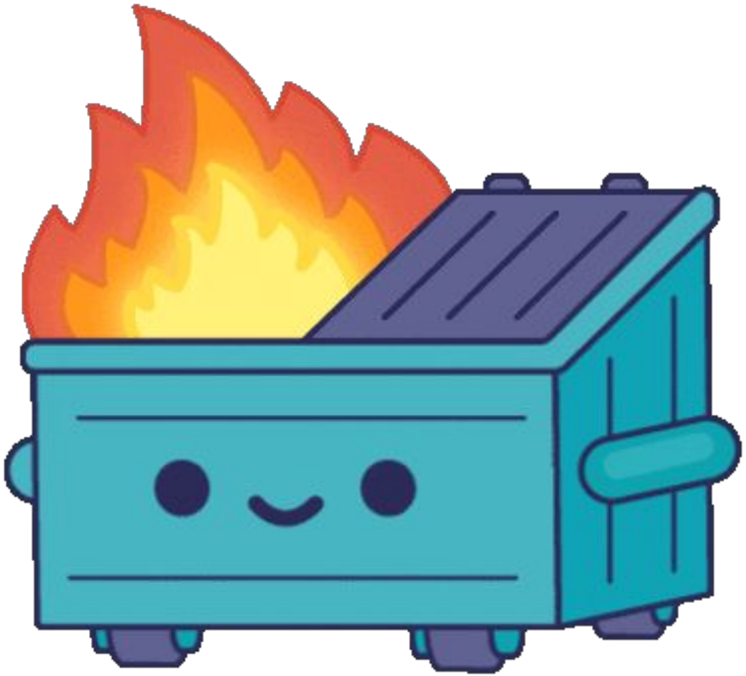
"For both FIPS 203 and FIPS 204, a KeyGen seed is considered an acceptable alternative format for a key-pair, or for the private (i.e., decapsulation or signing) key. In particular, generating the seed in one cryptographic module and then importing or exporting it into another cryptographic module is allowed. The internal key generation functions ML-KEM.KeyGen_Internal(d, z) and ML-DSA.KeyGen_internal(ξ) can be accessed for this purpose."

Good, seeds are allowed, where's the problem?

The problem is that, due to uncertainty about interpretation of the FIPS, many hardware crypto modules entered FIPS validation without support for seeds.

In essence, we already have "legacy" ML-DSA and ML-KEM hardware that PKI software needs to be compatible with. That means we're stuck supporting expanded private keys.

Alright, then let's support both. How hard can it be?



Lessons Learned from LAMPS

- The purpose of this presentation is to share lessons learned so that other WGs don't repeat it.
- Issue: FIPS 203 / 204 allows either KEY or SEED as approved private key formats. So now protocols have to choose between
  `{ key-only, seed-only, choice, both }`.
- Solution: LAMPS has settled on the ASN.1-based encoding

```
ML-KEM-1024-PrivateKey ::= CHOICE {
    seed [0] OCTET STRING (SIZE (64)),
    expandedKey OCTET STRING (SIZE (3168)),
    both SEQUENCE {
         seed OCTET STRING (SIZE (64)),
         expandedKey OCTET STRING (SIZE (3168))
         }
    }
```

Lessons Learned from LAMPS

- Solution: LAMPS has settled on `CHOICE { KEY, SEED, Both }`
- Other options that were explored and rejected:
  - SEED-only
  - KEY-only
  - Both-only
  - Alternative OIDs
  - CHOICE { KEY, SEED } – one or the other
  - SEQUENCE { optional KEY, optional SEED } – one or the other or both

- Let's go through Pros / Cons for each one.
  - Warning: you ***will*** have opinions 🔥 that's the whole point of sharing lessons learned
  - Note: I will use "ML-KEM (FIPS 203)", but everything here applies to ML-DSA (FIPS 204) as well.

SEED-only

<u>Idea</u>: X.509 private key formats would only support SEED-based priv keys.

<u>Pros</u>:

- Good for new applications.
- Good for bandwidth.
- Good for binding security properties.
- Compatible with crypto module imports that only support KEY because you can just do the KeyGen(seed) expansion before calling the import function.
  - Unless policy does not allow a software module to be involved in key import.

<u>Cons</u>:

- Not compatible with crypto module exports that only support KEY since you cannot convert from KEY to SEED.
- Bad for ML-KEM implementations that only support KEY import / export, which is allowed by FIPS, and we already have “legacy” ML-KEM implementations that do this.

KEY-only 🥈

Idea: X.509 private key formats would only support KEY priv keys.

Pros:

- Lowest common denominator: every crypto module can import / export KEYs.
- If we can't reach consensus on any other format, then by process-of-elimination, it will be this one.

Cons:

- Bad for bandwidth.
- Bad for binding security properties.
- FIPS 203 allows SEED, so we should support it somehow for completeness.

Both-only

Idea: Always carry both the SEED and the KEY.

Pros:

- Will be compatible with any SEED-based or KEY-based crypto module import function.

Cons:

- Will be incompatible with KEY-based crypto module export functions (since you can't derive KEY from SEED).
- Adds the potential for SEED and KEY that describe different keys. MUST perform consistency check KeyGen(seed) == KEY, which somewhat defeats the purpose.

Alternative OIDs

Idea: Pick one as the "default" priv. key format for the ML-KEM OIDs assigned by NIST, and assign alternate OIDs for the alternate priv. key format.

Pros:

- Allows parsers to easily know exactly what to expect in the private key bytes.

Cons:

- Either we have to declare private key format in the public certificate (privacy issue), or we have OID mismatch between public key and private key (leads to bugs).

CHOICE { KEY, SEED }

Idea: Producers can do either (but not both), consumers have to accept either.

Pros:

- Will be compatible with any SEED-based or KEY-based crypto module export function.

Cons:

- This forces exporters to know which format the intended importer needs.

SEQUENCE { optional KEY, optional SEED }

Idea: The ASN.1 SEQUENCE with OPTIONAL elements allows natively for “one or the other or both”.

Pros:

- Trivial to implement in ASN.1.
- Can carry both to support any kind of import.

Cons:

- Adds extra bytes (4?), which is significant for a 32 byte seed.
- Relies heavily on ASN.1 encoding, so will not translate well to other protocols (TLS, COSE, etc).

CHOICE { KEY, SEED, Both } 🏅

Idea: Export whatever you want.

Pros:

- Maximum interoperability flexibility with limited bytes-on-the-wire overhead.
- Can carry both to support any kind of import.

Cons:

- Has the potential for a mis-match between key format and a crypto module's import function.
  - Forwards complexity to user to either export in the correct format for the target import, or run an external KeyGen(seed) prior to import(KEY).
- Adds the potential for SEED and KEY that describe different keys. MUST perform consistency check (ie KeyGen(seed) == KEY ?).
- BOTH is an ASN.1 structure, so will not translate well to other protocols (TLS, COSE, etc).

Bonus topic: it looks like ML-DSA External Mu is not allowed in FIPS mode either …

As of an FAQ posted **Feb 24, 2025** and a discussion held on **Mar 10, 2025**, it looks like ML-DSA external mu is allowed specifically when one FIPS-approved module computes Mu and then calls a second FIPS-approved module that completes the signature.

This helps, for example in the case of an HSM and its proprietary driver.

But in general we would like Compute_mu(pk, M, ctx) and Sign_mu(sk, mu) to be fully separable so that, for example, I can compute Mu in an email application and hand it into my smartcard over a PKCS#11 API that accepts Mu.

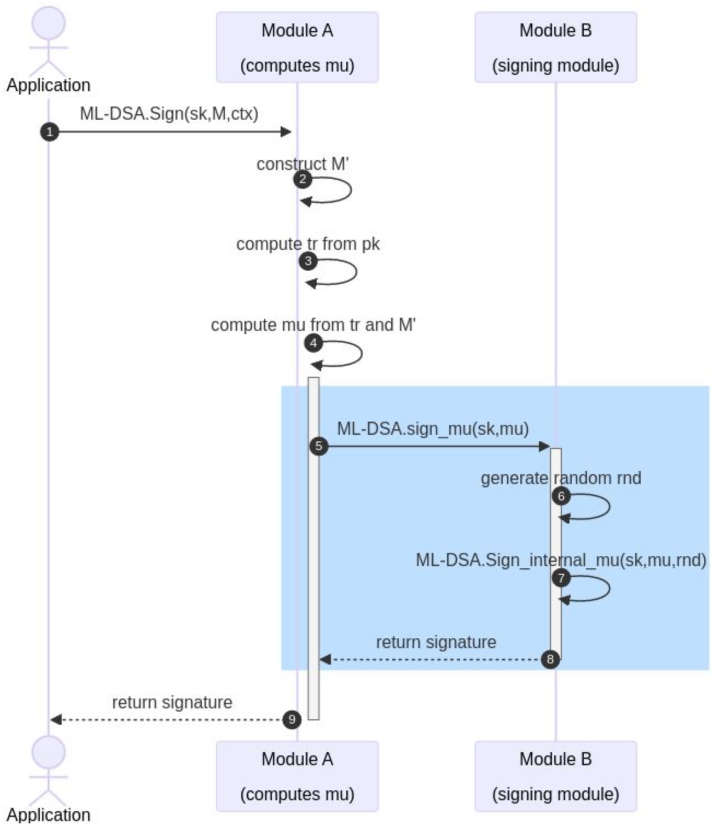
NIST has heard the objection and will give us an answer.

Bonus topic: ML-DSA External Mu actually FIPS-allowed?

As of an FAQ posted **Feb 24, 2025** and a discussion held on **Mar 10, 2025**.

It specifically allows this flow where Module A computes mu, and calls Module B that computes the signature.

This mode probably covers 80% usecases – ex.: an HSM and its proprietary client.



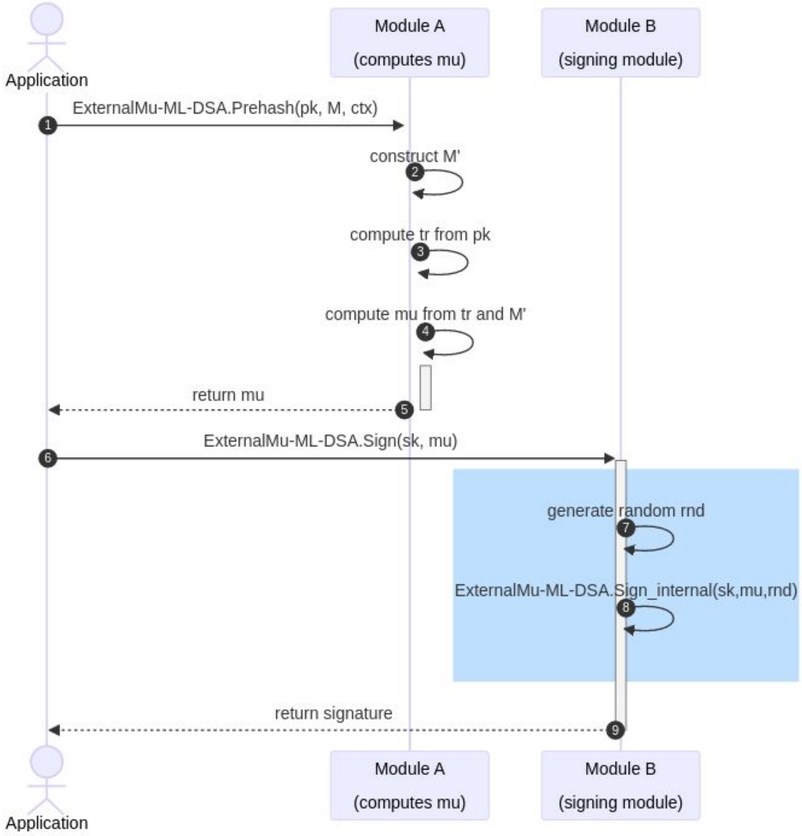
Bonus topic: ML-DSA External Mu actually FIPS-allowed?

But what about the mode where, for example, you want to compute Mu in a local client and then pass it over a REST API?

Or compute Mu in an email client and pass it through many layers of software before it reaches a smartcard?

(many other usecases look like this)

NIST has heard this compliant will get back to us (hopefully by updating the FAQ).



Bonus topic: what does this mean for hybrids?

- X-Wing mandates SEED-only.
- Composites are a mess (we have not solved this yet) because both the PQ part and the RSA / EC part could have multiple valid encodings (or at least be non-fixed length).
  - Great progress was made during the hackathon weekend.
  - … we'll get back to you on this.

Summary

- FIPS 203 / 204 gave us two allowed formats for private keys: SEED and KEY.
- This lead to an all-mighty bru-ha-ha for what to support in LAMPS / X.509.
- These slides share lessons learned so that other WGs can hopefully avoid re-hashing the same debate 🤞

- Solution: LAMPS has settled on the ASN.1-based encoding

```
ML-KEM-1024-PrivateKey ::= CHOICE {
   seed [0] OCTET STRING (SIZE (64)),
   expandedKey OCTET STRING (SIZE (3168)),
   both SEQUENCE {
       seed OCTET STRING (SIZE (64)),
       expandedKey OCTET STRING (SIZE (3168))
       }
   }
```

Summary

- Other options that were explored and rejected:
  - SEED-only
  - KEY-only
  - Both
  - Alternative OIDs
  - CHOICE { KEY, SEED }   – one or the other
  - SEQUENCE { optional KEY, optional SEED }   – one or the other or both

- Russ Housley:
  "My conclusion is that no algorithm should have more than one private key format. Cryptographers need to never do that again!"