PQC: Practical issues that will impact the future of hardware protected security environments

Mike Ounsworth

Global Platform Automotive Security

December 4, 2024

ENTRUST

SECURING A WORLD IN MOTION

MIKE'S INTERNET STANDARDS WORK

- I contribute heavily to the Internet Engineering Taskforce (IETF)!
- Getting the Internet ready for Post-Quantum Cryptography
- Internet Drafts that I am an author or contributor on:

LAMPS : Limited Additional Mechanisms for PKIX and SMIME

- PQC and migration:
  - X.509 / CMS: draft-ounsworth-pq-composite-sigs
  - X.509 / CMS: draft-ietf-lamps-pq-composite-kem
  - X.509: draft-bonnell-lamps-chameleon-certs
  - X.509: draft-ounsworth-lamps-pq-external-pubkeys
  - X.509: draft-lamps-okubo-certdiscovery
  - CMS: RFC 9629 - adding KEMs to CMS
  - CMS: draft-ietf-lamps-cms-kyber
- CMPv3:
  - RFC 9480 (CMPv3)
  - RFC 9481 (CMP Algorithm Updates)
  - draft-ietf-lamps-rfc4210bis
  - draft-ietf-lamps-rfc6712bis
- Attestation: draft-ietf-lamps-csr-attestation-00

CFRG: Cryptographic Research Forum

- draft-fluhrer-cfrg-ntru-00
- draft-ounsworth-cfrg-kem-combiners

OpenPGP

- draft-wussler-openpgp-pqc-00

ACME

- draft-vanbrouwershaven-acme-auto-discovery-01
- draft-acme-device-attest

PQUIP: Post-Quantum Use in Protocols

- https://datatracker.ietf.org/doc/draft-ietf-pquip-pqc-engineers/
- draft-vaira-pquip-pqc-use-cases

Remote Attestation (RATS)

- draft-ietf-rats-pkix-evidence

ENTRUST

Deep-dive on "surprising points" with deploying ML-DSA and ML-KEM

ENTRUST

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

PrivKeys, P12, P11, Hybrids, and beyond!

- Mike Ounsworth was asked to give a 30 min presentation to the PQUIP WG (Post-Quantum Use In Protocols) on “friction points” with how FIPS 203 (ML-KEM) and FIPS 204 (ML-DSA) are written.
  - https://datatracker.ietf.org/doc/slides-121-pquip-fips-issues-with-deploying-ml-kem-and-ml-dsa/
- The main points.
  - ML-DSA Context (*ctx*)
  - ML-DSA and ML-KEM private keys – seeds vs expanded
  - Direct Seed vs Derived Seed
  - Hybrids - KDF(mlkem || ec) vs KDF(ec || mlkem)
  - ML-DSA pre-hash mode (“HashML-DSA” vs “ExternalMu-ML-DSA”)
- Audience: developers of embedded crypto systems.

ENTRUST

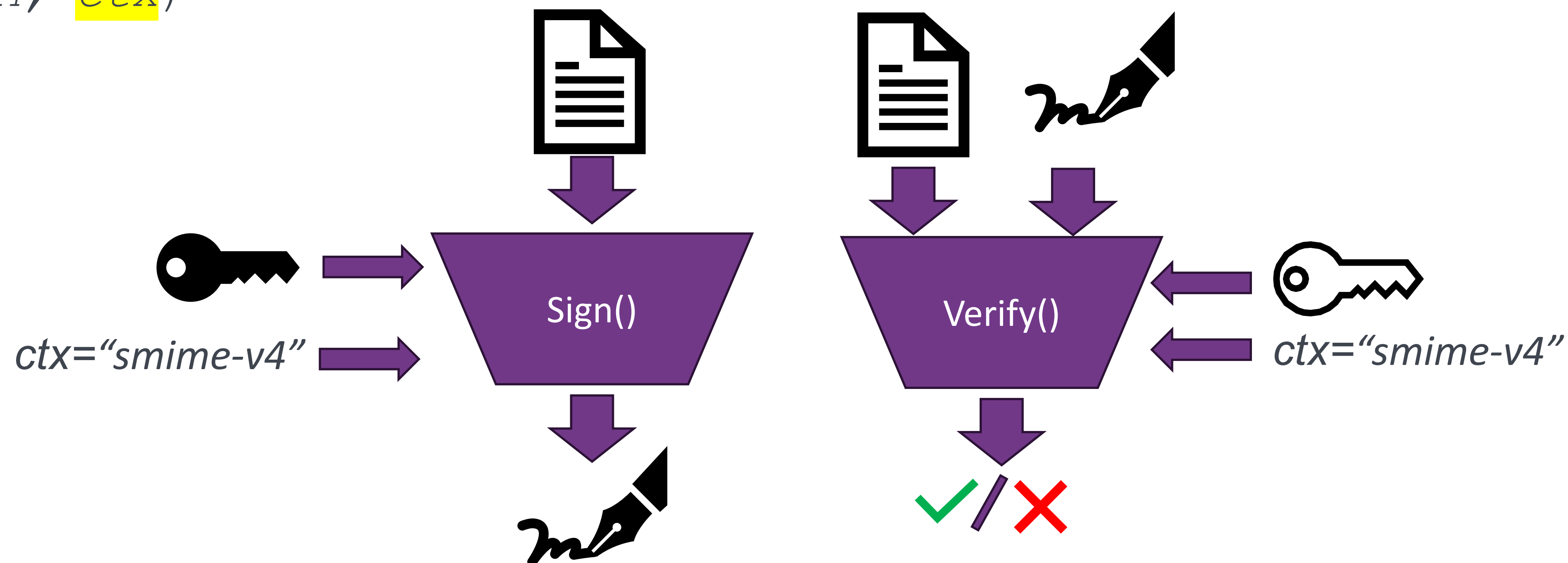
ML-DSA Context (*ctx*) parameter

ENTRUST

ML-DSA Context (ctx)

- Since EdDSA (RFC 8032, published 2017), and now with ML-DSA (FIPS 204) and SLH-DSA (FIPS 205) signature APIs accept a "context string":

```
Sign(sk, M, ctx)
```



- The benefit here is that, for example, S/MIME email and signed PDF use the same message structure, so a client might be tricked into confusing them.
- A well-chosen *ctx* hard-coded into both signer and verifier strongly prevents this by failing the signature.

ENTRUST

Signature context *ctx*

- The problem is that very few protocols used the *ctx* in EdDSA, so many crypto libraries never implemented an API for it. Ex.: python cryptography:

```
class cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PrivateKey

    sign(data)    [source]

        Parameters:     data (bytes-like) – The data to sign.
        Returns bytes:  The 64 byte signature.
```

- IETF asks: “When is *ctx* no longer ‘new’?”

- Can we just start designing network protocols to require
  `ML-DSA.Sign(sk, M, ctx)`
  and hope that crypto libraries, HSMs, smartcards, etc will catch up?

ENTRUST

ML-DSA and ML-KEM private keys seeds vs expanded

ENTRUST

ML-DSA and ML-KEM KeyGen() – seeds vs expanded

Algorithm 1 ML-DSA.KeyGen()

Generates a public-private key pair.

Output: Public key $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$
and private key $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot\text{bitlen}(2\eta)+dk)}$.

1: $\xi \leftarrow \mathbb{B}^{32}$ ▷ choose random seed
2: **if** $\xi =$ NULL **then**
3: **return** $\bot$ ▷ return an error indication if random bit generation failed
4: **end if**
5: **return** ML-DSA.KeyGen_internal (ξ)

Algorithm 19 ML-KEM.KeyGen()

Generates an encapsulation key and a corresponding decapsulation key.

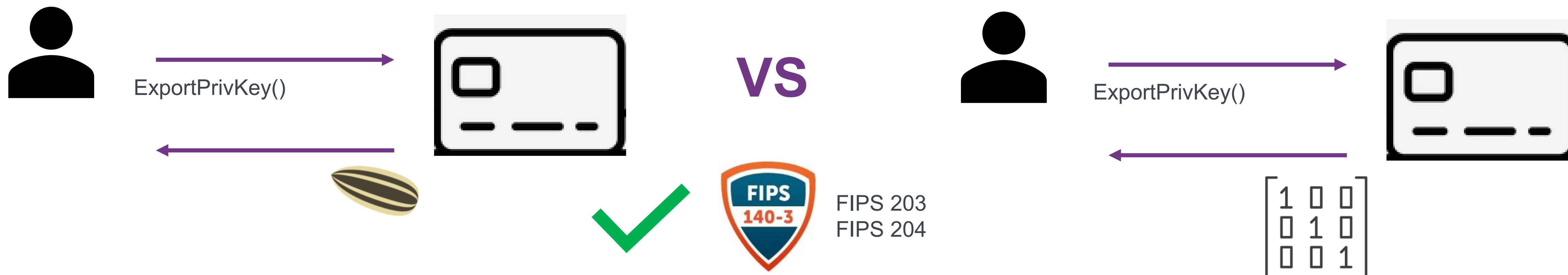
Output: encapsulation key ek $\in \mathbb{B}^{384k+32}$.
Output: decapsulation key dk $\in \mathbb{B}^{768k+96}$.

1: $d \xleftarrow{\$} \mathbb{B}^{32}$ ▷ d is 32 random bytes (see Section 3.3)
2: $z \xleftarrow{\$} \mathbb{B}^{32}$ ▷ z is 32 random bytes (see Section 3.3)
3: **if** $d ==$ NULL **or** $z ==$ NULL **then**
4: **return** $\bot$ ▷ return an error indication if random bit generation failed
5: **end if**
6: (ek, dk) $\leftarrow$ ML-KEM.KeyGen_internal(d, z) ▷ run internal key generation algorithm
7: **return** (ek, dk)

- Both output a big complicated private key object.
- Both chain to `KeyGen_internal( seed )`
  - ML-DSA: ξ is 32 bytes
  - ML-KEM: (d, z) is 64 bytes
- `KeyGen_internal( seed )` is actually very fast, fast enough that there's no real penalty to doing it every time I need to use the private key
- So, can I just store those seeds instead of storing the expanded key? 🤨
  … the answer is **<u>Yes</u>** (but this is something "you just have to know",
  FIPS 203 / 204 does not say it clearly enough for my liking).

ENTRUST

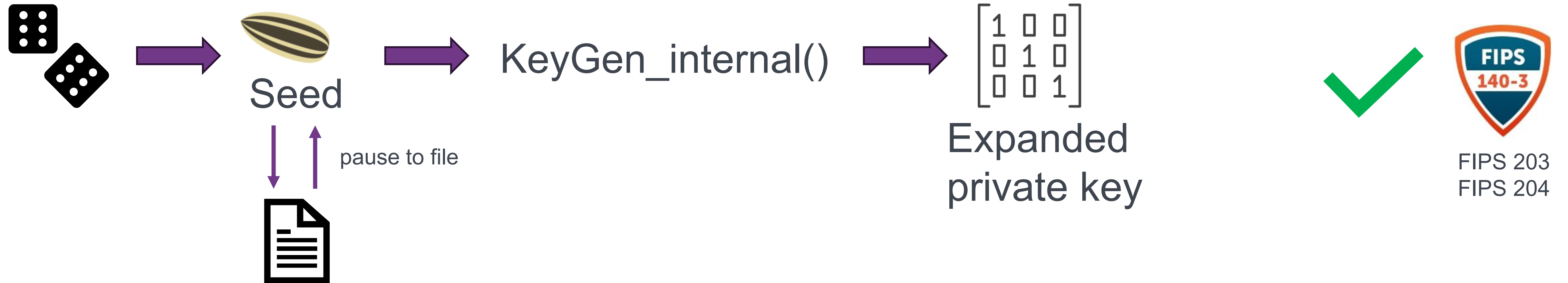
ML-DSA and ML-KEM KeyGen() – seeds vs expanded

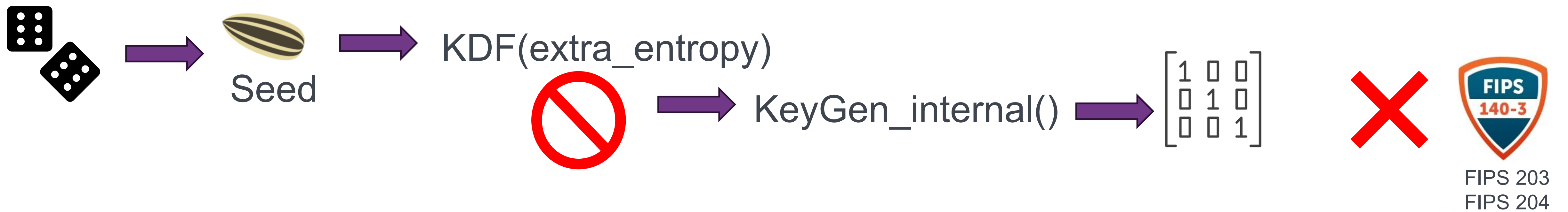


- The IETF wants “Internet” private key file formats like PKCS#12, JWK to only support seed-based private keys because of both performance and security gains.
  - Performance: obviously, size.
  - Security: if you re-derive the key from a seed, then you know that it is well-formed and not tampered with.
- That means hardware needs to keep the seed when doing a KeyGen() so that it can later export it.
- … I expect this will be a compatibility issue that will affect us for many years.

ENTRUST

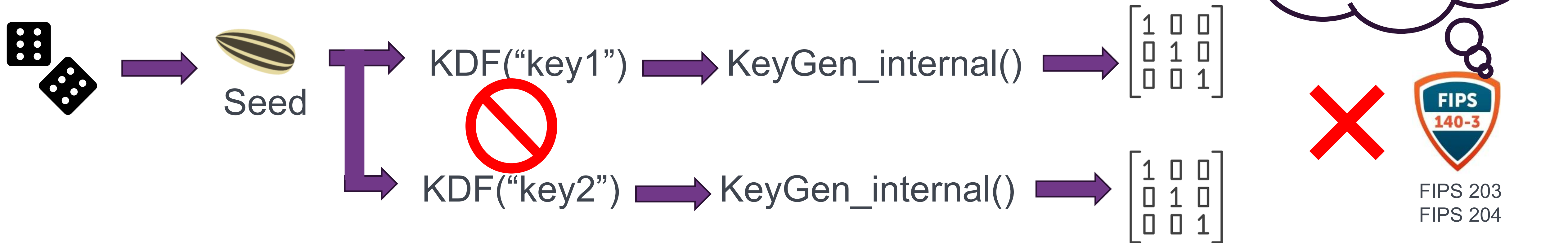
Direct Seed vs Derived Seed



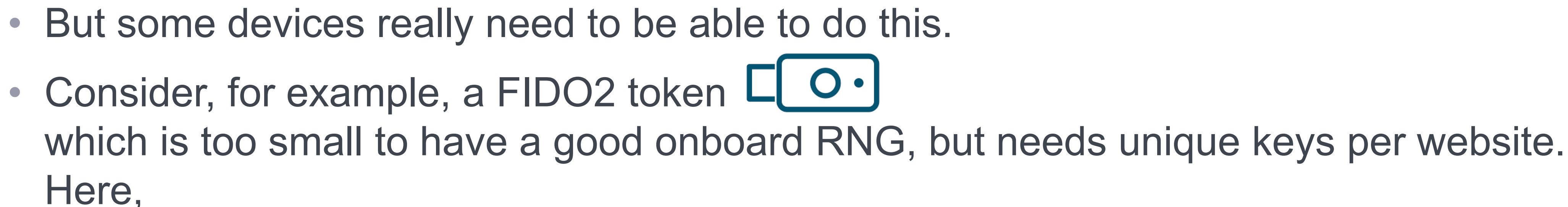


ENTRUST

Direct Seed vs Derived Seed



- But some devices really need to be able to do this.
- Consider, for example, a FIDO2 token which is too small to have a good onboard RNG, but needs unique keys per website. Here,



KDF(*high_entropy_seed* + *website_url*) → KeyGen_internal()

is a totally reasonable strategy.
- So, if you make a device like this, be aware that there is (currently) no way to do it and be compliant with FIPS 203 / 204.

ENTRUST

Hybrids and Composites

ENTRUST

Hybrids and Composites

```
LAMPS                                                        M. Ounsworth
Internet-Draft                                                    J. Gray
Intended status: Standards Track                                  Entrust
Expires: 24 April 2025                                            M. Pala
                                                              OpenCA Labs
                                                             J. Klaussner
                                                   Bundesdruckerei GmbH
                                                               S. Fluhrer
                                                            Cisco Systems
                                                          21 October 2024


  Composite ML-KEM for use in X.509 Public Key Infrastructure and CMS
                 draft-ietf-lamps-pq-composite-kem-05
```

```
LAMPS                                                        M. Ounsworth
Internet-Draft                                                    J. Gray
Intended status: Standards Track                                  Entrust
Expires: 24 April 2025                                            M. Pala
                                                              OpenCA Labs
                                                             J. Klaussner
                                                   Bundesdruckerei GmbH
                                                               S. Fluhrer
                                                            Cisco Systems
                                                          21 October 2024


  Composite ML-DSA For use in X.509 Public Key Infrastructure and CMS
                 draft-ietf-lamps-pq-composite-sigs-03
```

```
CFRG                                                         M. Ounsworth
Internet-Draft                                                    Entrust
Intended status: Informational                                  A. Wussler
Expires: 3 August 2024                                             Proton
                                                              S. Kousidis
                                                                      BSI
                                                          31 January 2024


Combiner function for hybrid key encapsulation mechanisms (Hybrid KEMs)
                 draft-ounsworth-cfrg-kem-combiners-05
```

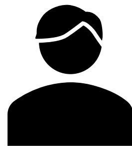
- I have done a lot of work on hybrids.
- These are mostly progressing through the IETF standardization process without issue.
- Except …

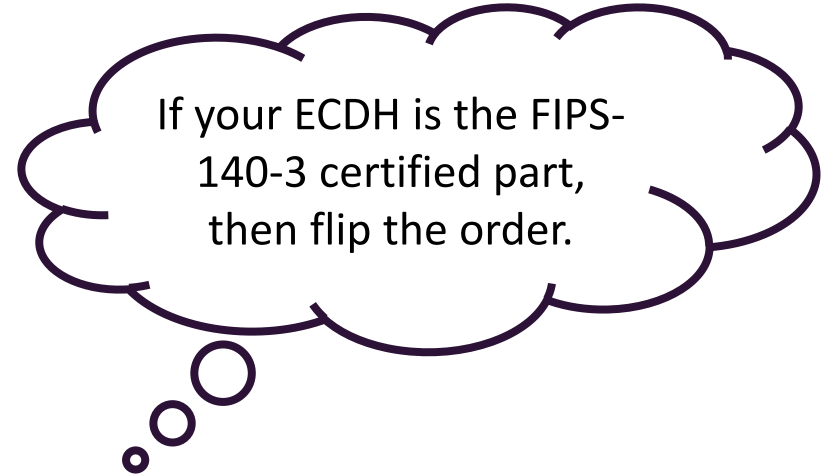
ENTRUST

Hybrid ML-KEM - KDF(mlkem || ec) vs KDF(ec || mlkem)

ML-KEM -> mlkemSS

ECDH -> ecSS





ss = KDF(**mlkemSS**, **ecSS**, fixedInfo)

SP 800-56Cr2

ss = KDF(**ecSS**, **mlkemSS**, fixedInfo)

- This is somewhat crazy that we would need different algorithm codepoints depending on which component is currently the FIPS-approved one (which will change over time).
- NIST agrees and has promised to fix this in SP 800-227 (we will see a draft in February 2025).

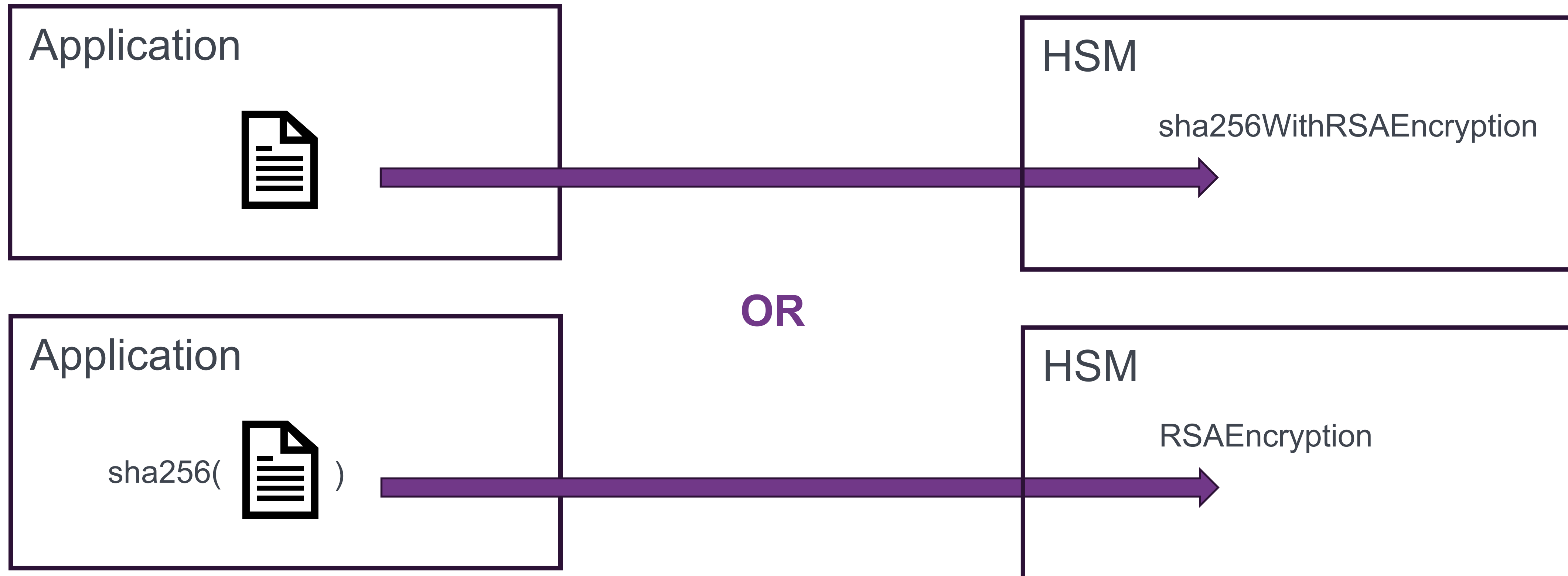
ENTRUST

ML-DSA and pre-hash modes

ENTRUST

Background: Pre-hashed Modes

- With RSA or ECDSA, you are free to split the pre-hash step from the core signature step.



- Both “modes” produce the same output, so this is just “implementation detail”.

ENTRUST

Background: ML-DSA

- ML-DSA's design includes a security improvement by including the hash of the public key (*tr*) in the message digest that will be signed:

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

6: $\mu \leftarrow$ H(BytesToBits$(tr)||M', 64)$

- This makes collision attacks harder – the attacker would need to perform per-public-key collision searches – and prevents some types of "key swapping" attacks.
  - If RSA / ECDSA had done this, then we would not have had any panic when collision attacks were discovered in SHA-1. 🧐
- ☝🧐 but we still want to do pre-hashing for performance reasons!

ENTRUST

ML-DSA pre-hash mode (“HashML-DSA” vs “ExternalMu-ML-DSA”)

- In the final FIPS 204, NIST gave us this:

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.

- Problem solved? … NOPE!
  1. Security: Cryptographers are unhappy that this completely un-does the security gains of hashing in *tr*.
  2. Implementation: These are different, incompatible, algorithms with different `.Verify()` functions, and because of … stupid OID-related reasons … you have to choose “pure” or “prehash” mode when you generate the key, and then that key can only ever be used in that mode for the lifetime of the key (well, technically of the cert).
- ☝🧐BUT WAIT … FIPS 204 gives us a 3rd (hidden) option!

ENTRUST

ML-DSA pre-hash mode (“HashML-DSA” vs “ExternalMu-ML-DSA”)

- ☝🧐BUT WAIT … FIPS 204 gives us a 3rd (hidden) option!

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

- Great! Being able to pull *mu* out front is actually what we wanted in the first place! It makes everybody happy.
- Good: this mode is clearly allowed by FIPS 204.
- Bad: they have not written this in an obvious way.
  - The “External Mu mode” deserves to be written out in full.
  - There are two allowed ways of doing a pre-hash: External Mu and HashML-DSA.

ENTRUST

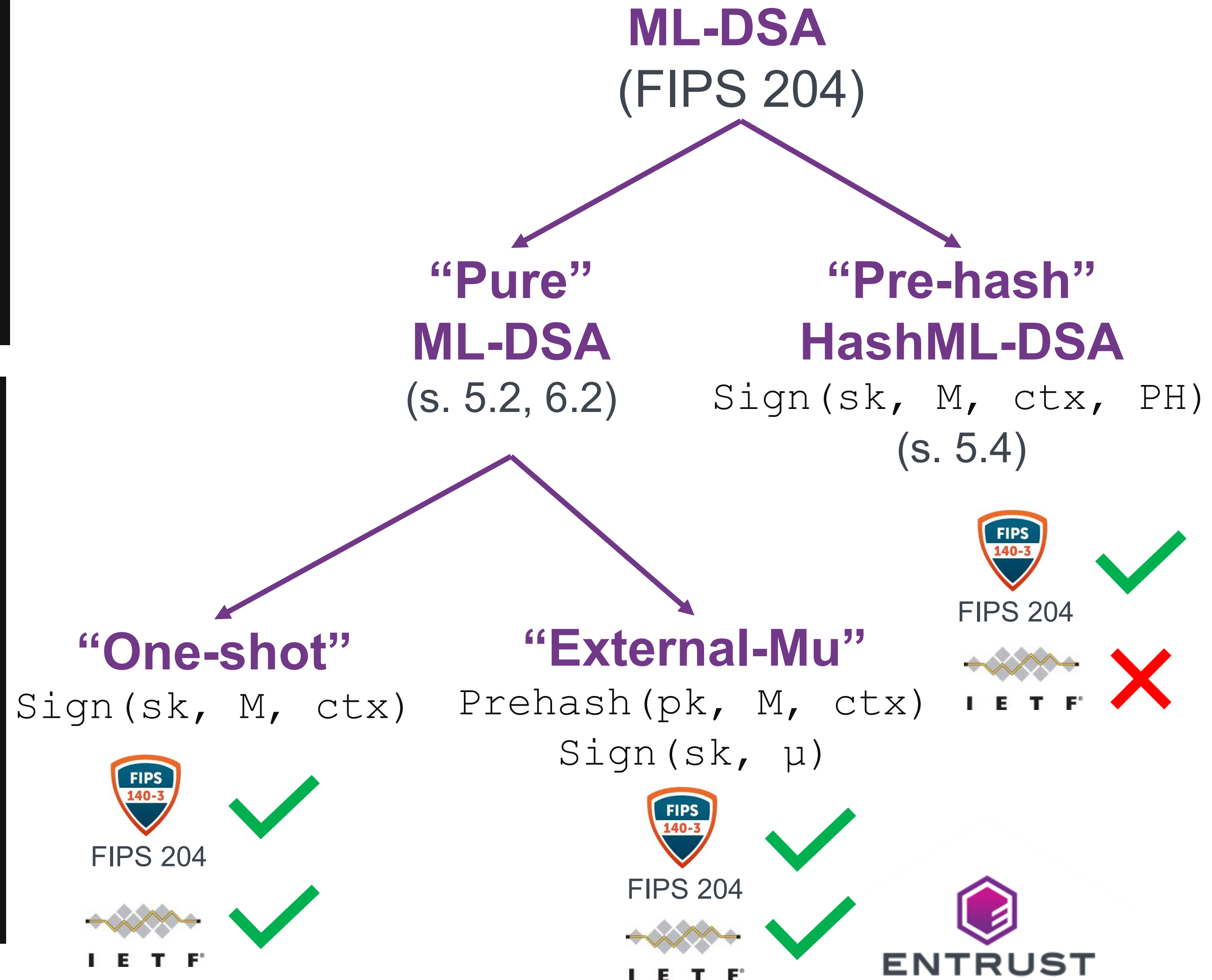
ML-DSA pre-hash mode ("HashML-DSA" vs "ExternalMu-ML-DSA")

```
ExternalMu-ML-DSA.Prehash(pk, M, ctx):

  if |ctx| > 255 then
    return error  # return an error indication if the context string is
                  # too long
  end if

  M' = BytesToBits(IntegerToBytes(0, 1) || IntegerToBytes(|ctx|, 1)
                                              || ctx) || M

  mu = H(BytesToBits(pk.tr) || M', 64)
  return mu
```

Figure 1: External steps of ExternalMu-ML-DSA

```
ExternalMu-ML-DSA.Sign(sk, mu):

  if |mu| != 512 then
    return error  # return an error indication if the input mu is not
                  # 64 bytes (512 bits).
  end if

  rnd = rand(32)  # for the optional deterministic variant,
                  # substitute rnd = 0x0 * 32
  if rnd = NULL then
    return error  # return an error indication if random bit
                  # generation failed
  end if

  sigma = ExternalMu-ML-DSA.Sign_internal(sk, mu, rnd)
  return sigma


ExternalMu-ML-DSA.Sign_internal(sk, mu, rnd):
  ... identical to FIPS 204 Algorithm 7, but with Line 6 removed.
```

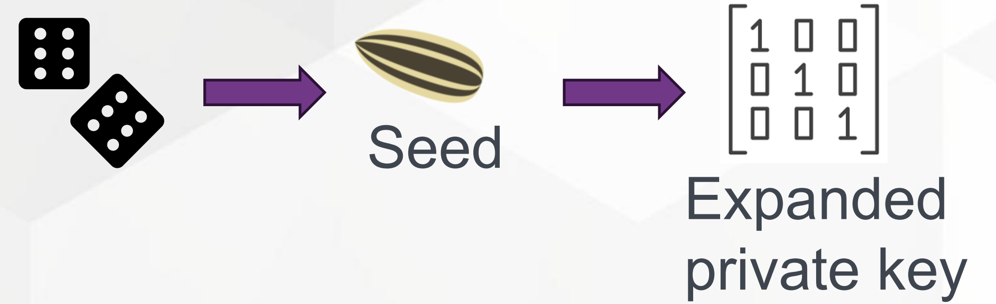
Figure 2: Internal steps of ExternalMu-ML-DSA

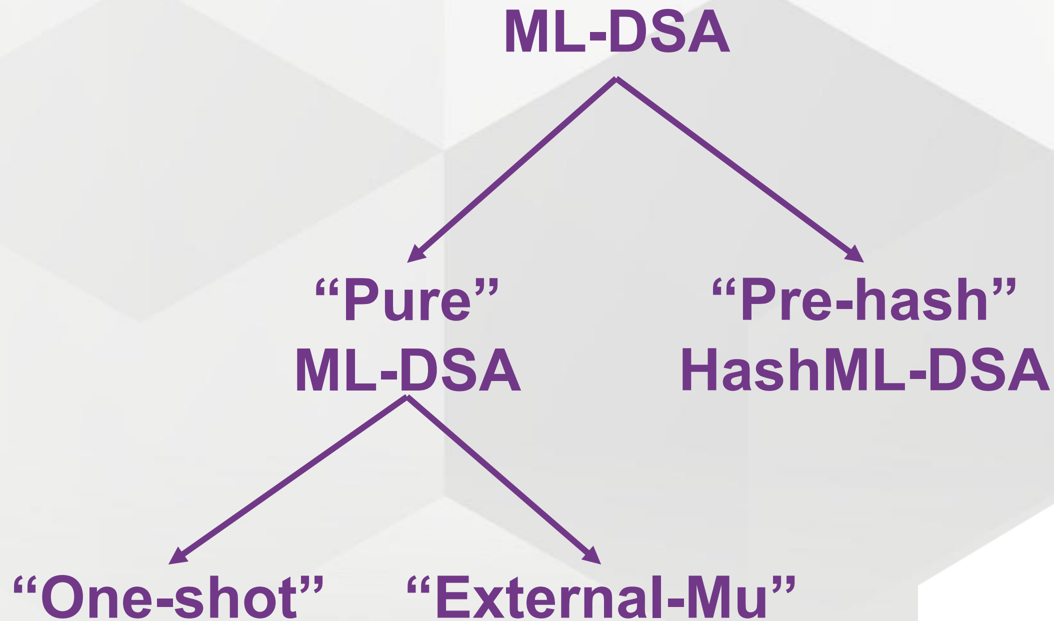


https://datatracker.ietf.org/doc/draft-ietf-lamps-dilithium-certificates/

Thank you!

mike.ounsworth@entrust.com





entrust.com

© Entrust Corporation

ENTRUST

SECURING A WORLD IN MOTION