# Request to Extend IETF WGLC for PQ Key Specifications

We, the undersigned, would respectfully appreciate the IETF WGLC on private format key specifications for PQ algorithms, namely draft-ietf-lamps-kyber-certificates-07 and draft-ietf-lamps-dilithium-certificates-06, being **extended until the end of January** for WG members and other interested parties to be able to fully consider the information outlined in this document. Presently, we feel we are unable to both serve our user bases and conform to the proposed specification, and we have little choice but to offer our users alternative private key formats.

We will represent keys in a manner that maximizes their usability for our users. Specifically, the simplest ASN.1 encoding (see below) can represent the {SEED}, the private/secret {KEY}, or both. We do acknowledge that our case is from the perspective of hindsight and that it is very unlikely that previous comments on the IETF WGLC drafts would have been in a position to take these things into account.

For those who do not closely follow the PKCS#11 and KMIP technical committees in OASIS Open, both of those standards have been tracking the various NIST specification updates and final publication and have early adopters working across ML-KEM, ML-DSA, and SLH-DSA.

The consolidated viewpoint within the PKCS#11 and KMIP TCs is that we cannot mandate that all implementations (especially HW-based) must be able to support the import and export of the SEED.

For those not familiar with PKCS#11 version 3.2, which adds support for ML-KEM and ML-DSA, the definitions for the private key for each algorithm are formed using three attributes - CKA_PARAMETER_SET, CKA_SEED, and CKA_VALUE. The latest draft of PKCS#11 v3.2 is available at https://groups.oasis-open.org/higherlogic/ws/public/download/72453/pkcs11-spec-v3.2-wd08.docx

The **PKCS#11 specification** explicitly states the following:

> *At least one of CKA_SEED and CKA_VALUE must be specified on C_CreateObject. Tokens may reject creation requests that only specify one of these values. For the highest compatibility, applications should set both.*

Similarly, for KMIP version 3.0 which adds support for ML-KEM and ML-DSA, the technical committee is closely aligned with the PKCS#11 technical committee and mirrors the recommendation to return both for maximum interoperability. The update delta to the latest draft of KMIP v3.0 is available at https://groups.oasis-open.org/higherlogic/ws/public/download/72473/kmip-spec-v3.0-wd18-markup.pdf/latest

The **KMIP specification** explicitly states the following:

> *At least one of Seed and Key SHALL be specified. Implementations MAY reject operations involving managed objects using this Key Format if only one of these values is specified and that value is not supported by the underlying implementation. For maximum interoperability, applications SHOULD set both Seed and Key.*

The various IETF drafts have operated on the assumption that it is possible to mandate {SEED} and that all other standards will correspondingly mandate it, and that all the various implementations in cryptographic toolkits will also provide interfaces that always support the retention, import, and export of the {SEED}.

The PKCS#11 and KMIP standards do not share this IETF assumption. Once you accept that multiple formats for import and export need to be supported then an encoding format to support multiple formats must be defined.

For a variety of reasons other standards groups and users have reached a different view to that of the IETF. We feel it would be a distraction at this time to debate the legitimacy of those reasons, our view is that we simply need to be able to support those users by implementing a more flexible private key format.

Maximum interoperability necessitates that implementations should export the {SEED} and the private {KEY} if possible, or just the private {KEY} (if the {SEED} is unavailable). For imports, whichever of these are supported by the ultimate implementation should be used (i.e., if the underlying cryptographic module or hardware solution supports {SEED} as an input, it should be used; if it does not, then the private {KEY} should be used). In short, {SEED + KEY} is recommended, {KEY} is mandatory to support, and {SEED} is optional to support.

Separately, the decision to remove the ASN.1 encoding from the PrivateKey in PKCS#8 format is not only against standing and expected practice but doesn't leave a sustainable solution to support both formats. PKCS#8 is ASN.1 at its core, and the use of PKCS#8 should reflect that (and does with every other algorithm). When combined with the requirement to support multiple formats, using ASN.1 to represent the alternatives is the natural outcome.

Like the IETF (""rough consensus and running code"), actual implementation experience is critically important to PKCS#11, KMIP, OpenSSL, Bouncy Castle, and other major standards and implementations. Within OpenSSL, our collective experience is that length-based discrimination to distinguish input formats is fragile and should be avoided. Unambiguous ASN.1 tagging avoids such heuristics and better aligns with how the keys of all other algorithms are handled.

Both OpenSSL and Bouncy Castle have implemented support for the various changing formats over time - and attempted to maximize interoperability. The resulting code necessitated the determination of the format by performing a length check and other heuristics and, as such, is considered an undesirable approach. The history of such heuristics based code suggests that it is likely to result in CVEs as it results in additional complications.

Flexibility on input allows us to interoperate with other implementations, however, absent a format that can represent both the {SEED} and {KEY}, when we come to selecting an output format, we'd have to place the burden on the user to determine which to choose. If {SEED}-only is selected, there are interoperability issues, if {KEY}-only is selected, there are interoperability issues. This leads to a conclusion that the maximally interoperable format is {SEED + KEY}.

As interoperability is an industry-wide guiding principle, we have selected to output {SEED + KEY}, when possible, and {KEY} or {SEED} (whichever is available) otherwise. This will be in our next release (for OpenSSL, this is version 3.5, which will be released in April 2025; for Bouncy Castle, this is version BC Java 1.81, which will be released in April 2025).

Therefore, the private key format that best serves our users and we are advocating the IETF adopt is

```
PrivateKey ::= SEQUENCE {
          seed OCTET STRING OPTIONAL,
          expandedKey [1] IMPLICIT OCTET STRING OPTIONAL
 }
```

As opposed to the current definition at the end of section 5 of the draft:

```
When used in a OneAsymmetricKey type, the privateKey OCTET STRING
contains the raw octet string encoding of the 64-octet seed.
```

If the IETF Working Group decides to output {SEED}-only and not support {SEED + KEY} or {KEY}-only, it would interoperate on **input** with OpenSSL and Bouncy Castle, provided the seed is preceded by exactly four bytes of ASN.1 encapsulation. In hexadecimal, these are: **30420440**, indicating a sequence of length 66 enclosing an OCTET STRING of length 64. The additional four bytes cannot reasonably be argued to be overly burdensome to produce on output or to match and discard on input. Of course, if that is the only selected option, interoperability **will** suffer as the specification would be unable to represent {SEED + KEY} or {KEY} alone.

## OpenSSL Statement:

We have a working code that can accept all three input variants and generate all three output variants proposed. We also accept the existing **oqs_provider** ASN.1 encoded variants but we don't expect that others will do the same. The reworked code is cleaner and clearer than our code based on length heuristics and processes the ASN.1 encapsulation using a very simple table-driven parser without resort to our general-purpose ASN.1 parser (this demonstrates how simple the proposed format is to encode and decode).

## Bouncy Castle Statement:

We have a working code that can accept both the proposal detailed here and the {SEED}-only proposal. For output, we will attempt to echo the original encoding, although, particularly for the case of ML-KEM, it has become obvious that we need to follow the proposal here (for the sake of our user community) as import of keys for the {SEED}-only case may be impossible for other FIPS providers where the direct processing of encodings is outside of the FIPS boundary. We will attempt to migrate {KEY}-only users to the proposal here, which will (hopefully) limit us to just {SEED}, or the {SEED + KEY} SEQUENCE.

### Authors and Contributors:

- Viktor Dukhovni (OpenSSL)
- Dmitry Belyavskiy (Red Hat, OpenSSL Maintainer)
- Matt Caswell (OpenSSL Foundation President)
- David Hook (Legion of the Bouncy Castle Inc., Co-Founder and maintainer)
- Tim Hudson (OpenSSL Corporation President, Cryptsoft CTO, OASIS Open PKCS#11 Profiles Co-Editor, OASIS Open KMIP Profiles Co-Editor, Formal liaison between the OASIS Open PKCS#11 and KMIP Technical Committees)
- Bob Relyea (Red Hat, NSS Developer, OASIS Open PKCS #11 Co-Chair)
- Simo Sorce (Red Hat, OASIS Open PKCS #11 member)