IPsecME
Internet-Draft
Intended status: Informational
Expires: 29 January 2026

S. Fluhrer
Cisco Systems

28 July 2025

# IKEv2 Support of ML-DSA
## draft-sfluhrer-ipsecme-ikev2-mldsa-01

## Abstract

One IPsec area that would be impacted by Cryptographically Relevant Quantum Computer (CRQC) is IKEv2 authentication based on traditional asymmetric cryptograph algorithms: e.g RSA, ECDSA; which are widely deployed authentication options of IKEv2. NIST has recently standardized ML-DSA, which is a signature algorithm believed to be secure against Quantum Computers. This document describes how to use ML-DSA with IKEv2 as an authentication scheme.

## About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://sfluhrer.github.io/ikev2-mldsa-support/draft-sfluhrer-ikev2-mldsa-support.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-sfluhrer-ipsecme-ikev2-mldsa/.

Discussion of this document takes place on the IP Security Maintenance and Extensions Security Area mailing list (mailto:cfrg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/ipsecme/. Subscribe at https://www.ietf.org/mailman/listinfo/cfrg/.

Source for this draft and an issue tracker can be found at https://github.com/sfluhrer/ikev2-mldsa-support.

## Status of This Memo

## Copyright Notice



## Table of Contents



## 1. Introduction

A Cryptographically Relevant Quantum Computer (CRQC) could break traditional asymmetric cryptograph algorithms: e.g RSA, ECDSA; which are widely deployed authentication options of IKEv2. NIST has recently published the postquantum digital signature algorithm ML-DSA [FIPS204].

This document describes how to use this algorithm for authentication within IKEv2 [RFC7296], as a replacement for the traditional signature algorithms (RSA, ECDSA).

Each IPsec peer announce the support for ML-DSA authentication via SUPPORTED_AUTH_METHODS notification as defined in [RFC9593], generates and verifies AUTH payload using ML-DSA.

### 1.1. Background on ML-DSA

ML-DSA (as specified in FIPS 204) is a signature algorithm that is believed to be secure against attackers who have a Quantum Computer available to them. There are three strengths defined for it (with the parameter sets being known as ML-DSA-44, ML-DSA-65 and ML-DSA-87). In addition, for each defined parameter set, there are two versions, the 'pure' version (where ML-DSA directly signs the message) and a 'prehashed' version (where ML-DSA signs a hash that was computed outside of ML-DSA). For this protocol, we will always use the pure version.

In addition, ML-DSA also has a 'context' input, which is a short string that is common to the sender and the recceiver. It is intended to allow for domain separation between separate uses of the same public key.

FIPS 204 also allows ML-DSA to be run in either determanistic or 'hedged' mode (where randomness is applied to the signature operation). We place no requirement on which is used; the implementation should select based on the quality of their random number source.

## 2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. Protocol Changes

### 3.1. Initial Negotiation

Both sides will need to inform the other that they implement ML-DSA signatures. To do so, they will use the [RFC9593] mechanism to specify support for ML-DSA signatures, using the Multi-octet Announcement, with the following Algorithm Idenfifiers (in hex):

* ML-DSA-44 -> 06 09 60 86 48 01 65 03 04 03 11

* ML-DSA-65 -> 06 09 60 86 48 01 65 03 04 03 12

* ML-DSA-87 -> 06 09 60 86 48 01 65 03 04 03 13

If an implementation supports multiple ML-DSA parameter sets, it will list every parameter set it does support.

If the peer has not specified support for a parameter set in a SUPPORTED_AUTH_METHODS notify, that ML-DSA parameter set MUST NOT be used.

In addition, the SIGNATURE_HASH_ALGORITHMS Notify payload must also be sent (see [RFC7427]), listing the supported hash functions.

## 3.2. Auth Payload Generation

If this implementation has an ML-DSA private key and the corresponding ML-DSA public certificate, and the peer has indicated support for the parameter set, the implementation will generate the AUTH payload as specified in section 3 of [RFC7427], using the ML-DSA algorithm as the signature algorithm, and using the fixed context string "IKEv2 AUTH" (49 4b 45 76 32 20 41 55 54 48).

That is, the implementation would take either the InitiatorSignedOctets string or the ResponderSignedOctets string (depending on whether they are the initiator or the responder, see section 2.15 of RFC 7296 for how those strings are constructed), compute the hash of that string (using one of the hashes listed in the peer’s SIGNATURE_HASH_ALGORITHMS notify). Then, the implementation hands that hash to ML-DSA to be signed (in pure mode, using the fixed context string "IKEv2 AUTH". The resulting signature is the Signature Value. Note that ML-DSA will hash the message to be signed again; this is expected.

TODO: We’ve defined the method two different ways - if we keep both, we need to make sure that they are equivalent.

## 3.3. Auth Payload Validation

If this implementation receives a certificate with an ML-DSA public key, it will process the AUTH payload as implied by RFC7427. That is, it will recover the hash function from the AlgorithmIdentifier ASN.1 object. Then, it will take the ResponderSignedOctets or InitiatorSignedOctets string (depending on the whether they are the initiator or the responder), and then hash the string. Then, it will before an ML-DSA verification, using the hash as the message, the

public key from the certificate, the string "IKEv2 AUTH" as the context string, and the signature value from the AUTH payload. If this signature verification fails, the implementation MUST reject the IKEv2 message.

## 4. Security Considerations

The only security consideration that this adds is that the user must trust the strength of the ML-DSA signature operation.

## 5. Discussion

We made several arbitrary design decisions in this draft. This section contains the reasoning behind those design decision, and why we did not select the alternative possibilities. Of course, these decisions are open to change - this is just a first cut.

### 5.1. ML-DSA and Prehashing

The signature architecture within IKE was designed around RSA (and later extended to ECDSA). In this architecture, the actual message (the SignedOctets) are first hashed (using a hash that the verifier has indicated support for), and then passed for the remaining part of the signature generation processing. That is, it is designed for signature algorithms that first apply one of a number of hash functions to the message and then perform processing on that hash. ML-DSA doesn’t fit cleanly into this architecture; internally it adds a prepend to the message to be signed, and then does a fixed SHAKE256 (generating 64 bytes).

We see three ways to address this mismatch

The first is to note that ML-DSA has prehashed parameter sets; that is, ones designed to sign a message that has been hashed by an external source. At first place, this would appear to be an ideal solution, however it turns out that there are a number of practical issues. The first is that the prehashed version of ML-DSA would appear to be rarely used, and so it is not unlikely that support for it within crypto libraries may be lacking. The second is that the public keys for the prehashed versions of ML-DSA parameter sets use different OIDs; this means that the certificates for IKEv2 would necessarily be different than certificates for other protocols (and some CAs might not support issuing certificates for prehashed ML-DSA, again because of the lack of use). The third is that some users have expressed a desire not to use the prehashed parameter sets of ML-DSA.

The second is to note that, while IKEv2 normally acts this way, it doesn't always. EdDSA has a similar constraint on not working cleanly with the standard 'hash and then sign' paradigm, and so the existing [RFC8420] provides an alternative method, which ML-DSA would cleanly fit into. We could certainly adopt this same strategy; our concern would be that it might be more difficult for IKEv2 implementors which do not already have support for EdDSA.

The third way (which this current draft adopts) is what we can refer to as 'fake prehashing'; IKEv2 would generate the hash as current, but instead of running ML-DSA in prehash mode, we have ML-DSA sign it in pure mode as if it was the message. This is a violation of the spirit, if not the letter of FIPS 204. However, it is secure (assuming the hash function is strong), and fits in cleanly with both the existing IKEv2 architecture, and what crypto libraries provide. Because this doesn't have any practical downsides, we opted for this option.

### 5.2. ML-DSA Context

An additional feature that ML-DSA provides it allows the signer and the verifier to provide a 'context string'. The signature would verify only if the context strings that are provided by both the signer and the verifier match. The reason behind this is to ensure that if a public key is used for multiple purposes, a signature for one purpose cannot be used by an adversary in another. In our case, if the same certificate where used to sign both IKEv2 and TLS exchanges, an adversary could not possibly take the signature from an TLS exchange and try to use it within an IKEv2 exchange.

This really is not a necessary security safe guard; the messages that are actually signed in both cases are distinct enough that an adversary could not actually take advantage of this, even without the protection.

However, given that ML-DSA does provide such a service, and it appears that crypto libraries do support a nonempty context, we cannot see a reason not to use it.

## 6. IANA Considerations

This document has no IANA actions.

The additional OIDs that this uses have been defined by NIST and do not need to be registered by IANA

## 7. References

### 7.1. Normative References

[FIPS204] "Module-Lattice-Based Digital Signature Standard", NIST FIPS 204, August 2024, <https://doi.org/10.6028/NIST.FIPS.204>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

### 7.2. Informative References

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/rfc/rfc7296>.

[RFC7427] Kivinen, T. and J. Snyder, "Signature Authentication in the Internet Key Exchange Version 2 (IKEv2)", RFC 7427, DOI 10.17487/RFC7427, January 2015, <https://www.rfc-editor.org/rfc/rfc7427>.

[RFC8420] Nir, Y., "Using the Edwards-Curve Digital Signature Algorithm (EdDSA) in the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 8420, DOI 10.17487/RFC8420, August 2018, <https://www.rfc-editor.org/rfc/rfc8420>.

[RFC9593] Smyslov, V., "Announcing Supported Authentication Methods in the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 9593, DOI 10.17487/RFC9593, July 2024, <https://www.rfc-editor.org/rfc/rfc9593>.

## Acknowledgments

No acknowledgements yet

## Author’s Address

Scott Fluhrer
Cisco Systems
Email: sfluhrer@cisco.com