Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 September 2026

A. Mitra
S. S. Chakkaravarthy
A. Ghosh
D. P. VS
VIT-AP University

23 March 2026

ML-DSA for Web Authentication
draft-vitap-ml-dsa-webauthn-04

Abstract

This document describes implementation of Passwordless authentication in Web Authentication (WebAuthn) using Module-Lattice-Based Digital Signature Standard (ML-DSA), a Post-Quantum Cryptography (PQC) digital signature scheme defined in FIPS 204.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 September 2026.

Table of Contents

1. Introduction

This document describes how to use ML-DSA keys and signature as described in [FIPS-204] with [WebAuthn].

2. Terminology

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.

Some examples in this specification are truncated with "..." for readability.

3. Background on Post-Quantum Cryptography

This section describes a generic backround of Post-Quantum Cryptography, Web Authentication and Phishing Resistance. Post-Quantum Cryptography is defined to be a set of cryptographic algorithms that are not vulnerable against a malicious actor in possession of a large scale Quantum Computer. [FIPS-204] has described Module-Lattice-Based Digital Signature Algorithm which is not vulnerable to attacks involving a large-scale quantum computer.

3.1. Motivation for ML-DSA in WebAuthn

With the wide range adoption of phishing-resistant passwordless authentication standard like Security Keys, Passkeys and Device attestation following the FIDO2 Specifications, the adopted cryptographic standards for authentication have been primarily ES256 (Elliptic Curve Digital Signature Algorithm with SHA-256) and RS256 (RSA with SHA-256) as defined in [FIPS-186-5]. Though most authenticators support other algorithms as well, the widely used default algorithms- ES256 and RS256 are deemed to be insecure against adversaries employing large-scale quantum computers.

Hence, the adoption of ML-DSA for WebAuthn would be necessary to secure digital identities and accounts from such adversaries.

4. ML-DSA Integration in WebAuthn

This section describes the implementation of WebAuthn with ML-DSA.

4.1. ML-DSA Key Representation in COSE

[I-D.draft-ietf-cose-dilithium-05] describes CBOR Object Signing and Encryption (COSE) Serialization for ML-DSA. It is to be noted that the COSE representation of only ’Public key’ or ’Verifying key’ is used in WebAuthn. Hence, the COSE Representation of private keys are beyond the scope of this document.

ML-DSA Signature Scheme is parameterized to support different security levels. In this document, the abbreviations of ML-DSA-44, ML-DSA-65 and ML-DSA-87 are used to refer to ML-DSA with the parameter choices given in Table 1 of FIPS-204.

This document requests the registration of the ML-DSA algorithms in [IANA.cose] as mentioned in [I-D.draft-ietf-cose-dilithium-05] as :

Name	value	Description
ML-DSA-44	TBD (requested assignment -48)	CBOR Object Signing Algorithm for ML-DSA-44
ML-DSA-65	TBD (requested assignment -49)	CBOR Object Signing Algorithm for ML-DSA-65
ML-DSA-87	TBD (requested assignment -50)	CBOR Object Signing Algorithm for ML-DSA-87

Table 1: COSE algorithms for ML-DSA

In accordance with the Algorithm Key Paid Type section of [I-D.draft-ietf-cose-dilithium-05], when present in AKP Keys, the "pub" parameter has the following constraints:

The "pub" parameter is the ML-DSA public key, as described in Section 5.3 of FIPS-204.

The size of "pub", and the associated signature for each of these algorithms is defined in Table 2 of FIPS-204, and repeated here for convenience:

Algorithm	Private Key	Public Key	Signature Size
ML-DSA-44	2560	1312	2420
ML-DSA-65	4032	1952	3309
ML-DSA-87	4896	2592	4627

Table 2: Sizes (in bytes) of keys and signatures of ML-DSA

These algorithms are used to produce signatures as described in Algorithm 2 of FIPS-204.

Signatures are encoded as bytestrings using the algorithms defined in Section 7.2 of FIPS-204.

4.2. Signature Generation and Verification

Signature generation is done in accordance with Section 5.2 of FIPS-204. Signature Verification is done in accordance with Section 5.3 of FIPS-204.

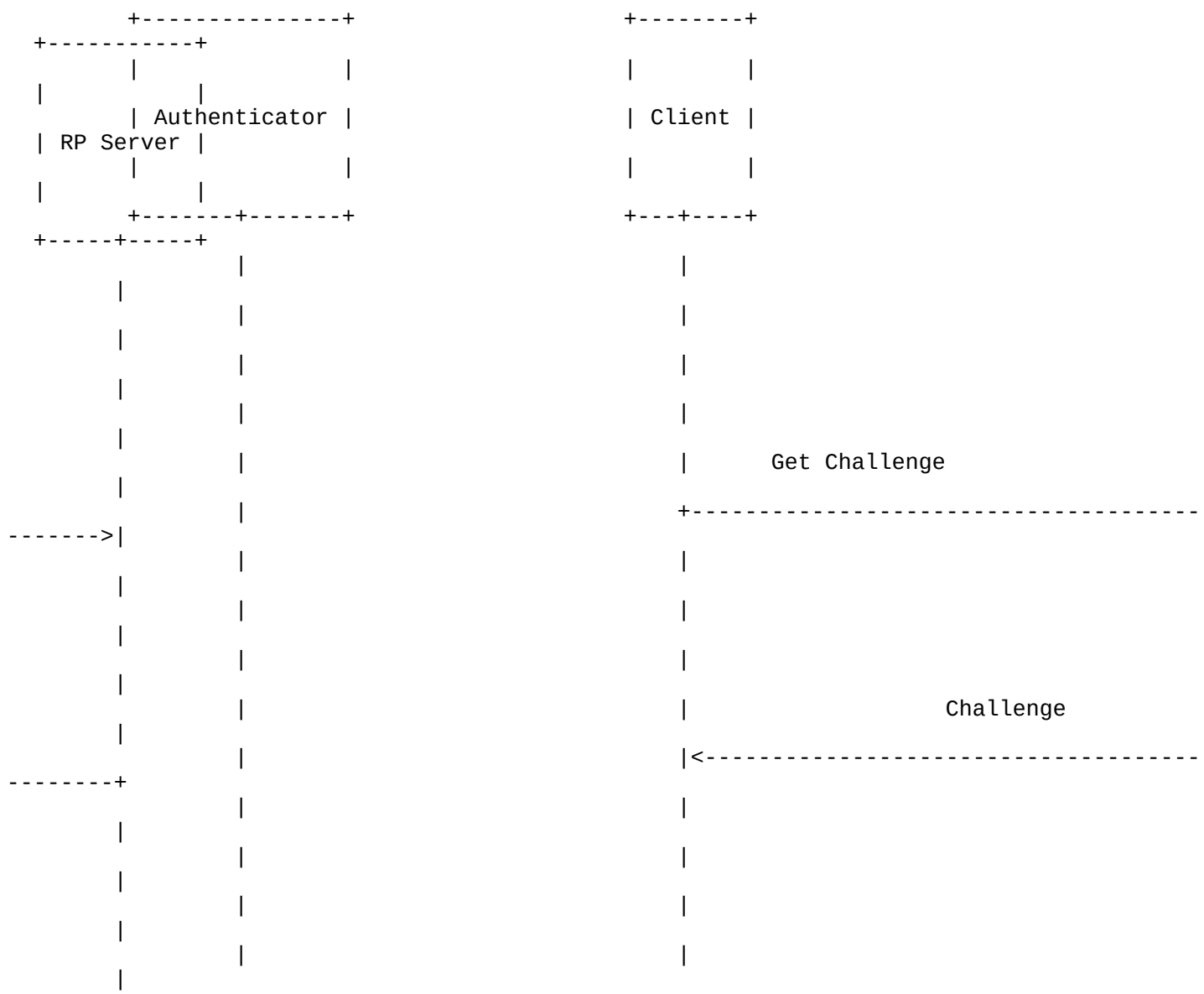
If small keys or signature is needed, ML-DSA might not be the ideal option. Furthermore, in usage scenarios expecting swifter processing, ML-DSA-87 might not be the best option. Therefore, using ML-DSA-44 and ML-DSA-65 with WebAuthn is advised.

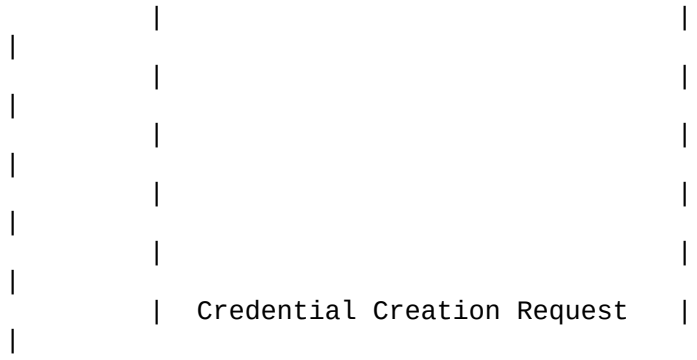
5. Authenticator Behavior

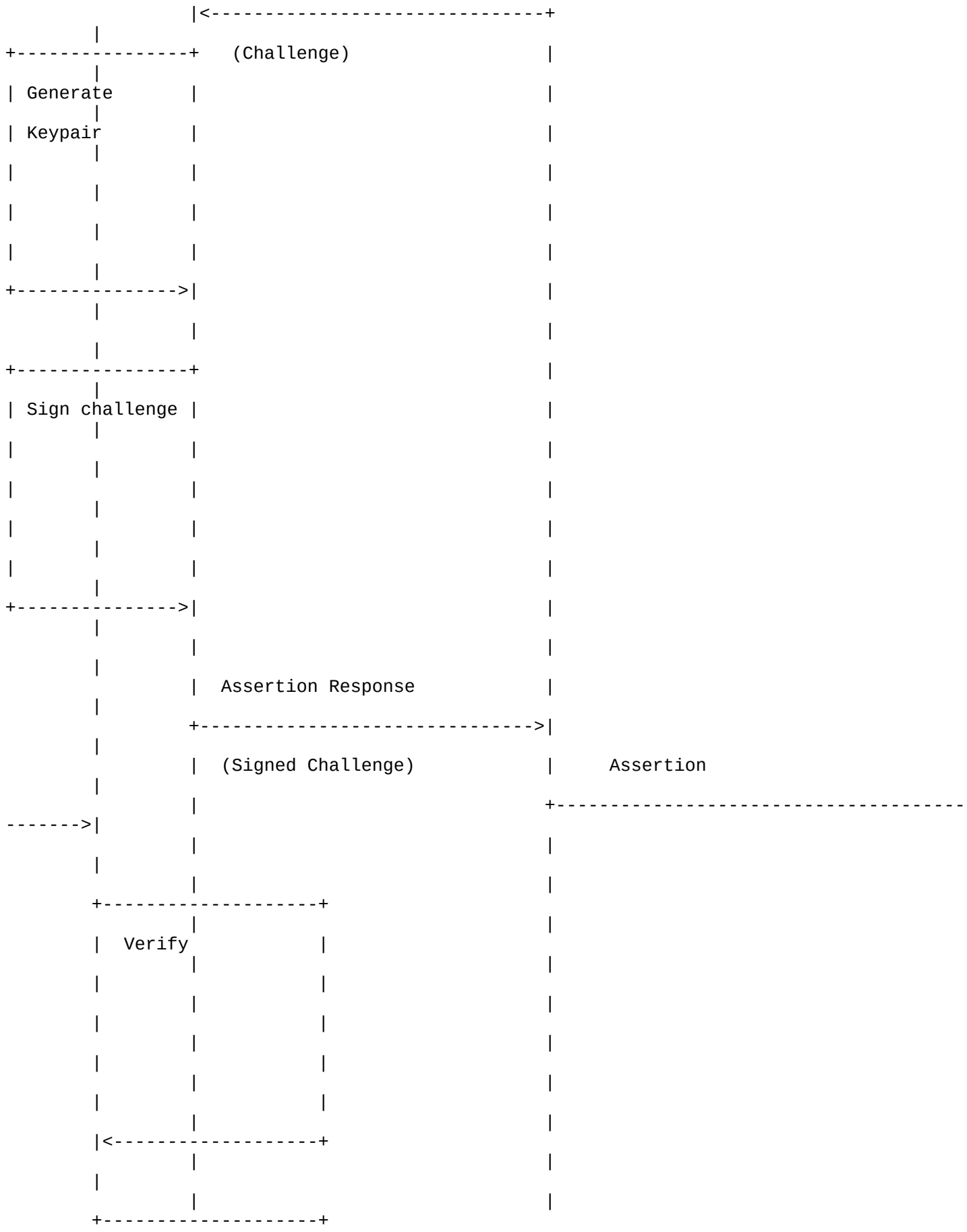
This section describes how an authenticator, roaming or platform would implement ML-DSA.

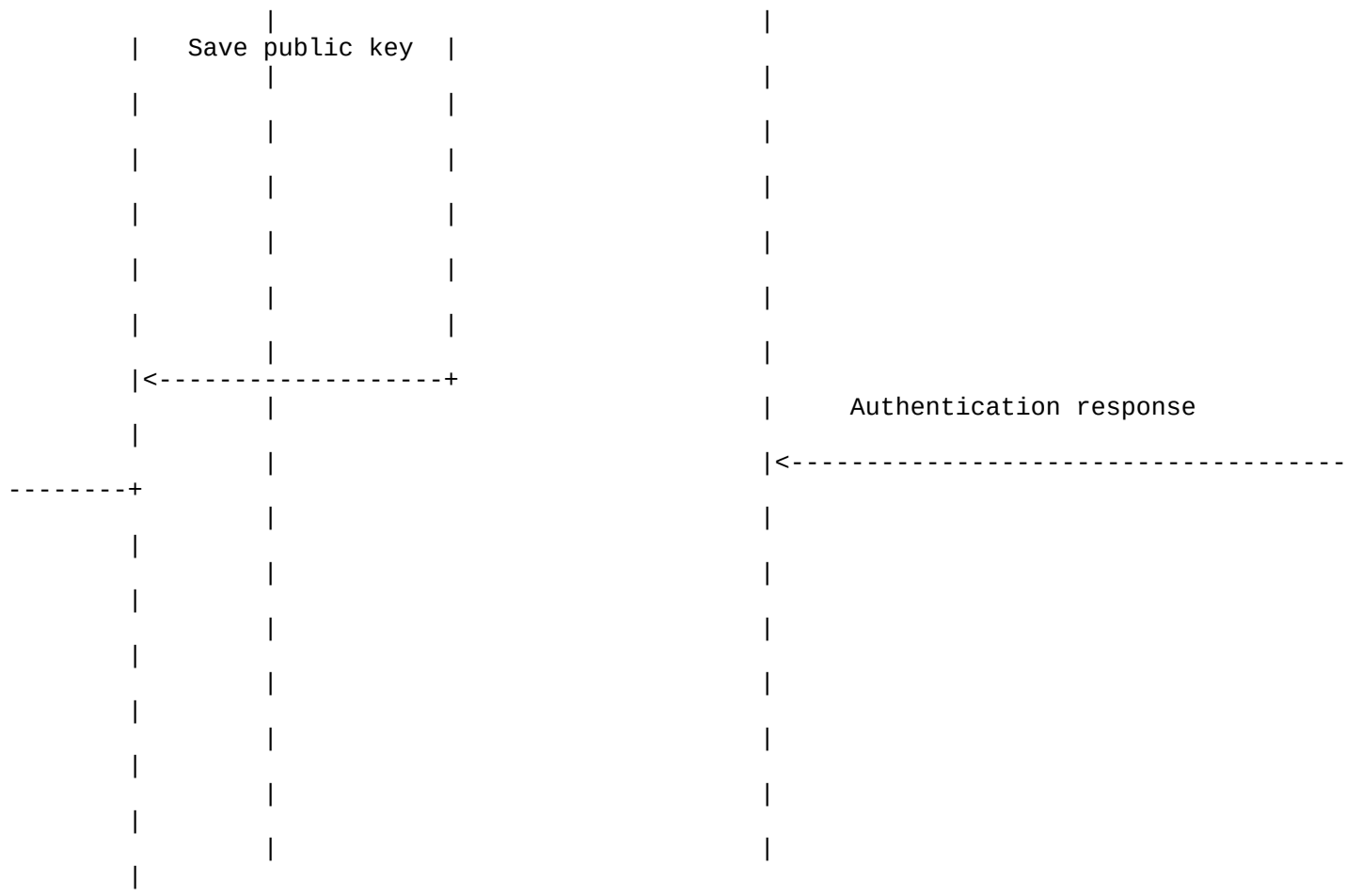
The authenticator MUST have a secure storage for storing cryptographic secrets which SHOULD NOT be able to export the secrets in an unauthorized manner.

5.1. Credential Creation




Credential Creation Request


(Challenge)
Generate Keypair
Sign challenge
Assertion Response
(Signed Challenge)
Assertion
Verify


Save public key
Authentication response

[WebAuthn] defines the credential creation API. This API takes a public key credential creation object, containing a cryptographic challenge, the Relying Party ID (RP ID), User details, and public key credential parameters. The public key credential parameters define the accepted algorithms.

An example of public key credential creation object is:

```
{
  challenge: new Uint8Array([215, 150, 119, 213, 215, 247, 188, 15, 142, 10, 53, 135, 17,
 205, 130, 133, 158, 32, 113, 0, 62, 67, 112, 191, 123, 180, 224, 151, 233, 114, 68, 225]
),
  rp: { id: "example.com", name: "Example Corporation" },
  user: {
    id: new Uint8Array([79, 252, 83, 72, 214, 7, 89, 26]),
    name: "johndoe",
    displayName: "John Doe",
  },
  pubKeyCredParams: [{ type: "public-key", alg: -7 }, { type: "public-key", alg: -49 }],
}
```

The web application invokes the Navigator.credentials.create() function with the Public Key Credential Creation Object. The client web browser, or an application invokes the authenticator API defined in [CTAP]. The public key credential creation object is CBOR encoded and sent to the authenticator device over a chosen transport method which includes but is not limited to USB HID, BLE and NFC.

An example of CBOR Encoded Public Key Credential Creation object is:

```
h’A50158201637B26333915747DBDC6C630C0165405A64939AE8F6E4FC39414F853F702F1602A2626964696C6
F63616C686F7374646E616D656B44656D6F207365727665203A3626964504EC1D4219F294FB4A0BC0CD29D48
5AFC646E616D6566615F757365726B646973706C61794E616D6567412E20557365720481A263616C67382F647
47970656A7075626C69632D6B657907A1627576F5’
```

The authenticator MUST verify whether any credential mentioned in the list of "excludeCredentials" in the public key credential creation object is already present on the authenticator, and in such cases it will return the error code in accordance with [CTAP]. Further authenticator MUST perform all additional checks involving authenticator PIN, User presence, user verification etc in accordance with Section 5.1 of CTAP.

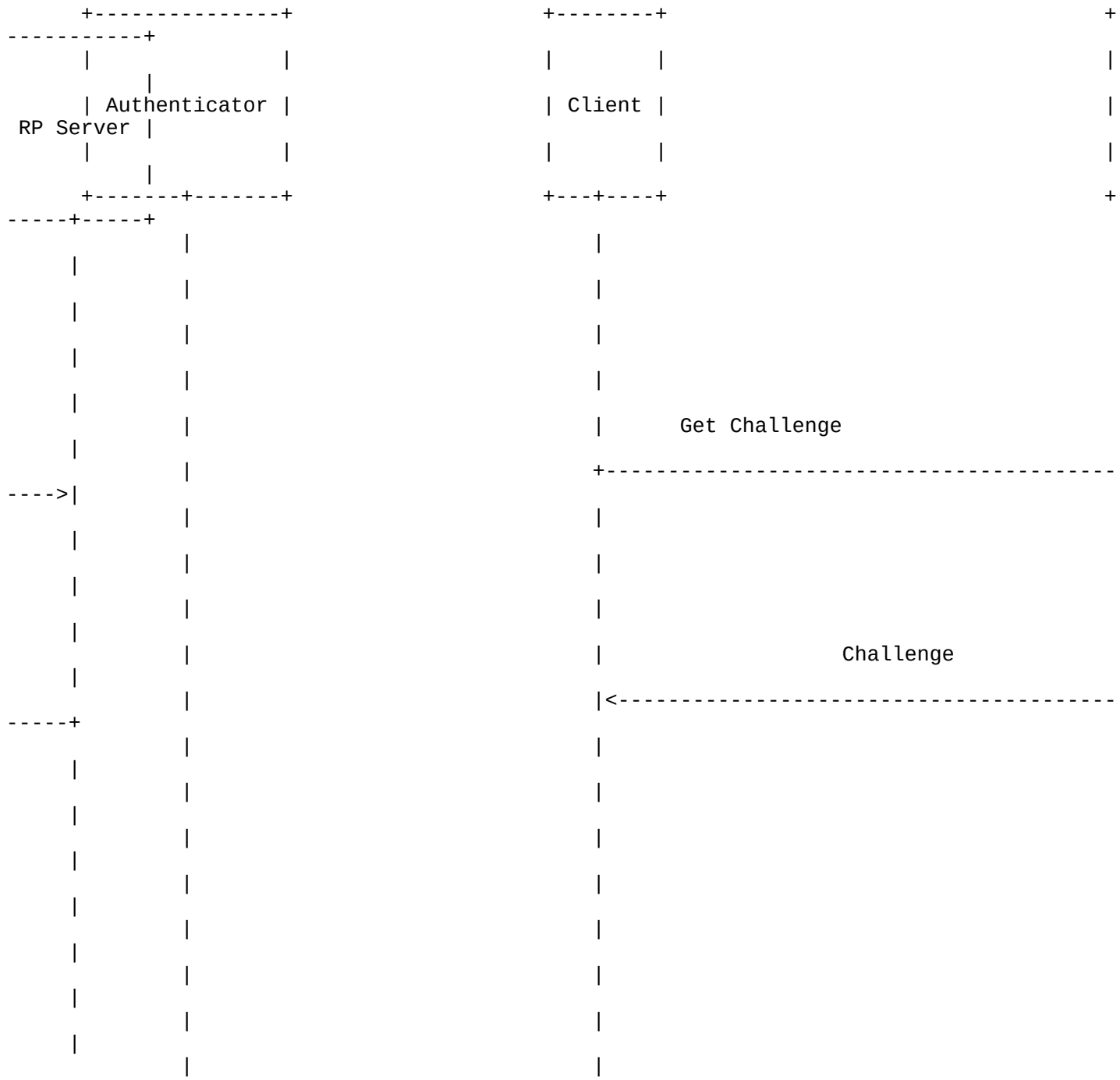
The authenticator generates ML-DSA keypair in accordance with Section 5.1 of FIPS 204 and unique Key ID. The public key is COSE encoded following [I-D.draft-ietf-cose-dilithium-05] and is a part of the attestedCredentialData.

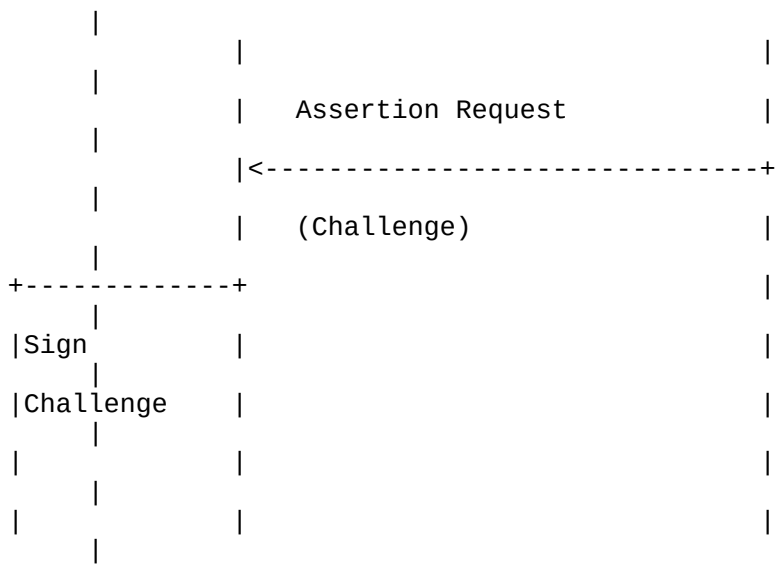
After passing all checks in accordance with section 5.1 of CTAP, the authenticator SHOULD create the attestation statement. The authData is created in accordance with WebAuthn and CTAP specifications and the clientDataHash is appended to it. This is signed with the private key of the generated keypair. The attestation statement is generated in accordance with CTAP and WebAuthn.

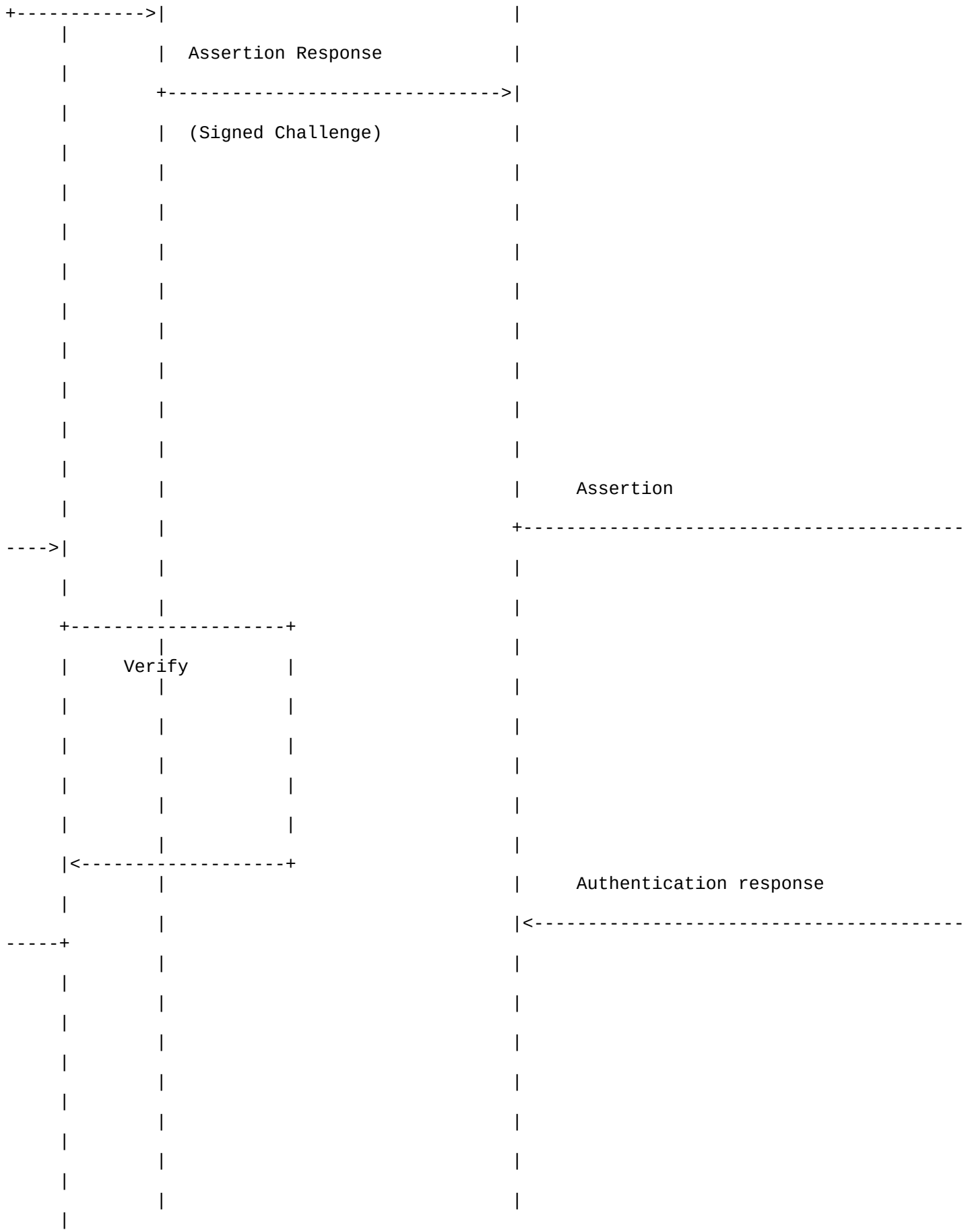
The ML-DSA private key is to be stored in accordance with the "Storage security and Key management" section of this document.

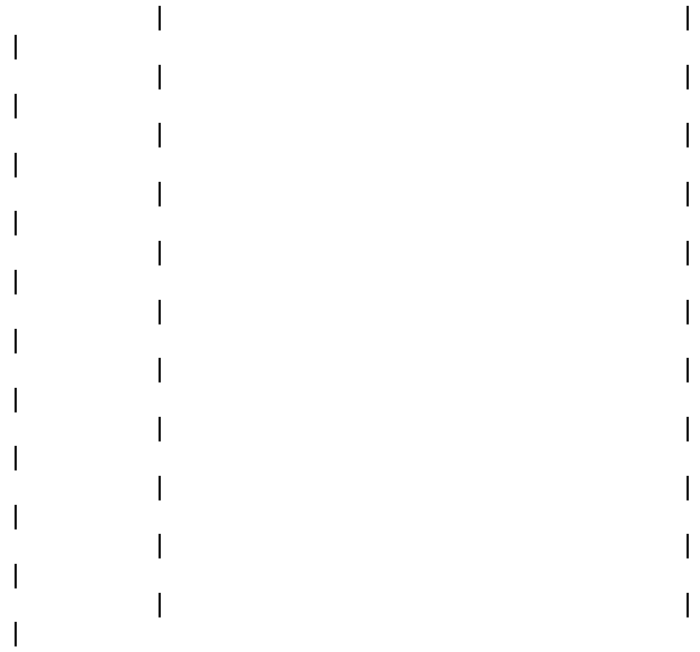
The attestation statement is encoded in CBOR and returned to the client application via the same transport method.

5.2. Authentication Flow




Assertion Request
(Challenge)
Sign
Challenge


Assertion Response
(Signed Challenge)
Assertion
Verify
Authentication response



[WebAuthn] defines the credential request API. This API takes a public key credential request object, containing a cryptographic challenge, the Relying Party ID (RP ID) and optionally a list of allowed credentials.

An example of public key credential request object is:

```
{
  challenge: new Uint8Array([215, 150, 119, 213, 215, 247, 188, 15, 142, 10, 53, 135, 17,
 205, 130, 133, 158, 32, 113, 0, 62, 67, 112, 191, 123, 180, 224, 151, 233, 114, 68, 225]
),
  rpId: "example.com",
}
```

The web application invokes the Navigator.credentials.get() function with the Public Key Credential Request Object. The client web browser, or an application invokes the authenticator API defined in [CTAP]. The public key credential request object is CBOR encoded and sent to the authenticator device over a chosen transport method which includes but is not limited to USB HID, BLE and NFC.

If a list of allowedCredentials is present, the authenticator must discover credentials bound to the same RP ID which is present in the list. If no such credential is present, it will return the error code in accordance with [CTAP]. Further authenticator MUST perform all additional checks involving authenticator PIN, User presence, user verification etc in accordance with Section 5.2 of CTAP.

The authenticator, on discovering a suitable credential for authentication SHOULD verify the algorithm. If it is not ML-DSA, the authenticator SHALL behave in accordance with the "Backward Compatibility Considerations" section of this document.

The credential is retrieved in accordance with the "Storage security and Key management" section of this document.

After passing all checks in accordance with section 5.2 of CTAP, the authenticator SHOULD create the assertion statement. The authData is created in accordance with WebAuthn and CTAP specifications and the clientDataHash is appended to it. This is signed with the decrypted ML-DSA private key. The assertion response is generated in accordance with CTAP and WebAuthn.

The assertion response is encoded in CBOR and returned to the client application via the same transport method.

All memory addresses used to handle the ML-DSA private key is immediately zeroized.

The authenticator getNextAssertion() function specification is to be similarly implemented in accordance with [CTAP].

5.3. Backward Compatibility Consideration

The authenticator SHOULD choose the algorithm to be used in accordance with CTAP, and hence not choose ML-DSA if not supported by the RP.

6. Client and Platform Considerations

This section describes the considerations about the Client and Platform.

6.1. COSE Algorithm Support in WebAuthn Clients

The CTAP implementations on client, for example Windows Hello for Microsoft Windows based systems, iCloud Keychain for Apple systems, Google Password Manager, Dashlane, OnePassword and other browser based implementations for Linux SHOULD have support for ML-DSA Algorithms in accordance with COSE. Further, Platform Authenticators for such devices are RECOMMENDED to have support for ML-DSA Key Generation and signing in accordance with this document.

6.2. Handling large signatures and keys

It is considered that the chosen transport methods including but not limited to USB HID, BLE and NFC are able to perform exchanges of the CBOR encoded data which may be often over 2000 bytes. The use of attestion certificates may increase the length even more.

6.3. Error Handling and Fallback mechanisms

In case of errors involving ML-DSA key generation and signing, the authenticator may fallback to using ES256 or RS256 algoritms. In case of errors involving communication with the client, the authenticator may handle it in accordance with [CTAP].

7. Attestation Considerations

Using Post Quantum Crptography for creating attestation certificates for credentials implies the presence of additional ML-DSA signatures and public keys in the x.509 certificate, depending upon the attestation format, defined in the [WebAuthn] flow. A second ML-DSA-44 Signature size is around 2420 bytes and a public key is around 1312 bytes, and bigger for others. On the other hand, [CTAP] using a 7-bit sequence continuation packet numbers for CTAP-HID constrains it to have a maximum of 129 frames only. CTAP-HID frame can be of size 64 bytes at max. A makeCredential response with a valid attestation certificate would contain atleast 1312+2420+1312+2420=7464 bytes.

The first frame has 7 bytes of header (4 byte channel identifier, 1 byte CMD, 2 bytes BCNT), while continuation frames have 5 bytes of header (4 bytes of channel identifier, 1 byte of sequence number).

Further, the first frame will have one byte of status code. The makeCredential response will contain the authData. This further contains 32 bytes of RP ID Hash, 1 byte of flags, 4 bytes of sign count and variable length attestedCredential data. The attestedCredential data further contains 16 bytes of AAGUID, 2 bytes of credential ID length. The credential ID is to be atleast 16 bytes, followed by the public key.

This covers a total of 8182 bytes, leaving only 74 bytes for CBOR encoding, and the full x.509 certificate, including fields like the version, subjects, certificate chain and so on. Further, when other algorithms like ML-DSA-65 or higher is used, the available bytes in CTAP-HID would not suffice for the attestation certificate.

7.1. WebAuthn Considerations

Due to the attestation certificate size limitations, Web Authentication standard is requested to recognize an attestation format ’minimal’.

The syntax of Minimal Attestation is defined by:

```
$$attStmtType //= (
                      fmt: "minimal",
                      attStmt: minimalStmtFormat,
                  )

minimalStmtFormat = {
                       alg: COSEAlgorithmIdentifier,
                       keyid: bytes,
                       sig: bytes,
                    }
```

sig here is a signature over the authenticatorData and clientDataHash. The authenticator produces the sig by concatenating authenticatorData and clientDataHash, and signing the result using an attestation private key selected through an authenticator-specific mechanism.

keyid is an Identifier that identifies the key used to sign. It is a 16-byte unique identifier.

7.1.1. Verification and FIDO MDS Database Considerations

The FIDO MDS database is requested to maintain the set of Public Key/ Verifying key certificates, identifiable by the AAGUID and the KeyId together. The RP may verify the attestation signature with the public key identified by the AAGUID and KeyID from the FIDO MDS database. Enterprise based implementations or implementations requiring self-attestation without FIDO MDS database MAY maintain their on keystores instead of the FIDO MDS database.

8. Security Considerations

The security considerations of [RFC9053] applies to this specification as well.

A detailed security analysis of ML-DSA is beyond the scope of this specification, see [FIPS-204] for additional details.

8.1. Resistance to Quantum Attacks

See [FIPS-204] for details on resistance to quantum attacks.

8.2. Storage security and Key management

It is to be noted that at the time of writing this draft, there is no suitable hardware security module (HSM), trusted platform module (TPM), Secure Element (SE), Trusted Execution Environment (TEE) with native support for ML-DSA. Hence, secure credential storage is a challenge. To overcome the same, the ML-DSA keys, also referred to as credentials, MUST be encrypted with Advanced Ecnryption Standard (AES), which is a Post Quantum Symmetric encryption algorithm in Galosis Counter Mode (GCM) with 256-bit keys.

The AES Keys MUST be generated and stored in secure storage, which may include a HSM, TPM, SE or TEE. The ML-DSA Keys may be generated on the standard computing environment, outside the secure storage. The ML-DSA Credential MUST be encrypted by AES as described above before being written to the Flash memory or Disk. Conversely, the same MUST be decrypted by AES in the secure storage before being used.

Any memory location, pointer or heap that has been used to handle the ML-DSA Credentials MUST be zeroized immediately after the operation is performed.

This section is to be updated when suitable secure storage modules supporting ML-DSA becomes widely available.

8.3. Implementation Best Practices

If the amount of space in the secure storage permits, each ML-DSA Private key is RECOMMENDED to be encrypted with unique AES keys. * Prior to storage, each ML-DSA private key is to be encrypted independently using a distinct AES encryption key. * To guarantee robust encryption, the AES key size must be at least AES-256. * To avoid unwanted access, encrypted keys ought to be kept in Hardware Security Modules (HSMs), Secure Elements (SEs), or Trusted Platform Modules (TPMs). * NIST SP 800-57 recommendations should be followed by key management to provide secure AES key lifecycle management (creation, storage, rotation, and disposal). * Only in a trusted execution environment (TEE) or secure enclave should the private key be decrypted in order to avoid memory leakage.

9. IANA Considerations

9.1. Additions to Existing Registries

This document requests the registration of the ML-DSA entries to the COSE Algorithm Registry as mentioned in [I-D.draft-ietf-cose-dilithium-05].

10. References

10.1. Normative References

[I-D.draft-ietf-cose-dilithium-05]
Prorock, M., Steele, O., Misoczki, R., Osborne, M., and C. Cloostermans, "ML-DSA for JOSE and COSE", Work in Progress, Internet-Draft, draft-ietf-cose-dilithium-05, 18 December 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-dilithium-05>.

[IANA.cose]
IANA, "CBOR Object Signing and Encryption (COSE)", <https://www.iana.org/assignments/cose>.

[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>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[RFC9679] Isobe, K., Tschofenig, H., and O. Steele, "CBOR Object Signing and Encryption (COSE) Key Thumbprint", RFC 9679, DOI 10.17487/RFC9679, December 2024, <https://www.rfc-editor.org/rfc/rfc9679>.

10.2. Informative References

[CTAP] "Client To Authenticator Protocol (CTAP)", n.d., <https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-client-to-authenticator-protocol-v2.0-id-20180227.html>.

[FIPS-186-5] "Digital Signature Standard (DSS)", n.d., <https://doi.org/10.6028/NIST.FIPS.186-5>.

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

[I-D.draft-ietf-cose-key-thumbprint] Isobe, K., Tschofenig, H., and O. Steele, "CBOR Object Signing and Encryption (COSE) Key Thumbprint", Work in Progress, Internet-Draft, draft-ietf-cose-key-thumbprint-06, 6 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-key-thumbprint-06>.

[I-D.draft-ietf-lamps-dilithium-certificates] Massimo, J., Kampanakis, P., Turner, S., and B. Westerbaan, "Internet X.509 Public Key Infrastructure - Algorithm Identifiers for the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)", Work in Progress, Internet-Draft, draft-ietf-lamps-dilithium-certificates-13, 30 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-dilithium-certificates-13>.

[SP-500-57] "Audit and Evaluation of Computer Security II: System Vulnerabilities and Controls", n.d., <https://doi.org/10.6028/NBS.SP.500-57>.

[WebAuthn] "Web Authentication: An API for accessing Public Key Credentials", n.d., <https://www.w3.org/TR/webauthn-2>.

Acknowledgments

We express our sincere gratitude to Dr. S. V. Kota Reddy, Vice Chancellor, VIT-AP University, for his unwavering support and encouragement throughout this research. We also extend our heartfelt thanks to Dr. Jagadish Chandra Mudiganti, Registrar, VIT-AP University, for facilitating a conducive research environment. Our appreciation goes to Dr. Hari Seetha, Director, Centre of Excellence, Artificial Intelligence and Robotics (AIR), VIT-AP University, for her invaluable guidance and insightful discussions that significantly contributed to this work.

We also acknowledge Indominus Labs Private Limited and Digital Fortress Private Limited for their generous support, which played a crucial role in the successful execution of this research.

Authors' Addresses

Aditya Mitra
VIT-AP University
Email: adityamitra5102@gmail.com

Sibi S. Chakkaravarthy
VIT-AP University
Email: chakkaravarthy.sibi@vitap.ac.in

Anisha Ghosh
VIT-AP University
Email: ghoshanisha2002@gmail.com

Devi Priya VS
VIT-AP University
Email: priya.21phd7042@vitap.ac.in